UNIVERSIDAD DE LA HABANA

Facultad de Matemática y Computación

PROYECTO FINAL
DISEÑO Y ANÁLISIS DE ALGORITMOS

Diseño Computacional de Proteínas

Autor: Lidier Robaina Caraballo

Grupo: C-411

20 de febrero de 2025

Índice general

Capítulo 1

Introducción

La bioinformática emerge como un campo revolucionario en la intersección entre la biología, la química y la ciencia de la computación, ofreciendo herramientas innovadoras para abordar desafíos científicos y tecnológicos de alto impacto. Las proteínas, como máquinas moleculares esenciales para la vida, desempeñan roles críticos en procesos biológicos, aplicaciones médicas (como el desarrollo de fármacos y terapias) y soluciones industriales (como biocatalizadores y materiales biodegradables). Sin embargo, el diseño tradicional de proteínas, basado en métodos experimentales de ensayo y error, resulta costoso, lento y limitado en complejidad. Aquí es donde los algoritmos computacionales se convierten en un aliado indispensable: permiten modelar, predecir y optimizar estructuras proteicas con precisión, acelerando el descubrimiento de soluciones biológicas personalizadas.

La **motivación** de este proyecto radica en dos pilares fundamentales. Primero, la necesidad de explorar estrategias algorítmicas eficientes para resolver problemas NP-duros asociados al diseño de proteínas, como el plegamiento tridimensional y la estabilidad termodinámica. Segundo, la oportunidad de contribuir al avance de áreas como la medicina personalizada, la bioingeniería y la sostenibilidad ambiental, donde proteínas diseñadas a la medida podrían transformar paradigmas actuales.

Los **objetivos** del proyecto se centran en:

1. Modelar el problema de diseño de proteínas desde una perspectiva algorítmica, identificando sus componentes críticos (espacio de búsqueda, funciones de energía, restricciones biológicas).
2. Analizar la complejidad del problema.
3. Implementar un algoritmo que resuelva el problema de forma exacta, evaluando su eficiencia (tiempo y espacio).

Capítulo 2

Definición del Problema DCP

2.1 Marco Teórico

Las proteínas son macromoléculas esenciales para la vida dado que están involucradas en casi todas las funciones estructurales, catalíticas, sensoriales y regulatorias de los organismos. Están formadas por una secuencia de compuestos orgánicos llamadas aminoácidos, y su función está determinada fundamentalmente por la estructura tridimensional que adopta la cadena de aminoácidos.

El objetivo del diseño de proteínas es encontrar, dentro de una colección de proteínas, la que más probabilidades tiene de ajustarse a una función. Puesto que existen veinte aminoácidos posibles para cada posición en una secuencia proteica, y cada uno de ellos admite diversas variantes estructurales, la cantidad de combinaciones factibles a evaluar supera las posibilidades de cualquier método experimental, incluso en el caso de secuencias muy cortas.

2.1.1. Esqueleto

Como ilustra la figura 2.1, los aminoácidos están formado por un núcleo común (átomos de H, C, N y O) y una cadena lateral (R) que es específica para cada uno de los 20. En una proteína, los núcleos de los aminoácidos están enlazados en una secuencia que forma el *esqueleto* de la proteína.

El esqueleto es relativamente rígido y define la estructura tridimensional de la proteína (figura 2.2), por tanto se asume que la proteína resultante del diseño conservará el plegamiento global de la estructura base elegida. Es decir, se considera que el esqueleto de la proteína está fijo y es entrada del problema del diseño de proteínas.

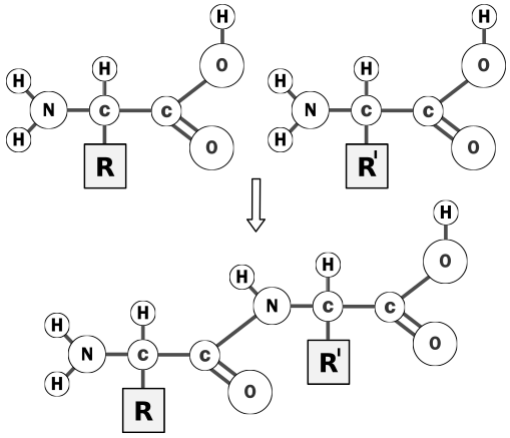

Figura 2.1: Representación de cómo dos aminoácidos, con cadenas laterales específicas R y R', enlazan sus núcleos para formar una cadena

(a) Modelo con cadenas laterales

(b) Esqueleto

Figura 2.2: Modelo 3D de una proteína

2.1.2. Rotámeros

Debido a la rotación alrededor de los enlaces simples, las cadenas laterales de los aminoácidos pueden adoptar distintas conformaciones tridimensionales, que reciben el nombre de *rotámeros*. Hay una cantidad infinita de rotámeros posibles, puesto que las moléculas pueden girar alrededor del eje de forma continua. No obstante, suele ser suficiente considerar un conjunto discreto y representativo de rotámeros (figura 2.3), el cual se puede determinar a través de un análisis estadístico de las conformaciones reales presentes en las bases de datos de estructuras de proteínas.

En el problema del diseño de proteínas, los rotámeros constituyen el espacio de búsqueda primario cuando el esqueleto está fijo.

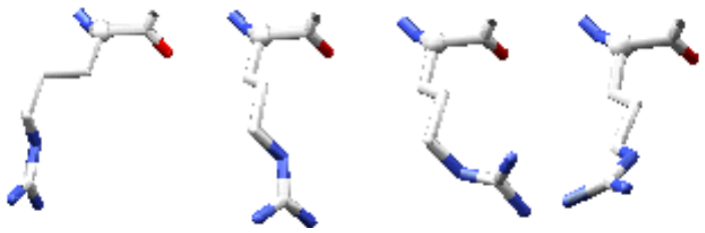

Figura 2.3: Rotámeros del aminoácido arginina: algunas posibles geometrías de la cadena lateral

2.1.3. Funciones de energía

La estabilidad y la eficiencia funcional de una proteína están correlacionadas con su energía, por lo tanto, el objetivo es encontrar la conformación que posea la energía total mínima.

La energía total de la proteína está dada por la energía del esqueleto, la energía de interacción entre los rotámeros y el esqueleto, y la energía de interacción entre los diferentes rotámeros. Cuando el esqueleto está fijo, la minimización de la energía depende únicamente de la energía de interacción entre los rotámeros y entre estos y el esqueleto. Sean i_r el rotámero en la posición i, $E(i_r)$ la energía de interacción entre el rotámero i y el esqueleto, y $E(i_r, j_{r'})$ la energía de interacción entre los rotámeros r en la posición i y r' en la posición j, la fórmula a minimizar se convierte en

$$E = \sum_i E(i_r) + \sum_i \sum_{j<i} E(i_r, j_{r'})$$

2.2 Modelación como problema de teoría de grafos

Una proteína con k aminoácidos puede ser representada por un grafo k-partito $G = (V, E)$, de forma tal que:

- hay una partición V_i por cada aminoácido i
- hay un vértice $v \in V_i$ por cada rotámero candidato del aminoácido i
- la arista $\langle u, v \rangle$ denota la interacción entre los rotámeros u y v
- el vértice v tiene un costo c_v que es igual a la energía de interacción entre el rotámero v y el esqueleto
- la arista $\langle u, v \rangle$ tiene un costo $c_{\langle u,v \rangle}$ que es igual a la energía de interacción entre el rotámero u y el rotámero v

Cada rotámero puede potencialmente interactuar con cualquier rotámero de cualquier otra posición distinta a la suya, por por tanto G es un grafo k-partito completo. Luego, seleccionar un rotámero para cada aminoácido es equivalente a seleccionar un clique de tamaño máximo (necesariamente tamaño k) (figura 2.4).

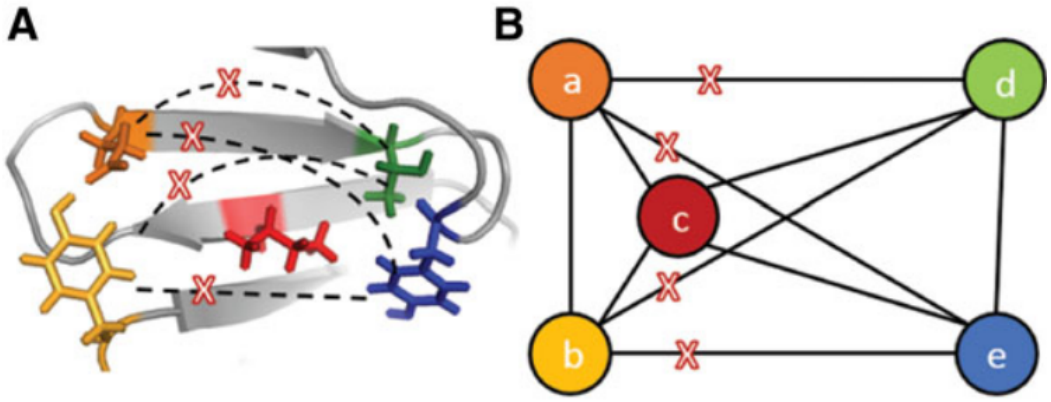

Figura 2.4: **(A)** Ejemplo de diseño de una proteína, cada color corresponde a una partición del grafo **(B)** Ejemplo de clique solución del problema A

El problema del **Diseño Computacional de Proteínas (DCP)** se define como:

> Dado un grafo k-partito completo $G = (V, E)$, $V = V_1 \cup V_2 \cup ... \cup V_k$, ponderado en vértices y aristas, hallar el clique C de tamaño k tal que el grafo inducido por C es de costo mínimo.

Lo anterior es equivalente a seleccionar un $v_i \in V_i \ \forall \ 1 \leq i \leq k$, tal que

$$E_{total} = \sum_i c_{v_i} + \sum_i \sum_{j<i} c_{\langle v_i, v_j \rangle}$$

sea mínima.

Capítulo 3

Análisis de complejidad

3.1 DCP $\in$ NP-Hard

3.1.1. Definiciones

- **Clique(G,k)**: Dado un grafo G y un entero k, determinar si en G existe un clique de tamaño k.
- **DCP_E(G,E)**: Dado un grafo n-partito completo G y una constante E, determinar si G tiene un clique de tamaño n y costo $E_{total} \leq E$.

3.1.2. Reducción Clique $\propto$ DCP_E

Sea $(G = (V, E), k)$ una instancia de Clique, $n = |V|$, se construye un grafo n-partito completo G' como sigue:

1. Particiones de G'
   - Cada vértice $v_i \in V$ corresponde a una partición V_i en G'.
   - Cada partición P_i tiene dos vértices:
     - a_i (representa incluir v_i en el clique), con peso 0.
     - b_i (representa excluir v_i), con peso 1.
2. Aristas en G'
   - Para cada par de particiones V_i, V_j $(i \neq j)$:
     - Si $\langle v_i, v_j \rangle \in E$, la arista $\langle a_i, a_j \rangle$ tiene peso 0.
     - Si $\langle v_i, v_j \rangle \notin E$, la arista $\langle a_i, a_j \rangle$ tiene peso 1.
   - Todas las aristas que involucran algún b_i tienen peso 0.

3.1.3. Equivalencia de soluciones

Teorema. *G tiene un clique de tamaño k si y solo si G' tiene un clique de tamaño n y costo $E_{total} \leq 0$.*

Demostración

($\Rightarrow$)

Seleccionando los vértices a_i correspondientes a los k nodos del clique en G, el conjunto formado por esos vértices es un clique de tamaño n y cumple que:

- Los vértices a_i tienen peso 0.
- Las aristas entre ellos tienen peso 0 (porque forman un clique en G).

Por tanto, tiene costo $E_{total} = 0 \leq 0$.

($\Leftarrow$)
Seleccionando el clique en cuestión:

- Todos los vértices del clique deben ser a_i (ya que cualquier b_i añadiría peso).
- Las aristas entre estos a_i deben tener peso 0, lo que implica que $\langle v_i, v_j \rangle \in E$ para todo par.

Esto corresponde a un clique de tamaño k en G. □
Dado que Clique es un conocido problema NP-Completo, con la reducción anterior se ha demostrado que DCP_E es un problema NP-Hard. También es fácil comprobar que DCP_E es NP, por tanto, es NP-Completo. Luego, como DCP es un problema de optimización correspondiente a un problema de decisión NP-Completo, $DCP \in NP - Hard$

3.2 DCP no es aproximable

Supondremos la existencia de un algoritmo de aproximación con factor constante $\alpha \geq 1$ y mostraremos que esto permitiría resolver 3-SAT en tiempo polinomial, contradiciendo la NP-dureza del problema.

3.2.1. Reducción 3-SAT $\propto$ DCP

Para cualquier $\alpha \geq 1$, construimos un grafo m-partito completo G desde una fórmula 3-CNF ϕ con m cláusulas:

- **Particiones**: m particiones (uno por cláusula)
- **Nodos**: 3 por partición (literales de la cláusula)

- **Pesos en nodos**: 1 para todos los nodos
- **Pesos en aristas**:

$$w(u,v) = \begin{cases} 0, & \text{si } u \text{ y } v \text{ son consistentes} \\ W, & \text{con } W = \lceil(\alpha-1)m\rceil + 1 \quad \text{(para literales inconsistentes)} \end{cases}$$

3.2.2. Análisis de Costos

- Caso 1: ϕ es satisfacible
  - **Costo nodos**: $m \times 1 = m$
  - **Costo aristas**: 0 (sin inconsistencias)
  - **OPT** $= m$
- Caso 2: ϕ no es satisfacible
  - **Costo nodos**: $m \times 1 = m$
  - **Costo aristas**: $\geq W$ (al menos una inconsistencia)
  - **OPT** $\geq m + W = m + \lceil(\alpha-1)m\rceil + 1 \geq \alpha m + 1$

3.2.3. Ratio de Aproximación

Para cualquier algoritmo $\mathcal{A}$ con factor α:

$$\frac{\text{Costo}(\mathcal{A}(G))}{\text{OPT}} \leq \alpha$$

Esto implica:

$$\begin{cases} \mathcal{A}(G) \leq \alpha m & \text{si } \phi \text{ es satisfacible} \\ \mathcal{A}(G) \geq \alpha m + 1 & \text{si } \phi \text{ no es satisfacible} \end{cases}$$

3.2.4. Contradicción

Un algoritmo α-aproximado distinguiría entre los dos casos en tiempo polinomial:

1. Si $\mathcal{A}(G) \leq \alpha m \Rightarrow \phi$ es satisfacible
2. Si $\mathcal{A}(G) \geq \alpha m + 1 \Rightarrow \phi$ no es satisfacible

Esto resolvería 3-SAT en tiempo polinomial, por tanto DCP no puede ser aproximado dentro de ningún factor constante salvo que P = NP.

Capítulo 4

Solución

4.1 Branch and Bound (B&B)

El algoritmo B&B construye progresivamente el clique seleccionando un residuo por partición. En cada paso:

- **Ramificación**: Genera subproblemas al elegir diferentes nodos de la partición actual
- **Acotamiento**: Calcula una cota inferior (LB) del costo mínimo posible para completar el clique
- **Poda**: Si LB $\geq$ mejor solución actual, abandona esa rama

4.1.1. Pseudocódigo

Algorithm 1 Branch and Bound para DCP

```
Entrada: Grafo k-partito completo con costos en nodos y aristas
Salida: Clique de costo mínimo
Inicializar mejor_costo ← ∞
procedure BranchAndBound(S, costo, P)
    if P = ∅ then
        if costo < mejor_costo then
            Actualizar solución óptima
        end ifreturn
    end if
    Seleccionar partición p ∈ P
    for cada nodo v ∈ p do
        nuevo_costo ← costo + C(v) + Σ_{u∈S} C(u, v)
        lb ← CotaInferior(S ∪ {v}, P \ {p})
        if nuevo_costo + lb < mejor_costo then
            Llamar recursivamente BranchAndBound
        end if
    end for
end procedure
```

4.1.2. Correctitud

- **Exhaustividad**: Explora todas posibles selecciones de nodos mediante recursión
- **Poda segura**: La cota inferior nunca subestima el costo real (admisible)
- **Optimalidad**: Mantiene registro del mejor clique encontrado durante la exploración

4.1.3. Complejidad

$$T(n,k) = \underbrace{O(n)}_{\text{1er nivel}} \times \underbrace{O(n)}_{\text{2do nivel}} \times \cdots \times \underbrace{O(n)}_{k\text{-ésimo nivel}} = O(n^k)$$

La poda reduce la constante pero no el orden polinómic.

4.2 Prepodado

Técnica de reducción previa basada en dominancia estricta entre residuos:

Teorema. *Un residuo i puede eliminarse si $\exists j$ en su misma partición tal que:*

$$C(i) + \sum_{q \neq p} \min_{r \in q} C(i,r) > C(j) + \sum_{q \neq p} \min_{r \in q} C(j,r)$$

Supongamos que i está en la solución óptima. Reemplazando i por j, el nuevo costo sería:

$$C(j) + \sum_{q \neq p} C(j, r^*) \leq C(j) + \sum_{q \neq p} \min C(j, r) < C(i) + \sum \min C(i, r)$$

Contradice la optimalidad de i. Luego, i no puede ser parte de la solución óptima.

4.2.1. Pseudocódigo

Algorithm 2 Dead End Elimination

```
for cada partición p ∈ {1, ..., k} do
    for cada nodo i ∈ p do
        for cada nodo j ∈ p \ {i} do
            Calcular Δ = C(i) − C(j)
            for cada partición q ≠ p do
                Δ ← Δ + mín_{r∈q}[C(i, r) − C(j, r)]
            end for
            if Δ > 0 then
                Eliminar nodo i de p
                break
            end if
        end for
    end for
end for
```

4.2.2. Complejidad

$$\begin{aligned} \text{Particiones} &: O(k) \\ \text{Residuos por partición} &: O(n^2) \text{ (todas parejas } i, j) \\ \text{Particiones restantes} &: O(k) \\ \text{Residuos en } q &: O(n) \\ \Rightarrow \text{Total} &: O(k^2 n^3) \end{aligned}$$

4.3 Cotas Inferiores: Greedy vs Programación Dinámica

4.3.1. Cota Greedy

Estrategia que considera mínimos locales:

$$lb_{greedy} = \sum_{q \in rest} \left[\min_{v \in q} \left(C(v) + \sum_{u \in S} C(u, v) \right) \right]$$

Límites:

- Subestima costos entre particiones no seleccionadas
- No considera interdependencias futuras
- Fácil de calcular pero poco ajustada

4.3.2. Cota con Programación Dinámica

Precomputa costos mínimos desde cada partición:

$$dp[p][v] = C(v) + \min_{q > p} \left(\min_{u \in q} [C(v, u) + dp[q][u]] \right)$$

$$lb_{dp} = \min_{v \in p_1} \left(dp[p_1][v] + \sum_{u \in S} C(u, v) \right)$$

Ventajas Clave:

- Captura costos acumulativos óptimos hacia el futuro
- Considera relaciones entre todas las particiones restantes
- Requiere preprocesamiento pero ofrece cotas más precisas

4.3.3. Análisis Experimental Teórico

Método	Tiempo Preproc.	Tiempo por Cota	Memoria	Efectividad Poda
Greedy	0	$O(k^2n^2)$	$O(1)$	Baja
DP	$O(k^2n^2)$	$O(n)$	$O(kn)$	Alta

4.4 Conclusión

- DEE reduce efectivamente el espacio de búsqueda
- B&B con DP logra mejores podas pero requiere más memoria
- La combinación DEE + B&B + DP es óptima para instancias medianas