Implementación de un algoritmo basado en la naturaleza para la predicción de la estructura péptido-proteína conociendo el sitio de unión.

Enrique Delgado Rodríguez

Máster Universitario en Bioinformática y Bioestadística

Programación para la bioinformática

Tutor: Brian Jiménez García

junio del 2018

A mi madre, por creer en mí sin importar nada ni
lo descabellado de mis ideas.

A Pilar, a Manolo, a Araceli, a Rubén, a Inma, a Esther...
por aguantarme hablar durante horas de mis cucos.

A Brian, por su ayuda y apoyo durante todo este largo proceso.

A Xema, porque sin él esto nunca habría sido posible.

A todos... Gracias.

GNU Free Documentation License (GNU FDL)

FICHA DEL TRABAJO FINAL

Título del trabajo:	Implementación de un algoritmo basado en la naturaleza para la predicción de la estructura péptido-proteína conociendo el sitio de unión.
Nombre del autor:	Enrique Delgado Rodríguez
Nombre del consultor/a:	Brian Jiménez García
Fecha de entrega (mm/aaaa):	06/2018
Titulación:	Máster en Bioinformática y Bioestadística
Área del Trabajo Final:	Estadística y bioinformática
Idioma del trabajo:	Español
Palabras clave	proteína, péptido, algoritmo basado en la naturaleza, reconstrucción

Resumen del Trabajo (máximo 250 palabras): Con la finalidad, contexto de aplicación, metodología, resultados i conclusiones del trabajo.

Seidh es una librería que implementa un método ab initio, no-ciego, “fine-grained” para la predicción de la estructura de un péptido unido a una proteína, partiendo de la estructura de la proteína y del punto de unión. Contextualizado dentro de la biología estructural computacional, esta librería constituye un aporte a uno de los retos actuales de la bioinformática: la predicción de estructuras de biomoléculas.

Seidh implementa un algoritmo basado en la naturaleza: el vuelo del ave cuco, un ave cuyos movimientos vienen determinados por la distribución de frecuencias de Lévy. Al tratarse de la implementación de un método ab initio, se emplea una función de evaluación basada en las interacciones que rigen la física molecular, considerando radios atómicos, fuerzas de van der Waals, distancias, potenciales electrostáticos, etc. Posee funciones para detectar y prevenir

colisiones, una mejora del rendimiento basada en computación paralela y un sistema de refinado de resultados basado en clustering de conformaciones similares, entre otras características.

La librería, exportable como pone de manifiesto su implementación en un servidor web Flask, responde al contexto de partida con la posibilidad de variar su configuración y amoldarla a distintas situaciones de uso, demostrando así ser una solución interesante para la tarea propuesta.

Abstract (in English, 250 words or less):

Seidh is a library which implements an ab initio method, unblinded, "fine-grained" for the structure prediction of protein binded peptides, parting from the protein structure itself and the binding point. Contextualized within computational structural biology, this library constitutes a contribution to one of the current challenges of bioinformatics: the prediction of biomolecular structures.

Seidh implements a nature-based algorithm: the flight of the cuckoo bird, whose movements are determined by a Lévy distribution of frequencies. Being an ab initio method implementation, it employs a scoring function based on the interactions that rule molecular physics, considering atomic radii, Van der Waals forces, distances, electrostatic potentials, etc. It counts also with functions to detect and prevent collisions, an improvement in performance through parallel computation and a result refinement system based on similar conformation clustering, among other characteristics.

The library, exportable as proved by its implementation within a Flask webserver, answers to the given starting context with the possibility of altering its configuration and fit different scenarios, thus rendering itself as an interesting solution for the proposed task.

Índice

Lista de figuras

Índice de tablas

1. Introducción

1.1 Contexto y justificación del Trabajo

Las interacciones entre proteínas y péptidos representan una gran fracción de todos los procesos bioquímicos que tienen lugar en cualquier organismo vivo. Hoy en día, en pleno desarrollo de la medicina personalizada, el conocimiento sobre cómo se acoplan las proteínas y los péptidos resulta de vital importancia para el desarrollo de nuevos tratamientos y terapias encaminadas a la cura de enfermedades genéticas, oncológicas, autoinmunes y un largo etcétera.

Tradicionalmente, las estructuras de unión proteína-péptido eran recuperadas a través de métodos experimentales, como la microscopía electrónica de alta resolución, la resonancia magnética nuclear o la cristalografía de rayos X. Estos resultan costosos, lo que reduce enormemente sus posibilidades de uso.

En la actualidad, los recursos bioinformáticos para predecir estas estructuras no son infalibles: no obstante, en su diversidad pueden encontrarse soluciones para problemas más o menos específicos que justifiquen el empleo de estos métodos.

Algunos de estos enfoques incluyen aprendizaje automático (sea por algoritmos tradicionales, como NearestNeighbour o por métodos más modernos, como redes neuronales y máquinas de vectores de soporte), algoritmos de búsqueda, o algoritmos basados en la naturaleza.

Estos últimos, también llamados algoritmos inspirados en la naturaleza, aprovechan el comportamiento observado de un fenómeno natural (la organización de las hormigas en una colonia, el orden en un enjambre, el vuelo de los pájaros…) para plantear un conjunto de pasos para solucionar un problema computacional.

Estos métodos emulan de esta manera procedimientos pulidos a lo largo de toda la evolución, permitiendo así la solución de problemas de difícil solución.

Es preciso mencionar que a menudo las soluciones propuestas por estos sistemas no son perfectas: pueden darse estructuras no observadas en la naturaleza, o bien no del todo exactas. Esto se debe a que, en última instancia, el sistema diseñado trata de reproducir las condiciones que se dan en la

naturaleza (incluyendo, en este caso, elementos como las características de cada núcleo atómico, las fuerzas de Van der Waals implicadas, o el potencial electrostático de cada átomo), lo que solo permite dar una aproximación a la realidad.

Con todo, una buena definición de las limitaciones del sistema puede procurar, si bien no soluciones perfectas, otras que son lo suficientemente buenas como para permitir estudiar el comportamiento de un péptido ante una proteína.

Asimismo, también sería posible estimar el comportamiento de la estructura al modificar el péptido, lo que revestiría especial importancia en el desarrollo de un fármaco nuevo. Es el caso de los péptidos que se compongan de una o más partes.

1.2 Objetivos del Trabajo

El presente proyecto apunta fundamentalmente a la implementación de un algoritmo basado en la naturaleza para la predicción de la estructura proteína-péptido, conociendo el sitio de unión. Lo mínimo exigible en este aspecto, sería una librería basada en Python capaz de partir de un fichero PDB con una proteína, una secuencia de aminoácidos, y un segundo fichero PDB con el punto de unión; para devolver un fichero con la estructura del aminoácido una vez acoplado a la proteína.

Para ello, se plantean objetivos específicos de cara al desarrollo y testeo del mismo, así como su disponibilidad a través de un servidor web. Para estos objetivos específicos se derivan asimismo tareas concretas. Los objetivos específicos están sujetos a cambios.

1.3. Objetivos específicos

1. Desarrollo de la librería Seidh
    a. Identificación del algoritmo basado en la naturaleza y justificación.
    b. Implementación de librerías auxiliares para el manejo de ficheros y estructuras de proteínas y péptidos.
    c. Implementación del algoritmo.
    d. Diseño de un banco de pruebas.
    e. Optimización.
2. Implementación de Seidh
    a. Diseño del servidor Flask.
    b. Implementación de la librería Seidh y testeo.

1.4. Enfoque y método seguido

La predicción de estructuras tridimensionales del complejo proteína-péptido (también proteína-proteína) es un sector muy concreto de la biología computacional y estructural. Implica un profundo conocimiento de las interacciones entre las moléculas implicadas, y cómo estas pueden afectar a las estructuras secundaria y terciaria de los complejos; además de un potente banco de recursos bioinformáticos para el manejo de esta información.

En la primera etapa de desarrollo del proyecto, se busca la implementación de la librería, nombrada Seidh, para poder ser utilizada en terminal. No es hasta la segunda fase cuando se implementará en un servidor web, basado en el framework python Flask.

En las siguientes líneas se describen los recursos implicados.

1.4.1. Software

- La librería será escrita en lenguaje python, apoyándose en su versión 2.7. Gran parte del resto de herramientas bioinformáticas están

escritas para esta misma versión, con lo que se asegura una buena compatibilidad con estas.

- Para el manejo de estructuras tridimensionales se emplearán ficheros PDB; así como las herramientas proporcionadas por BioPython para estos archivos.
- Los cálculos de la geometría de aminoácidos y moléculas serán llevados a cabo por la librería PeptideBuilder.
- Para el desarrollo del servidor web se ha elegido el framework Flask, que permite la implementación de soluciones sencillas y livianas, ideales para esta tarea.
- Para la gestión de colas en el servidor Flask se ha elegido la librería celery.
- Para el diseño de las interfaces del servidor web, se emplearán lenguajes de marcado como HTML y CSS, junto con el motor de plantillas de Python Jinja2. Opcionalmente, también puede emplearse algún código en javascript para mejorar la experiencia del usuario
-

1.4.2. Hardware

El desarrollo, implementación y testeo se llevará a cabo en un PC ASUS GL552VW-DM141 con las siguientes características:

- CPU: Core i7 6700HQ 2.6GHz
- GPU: Nvidia GTX960M 4GB
- RAM: 16GB DDR4 2133MHz, CL13
- HDD: 1000GB 7200RPM SATA3.0
- SSD: 240GB 550/330MB/s SATA3.0
- OS: Ubuntu 16.04.3 LTS sobre Windows 10 Pr (subsistema Linux)

1.4.3. Documentación

La librería Seidh posee una documentación sobre su uso básico, así como sus funciones internas.

1.5. Planificación del Trabajo

1.5.1. Etapa 0 [1 febrero – 20 febrero]

Ilustración 1. Diagrama de Gantt de la Etapa 0

En la etapa 0 se planteaba el proyecto desde una perspectiva naïve. Por esto, durante los primeros días se llevó a cabo una búsqueda bibliográfica para documentar y asentar la base teórica en biología estructural y el estado del arte de las soluciones publicadas hasta la fecha.

1.5.2. Etapa 1 [5 febrero – 21 marzo]

Ilustración 2. Diagrama de Gantt de la Etapa 1

En la etapa 1 se establecía la base de lo que sería la librería Seidh. Se diseñó un banco de pruebas que respondiera a los criterios de inicio (una proteína en formato PDB, un sitio de unión en formato PDB y la secuencia FASTA para la reconstrucción del péptido), y se definió la base funcional del algoritmo (lectura y escritura de ficheros PDB, y modificación de ángulos de los aminoácidos, para ir reproduciendo la estructura secundaria/terciaria).

1.5.3. Etapa 2 [22 abril – 7 mayo]

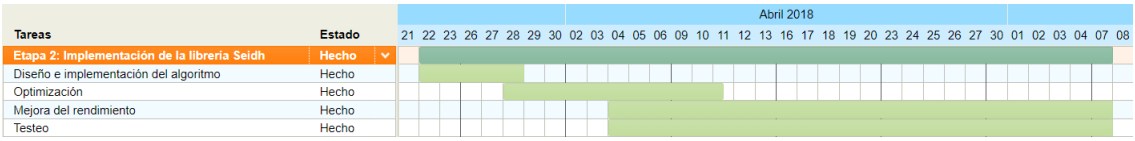

Ilustración 3. Diagrama de Gantt de la Etapa 2.

En la etapa 2 se diseña e implementa el algoritmo basado en la naturaleza, y comienza la fase fundamental del proyecto: su optimización y la mejora del rendimiento. Esta fase sufre modificaciones con respecto a la planificación original, que la estimaban acabada a finales de abril. Esto se debe a la detección

de mejoras como la implementación de la librería multiprocessing y la detección de clústeres de predicciones similares.

1.5.4. Etapa 3 [20 abril – 25 mayo]

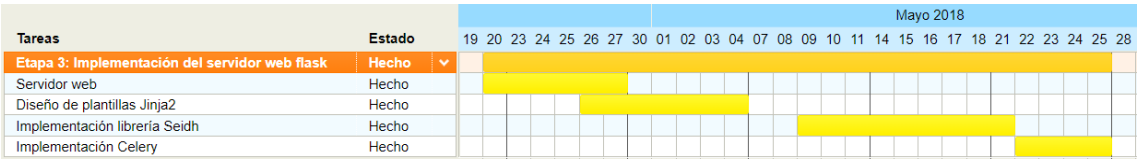

Ilustración 4. Diagrama de Gantt de la Etapa 3

En la etapa 3, finalmente, se comienza con la implementación de la librería Seidh en un servidor web flask. Si bien esta fase comienza a finales de abril, la implementación de la librería Seidh se ve retrasada por lo especificado en la Etapa 2. Finalmente, la etapa 3 concluye con la implementación de celery: una librería Python diseñada para la gestión de colas de tareas en segundo plano.

1.5.5. Etapa 4 [22 abril – 7 junio]

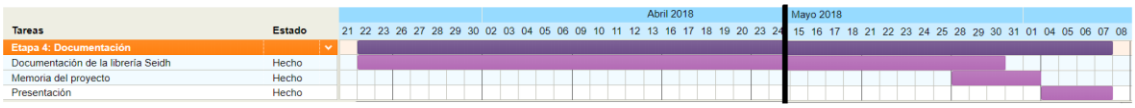

Ilustración 5. Diagrama de Gantt de la Etapa 4

En la etapa 4 se define la documentación del proyecto: tanto la presente memoria como los documentos que describen el funcionamiento de la librería Seidh y del servidor web flask.

1.6. Breve sumario de productos obtenidos

Documentos de desarrollo del trabajo:

1. **Estado del arte de la predicción del complejo proteína péptido mediante métodos ab initio**: este documento sirvió para establecer las bases del punto de partida de la librería Seidh.
2. **Evaluación continua de la UOC**
    a. **PEC0.** Definición del trabajo, detalle sobre las líneas a desarrollar, bibliografía de apoyo.
    b. **PEC1.** Definición de objetivos, planteamiento de la metodología, planificación del trabajo.

c. **PEC2.** Informe de seguimiento. Primeros resultados de la librería Seidh, identificación de problemas y propuesta de resolución. Identificación de aspectos a mejorar.

d. **PEC3.** Informe de seguimiento. Implementación de mejoras, detalle sobre la identificación y resolución de problemas antes expuestos.

Código del proyecto:

1. **Librería Seidh**: la librería python Seidh ofrece una solución sencilla para la predicción de estructuras de péptidos ligados a proteínas conociendo el sitio de unión. Además del algoritmo basado en la naturaleza, contiene una serie de herramientas que facilitan el trabajo con ficheros PDB y secuencias FASTA, derivados de la librería BioPython.
2. **Servidor web**: la librería Seidh se ha implementado exitosamente en un servidor web basado en flask, un framework de Python especialmente diseñado para la creación de páginas web y servidores. El servidor permite el almacenamiento de los ficheros subidos por el usuario, la predicción de las estructuras de los péptidos ligados a proteínas conociendo el sitio de unión, y la gestión de la cola de tareas mediante la librería celery.

1.7. Breve descripción de los otros capítulos de la memoria

En las páginas siguientes se detallan los siguientes capítulos:

1.7.1. Capítulo 2: Predicción de estructuras proteína-péptido y algoritmos basados en la naturaleza

En este capítulo se recoge el fundamento teórico que sustenta la metodología empleada al implementar un algoritmo basado en la naturaleza para la predicción de estructuras. Además, se resume el estado del arte de la cuestión, ofreciendo un pequeño catálogo de otras soluciones y sus características.

1.7.2. Capítulo 3: Diseño de Seidh

En este capítulo se recoge el fundamento operativo de Seidh y el algoritmo del cuco, el algoritmo basado en la naturaleza en el que se inspira, así como las restricciones que impone.

1.7.3. Capítulo 4: Revisión de las versiones de Seidh

Antes de su primera versión estable, considerada la primera reléase, Seidh pasó por cinco etapas de desarrollo mientras se realizaban pruebas y se obtenían fortalezas y debilidades. En este capítulo se explica cómo funcionaban cada una de ellas y qué pasos se dieron para considerar el cambio de versión.

1.7.4. Capítulo 5: Evaluación del rendimiento y precisión

En este capítulo se muestran los resultados de diversas pruebas de diagnóstico para observar el rendimiento de Seidh y la precisión de sus simulaciones.

1.7.5. Capítulo 6: Conclusiones

En este capítulo final se recogen las conclusiones y reflexiones del autor tras el diseño del algoritmo y su implementación en la librería Seidh.

2. Predicción de estructuras proteína-péptido y algoritmos basados en la naturaleza

2.1. Base de la predicción de estructuras proteína-péptido

La predicción de la estructura de proteínas y péptidos continúa siendo un reto en el contexto de la biología estructural computacional. Desde la década de los noventa hasta nuestros días, se han utilizado múltiples estrategias para realizar una aproximación a la misma; partiendo desde métodos experimentales como la cristalografía de rayos X, la resonancia magnética nuclear o la microscopía electrónica de alta resolución [1]; hasta las más recientes técnicas de computación: machine learning y algoritmos inspirados en la naturaleza. Los primeros constituyen un buen primer paso, rindiendo resultados precisos y estableciendo los puntos base de su posterior desarrollo. No obstante, considerando la cantidad de proteínas conocidas secuenciadas, estos métodos son caros y poco eficientes. Entonces aparecen los últimos métodos descritos: soluciones asistidas por el ordenador, que se demuestran lo suficientemente precisas como para considerarlas una alternativa prometedora.

Algunos de los métodos computacionales tradicionales incluyen la comparación entre estructuras ya conocidas, obtenidas experimentalmente, y sus secuencias, lo que se conoce como "proteínas plantilla". Este conocimiento permite la predicción de estructuras desconocidas de otras proteínas. Este proceso se conoce como modelado por homología [1]. Una alternativa a este método es la comparación de fragmentos entre proteínas desconocidas y proteínas conocidas, para identificar el plegado de estos.

Finalmente, hay una familia de métodos adicional: los métodos de novo o ab initio [1], que se apoyan en los principios de la física y de la información provista por otras estructuras, sin llegar a considerar los pliegues de otras proteínas conocidas ni el desarrollo evolutivo de la proteína como tal. La literatura afirma que estos métodos funcionan mejor con secuencias cortas de aminoácidos [2].

Es preciso mencionar, no obstante, que a menudo se emplean métodos híbridos entrelazando los tres propuestos [1].

2.2. Interacciones proteína-péptido y la predicción de la estructura

Pese a ser similares en cuanto a estructura al estar ambas entidades compuestas de aminoácidos, las interacciones proteína-péptido constituyen un campo de estudio bien delimitado dentro del estudio de interacciones proteína-proteína, y por tanto dentro de la biología estructural computacional [3]. Son responsables de un buen número de procesos clave dentro de la célula, incluyendo procesos de señalización y regulación; y debido a la naturaleza de sus propiedades bioquímicas se comportan de forma distinta.

Los péptidos, a diferencia de las proteínas, no se encuentran como moléculas independientes, y la mayoría no poseen un pliegue característico. Se las conoce como proteínas intrínsecamente desordenadas. La forma en la que se plieguen a una proteína en particular afecta al resultado final de su plegado.

Como se señalaba en el párrafo anterior, median numerosos procesos, tales como procesos proteína-proteína. Por tanto, conocer las relaciones proteína-péptido implica la posibilidad de diseñar péptidos inhibidores.

A pesar de su aparente variabilidad (pues la longitud de la secuencia de un péptido varía desde los dos aminoácidos hasta varias docenas), la predicción de la estructura del péptido una vez unido a la proteína sigue siempre unos pasos similares [3]:

1. Modelado de la estructura receptora (una proteína)
2. Predicción del punto de unión (la región de la proteína a la que se unirá el péptido)
3. Modelado del péptido
4. Refinado de las predicciones

Los primeros tres pasos son, en sí mismos, problemas independientes. No obstante, considerando el cuarto paso todos se vuelven interdependientes, ya que como se ha mencionado anteriormente, la estructura del péptido variará en función del sitio de unión. En ocasiones, incluso, la estructura de la proteína

puede llegar a cambiar, aumentando así la dificultad de predecir la estructura final real de la interfaz proteína-péptido [4].

2.3. Estado del arte de las soluciones disponibles

Un ejemplo de método ab initio sería el método Rosetta, implementado en el software Rosetta desarrollado por David Baker. Rosetta incluye dos funciones de energía, una que se define como "coarse-grained" o "boceto": una versión simplificada (sin todos los átomos que conforman las proteínas); y otra versión que sí incluye todos los átomos (más precisa), que representa la proteína en su conjunto, tal y como sucedería en el mundo real [5].

Otros métodos híbridos, como I-TASSER (iterative threading assembly refinement), parten de un método ab-initio y combinan otras tecnologías. I-TASSER, concretamente, emplea PSSpred para hallar la estructura secundaria, y después identifica las proteínas plantilla a través de LOMETS [6]. Después, los fragmentos son ensamblados a través de una simulación basada en el método de Monte Carlo [6]. Finalmente, SPICKER selecciona los modelos clusterizados, y la estructura es redefinida por FG-MD a nivel atómico [6].

Algunas de las soluciones descritas en la literatura para el modelado de complejos proteína-péptido incluyen FlexPepDock, DynaDock, Autodock y pepATTRACT. Otras soluciones, como EpiDock, pDock, SVMHC o DynaPred también buscan la predicción del complejo proteína-péptido, con otro enfoque. Todos tienen en común el esquema descrito anteriormente. A modo de resumen del estado del arte, se describen sucintamente algunos de estos métodos:

- **FlexPepDock** está implementado en el framework Rosetta, empleando el ya mencionado enfoque a través del método de Monte Carlo [3]. Los resultados de los tests muestran una precisión aceptable de menos de 2 Å en el 91% de los casos [3]. Puede accederse a través de un servidor web, y su código está disponible en la suite Rosetta.
- **DynaDock** permite la predicción de pequeños péptidos (de 2 a 16 residuos) empleando un gran número de muestras. Comienza con la creación de 500 conformaciones iniciales aleatorias, y se irán refinando a través de su algoritmo, para finalmente ser evaluadas y ordenadas según un criterio de ranking [3]. Los tests demuestran una precisión de menos

de 2.1 Å para la totalidad de los casos [3]. Es accesible previa petición a sus autores.

- **Autodock** fue diseñado en un principio para péptidos pequeños (de 2 a 4 residuos), de forma ciega (sin conocimiento de cuál es el lugar de unión a la estructura receptora) [3]. Los tests muestran una buena precisión incluso para péptidos más grandes (de 3 a 7 residuos) [3]. Es gratuito, y puede accederse a través de la web de su autor.
- **pepATTRACT** combina un método "coarse-grained" (boceto), similar al de Rosetta, con diferentes mecanismos de flexibilidad en su motor de unión ATTRACT [7]. El método no requiere la descripción del sitio de unión (es un método ciego, al igual que Autodock). Los tests de benchmark le dan un 70% de precisión en predicciones a ciegas, con menos de 2 Å [8]. Posee una interfaz web, y la librería puede ser instalada en sistemas basados en Unix.
- **CabsDock** unifica los cuatro procesos descritos en un único método ciego que permite que un péptido completamente flexible explore la superficie de la proteína, buscando puntos de unión aceptables [9]. También posee un enfoque "coarse-grained" (boceto) con su propio motor de unión. Su módulo principal modela diez mil diferentes modelos para después filtrar y ordenar. Hasta el 70% de los modelos encuentran el sitio de unión nativo; con menos de 1.4 Å de precisión [10]. Está disponible a través de un servidor web y una API restFUL. También está disponible una versión de escritorio, tanto para Unix como para Windows.
- **HADDOCK** es un método no-ciego para el ligado de péptidos. Su protocolo se divide en cinco pasos, que incluyen la creación del péptido, la propuesta de distintos pliegues, una simulación de unión proteína-péptido y un análisis de la misma [11]. Los test de benchmark proponen resultados interesantes, con menos de 1.6 Å para péptidos pequeños (en torno a 6 residuos) [11]. Los resultados para péptidos más grandes resultan poco precisos. Es gratuito y se encuentra disponible en formato de servidor web y de instalación independiente para sistemas basados en Unix.

En pocas palabras, el software disponible para predecir el pliegue de un péptido una vez unido al punto de unión con la proteína receptora da unos resultados aceptables. La longitud de la estructura primaria del péptido condiciona notablemente la precisión de los modelos propuestos.

La suite Rosetta proporciona una gran cantidad de herramientas y recursos para el modelado ab Initio, si bien otras soluciones (pepATTRACT, CabsDock o HADDOCK) podrían ser más sencillas de usar por su implementación y disponibilidad. Estrictamente hablando, y volviendo a la casuística planteada para este proyecto, HADDOCK puede resultar la opción más similar, ya que es el único enfocado a un punto de unión concreto (método no ciego). Paralelamente, entre los métodos ciegos destacarían pepATTRACT y CabsDock, si bien la literatura muestra diferencias en la casuística de uso.

Esto demuestra que, con independencia del origen y planteamiento de la herramienta para la simulación de la interfaz proteína-péptido, cada una resulta más eficaz para un planteamiento que otro, siendo algunas de las variables implicadas el conocimiento o no del sitio de unión, la longitud del péptido y las características de la proteína receptora (si es flexible o no).

2.4 Algoritmos inspirados en la naturaleza

Con el paso del tiempo, muchos animales se han vuelto, sin ser conscientes de ello, expertos en aplicar soluciones óptimas a problemas de su día a día. Algunos ejemplos clásicos incluyen la búsqueda de alimento de una colonia de hormigas y el retorno al hormiguero; el vuelo de las abejas; la aerodinámica de algunas especies o incluso la capacidad de aprendizaje. Todo se basa en la evolución: el proceso por el cual algunas características son más deseables frente a otras en un ecosistema en continuo cambio. En un entorno mutable, una especie incapaz de experimentar ninguna alteración de sus características está condenada a la desaparición.

Siguiendo esta analogía, el enfoque clásico o tradicional a numerosos problemas resultaba insuficiente por la propia naturaleza del problema. Problemas como la optimización de un algoritmo, el diseño de un sistema de vuelo, o la simulación de un sistema real (como las interacciones entre un péptido y una proteína)

requieren un enfoque nuevo, un enfoque más relacionado con la propia naturaleza del problema.

Los algoritmos suelen tener un esquema común, en el que se parte de un supuesto y se introducen una serie de cambios que deben respetar unas limitaciones (constraints). Estos cambios son seguidamente evaluados por una función o un criterio (fitness), que determina si el cambio se mantiene o no. Finalmente, los elementos son ordenados en base a dicho fitness.

Paso 1: Definiciones. Aquí comúnmente se determinan los puntos de partida y los limites (constraints) sobre los que se realizará la simulación

Paso 2: Inicialización. Según las definiciones del paso 1, se inicia el momento inicial acorde a dicha configuración.

Paso 3: Bucle. Para cada unidad, se realiza una evaluación seguida de una modificación. Si la modificación es válida, se admite; si no, se desecha o se vuelve a la versión original. Esto puede realizarse varias veces hasta llegar a una solución única o a un conjunto de soluciones menor que el conjunto de partida. Orden de las unidades en la simulación.

Paso 4. Fin del algoritmo si se ha llegado al criterio de fin.

La característica fundamental de los algoritmos basados en la naturaleza es que uno o más de sus elementos nacen de un fenómeno observable. A menudo puede ser la generación de cambios en las unidades del algoritmo, aunque también podría ser la función de evaluación y los constraints.

3. Diseño de Seidh: del vuelo del cuco al vuelo de Lévy

3.1. Descripción del planteamiento de partida

El objeto de la librería Seidh es proporcionar una solución para la predicción de la estructura (plegado) de un péptido unido a una proteína, partiendo únicamente de la estructura de la proteína, del sitio de unión (binding point) y de la secuencia del péptido. Por las características de este planteamiento, diremos que Seidh es un método no-ciego, pues el binding point es conocido. Además, ya que Seidh incorpora todos los átomos de la proteína y del péptido a la simulación, podemos considerarlo un método “fine grained” o nítido, ya que el resultado será más fiel a lo observable en la naturaleza.

3.2. Algoritmo del cuco y vuelo de Lévy

Seidh se inspira en el comportamiento de cuco, un ave parásita que en época de puesta trata de encontrar el nido idóneo para su prole. Este deberá constar de huevos, prueba de que otra ave se encuentra en los alrededores y está en época de cría, pero deberá tener asimismo espacio suficiente para albergar el suyo propio, de un tamaño sensiblemente mayor al del resto de pájaros.

El movimiento que el cuco realiza para la búsqueda del nido sigue la llamada distribución de Lévy: una distribución con una función de densidad de probabilidad centrados en el intervalo [0,1], y una cola densa y pesada. Esto se traduce en un conjunto de números relativamente pequeños alternados con números más grandes, pero con escasa frecuencia. Este movimiento ha sido observado en numerosas aves, y por ello recibe el sobrenombre de vuelo de Lévy.

Cada aminoácido es representado en el algoritmo como un cuco. Por cada residuo del péptido a predecir, se crea un número determinado de cucos que realizarán, cada uno, un vuelo de Lévy: algunos (la mayoría) variarán sus parámetros (los ángulos psi y phi del marco de átomos de carbono) mínimamente, mientras que otros (una minoría) variarán considerablemente.

Una vez los cucos han realizado el vuelo de Lévy, son evaluados mediante una función de cálculo de energía basada en el trabajo de Torchala con SwarmDock (12), que considera la distancia entre átomos, la carga eléctrica, y la energía y radio de Van der Waals. Tras esta evaluación, los cucos son ordenados en un ranking, para después eliminar los diez peores.

El proceso se repite hasta que queden 10 cucos, momento en el que el primero queda aceptado como predicción del aminoácido y el proceso comienza de nuevo.

Además, el algoritmo es ejecutado a su vez varias veces, de forma que cuando la simulación finaliza el programa ha calculado un número considerable de predicciones. Estas predicciones son también ordenadas en un ranking según el resultado de la función de cálculo de energía. El resultado final de la simulación es, por tanto, la predicción que mejor puntuación obtenga.

A modo de esquema:

Condiciones iniciales:
Secuencia S con R residuos
Binding point I
M péptidos a simular
Con N pruebas por residuo
Distancia mínima X
...

1. Para el péptido m de M péptidos
   1.1 Por cada residuo R de la secuencia S:
      1.1.1 Generar N pruebas del binding point I
      1.1.2 Añadir el residuo R (aleatorio) al binding point I
         1.1.2.1 Mientras que no cumpla las restricciones, borrar y añadir el residuo R (aleatorio) al binding point I
      1.1.3 Mientras que N sea mayor a 10:
         1.1.3.1 Modificar el ángulo de los aminoácidos según la distribución de Lévy [desechar cambios si no respetan los restricciones]
         1.1.3.2 Evaluar según la función de scoring
         1.1.3.3 Eliminar los 10 peor evaluados
      1.1.4 Devolver el residuo R al péptido m
   1.2 Evaluar péptido m
      1.2.1 Si supera la distancia mínima, clasificar. Si no, desechar.
2. Analizar péptidos clasificados e identificar conformaciones similares
3. Distribuir por clústeres
   3.1 Identificar péptido de mayor puntuación según función de scoring y proponer como solución del clúster
4. Devolver clústeres y sus soluciones

Las restricciones que implementa el algoritmo son varias.

La principal restricción está basada en la posición de cada par de átomos (a_i,a_j; i para cada átomo de la proteína; j para cada átomo del péptido), que impide que estos colisionen.

Otra limitación, derivada de la anterior, establece un número de intentos máximo por residuo del péptido. Si superado este número de intentos no se ha conseguido añadir el residuo, el péptido entero es descartado. Esto es debido a que es posible que haya incurrido en una configuración no permitida por las restricciones.

A la hora de modificar los ángulos de un aminoácido, se consideran los límites de torsión entre -180º y 180º: cualquier valor que exceda estos números se descarta, tomando el valor original en su lugar.

Por otro lado, podemos considerar la restricción por excelencia, la función de scoring. La función de scoring de Seidh considera las distancias entre los átomos, sus radios atómicos, sus radios de Van der Waals, sus potenciales electrostáticos y sus fuerzas de Van der Waals. Es una función “more is better”, significando esto que cuanto mayor es la puntuación obtenida mejor es el resultado obtenido. Esto permite clasificar y ordenar los péptidos conforme se añaden aminoácidos, descartando así soluciones poco óptimas desde el principio.

Por último, se descartan aquellos péptidos que no consigan un mínimo de contacto con la superficie de la proteína: es preciso obtener una orquilla de contactos de aproximadamente un 80% para considerar una solución válida: un número menor podría favorecer determinadas conformaciones difícilmente observables en la naturaleza en las que el péptido se pliega sobre sí mismo; y niveles (muy) superiores podrían dificultar la aceptación de algún péptido. Los contactos se calculan átomo a átomo, teniendo como referencia al péptido.

Por la naturaleza de este método ab initio, es posible identificar diversas estructuras que pueden darse simultáneamente en la naturaleza. Por ello, se implementa una función de clusterización para identificarlas y proponerlas como soluciones alternativas al usuario.

4. Revisión de las versiones de Seidh: descripción de las mejoras implementadas

4.1. Seidh 0.1: versión base

La versión original de Seidh consideraba la proteína como una mera referencia, y su única misión consistía en ir añadiendo aminoácidos al binding point según una simplificación del algoritmo actual.

Al no existir una limitación real que descartara prematuramente las soluciones que violasen alguna restricción, su rendimiento era rápido, pero los resultados a menudo resultaban ser antinaturales (ya que atravesaban la proteína, o el resto de átomos del péptido).

En esta versión se diseñó el generador de la librería peptiDB, que permite el acceso al banco de pruebas de Seidh; la librería auxiliar, que permitía la lectura y escritura de ficheros *.ent, *.pdb y *.fas; y la librería levy, que permitía el acceso al generador de números de la distribución de Lévy.

Además, la función de scoring y la función adaptadora (driver) también fueron implementadas en esta versión.

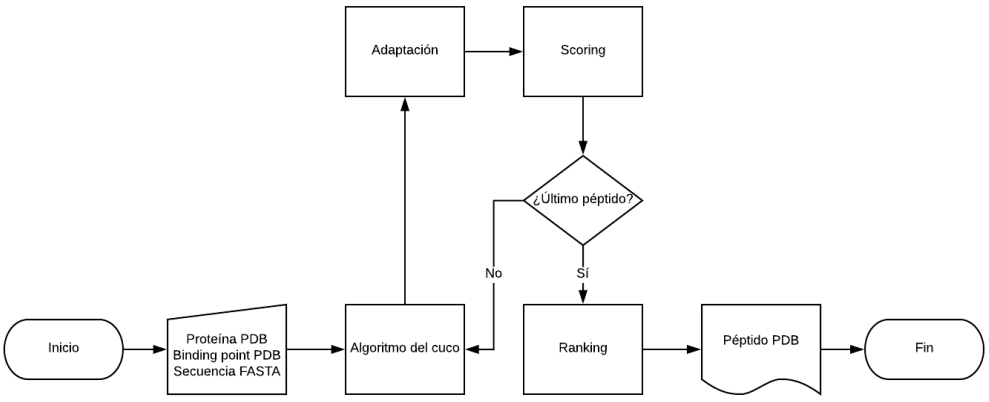

Ilustración 6. Esquema de funcionamiento de Seidh 0.1

4.2. Seidh 0.2: mejora de las restricciones y bloqueo de colisiones

Cuando los problemas de Seidh 0.1 se hicieron patentes al poco tiempo, se resolvió la implementación de nuevas restricciones: un sistema de bloqueo de colisiones externas (péptido-proteína) e internas (péptido-péptido). Este sistema se recoge en la librería distance, que en esta etapa son tan solo dos funciones.

La función VdW_minima, que emplea la versión adaptada de las moléculas, comprueba que la distancia entre cada par de átomos sea menor que la suma de los radios de Van der Waals. En caso de que se detecte una colisión, la función se detiene y devuelve una señal (False), como criterio de que el par de moléculas no cumplen los criterios de restricción.

De forma similar, la función inner_stability determina si existe alguna colisión interna. Es diferente en cuanto a VdW_minima, ya que al buscar los pares de átomos respeta aquellos que se encuentren enlazados por un enlace (que, a tenor del criterio empleado, violarían la restricción). En caso de encontrar una colisión interna, devuelve una señal (False), del mismo modo que VdW_minima.

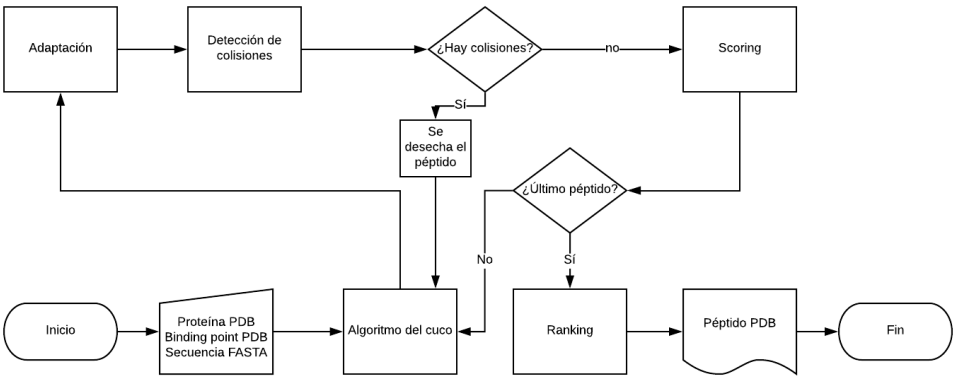

Ilustración 7. Esquema de funcionamiento de Seidh 0.2

4.3. Seidh 0.3: multiprocessing

Con la implementación de la librería distance se mejoraban enormemente los resultados: ya no se incurría en la situación de tener átomos superpuestos, o a distancias antinaturales. Sin embargo, el rendimiento había bajado enormemente, ya que a menudo se desechaban péptidos con apenas uno o dos residuos para su finalización, lo que implicaba un retraso importante.

Para aumentar la velocidad de procesado, se decide emplear una librería de computación paralela haciendo uso de más de un núcleo del procesador (multiprocessing). Esto exigió en cierta manera la redefinición de la librería base, ya que en función de que el código se ejecutara en un ordenador con un solo núcleo, o con dos o más, cambiaría completamente la forma de proceder.

Hasta ahora, por cada residuo, se manejaba una lista de un número máximo de pruebas, determinado por las propiedades definidas en el código. Esta lista era procesada secuencialmente. Con la implementación de multiprocessing, ahora es posible procesar varios residuos simultáneamente, lo que permite reducir tiempos de cálculo en funciones como el cálculo de energía:

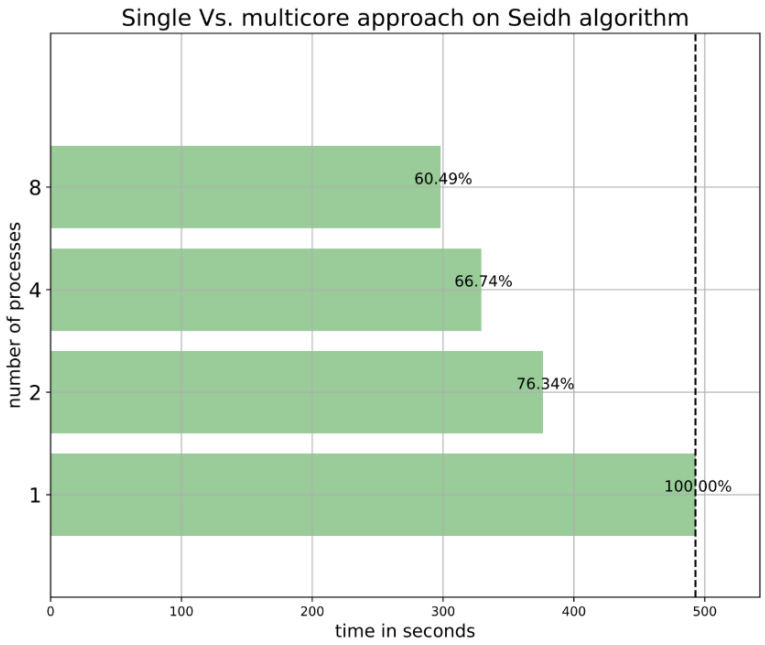

Ilustración 8. Comparativa de ejecucíón según el número de núcleos empleados.

El ahorro de hasta un 40% del tiempo de predicción, de media por péptido, supone una gran mejora en simulaciones que comprendan un número importante de péptidos a simular.

El código debe, no obstante, respetar la posibilidad de que se ejecute en sistemas de un solo núcleo (bien por limitaciones de hardware, bien porque sea ejecutado paralelamente).

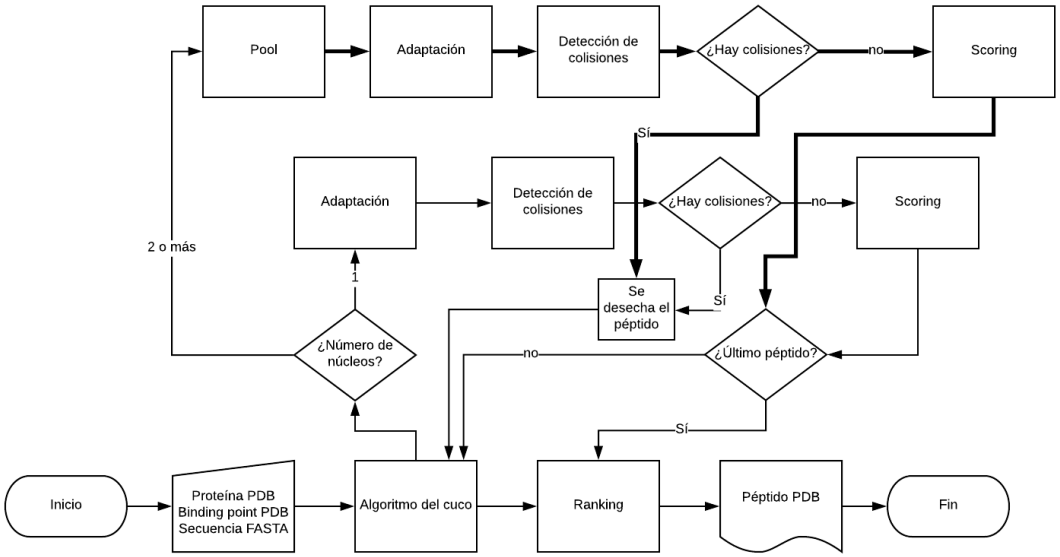

Ilustración 9. Esquema de funcionamiento de Seidh 0.3

En este esquema, más complejo que los anteriores, se denota en línea de puntos el procesado paralelo que se hace a través del “pool” de multiprocessing: siguiendo un esquema similar a la ejecución mononúcleo, cada aminoácido propuesto sigue el mismo camino, pero simultáneamente.

Es preciso destacar que en el uso de multiprocessing se hace uso de una llamada asíncrona, lo que permite recuperar los resultados en el orden en el que fueron ejecutados, algo fundamental teniendo en cuenta que se devuelven tanto los aminoácidos modificados como su evaluación con cada ejecución.

4.4. Seidh 0.4: mejora de la calidad de las predicciones. Contactos entre moléculas

Al introducir nuevas restricciones en el algoritmo, simultáneamente se estaban beneficiando determinadas soluciones no deseables en la naturaleza, como pequeñas hélices alfa en péptidos lo suficientemente pequeños como para que esta conformación carezca de sentido. Por tanto, se incluye una nueva restricción: que el péptido esté al menos un 80% en contacto con la proteína a la que se acople.

Esto se lleva a cabo mediante la definición de la función molecule_contacts, también de la librería distance. Esta función se limita ir aumentando un contador conforme se detecta que cada átomo del péptido está, al menos, a una distancia inferior al radio de Van der Waals más 0.5 Å de un átomo de la proteína. Después, devuelve este contador dividido entre el número de átomos del péptido, lo que resulta en un índice de contacto. Si este índice es de, al menos, 0.8, se considera que el péptido está unido a la proteína y por tanto es una conformación válida. En caso contrario, el péptido es desechado ya que en la naturaleza no estarían unidos.

Esta modificación no entraña un gran cambio en el modo en que Seidh o el algoritmo del cuco funcionan, sino que más bien modifica la forma en la que Seidh refina sus resultados.

4.5. Seidh 0.5: identificación de estructuras semejantes mediante clusterización

Finalmente, tras la inclusión de todas las restricciones consideradas y la enmienda de los problemas identificados, se constató que Seidh continuaba prediciendo cierta diversidad de conformaciones.

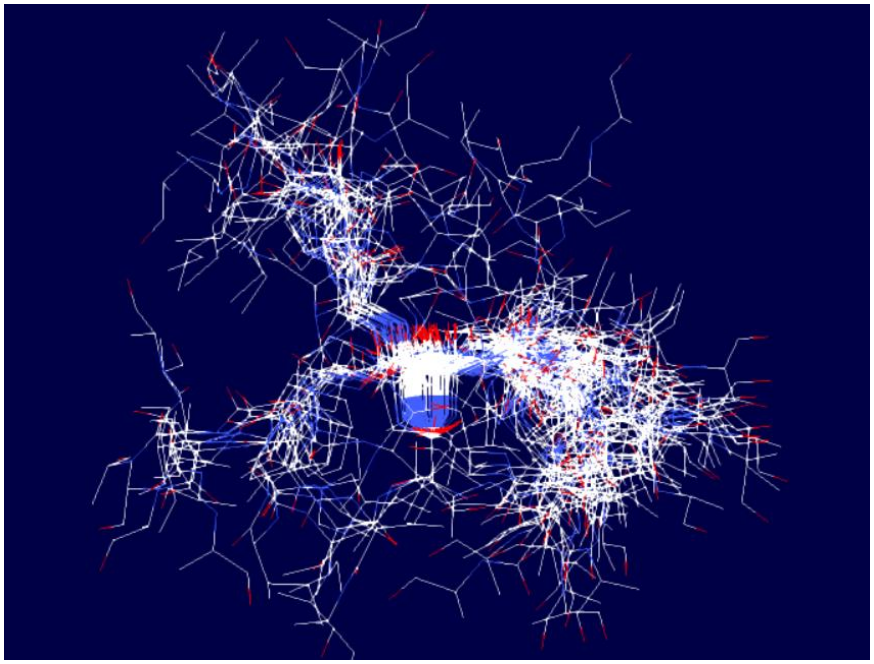

Ilustración 10. Cien simulaciones del péptido 1AWR-I

Como se puede apreciar en la ilustración 10, pueden intuirse al menos tres conformaciones distintas, si bien algunas son más frecuentes que otras. Ya que en la naturaleza existe la posibilidad de que un péptido se una de distintas maneras a una proteína, se determinó la necesidad de que Seidh reconociese estas distintas conformaciones y propusiera la mejor simulación de cada una como resultado. A tal efecto se implementa una librería de clusterización (clusterize), que realiza ese proceso calculando la RMSD entre cada una de los distintos péptidos obtenidos, basándose en el algoritmo BSAS. Finalmente, Seidh devuelve, para cada conformación, el péptido con un scoring más alto.

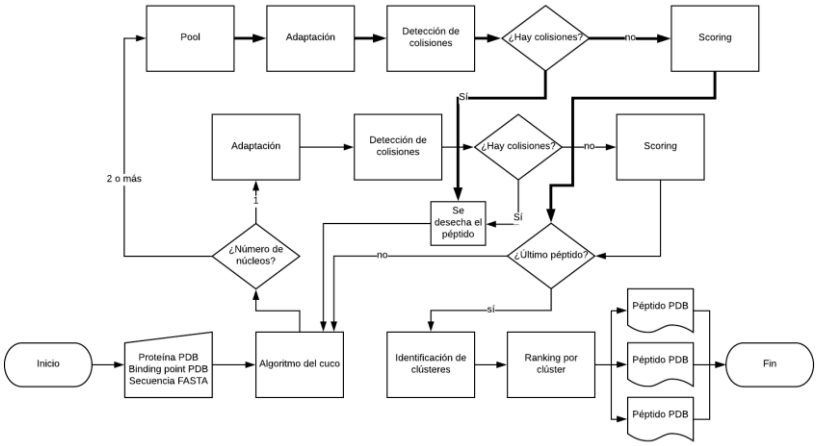

Ilustración 11. Esquema de funcionamiento de Seidh 0.5

En el esquema se pueden apreciar ahora todas las modificaciones incluidas.

4.6. Seidh 1.0: la versión definitiva

Tras las anteriores versiones, Seidh finalmente alcanza su primera release tras la implementación de la clusterización y de un pequeño logger que aporta información sobre la simulación.

Podemos definir Seidh como un conjunto de herramientas que incluye la aplicación del algoritmo del cuco modificado para predecir estructuras peptídicas unidas a proteínas, partiendo de esta proteína, del sitio de unión y de la secuencia FASTA que lo origina.

Seidh puede ser invocado de forma independiente, proporcionándole únicamente los objetos Biopython.PDB de la proteína y el sitio de unión, así como una cadena de caracteres conteniendo la secuencia FASTA. Adicionalmente, podemos indicar el número de núcleos de procesador que queremos emplear para su ejecución.

En las siguientes líneas se describen las funciones auxiliares de Seidh, un diagrama de flujo que resume su funcionamiento, y un esquema del almacenamiento de archivos que requiere. También se menciona el esquema de salida del logger, para comprobar los detalles sobre la ejecución.

4.6.1. Librerías y dependencias

Seidh necesita, para funcionar, algunas librerías de terceros como Biopython o Numpy. Otras librerías estándar como os y datetime también son empleadas aquí.

Librerías estándar

- os: empleada para la navegación por carpetas.
- datetime: empleada para generar horas y fechas, tanto en nombres de fichero como en el logger.
- copy: empleada para generar copias profundas (deepcopy()) de algunos objetos Biopython PDB, para evitar modificaciones involuntarias.

Librerías de terceros

- Biopython: empleada para la manipulación de ficheros PDB y la obtención de secuencias a partir de estos.
- PeptideBuilder: tanto el módulo principal, PeptideBuilder, como su accesorio, Geometry, son empleados para la generación y modificación de aminoácidos.
- Numpy: empleada para cálculos de distancias.
- Multiprocessing: librería base para la implementación de la computación paralela en Seidh.

4.6.2. Organización y ficheros. Librerías auxiliares

La librería está organizada en varios niveles. El fichero seidh.py incluye la función seidh(), que permite comenzar la simulación. Para su funcionamiento, encontramos varias librerías auxiliares en la carpeta lib, la función de scoring y su adaptador en la carpeta sd y, según la configuración por defecto, la carpeta pdb_files con todos los archivos.

En el archivo props.py, dentro de la carpeta lib, podemos configurar cómo queremos que se comporte Seidh: definir el máximo de péptidos por simulación, el máximo de pruebas por péptido, el porcentaje mínimo de contactos para considerar un péptido como adecuado, y las carpetas donde queremos que se guarden los distintos ficheros.

Por lo demás, en lib encontramos las siguientes librerías auxiliares:

- auxiliar.py: Contiene las funciones básicas para el manejo de ficheros PDB y secuencias FASTA. Puede emplearse para comprobar si una cadena es una secuencia FASTA válida, o si la secuencia FASTA contiene caracteres ambiguos o comodín. En este caso, también se puede obtener una secuencia no ambigua.
- clusterize.py: Contiene las funciones necesarias para identificar los clústeres dentro de la simulación.
- cuckoo.py: Contiene las funciones que definen el algoritmo del cuco.

- distance.py: Contiene las funciones empleadas en la detección de colisiones y el cálculo de contactos.
- levy_distribution.py: Contiene la función que genera valores de la distribución de Lévy.

Es recomendable evitar llamar las funciones de las librerías auxiliares de Seidh, con la excepción de las contenidas en auxiliar.py o levy_distribution.py, ya que el resto se encuentran diseñadas para funcionar con entidades de la propia librería.

Por otro lado, en sd encontramos:

- driver.py: Contiene la función SeidhAdapt(), que toma un objeto BioPython PDB y devuelve un array multidimensional con las coordenadas, radios de Van der Waals, potenciales electrostáticos y energías de Van der Waals.
- Scoring.py: Contiene la función seidh_calculate_energy(), que toma dos arrays proporcionadas por SeidhAdapt() y devuelve la puntuación energética. Según la función de scoring implementada, a mayores resultados mejor es la conformación.

4.6.3. Banco de pruebas

En el diseño de Seidh fue preciso contar desde el principio con un conjunto de proteínas, puntos de unión y secuencias fasta para hacer pruebas sobre el funcionamiento del algoritmo. A tal efecto, se crea el script pept_lib_gen.py, que a partir de un fichero peptidb [13], genera un banco de pruebas que satisface las condiciones de partida.

El banco de pruebas consiste en un árbol de directorios donde se encuentran los ficheros .ent de PDB, que contienen las múltiples cadenas de una proteína y péptidos varios; los ficheros .pdb correspondientes a la cadena concreta de cada proteína y péptido del fichero .ent; y ficheros .fas correspondientes a las secuencias de aminoácidos de cada péptido.

4.6.4. Esquema de funcionamiento

Seidh 1.0 emplea todos estos recursos para realizar cada simulación. Además de los péptidos solución, también ofrece la opción para guardar los péptidos prueba.

Por último, junto con los péptidos solución también crea un fichero resumen de la simulación, ofreciendo información, péptido por péptido, sobre los resultados de score y los tiempos que ha llevado cada uno.

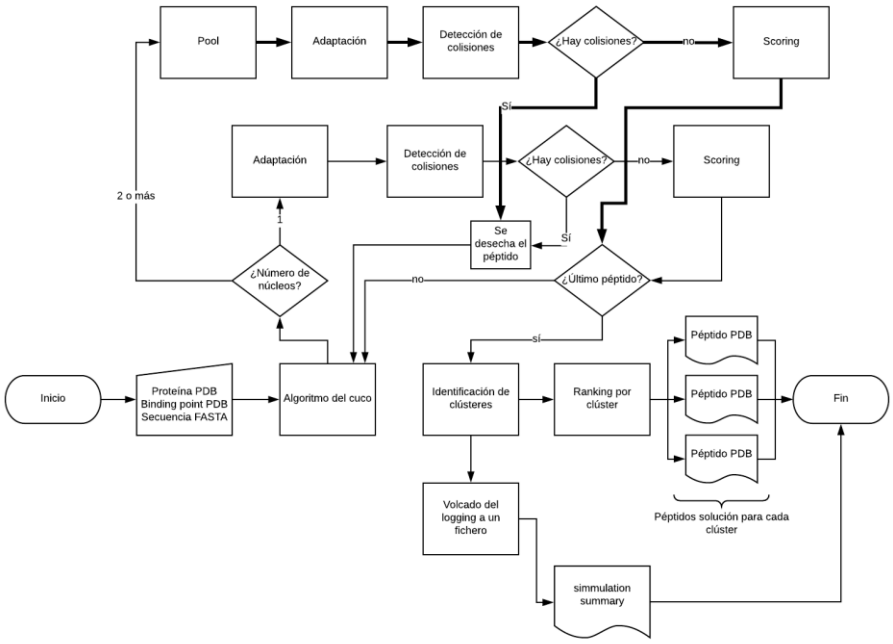

Ilustración 12. Esquema de funcionamiento de Seidh 1.0

4.6.5. Configuración

Seidh permite la modificación de sus parámetros a través del fichero props.py. También es posible cambiar los directorios por defecto.

- MAX_AA: determina el número máximo de pruebas por aminoácido. Como mínimo, para funcionar, debe establecerse en 10. Aumentarlo incrementa notablemente el tiempo de ejecución, pero mejora la precisión.
- MAX_PEPTIDES: determina el número máximo de péptidos por simulación. Como mínimo, para funcionar, debe establecerse en 1.

Aumentarlo incrementa el tiempo de ejecución y la posibilidad de hallar nuevas conformaciones.

- MIN_CONTACT: determina el índice de contactos mínimo entre proteína y péptido. No debería situarse por encima de 85. Aumentarlo incrementa el tiempo de ejecución y los falsos negativos.

4.6. Mejoras

Durante el desarrollo de Seidh, se detectaron problemas y aspectos a mejorar. Muchas de estas incidencias se tradujeron en las actualizaciones que cimentaron las nuevas versiones de la librería, pero otras, por sus implicaciones en tiempo y demora, no. Se recogen a continuación algunos aspectos, ideas y limitaciones que posee el código actual en su versión 1.0, tanto a efectos de documentación como de lista de tareas pendientes.

Diseño

Seidh se concebió originalmente como una librería con un fuerte diseño orientado a objetos. La clase péptido incluía como atributo el objeto BioPython PDB asociado al mismo y su scoring; así como métodos para añadir o quitar aminoácidos, modificarlos, comprobar colisiones o calcular el score. Fue en torno a la versión 0.2 cuando se hizo un fork del código original (que carecía de este enfoque orientado a objetos) para implementar esta clase, y los resultados fueron insatisfactorios: el consumo de memoria de Seidh se disparaba, y el tiempo de ejecución crecía enormemente. Dado que, como se verá más adelante en el capítulo 5, el tiempo es uno de los recursos más valiosos de Seidh, se desistió. El principal problema radica en tratar de mover objetos tan pesados como el objeto BioPython PDB con el péptido, a través de bucles y llamadas a varias funciones y métodos. Por tanto, como solución para una futura versión, se plantea limitar la dependencia de librerías de terceros como PDB o PeptideBuilder, y plantear la creación integral del objeto en base a las necesidades y aspiraciones de Seidh.

Binding point

Una limitación notable de Seidh es su dependencia del marco de lectura en un único sentido: el algoritmo es capaz de ir añadiendo aminoácidos de izquierda a derecha, pero no así de derecha a izquierda. Esto condiciona que el binding point sea siempre el primer aminoácido de la secuencia FASTA (o un conjunto de estos, ya que Seidh admite como binding point más de un aminoácido). La limitación viene impuesta por una librería de terceros, PeptideBuilder; y su solución viene dada por su modificación o rediseño para admitir la adición de aminoácidos en sentido derecha a izquierda.

Mejora del rendimiento

Seidh realiza algunas funciones muy costosas en cuanto a trabajo computacional: el cálculo del score, la detección de colisiones o el cálculo del índice de contactos son algunos ejemplos. Estas funciones están actualmente implementadas en Python, por lo que no son tan eficientes como otras implementaciones a más bajo nivel. Por ello, es planteable en un futuro portar estas funciones a otros lenguajes de programación como C, que permiten un cálculo más ágil.

Alternativas a multiprocessing

Multiprocessing es la librería elegida para la computación paralela en Seidh: es un recurso ampliamente utilizado y descrito en la programación con Python que emplea los núcleos del procesador para llevar a cabo esta tarea. Sin embargo, hay otras alternativas más atractivas, como el uso de una GPU para aumentar el rendimiento del trabajo. No obstante, dada la complejidad de trabajar con núcleos CUDA en Python –por poner une ejemplo de computación con GPU-, se desestimó su inclusión por falta de tiempo, en favor de multiprocessing, una alternativa más rápida y sencilla.

4.7. Seidh webserver

Seidh, como librería Python, es fácilmente exportable a otros entornos. Sin embargo, algunas implementaciones como la del servidor web Flask que forma

parte de este proyecto, requiere alguna modificación mínima en su fichero de propiedades.

Seidh webserver es una aplicación desarrollada en Flask, que permite al usuario subir dos ficheros PDB o ENT que contengan una proteína y un binding point, así como especificar la secuencia FASTA del aminoácido. Después, el servidor comprobará la integridad de los ficheros y si la secuencia introducida se corresponde o no con una secuencia FASTA, para incorporar toda esta información a una base de datos y devolver al usuario un enlace con un código hash. A través de este enlace, el usuario podrá comprobar el estado de su simulación. En cuanto termine, el enlace permitirá al usuario descargar cuantas soluciones halla identificado Seidh como posibles.

Para la gestión de varias peticiones al servidor, se emplea la librería celery. Esta, a través de un servidor broker basado en Redis, organiza las peticiones y las va poniendo en cola.

Seidh webserver, como aplicación Flask, no necesita ninguna otra aplicación o servicio para levantar el servicio web. No obstante, ya que el servidor que Flask incluye de serie está más enfocado a pruebas que a ofrecer un servicio en producción, su implementación final se realiza a través de apache y la herramienta wsgi.

5. Evaluación y pruebas

Las pruebas se han realizado en la máquina descrita en el apartado 1.4.2, empleando cuatro núcleos del procesador.

Se ha empleado el complejo con código PDB 1AWR, que incluye la proteína ciclofilina A y un péptido de 6 residuos (HAGPIA). Este es un complejo de muy alta resolución (1.58 Å), por lo que los resultados con un péptido con menos resolución podrían variar.

El criterio de calidad (índice de contacto) no ha sido tenido en cuenta en la realización de la simulación. La razón de esto es comprobar el efecto que tiene la modificación de los parámetros especificados en la calidad de las predicciones. En el tratamiento de los datos se considera una predicción de calidad cuando esta posee un índice de contacto igual o superior a 0.8.

Se han realizado un total de 35 pruebas con distintas configuraciones con los parámetros MAX_AA y MAX_PEPTIDES, con el objetivo de analizar distintos aspectos de la simulación tales como:

- Tiempo total de ejecución de simulación
- Tiempo total de ejecución de simulación ajustado [por intentos]
- Tiempo de cálculo por péptido
- Tiempo de cálculo por péptido ajustado [por intentos]
- Número de clústeres identificados
- Número de péptidos de calidad calculados
- Índice de calidad relativa
- Número de intentos medios por péptido

Las 35 pruebas consisten en la combinación de los parámetros MAX_AA : {10,20,30,40,50} y MAX_PEPTIDES : {1,5,10,15,20,30,50). Los resultados tabulados se encuentran en el anexo 2.

5.1. Tiempo de ejecución

MAX_AA y MAX_PEPTIDES influyen notablemente en el tiempo total de ejecución de la simulación. Esto puede comprobarse en las tablas 1 y 2:

Tiempo de ejecución							
		MAX_AA					
		10	20	30	40	50	PROM(MPEP)
MAX PEPT	1	0:00:31	0:01:30	0:06:54	0:03:35	0:12:15	0:04:57
	5	0:04:19	0:17:31	0:23:13	0:44:52	0:29:53	0:23:58
	10	0:06:38	0:22:26	0:45:12	0:58:26	1:59:11	0:50:23
	15	0:11:25	0:34:42	1:00:12	1:47:33	1:51:59	1:05:10
	20	0:18:18	0:57:53	1:27:48	1:59:36	2:48:14	1:30:22
	30	0:28:35	1:18:49	1:38:42	4:22:07	5:11:04	2:35:51
	50	1:00:01	1:43:48	2:59:43	4:23:24	9:52:54	3:59:58
	PROM(MAA)	0:18:32	0:45:14	1:11:41	2:02:48	3:12:13	

Tabla 1. Tiempo de ejecución

Tiempo de ejecución (por intento)							
		MAX_AA					
		10	20	30	40	50	PROM(MPEP)
MAX PEPT	1	0:00:31	0:01:30	0:01:23	0:03:35	0:03:04	0:02:01
	5	0:03:05	0:07:18	0:09:40	0:12:28	0:21:21	0:10:46
	10	0:05:32	0:14:57	0:18:50	0:34:22	0:37:15	0:22:11
	15	0:08:09	0:20:30	0:30:06	0:41:22	0:54:11	0:30:52
	20	0:08:56	0:28:14	0:35:50	1:01:20	1:16:28	0:42:10
	30	0:14:32	0:36:23	1:06:34	1:32:31	1:58:08	1:05:37
	50	0:24:48	1:01:04	1:39:51	2:36:47	3:15:02	1:47:30
	PROM(MAA)	0:09:22	0:24:17	0:37:28	0:57:29	1:12:13	

Tabla 2. Tiempo de ejecución ajustado por intentos

Conforme avanzamos en la lectura de las mismas, es posible advertir un abrupto cambio en las cuatro celdas de la esquina inferior derecha de las dos tablas, coloreadas de un tono de rojo más intenso. Esto se corresponde con lo que la lógica propondría (a mayores valores de ambos parámetros, mayor tiempo consumido en la ejecución de Seidh). Sin embargo, hay algunos valores que no se corresponden con la tónica general (tabla 1), como el correspondiente a (MAX_AA=30, MAX_PEPTIDES=1): 6 minutos y 54 segundos para el cálculo de un único péptido. Comprobando la tabla 3 y la tabla 2, comprobamos que en efecto esto se debe a sucesivos intentos frustrados para calcular ese péptido, lo que ha aumentando considerablemente el tiempo total.

Intentos (promedio)							
		MAX_AA					
		10	20	30	40	50	PROM(MPEP)
MAX PEPT	1	1,00	1,00	5,00	1,00	4,00	2,40
	5	1,40	2,40	2,40	3,60	1,40	2,24
	10	1,20	1,50	2,40	1,70	3,20	2,00
	15	1,40	1,69	2,00	2,60	2,07	1,95
	20	2,05	2,05	2,45	1,95	2,20	2,14
	30	1,97	2,17	1,48	2,83	2,63	2,22
	50	2,42	1,70	1,80	1,68	3,04	2,13
	PROM(MAA)	1,63	1,79	2,50	2,19	2,65	

Tabla 3. Promedio del número de intentos por péptido

Esto debe ser considerado como un tercer actor tenido en cuenta: los intentos suman tiempo de ejecución de Seidh, de la misma forma que los dos parámetros. Si observamos la tabla 3, que relaciona el número de intentos promedio por simulación según las configuraciones de los parámetros, es complicado intuir algún tipo de patrón: en principio esto se produce de forma aleatoria. No obstante, si tratamos de eliminar el factor de confusión que podría suponer (respecto al resto de incidencias) el valor de (MAX_AA=30, MAX_PEPTIDES=1), observaríamos como a pesar de que en las tablas no se aprecia de forma clara, en las columna y filas resumen sí: si bien el número de intentos crece conforme crecen los valores de los parámetros, la subida se hace más acusada en la subida de MAX_AA. Esto es explicable por el hecho de que, cuanto más tratamos de variar un residuo más conformaciones que resulten en un choque pueden darse.

En definitiva: al ajustar el valor de los parámetros debemos ser conscientes de que por sí mismos ya añaden tiempo de ejecución conforme aumentan. Además, influyen también en el número de intentos, lo que debe considerarse un tiempo de ejecución aleatorio adicional a tener en cuenta.

5.2. Tiempo de cálculo

Tiempo de cálculo por péptido							
		MAX_AA					
		10	20	30	40	50	PROM(MPEP)
MAX PEPT	1	0:00:30	0:01:29	0:06:54	0:03:34	0:12:14	0:04:56
	5	0:00:51	0:03:29	0:04:38	0:08:58	0:04:32	0:04:30
	10	0:00:39	0:02:14	0:04:31	0:05:38	0:11:55	0:04:59
	15	0:00:45	0:02:18	0:04:00	0:06:57	0:07:27	0:04:18
	20	0:00:54	0:02:53	0:04:23	0:05:58	0:08:02	0:04:26
	30	0:00:56	0:02:37	0:03:17	0:08:47	0:10:22	0:05:12
	50	0:01:11	0:02:04	0:03:35	0:05:15	0:11:51	0:04:47
	PROM(MAA)	0:00:50	0:02:26	0:04:28	0:06:27	0:09:29	

Tabla 4. Tiempo de cálculo por péptido

T. de cálculo por péptido p/ intentos							
		MAX_AA					
		10	20	30	40	50	PROM(MPEP)
MAX PEPT	1	0:00:30	0:01:29	0:02:09	0:03:34	0:03:03	0:02:09
	5	0:00:36	0:01:33	0:02:20	0:02:44	0:04:32	0:02:21
	10	0:00:33	0:01:33	0:01:58	0:03:43	0:04:01	0:02:22
	15	0:00:33	0:01:34	0:02:10	0:03:04	0:04:31	0:02:22
	20	0:00:30	0:01:32	0:02:00	0:03:18	0:03:55	0:02:15
	30	0:00:32	0:01:22	0:02:02	0:03:19	0:04:14	0:02:18
	50	0:00:31	0:01:17	0:01:23	0:03:20	0:04:32	0:02:12
	PROM(MAA)	0:00:32	0:01:28	0:02:00	0:03:17	0:04:07	

Tabla 5. Tiempo de cálculo por péptido ajustado por intentos

En las tablas 4 y 5 podemos observar cómo el número de péptidos apenas afecta al tiempo que conlleva calcular uno solo, más allá de la influencia que esta variable represente sobre el número de intentos.

Sí que podemos observar, sobre todo en la tabla 4, cómo el parámetro MAX_AA condiciona directamente el tiempo de cálculo por péptido, llegando prácticamente a no afectar el otro parámetro.

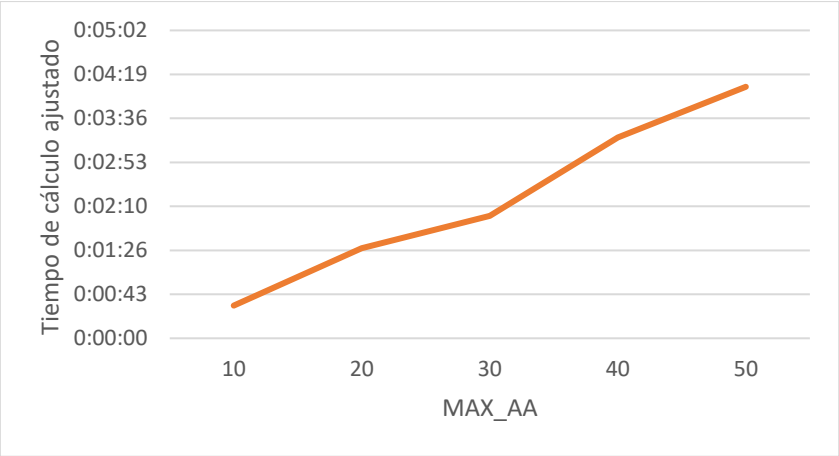

Gráfico 1. Relación entre los valores de MAX_AA (abscisas) y el tiempo de cálculo por péptido ajustado (ordenadas)

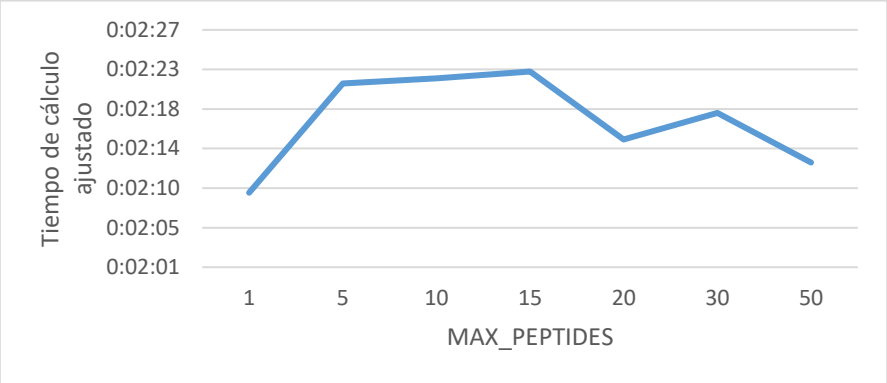

Gráfico 2. Relación entre los valores de MAX_PEPTIDES (abscisas) y el tiempo de cálculo por péptido ajustado (ordenadas)

En los gráficos 1 y 2 podemos además apreciar este fenómeno: la relación lineal directa entre el valor de MAX_AA y el tiempo, y la falta de relación entre MAX_PEPTIDES y el tiempo, además de su minúscula variación.

5.3. Número de clústeres y calidad

La detección de clústeres implica identificar las posibles conformaciones que puede alcanzar el péptido que se intenta predecir. Por tanto, puede ser una característica deseable, si bien es cierto que por sí solo aporta escasa utilidad: si no cruzamos los péptidos calculados con un criterio de calidad (en el caso de Seidh, los contactos proteína-péptido), podemos obtener un gran número de falsos positivos.

Para comprobar hasta qué punto esto es así, la simulación se ha llevado a cabo sin aplicar este criterio, de forma que los resultados puedan filtrarse a posteriori y considerar el efecto que posee eliminar o no los péptidos que no cumplan el criterio de calidad.

En la tabla 6 podemos ver la relación que se da entre los parámetros y el número de clústeres a detectar: a valores mayores de MAX_AA, menos clústeres; mientras que a mayores valores de MAX_PEPTIDES, más. Con todo, a tenor de la influencia de MAX_AA sobre el número total de clústeres tampoco podemos pensar que juegue un rol fundamental sobre el devenir de su número: en el promedio, la diferencia entre el menor y máximo resultados es de apenas 0.7 puntos: ni siquiera un solo clúster de diferencia.

Pépt. Calidad / simulación							
		MAX_AA					
		10	20	30	40	50	PROM(MPEP)
MAX PEPT	1	0	1	0	1	1	0,60
	5	1	2	3	4	4	2,80
	10	1	7	9	6	9	6,40
	15	4	10	11	9	13	9,40
	20	8	8	17	14	14	12,20
	30	4	18	22	23	20	17,40
	50	16	30	21	39	36	28,40
	PROM(MAA)	4,86	10,86	11,86	13,71	13,86	

Tabla 6. Número de péptidos de calidad identificados por simulación

Donde sí afecta considerablemente es en el número total de péptidos de calidad (computables a efectos de detección de clústeres). En las tablas 6 y 7 esto se evidencia notablemente, donde la cantidad de péptidos de calidad es directamente proporcional al aumento de los valores de los parámetros.

Considerar las tablas 7 y 8 juntas justifica que para identificar el máximo número de clústeres de calidad posible es preciso aumentar ambos parámetros.

Clústeres por simulación

		MAX_AA					
		10	20	30	40	50	PROM(MPEP)
MAX PEPT	1	1	1	1	1	1	1,00
	5	4	3	2	2	4	3,00
	10	3	4	4	3	3	3,40
	15	4	4	3	4	4	3,80
	20	4	4	4	4	5	4,20
	30	5	4	6	5	5	5,00
	50	8	6	5	5	4	5,60
	PROM(MAA)	4,14	3,71	3,57	3,43	3,71	

Tabla 7. Número de clústeres identificados por simulación

Pépt. Calidad / simulación (rel)

		MAX_AA					
		10	20	30	40	50	PROM(MPEP)
MAX PEPT	1	0,00	1,00	0,00	1,00	1,00	0,60
	5	0,20	0,40	0,60	0,80	0,80	0,56
	10	0,10	0,70	0,90	0,60	0,90	0,64
	15	0,27	0,67	0,73	0,60	0,87	0,63
	20	0,40	0,40	0,85	0,70	0,70	0,61
	30	0,13	0,60	0,73	0,77	0,67	0,58
	50	0,32	0,60	0,42	0,78	0,72	0,57
	PROM(MAA)	0,20	0,62	0,61	0,75	0,81	

Tabla 8. Relación entre el número de péptidos de calidad identificados y el número total de péptidos simulados

5.4. Precisión de la simulación

Anteriormente, en otras secciones, se ha mencionado que los péptidos pueden tener distintas conformaciones, y que a tal efecto Seidh es capaz de detectar estas y clasificar los péptidos según pertenezcan a una o a otra. Sin embargo, con frecuencia tan solo una de estas conformaciones es considerada natural.

Para determinar la precisión de las simulaciones de Seidh se ha llevado a cabo una simulación con valores de MAX_AA=30 y MAX_PEPTIDES=200. El criterio para determinar la precisión es su distancia, en RMSD, respecto a la conformación natural del péptido registrada en el banco de PDB.

Los péptidos clasificados como péptidos de calidad (n=166) no fueron clasificados en clústeres, ya que el objetivo de esta prueba es únicamente comparar la posibilidad de encontrar un péptido con una conformación similar a la natural registrada en PDB, y que esta pueda ser clasificada en el top 10 de predicciones.

Tras calcular la RMSD de todos ellos, y ordenarlos según la función de scoring de Seidh, obtenemos los siguientes resultados para el top 10 (n=10):

- 40% de éxito para una RMSD de 2 Å
- 70% de éxito para una RMSD de 4 Å
- 100% de éxito para una RMSD de 6 Å

Y los siguientes para el top 100 (n=100):

- 9% de éxito para una RMSD de 2 Å
- 91% de éxito para una RMSD de 4 Å
- 100% de éxito para una RMSD de 6 Å

Por tanto, si empleamos los mismos criterios empleados en la literatura, podemos empezar a considerar soluciones aceptables por debajo de los 4 angstrom de diferencia, y soluciones muy buenas por debajo de los 2 angstrom de diferencia.

Hay que destacar que algunos péptidos cercanos a la conformación original recibieron una puntuación más baja, escapándose así del top 10. Es posible que se deba a factores como la distancia respecto a la proteína de referencia.

En la ilustración número 13 pueden observarse hasta cuatro distintas conformaciones (coloreadas en rojo, con un solo péptido; en azul, tres; naranja, dos; y morado; 4). La conformación nativa ha sido coloreada en verde. Además, se ha coloreado en blanco el punto de unión (el primer aminoácido) para destacarlo sobre el resto.

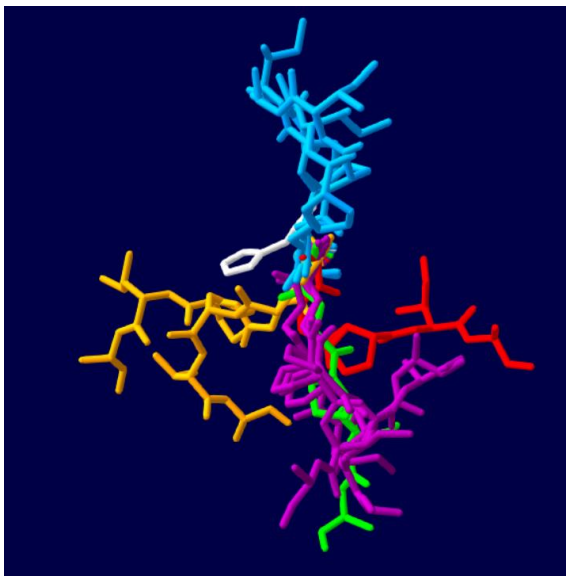

Ilustración 13. Superposición del top 10 de péptidos junto con la conformación original (en verde)

Por otro lado, en la ilustración número 14 podemos observar estos péptidos respecto a la proteína (en azul oscuro):

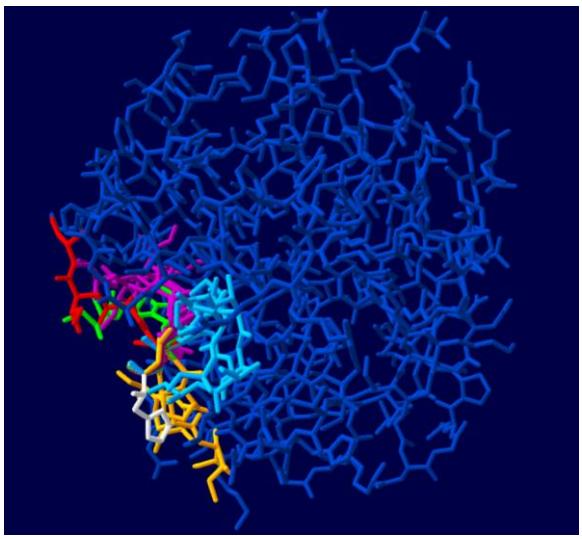

Ilustración 14. Superposición del top de péptidos junto con la conformación original (en verde) y la proteína (en azul oscuro).

5.5. Configuración recomendada

Si consideramos que los resultados aceptables comienzan a partir de que, al menos la mitad de los péptidos simulados sean de calidad, sería sensato proponer 20 como valor mínimo de MAX_AA. En cuanto a MAX_PEPTIDES, depende bastante de qué pretendamos hacer al ejecutar Seidh. Si tratamos de obtener una visión general de cómo el péptido puede ser afín a la proteína, posiblemente con valores entre 15-20 podamos identificar potenciales conformaciones que difieran lo suficiente. Por otro lado, si lo que pretendemos es realizar una simulación para tratar de obtener datos precisos, deberíamos recomendar un valor mínimo de 30.

A mayor disponibilidad de tiempo y potencia computacional, la configuración recomendada para ejecuciones habituales sería de 30 para MAX_AA y de, al menos, 50 o superiores para MAX_PEPTIDES. Si bien esto puede alargar el tiempo de ejecución total hasta 3-5 horas o incluso más, los resultados son más apropiados para trabajar con ellos posteriormente.

5.6. Conclusiones de la evaluación

Seidh depende altamente de los valores de sus parámetros MAX_AA y MAX_PEPTIDES. Con la configuración adecuada Seidh puede alcanzar a predecir las conformaciones naturales con cierta facilidad. Quizá la implementación de nuevas funciones de scoring permitan refinar aún más los resultados.

6. Conclusiones

La predicción de estructuras de moléculas como las proteínas o los ácidos nucleicos son uno de los grandes retos de la computación actualmente. Cada caso particular puede ser atajado con mayor o menor éxito con diferentes métodos, por lo que una tarea pendiente continúa siendo determinar en qué circunstancias es mejor decantarse por una metodología u otra.

Los algoritmos basados en la naturaleza son una renovada estrategia para afrontar tareas que tradicionalmente consumen tiempo y recursos. En este caso, el algoritmo del cuco demuestra ser un método eficaz para hallar estructuras de péptidos en las condiciones iniciales especificadas, ya que la modificación de los ángulos de los aminoácidos y su paulatina adición al péptido pueden recordar en cierta manera al ave buscando el nido idóneo para depositar su prole.

La implementación de un algoritmo en una librería para su explotación es un proceso largo, que requiere de múltiples pruebas y una gran tarea de investigación previas. En el diseño de la simulación hay varios aspectos que han de ser tenidos en cuenta a la hora de determinar qué parámetros son correctos y cuales pueden implicar un enlentecimiento inaceptable. No obstante, ya que como se ha demostrado la simulación puede ser diferente de un complejo proteína-péptido a otro, es preferible dejar cierto margen de maniobra al usuario que desee modificar estos aspectos de la simulación.

Seidh presenta algunos puntos clave muy positivos:

- Es un método “fine-grained” (nítido), que tiene en cuenta todos los átomos de las moléculas en sus predicciones.
- Es rápido y preciso, ya que incluso en configuraciones mínimas puede ofrecer resultados bien orientados.
- Es fácilmente exportable, e implementable en otros sistemas. Ya que se basa en Python 2.7 y la librería BioPython, dos estándares de la bioinformática, puede utilizarse en otros proyectos sin mayor problema.

Sin embargo, hay algunos aspectos que deben pulirse, como la posibilidad de predecir el aminoácido con independencia del lugar que ocupe en la secuencia el punto de unión proporcionado por el usuario.

También es posible comprobar cómo Seidh se comporta de forma diferente en función de la longitud del péptido a predecir: a cadenas más cortas (5-6 residuos) ofrece resultados más similares a la conformación registrada en PDB que con cadenas más largas (7-10 residuos).

En definitiva, Seidh es la prueba de que los algoritmos inspirados en la naturaleza son una herramienta útil en el reto que supone la predicción de estructuras de péptidos ligados a proteínas. También pone de manifiesto la creciente necesidad por ordenadores capaces de realizar más operaciones simultáneamente, y más rápido.

En el futuro, las sucesivas versiones de Seidh tratarán de incorporar mejoras y correcciones para los aspectos destacados en la memoria, incluyendo:

- Soporte para predicción en sentido inverso, mediante el uso de una librería alternativa a PeptideBuilder o su modificación.
- Mejora del rendimiento reescribiendo algunas funciones en C.
- Soporte de predicciones ciegas, en las que se desconozca el punto de unión.
- Soporte de computación paralela empleando GPU.
- Implementación de otras funciones de scoring.

El código, tanto de la librería Seidh como de su implementación en un servidor web basado en Flask pueden encontrarse en los siguientes repositorios de GitHub:

Seidh

https://github.com/i22deroe/seidh_1.0

Seidh webserver

https://github.com/i22deroe/seidh_webserver

7. Glosario

Términos relacionados con el área de conocimiento de las ciencias naturales

Aminoácido: molécula orgánica con un grupo amino ($NH2^-$) y un grupo carboxilo ($COOH^-$). Son la unidad básica de las proteínas.

Angstrom (Å): unidad de medida equivalente a 10^{-10} m.

Ángulo phi: ángulo de torsión de un aminoácido determinado por el carbono alfa y el átomo de nitrógeno.

Ángulo psi: ángulo de torsión de un aminoácido determinado por el carbono alfa y el carbono beta.

Cristalografía de rayos X: técnica de visión para el estudio de materiales basada en la difracción de rayos X.

Fuerzas de Van der Waals: conjunto de fuerzas atractivas y repulsivas entre moléculas.

Microscopía electrónica de alta resolución: técnica de visión basada en el bombardeo de electrones de una superficie para obtener una imagen tridimensional.

PDB: Banco de datos de proteínas, del inglés *Protein data bank.*

Potencial electrostático: variación de energía necesaria para desplazar una carga positiva.

Proteína: molécula orgánica compuesta por aminoácidos. Suelen ser de gran tamaño.

Proteína intrínsecamente desordenada: proteína caracterizada por carecer de una estructura terciaria definida.

Proteína: estructura: una proteína tiene hasta cuatro estructuras:

- Estructura primaria: su secuencia de aminoácidos.
- Estructura secundaria: pliegues y estructuras básicos, como hélices álfa y pliegues beta.

- Estructura terciaria: uniones y pliegues sobre sí misma, fruto de puentes de hidrógeno y disulfuro.
- Estructura cuaternaria: unión de varias cadenas polipeptídicas.

Péptido: molécula orgánica compuesta por aminoácidos, de sensiblemente menor tamaño que una proteína.

Residuo: cada uno de los aminoácidos que componen una proteína o un péptido.

Resonancia magnética nuclear: técnica de visión mediante campos magnéticos y radiación no ionizante.

Términos relacionados con el área de conocimiento de (bio) informática

Algoritmo: conjunto ordenado de pasos para la consecución de un objetivo.

Algoritmo basado en la natualeza: algoritmo que emula el comportamiento de algún aspecto de la naturaleza, como los animales.

Algoritmo de búsqueda: algoritmo optimizado para dar con un valor determinado en el mínimo de pasos posible.

Apache: servidor web HTTP.

Biopython: librería de Python diseñada para bioinformática.

Benchmark, test o prueba de: conjunto de pruebas que tratan de comprobar el rendimiento de un sistema.

C: lenguaje de programación de bajo nivel desarrollado por Dennis Ritchie entre 1969 y 1972, altamente eficiente.

Celery: librería de Python para el encadenamiento de tareas en segundo plano.

Clúster: grupo de entidades con características comunes.

CSS: lenguaje de marcado de estilos

Diagrama de Gantt: gráfico de organización temporal para la distribución de tareas y la gestión de recursos.

FASTA: código para la representación de nucleótidos y aminoácidos.

Fichero ENT: fichero entity, del Protein Data Bank. Posee varios cadenas de proteínas y péptidos.

Fichero FAS: fichero fasta. Contiene una cadena de nucleótidos o aminoácidos.

Fichero PDB: fichero protein data bank. Posee una o varias cadenas de proteínas o péptidos.

Flask: framework y librería Python para el desarrollo de servidores web.

Framework: conjunto de herramientas, librerías y gramáticas construidas sobre un lenguaje de programación para llevar a cabo una tarea de una forma concreta.

GPU: unidad de procesamiento gráfico, comúnmente denominada “tarjeta gráfica”.

HTML: lenguaje de marcado estándar en páginas web.

Javascript: lenguaje de programación orientado a objetos que complementa a HTML.

Jinja2: librería de lenguaje de plantillas para Python.

Librería: conjunto de funciones y constantes importables en un lenguaje de programación.

Machine Learning: conjunto de técnicas y área de conocimiento que tiene por fin dotar de procesos asimilables al aprendizaje a las máquinas.

Multiprocessing: librería de Python para permitir la compatibilidad con el procesado multinúcleo.

Máquina de vectores de soporte: algoritmo de machine learning perteneciente a la categoría de “algoritmos caja-negra”. Limitado a algoritmos de predicción y lasificación.

NearestNeighbour: algoritmo de machine learning para la clasificación de entidades.

Numpy: librería de Python con numerosas funciones relacionadas con los números y las matemáticas.

PeptideBuilder: librería de Python para calcular la geometría de aminoácidos y construir estructuras PDB.

Python: lenguaje de programación orientado a objetos diseñado en 1991 por Guido van Rossum. Especialmente extendido en el ámbito de la bioinformática.

Red neuronal: algoritmo de machine learning perteneciente a la categoría de "algoritmos caja negra". Es enormemente versátil, empleado en reconocimiento de caracteres, voz, imágenes… entre otras muchas.

Redis: motor de bases de datos clave:valor.

Servidor web: aplicación informática que permite conexiones externas para proveer un servicio fuera del entorno donde se ejecuta, que es accesible por un protocolo web.

Unix: sistema operativo que sustentó la base de sistemas operativos modernos como MacOSX o Linux.

Wsgi: interfaz que permite que aplicaciones y servicios que implementan servidores web lancen aplicaciones basadas en código Python.

Términos relacionados con Seidh

Binding point: posición relativa ficticia de un aminoácido perteneciente a un péptido. En bioinformática representa el punto de partida para la predicción de la estructura del péptido.

Función de scoring: función que evalúa una predicción de un algoritmo. En el caso de Seidh, evalúa las conformaciones de los péptidos.

Limitaciones: del inglés, *constraints*, representa las líneas rojas no cruzables por el algoritmo. Determina y perfila los resultados.

Modelado por homología: proceso de predicción de estructuras de proteínas y péptidos basándose en *templates* o plantillas de otras estructuras conocidas.

Método "coarse grained": método de predicción de estructuras de proteínas y péptidos que no consideran la totalidad de átomos implicados. Suelen ser rápidos y eficientes.

Método "fine grained": método de predicción de estructuras de proteínas y péptidos que consideran la totalidad de los átomos implicados. Suelen ser más lentos, pero ofrecen resultados más realistas.

Método ab initio: método de predicción de estructuras de proteínas y péptidos que parte de cero, sin considerar estructuras semejantes conocidas. Se basa únicamente en la interpretación de las leyes de la física que rigen la mecánica de las moléculas orgánicas.

Métodos ciegos: método de predicción de estructuras de proteínas y péptidos en los que el binding point o sitio de unión no es conocido y, por tanto, requiere su búsqueda como primer paso.

Métodos no-ciegos: método de predicción de estructuras de proteínas y péptidos en los que el binding point o sitio de unión es conocido.

Peptidb: conjunto de entidades PDB ordenadas y clasificadas para la creación de un banco de pruebas que satisfaga las condiciones iniciales de Seidh.

Sitio de unión: ver binding point.

8. Bibliografía

1. Jayachandran, G., Vishal, V., García, A. and Pande, V. (2007). Local structure formation in simulations of two small proteins. Journal of Structural Biology, 157(3), pp.491-499.
2. Schwede, T. and Peitsch, M. (2008). Computational Structural Biology: Methods and Applications. World Scientific.
3. London N, Raveh B, Schueler-Furman O. Modeling Peptide–Protein Interactions. Methods in Molecular Biology. 2011;375-398.
4. London N, Movshovitz-Attias D, Schueler-Furman O. The Structural Basis of Peptide-Protein Binding Strategies. Structure. 2010;18(2):188-199.
5. Helles, G. (2008). A comparative study of the reported performance of ab initio protein structure prediction algorithms. Journal of The Royal Society Interface, 5(21), pp.387-396.
6. Roy, A., Kucukural, A. and Zhang, Y. (2010). I-TASSER: a unified platform for automated protein structure and function prediction. Nature Protocols, 5(4), pp.725-738.
7. Schindler C., Zacharias M. (2017) Application of the ATTRACT Coarse-Grained Docking and Atomistic Refinement for Predicting Peptide-Protein Interactions. In: Schueler-Furman O., London N. (eds) Modeling Peptide-Protein Interactions. Methods in Molecular Biology, vol 1561. Humana Press, New York, NY
8. pepATTRACT: Blind, proteome-wide peptide-protein docking [Internet]. Bioserv.rpbs.univ-paris-diderot.fr. 2017 [cited 29 may 2018]. Available from: http://bioserv.rpbs.univ-paris-diderot.fr/services/pepATTRACT
9. Ciemny M.P., Kurcinski M., Kozak K.J., Kolinski A., Kmiecik S. (2017) Highly Flexible Protein-Peptide Docking Using CABS-Dock. In: Schueler-Furman O., London N. (eds) Modeling Peptide-Protein Interactions. Methods in Molecular Biology, vol 1561. Humana Press, New York, NY

10. CABS-dock: server for protein-peptide docking [Internet]. Biocomp.chem.uw.edu.pl. [cited 29 may 2018]. Available from: http://biocomp.chem.uw.edu.pl/CABSdock/learn_more

11. Geng C., Narasimhan S., Rodrigues J.P.G.L.M., Bonvin A.M.J.J. (2017) Information-Driven, Ensemble Flexible Peptide Docking Using HADDOCK. In: Schueler-Furman O., London N. (eds) Modeling Peptide-Protein Interactions. Methods in Molecular Biology, vol 1561. Humana Press, New York, NY

12. Torchala M, Moal I, Chaleil R, Fernandez-Recio J, Bates P. SwarmDock: a server for flexible protein–protein docking. Bioinformatics. 2013;29(6):807-809.

13. London N, Movshovitz-Attias D, & Schueler-Furman O (2010). The structural basis of peptide-protein binding strategies. Structure (London, England : 1993), 18 (2), 188-99 PMID: 20159464

9. Anexos

Listado de anexos

Anexo I.- Documentación técnica de Seidh 1.0

Anexo II.- Tablas

Anexo I.- Documentación técnica de Seidh

(comienza en la página siguiente)

Seidh 1.0

Documentación técnica

Enrique Delgado Rodríguez

Brian Jiménez García

Contenido

Introducción

> *En el antiguo idioma nórdico,* ***seiðr*** *(seidh), era un tipo de brujería o hechizo practicado en la edad de hierro escandinava con la finalidad de predecir el futuro. Esta práctica fue asociada a menudo con Odín, por su vasto conocimiento atemporal.*

Las interacciones entre proteínas y péptidos representan una gran fracción de todos los procesos bioquímicos que tienen lugar en cualquier organismo vivo. Hoy en día, en pleno desarrollo de la medicina personalizada, el conocimiento sobre cómo se acoplan las proteínas y los péptidos resulta de vital importancia para el desarrollo de nuevos tratamientos y terapias encaminadas a la cura de enfermedades genéticas, oncológicas, autoinmunes y un largo etcétera.

La predicción de estructuras tridimensionales del complejo proteína-péptido (también proteína-proteína) es un sector muy concreto de la biología computacional y estructural. Implica un profundo conocimiento de las interacciones entre las moléculas implicadas, y cómo estas pueden afectar a las estructuras secundaria y terciaria de los complejos; además de un potente banco de recursos bioinformáticos para el manejo de esta información.

La librería Seidh implementa un algoritmo basado en el vuelo de Cuculus canorus (cuco común), un ave parásita que en época de puesta invade nidos de otros pájaros para depositar sus huevos. Este movimiento, que alterna movimientos cortos seguidos de vuelos más largos, resulta particularmente interesante para la predicción de estructuras.

La librería también incluye otros módulos auxiliares para llevar a cabo su cometido. Algunos de estos módulos son de recomendado acceso antes de la invocación de la función principal. Para ello, es recomendable leer detenidamente los requisitos de uso de Seidh y cómo pueden estos otros módulos ayudarte para obtener la estructura del péptido problema.

Requisitos y guía rápida de uso

Seidh 1.0 está escrito para Python 2.7 y requiere los siguientes módulos de Python:

- biopython 1.66 o superior
- numpy 1.14.0 o superior
- PeptideBuilder 1.0.3 o superior

Para utilizarla, basta con importar el módulo principal:

```
from seidh import seidh
```

Para llamar la función principal solo es necesario indicar los objetos BioPython PDB que contengan la proteína y el binding point, junto con un string que contenga la secuencia FASTA.

```
seidh ( protein, peptide, fasta )
```

Adicionalmente, también puede especificar el número de núcleos de procesador con los que ejecutar la simulación. Por ejemplo, para una simulación que emplee cuatro núcleos:

```
seidh ( protein, peptide, fasta, NUM_PROC=4 )
```

Es preciso notar que Seidh incluye en su librería auxiliar funciones para facilitar la tarea, como open_fasta_peptide(), que independientemente de que se le proporcione un fichero .FAS o una secuencia FASTA, la función comprueba que sea una secuencia válida y devuelve un string conteniendo la secuencia no-ambigua. Para más información, consulta la documentación.

Seidh lleva a cabo una predicción “fine-grained”. En su configuración básica, creará hasta 50 péptidos de prueba, que clasificará y ordenará en función de su calidad. Tenga en cuenta que puedes obtener varios péptidos como resultado, en caso de que Seidh determine que existen varias conformaciones posibles.

Módulos

Módulo principal: seidh.py

En seidh.py se encuentra la función principal de la librería, seidh().

Funciones

seidh (protein, peptide, fasta, NUM_PROC=1)

seidh() precisa de dos objetos BioPython PDB (protein y peptide), así como una secuencia FASTA en un objeto string. Adicionalmente, se puede indicar el número de núcleos con los que correr la simulación a través de NUM_PROC.

Tras su ejecución, seidh() creará los siguientes archivos:

- test_#.pdb y discarded_test_#.pdb. Son los ficheros que genera cada simulación de Seidh. Seidh puede configurarse para no guardarlos.
- test_finest_for_cluster_#.pdb. Son los ficheros solución. Cada uno representa la solución a cada uno de los clústeres (conformaciones diferentes) identificados.
- #_cores_simmulation_summary_#. Es un fichero con el resumen de la simulación. Contiene información útil sobre tiempos y calidad de los resultados de la simulación.

Por favor, tenga en cuenta que los ficheros .pdb se crearán dentro de la carpeta pdb_files/predictions si no se especifica otra cosa en lib/constants/constants.py. También es posible evitar la generación de ficheros test_ y discarded_test_ comentando las líneas 67 y 70 de seidh.py.

Seidh le irá informando sobre el resultado de la simulación. Es relativamente normal que se produzcan colisiones entre el péptido y la proteína, sobre todo en péptidos de un gran número de residuos. Tenga en cuenta que detener la simulación no borrará los ficheros ya creados (test_ y discarded_test_), pero no será posible realizar una clasificación de las predicciones ni identificación de clústeres.

Módulos auxiliares: auxiliar.py

En auxiliar.py se encuentran las funciones básicas para el manejo de ficheros PDB, así como otras funciones para manipular secuencias FASTA.

Constantes

fasta_ambiguous

fasta_ambiguous es un objeto de tipo list () que contiene todos los caracteres que pueden darse en una secuencia FASTA. Se utiliza para comprobar que una secuencia introducida por el usuario se corresponde en efecto con una secuencia FASTA.

fasta_unambiguous

fasta_unambiguous es un objeto de tipo list () que contiene los caracteres que pueden darse en una secuencia FASTA y coinciden con un aminoácido. Ya que en ocasiones el código FASTA es ambiguo, y un carácter puede representar un comodín, se hace necesario eliminar esta ambigüedad.

ambiguity_dictionary

ambiguity_dictionary es un objeto de tipo dict () que contiene la equivalencia entre los símbolos ambiguos y los no ambiguos.

Funciones

is_fasta (fasta)

is_fasta () lee una cadena de caracteres y determina si es una cadena FASTA válida (devolviendo True) o no (devolviendo False).

fasta_ambiguity (fasta)

fasta_ambiguity () lee una cadena de caracteres y, en caso de que esta sea una cadena FASTA válida, devuelve esa misma cadena sin ambigüedades. Cuando detecta un comodín, selecciona una de las alternativas que representa de forma aleatoria. En caso de que la secuencia FASTA no sea válida, devuelve False.

open_pdb_protein (file)

open_pdb_protein () lee una cadena de caracteres con el nombre de un fichero .pdb que se encuentre en el directorio especificado en la configuración para las proteínas. En caso de que el fichero exista, devuelve el objeto pdb asociado. En caso contrario, devuelve False.

open_pdb_peptide (file)

open_pdb_peptide() lee una cadena de caracteres con el nombre de un fichero .pdb que se encuentre en el directorio especificado en la configuración para los péptidos. En caso de que el fichero exista, devuelve el objeto pdb asociado. En caso contrario, devuelve False.

open_pdb_entity (file, chain)

open_pdb_entity () lee una cadena de caracteres con el nombre de un fichero .pdb que se encuentre en el directorio especificado en la configuración para los péptidos, junto con la cadena en la que se encuentra la estructura deseada. En caso de que el fichero y la cadena existan, devuelve el objeto pdb asociado. En caso contrario, devuelve False. Es preciso notar que por defecto open_pdb_entity () buscará la cadena en el modelo 0.

cut_peptide (peptide, fasta)

cut_peptide () lee un objeto Biopython PDB correspondiente a un péptido y una cadena de caracteres conteniendo la secuencia FASTA correspondiente completa, y devuelve el resto de caracteres pendientes. Ejemplo: si el péptido posee 3 residuos y la secuencia FASTA 7, se devolverán los últimos 4 residuos en formato FASTA. En caso de que la secuencia indicada no sea FASTA, devolverá False. Es preciso notar que cut_peptide () no detecta ambigüedades en la secuencia FASTA, por lo que esto correrá de cuenta del usuario.

save_structure (structure, name)

save_structure () lee un objeto Biopython PDB correspondiente a un péptido o una proteína y una cadena de caracteres correspondiente a un nombre. Esta función guardará la estructura en un fichero PDB con el nombre indicado en el

directorio especificado en la configuración. Es preciso notar que save_structure () no tiene en cuenta si existe o no el fichero donde se va a guardar la estructura, por lo que existe riesgo de sobreescritura si el usuario no toma las precauciones adecuadas.

Módulos auxiliares: clusterize.py

En clusterize.py se encuentran las funciones empleadas para la detección de distintas conformaciones del péptido (clústeres).

Funciones

get_ca_atoms (ids_list, folder):

get_ca_atoms () lee una lista de ids contenidas en un objeto tipo list () y una cadena de caracteres que contiene una carpeta. En esta carpeta buscará los archivos correspondientes a ids_list. Esta función devuelve una lista de coordenadas correspondientes únicamente al esqueleto de carbono de las estructuras leídas.

clusterize (sorted_ids, folder):

clusterize () lee una lista de ids contenidas en un objeto tipo list () y una cadena de caracteres que contiene una carpeta, para pasarlas a la función get_ca_atoms (). Una vez posee las coordenadas del esqueleto de carbono de cada estructura, calcula la RMSD entre las distintas estructuras y determina la aparición de un clúster cuando la distancia es mayor a 2.0 Å. Devuelve un objeto tipo dict () que contiene la ID del clúster y la lista de IDs correspondientes a las estructuras clasificadas.

Módulos auxiliares: distance.py

En distance.py se encuentran las funciones relacionadas con la distancia espacial de las estructuras.

Funciones

VdW_minima (molecule_1, molecule_2):

VdW_minima () lee dos moléculas en formato adaptado Seidh para calcular si estas colisionan o no. El criterio empleado implica los radios de Van der Waals y las posiciones relativas de los átomos de cada molécula. Los parámetros han sido ajustados experimentalmente. En caso de que se detecte una colisión, la función devuelve False; mientras que si no se detecta ninguna, devuelve True.

inner_stability (peptide, adapted_peptide):

inner_stabilit () lee un objeto BioPython PDB y una molécula en formato adaptado Seidh, correspondiente a dicho objeto BioPython PDB. Siguiendo la lógica de VdW_minima, la función detecta si se ha producido algún tipo de colisión interna. Los parámetros han sido ajustados experimentalmente. En caso de que se detecte una colisión, la función devuelve False; mientras que, si no se detecta ninguna, devuelve True.

molecule_contacts (peptide, protein):

molecule_contacts () lee dos moléculas en formato adaptado Seidh correspondientes a una proteína y un péptido, calcular el índice de contactos entre ellas. Es preciso notar que se asume la no-colisión, ya que esta función es llamada después de que VdW_minima e inner_stability hayan devuelto ambas True. La función devuelve el índice calculado en base al péptido.

Módulos auxiliares: levy_distribution.py

En levy_distribution.py se encuentra la función generadora de valores pertenecientes a la distribución de Lévy.

Funciones

levy_distro (mu):

levy_distro () lee un entero correspondiente al parámetro que determina la localización de la distribución. En las simulaciones de Seidh se emplea un valor de mu=0. La función devuelve un número perteneciente a la distribución de Lévy. Este conjunto de valores está fuertemente concentrado en torno a los valores (0,1), mientras que la cola de la distribución es más baja. Esto permite la alternancia de valores pequeños más frecuentes con valores grandes de frecuencia escasa.

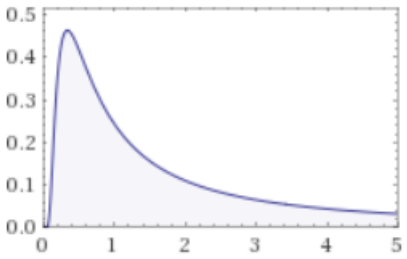

Ilustración 1, Distribución de frecuencias de la distribución Lévy con mu=0.

Módulos auxiliares: cuckoo.py

En cuckoo.py se encuentran todas las funciones que implementan el algoritmo del cuco en Seidh. Contiene tanto las funciones necesarias para llevar a cabo esta tarea en sistemas con un solo núcleo como en aquellos en los que es posible emplear más de uno.

Funciones

new_cuckoo (aminoacid):

new_cuckoo () lee una cadena de caracteres codificando un aminoácido, y devuelve un objeto Geometry que contiene la geometría del aminoácido. Los ángulos psi y phi del aminoácido son iniciados con valores aleatorios dentro del rango [-180,180]. No se modifica el ángulo omega.

modify_cuckoo (cuckoo):

modify_cuckoo () lee un objeto Geometry que contiene la geometría de un aminoácido. Entonces, genera un valor aleatorio según la distribución de Lévy para modificar los ángulos psi y phi, permitiendo únicamente valores dentro del rango [-180,180]. No se modifica el ángulo omega. Devuelve el objeto Geometry modificado, si los valores aleatorios caen dentro del citado rango, o el original, en caso contrario.

new_flock (peptide, aminoacid, protein):

new_flock () lee dos objetos BioPython PDB correspondientes al péptido que hace de binding spot y la proteína a la que fijarse, junto con un carácter correspondiente al aminoácido a añadir al péptido. Esta función es la encargada de generar las pruebas para el aminoácido en cuestión, y devolverá una señal False en caso de que no consiga resultados tras un número determinado de intentos. En caso de poder generar los aminoácidos sin colisiones, devolverá una lista de objetos Geometry (flock) y la versión adaptada de Seidh (adapted_flock).

flock_migration (flock, adapted_flock, protein, peptide):

flock_migration () lee una lista de objetos Geometry, una lista de versiones adaptadas de Seidh, y dos objetos BioPython PDB correspondientes al péptido que hace de binding spot y la proteína a la que fijarse. Es la función encargada de modificar los objetos Geometry contenidos en flock, y modificar la versión adaptada de Seidh (adapted_flock) de forma acorde. En caso de que la modificación sea mejor que la versión original, esta última es sustituida. En caso contrario, prevalece la versión original. flock_migration() devuelve una lista de energías (resultados de la función de scoring) asociadas a los objetos Geometry contenidos en la lista flock, que también es devuelta con las modificaciones pertinentes.

cuckoo_migration (cuckoo, adapted_cuckoo, protein, peptide):

cuckoo_migration () lee un objeto Geometry, una versión adaptada de Seidh de dicho objeto, y dos objetos BioPython PDB correspondientes al péptido que hace de binding spot y la proteína a la que fijarse. cuckoo_migration () se encarga de modificar el objeto Geometry y modificar la versión adaptada de Seidh de forma acorde. En caso de que la modificación sea mejor que la versión original, esta última es sustituida. En caso contrario, prevalece la versión original. cuckoo_migration () devuelve una energía (resultado de la función de scoring) asociada al objeto Geometry que también devuelve con las modificaciones pertinentes. Esta función es análoga, junto con multi_migration (), a flock_migration (), ya que se utiliza en entornos multiprocesador para su mismo fin.

multi_migration (flock, adapted_flock, protein, peptide, NUM_PROC): cuckoo_list, energy_list

multi_migration () lee una lista de objetos Geometry, una lista de sus versiones adaptadas de Seidh, dos objetos BioPython PDB correspondientes al péptido que hace de binding spot y la proteína a la que fijarse, y el número de procesadores a emplear durante la simulación. Esta función se encarga de invocar las funciones del módulo multiprocess de Python para llevar a cabo varias modificaciones de los objetos Geometry en paralelo, y así ahorrar tiempo de computación. Devuelve una lista de energías (resultados de la función de

scoring) asociadas a los objetos Geometry contenidos en la lista flock, que también es devuelta con las modificaciones pertinentes. Esta función es análoga, junto con cuckoo_migration (), a flock_migration (), ya que se utiliza en entornos multiprocesador para su mismo fin.

rank (flock_energy, flock, diff=1)

rank () lee dos listas: una contiene energías (resultados de la función de scoring) asociadas a los objetos Geometry contenidos en la otra. Además, se puede indicar un parámetro diff que determina el funcionamiento de la función rank (): ordenar los objetos Geometry de mayor a menor y eliminar los diez peores resultados (diff=0), o solo ordenarlos. rank () devuelve una lista ordenada de objetos Geometry.

cuckoo (aminoacid_sequence, peptide, protein, NUM_PROC)

cuckoo () es la función principal de cuckoo.py. Lee una cadena de caracteres que contiene una secuencia de aminoácidos en formato FASTA, y dos objetos BioPython PDB correspondientes al péptido que hace de binding spot y la proteína a la que fijarse, y el número de procesadores a emplear durante la simulación. Se encarga de llevar acabo el algoritmo del cuco para producir un objeto BioPython PDB que corresponde al péptido calculado, que devuelve si no se ha activado la señal False de new_flock (). En este último caso, a su vez, devuelve una señal False como indicativo de que el cálculo no ha ido bien.

Módulo de cálculos de energía: driver.py y scoring.py

En driver.py y scoring.py se encuentran dos funciones clave del funcionamiento de Seidh: SeidhAdapt y seidh_calculate_energy.

Funciones

SeidhAdapt (molecule):

SeidhAdapt () lee un objeto BioPython PDB correspondiente a una proteína o a un péptido, y mediante la base de datos interna genera una serie de listas que contienen la información necesaria para llevar a cabo múltiples funciones de Seidh, incluyendo el cálculo de energía y la detección de colisiones. Devuelve una molécula adaptada de Seidh, una lista multidimensional que comprende los siguientes datos:

- Índice 0: Coordenadas atómicas. Poseen a su vez tres índices, correspondientes a los ejes x, y, z.
- Índice 1: Cargas eléctricas.
- Índice 2: Energías de Van der Waals.
- Índice 3: Radios de Van der Waals.

seidh_calculate_energy (rec, lig):

seidh_calculate_energy () lee dos moléculas adaptas de Seidh correspondientes a una proteína y un péptido. Considerando la teoría de los potenciales de Lennard-Jones y el resto de elementos contenidos en la molécula adaptada de Seidh, devuelve un número: el score o puntuación. Este score es del tipo "cuanto más grande, mejor", significando esto que pares de moléculas que reciban una mayor puntuación serán más estables.

Módulos de propiedades y configuración: props.py

En constants.py se encuentra la configuración paramétrica de Seidh. Es posible modificarlo para ajustar su funcionamiento

Propiedades

predictions

Esta constante refiere la carpeta en la que se guardarán los ficheros PDB fruto de las predicciones de Seidh. Por defecto es *'pdb_files/predictions'.*

peptides

Esta constante refiere la carpeta de la que se leerán los ficheros PDB correspondientes a los péptidos (usualmente binding spots). Por defecto es *'pdb_files/peptides'.*

proteins

Esta constante refiere a la carpeta de la que se leerán los ficheros PDB correspondientes a las proteínas. Por defecto es *'pdb_files/proteins'*.

entities

Esta constante refiere a la carpeta de la que se leerán los ficheros ENT correspondientes a la base de datos de PDB. Por defecto es *'pdb_files/entities'*.

MAX_AA

Esta constante marca el número máximo de pruebas por aminoácido. Disminuirla mejora el rendimiento, pero disminuye sensiblemente la precisión en las simulaciones. El valor mínimo debe ser 10.

MAX_PEPTIDES

Esta constante marca el número máximo de péptidos por simulación. Disminuirla acorta directamente el tiempo de la simulación, pero disminuye sensiblemente la posibilidad de detectar todas las conformaciones posibles. El valor mínimo es 1.

MIN_CONTACT

Esta constante marca el índice de contacto mínimo entre las superficies del péptido y la proteína. Disminuirla permite la clasificación de un mayor número de péptidos, pero podrían ser conformaciones de baja calidad. No se recomienda aumentarla por encima de 0.95, ya que aumentarían así los falsos negativos.

Ejemplo de caso de uso

Como se detallaba en la guía rápida, Seidh puede ser invocado suficientemente con la función seidh (), especificando la proteína, el péptido (binding point) y la secuencia FASTA a añadir al péptido. Sin embargo, aprovechando las funciones contenidas en auxiliar.py, podemos reducir errores que precipiten una larga simulación al fracaso.

Ejemplo de uso, con los ficheros 1AWR_protein.pdb en 'pdb_files/proteins', 1AWR_peptide.pdb en 'pdb_files/peptides' y 1AWR_fasta.pdb en pdb_files/peptides. En el código, empleamos la función cut_peptide () para asegurarnos de que no repetimos los aminoácidos contenidos en el binding point. Si este por ejemplo consiste de los aminoácidos 'HA', y la secuencia completa del péptido es 'HAGPIA', podemos encontrarnos con que los péptidos simulados responden a la estructura primaria 'HAHAGPIA'. Podemos considerar una buena práctica el uso de cut_peptide () antes de pasar la secuencia FASTA a seidh ().

```
from lib.auxiliar import *
from seidh import seidh

protein = pdb_open_protein ( '1AWR_protein.pdb' )
peptide = pdb_open_peptide ( '1AWR_peptide.pdb' )
fasta = open_fasta_peptide ( '1AWR_fasta.pdb' )

fasta = cut_peptide ( peptide, fasta )

seidh ( protein, peptide, fasta, 4)
```

Anexo II – Tablas

Todas las tablas siguen el mismo diseño. Son tablas de doble entrada que relacionan los valores descritos en el título de la tabla según la configuración pareada de MAX_AA y MAX_PEPT. En la columna y fila final se recoge el promedio de cada variable, para así aislar el efecto que posee por separado la variación de cada una. Se han utilizado dos códigos de color en función de lo que resulte más deseable: blanco (más deseable) y rojo (menos deseable) para los datos de la tabla; y verde (más deseable) y amarillo (menos deseable) para el resumen en la fila y columna últimas.

Índice de tablas

Tiempo de ejecución							
		MAX_AA					
		10	20	30	40	50	PROM(MPEP)
MAX PEPT	1	0:00:31	0:01:30	0:06:54	0:03:35	0:12:15	0:04:57
	5	0:04:19	0:17:31	0:23:13	0:44:52	0:29:53	0:23:58
	10	0:06:38	0:22:26	0:45:12	0:58:26	1:59:11	0:50:23
	15	0:11:25	0:34:42	1:00:12	1:47:33	1:51:59	1:05:10
	20	0:18:18	0:57:53	1:27:48	1:59:36	2:48:14	1:30:22
	30	0:28:35	1:18:49	1:38:42	4:22:07	5:11:04	2:35:51
	50	1:00:01	1:43:48	2:59:43	4:23:24	9:52:54	3:59:58
	PROM(MAA)	0:18:32	0:45:14	1:11:41	2:02:48	3:12:13	

Tabla 3. Tiempo de ejecución

En esta tabla se describe el tiempo total de ejecución de una simulación en formato H:MM:SS. Según la clasificación, el resultado sigue la lógica “menos es mejor”, indicando que un tiempo total de ejecución es más deseable cuanto más pequeño sea.

Tiempo de ejecución (por intento)							
		MAX_AA					
		10	20	30	40	50	PROM(MPEP)
MAX PEPT	1	0:00:31	0:01:30	0:01:23	0:03:35	0:03:04	0:02:01
	5	0:03:05	0:07:18	0:09:40	0:12:28	0:21:21	0:10:46
	10	0:05:32	0:14:57	0:18:50	0:34:22	0:37:15	0:22:11
	15	0:08:09	0:20:30	0:30:06	0:41:22	0:54:11	0:30:52
	20	0:08:56	0:28:14	0:35:50	1:01:20	1:16:28	0:42:10
	30	0:14:32	0:36:23	1:06:34	1:32:31	1:58:08	1:05:37
	50	0:24:48	1:01:04	1:39:51	2:36:47	3:15:02	1:47:30
	PROM(MAA)	0:09:22	0:24:17	0:37:28	0:57:29	1:12:13	

Tabla 4. Tiempo de ejecución ajustado por intentos

En esta tabla se describe el tiempo total de ejecución de una simulación ajustado por el número de intentos en formato H:MM:SS. Según la clasificación, el resultado sigue la lógica “menos es mejor”, indicando que un tiempo total de ejecución es más deseable cuanto más pequeño sea.

Intentos (promedio)							
		MAX_AA					
		10	20	30	40	50	PROM(MPEP)
MAX PEPT	1	1,00	1,00	5,00	1,00	4,00	2,40
	5	1,40	2,40	2,40	3,60	1,40	2,24
	10	1,20	1,50	2,40	1,70	3,20	2,00
	15	1,40	1,69	2,00	2,60	2,07	1,95
	20	2,05	2,05	2,45	1,95	2,20	2,14
	30	1,97	2,17	1,48	2,83	2,63	2,22
	50	2,42	1,70	1,80	1,68	3,04	2,13
	PROM(MAA)	1,63	1,79	2,50	2,19	2,65	

Tabla 3. Promedio del número de intentos por péptido

En esta tabla se describe el número medio de intentos por simulación. Según la clasificación, el resultado sigue la lógica “menos es mejor”, indicando que un tiempo total de ejecución es más deseable cuanto más pequeño sea.

Tiempo de cálculo por péptido							
		MAX_AA					
		10	20	30	40	50	PROM(MPEP)
MAX PEPT	1	0:00:30	0:01:29	0:06:54	0:03:34	0:12:14	0:04:56
	5	0:00:51	0:03:29	0:04:38	0:08:58	0:04:32	0:04:30
	10	0:00:39	0:02:14	0:04:31	0:05:38	0:11:55	0:04:59
	15	0:00:45	0:02:18	0:04:00	0:06:57	0:07:27	0:04:18
	20	0:00:54	0:02:53	0:04:23	0:05:58	0:08:02	0:04:26
	30	0:00:56	0:02:37	0:03:17	0:08:47	0:10:22	0:05:12
	50	0:01:11	0:02:04	0:03:35	0:05:15	0:11:51	0:04:47
	PROM(MAA)	0:00:50	0:02:26	0:04:28	0:06:27	0:09:29	

Tabla 4. Tiempo de cálculo por péptido

En esta tabla se describe el tiempo de cálculo medio de cálculo por péptido en formato H:MM:SS. Según la clasificación, el resultado sigue la lógica “menos es mejor”, indicando que un tiempo total de ejecución es más deseable cuanto más pequeño sea.

T. de cálculo por péptido p/ intentos							
		MAX_AA					
		10	20	30	40	50	PROM(MPEP)
MAX PEPT	1	0:00:30	0:01:29	0:02:09	0:03:34	0:03:03	0:02:09
	5	0:00:36	0:01:33	0:02:20	0:02:44	0:04:32	0:02:21
	10	0:00:33	0:01:33	0:01:58	0:03:43	0:04:01	0:02:22
	15	0:00:33	0:01:34	0:02:10	0:03:04	0:04:31	0:02:22
	20	0:00:30	0:01:32	0:02:00	0:03:18	0:03:55	0:02:15
	30	0:00:32	0:01:22	0:02:02	0:03:19	0:04:14	0:02:18
	50	0:00:31	0:01:17	0:01:23	0:03:20	0:04:32	0:02:12
	PROM(MAA)	0:00:32	0:01:28	0:02:00	0:03:17	0:04:07	

Tabla 5. Tiempo de cálculo por péptido ajustado por intentos

En esta tabla se describe el tiempo de cálculo medio de cálculo por péptido, ajustado por número de intentos, en formato H:MM:SS. Según la clasificación, el resultado sigue la lógica “menos es mejor”, indicando que un tiempo total de ejecución es más deseable cuanto más pequeño sea.

Pépt. Calidad / simulación							
		MAX_AA					
		10	20	30	40	50	PROM(MPEP)
MAX PEPT	1	0	1	0	1	1	0,60
	5	1	2	3	4	4	2,80
	10	1	7	9	6	9	6,40
	15	4	10	11	9	13	9,40
	20	8	8	17	14	14	12,20
	30	4	18	22	23	20	17,40
	50	16	30	21	39	36	28,40
	PROM(MAA)	4,86	10,86	11,86	13,71	13,86	

Tabla 5. Número de péptidos de calidad identificados por simulación

En esta tabla se describe el número absoluto de péptidos que cumplirían el criterio de calidad (índice mínimo de contactos proteína-péptido superior a 0.8). Según la clasificación, el resultado sigue la lógica “más es mejor”, indicando que un tiempo total de ejecución es más deseable cuanto más pequeño sea.

Clústeres por simulación							
		MAX_AA					
		10	20	30	40	50	PROM(MPEP)
MAX PEPT	1	1	1	1	1	1	1,00
	5	4	3	2	2	4	3,00
	10	3	4	4	3	3	3,40
	15	4	4	3	4	4	3,80
	20	4	4	4	4	5	4,20
	30	5	4	6	5	5	5,00
	50	8	6	5	5	4	5,60
	PROM(MAA)	4,14	3,71	3,57	3,43	3,71	

Tabla 7. Número de clústeres identificados por simulación

En esta tabla se describe el número de clústeres identificados por simulación. Según la clasificación, el resultado sigue la lógica “más es mejor”, indicando que un tiempo total de ejecución es más deseable cuanto más pequeño sea.

Pépt. Calidad / simulación (rel)							
		MAX_AA					
		10	20	30	40	50	PROM(MPEP)
MAX PEPT	1	0,00	1,00	0,00	1,00	1,00	0,60
	5	0,20	0,40	0,60	0,80	0,80	0,56
	10	0,10	0,70	0,90	0,60	0,90	0,64
	15	0,27	0,67	0,73	0,60	0,87	0,63
	20	0,40	0,40	0,85	0,70	0,70	0,61
	30	0,13	0,60	0,73	0,77	0,67	0,58
	50	0,32	0,60	0,42	0,78	0,72	0,57
	PROM(MAA)	0,20	0,62	0,61	0,75	0,81	

Tabla 6. Relación entre el número de péptidos de calidad identificados y el número total de péptidos simulados

En esta tabla se describe el número relativo de péptidos que cumplirían el criterio de calidad (índice mínimo de contactos proteína-péptido superior a 0.8) respecto al total de péptidos simulados. Según la clasificación, el resultado sigue la lógica “más es mejor”, indicando que un tiempo total de ejecución es más deseable cuanto más pequeño sea.