UNIVERSITÀ
DI PARMA

Dipartimento di Scienze Matematiche, Fisiche e Informatiche

Corso di Laurea Magistrale in Scienze Informatiche

Studio di mobilità su larga scala della proteina Spike del covid-19 con ricerca locale

Long range mobility of covid-19 Spike protein through local search

Candidato:
Lorenzo Mora

Relatore:
Prof. Alessandro Dal Palù

Correlatori:
Prof. Pietro Cozzini
Dott.sa Federica Agosta

Anno Accademico 2021–2022

Dedica

Indice

Elenco delle figure

Elenco degli algoritmi

Elenco delle tabelle

Introduzione

La glicoproteina spike del SARS-CoV-2 è una proteina di superficie virale che svolge un ruolo cruciale nell'entrata del virus nelle cellule ospiti. Essa è costituita da tre domini principali: il dominio N-terminale, il dominio centrale e il dominio C-terminale. Il dominio N-terminale della proteina spike si lega ai recettori delle cellule ospiti, in particolare l'enzima di conversione dell'angiotensina 2 (ACE2), che si trova sulla superficie delle cellule del tratto respiratorio. La proteina spike è una delle principali destinazioni dei vaccini COVID-19, poiché è il bersaglio principale del sistema immunitario nell'identificare e combattere il virus.

L'obbiettivo principale è quello di studiare il comportamento della glicoproteina attraverso tecniche di ricerca locale, che tengano maggiormente in considerazione la stabilità della proteina stessa evitando di generare clash. Per l'appunto uno degli obbiettivi di questo framework è quello di preservare la catena laterale degli amminoacidi in qualsiasi movimento. Non vengono però applicate delle rotazioni alla catena laterale dell'amminoacido che rimane fissa, ma si agisce sulla backbone (catena principale), cercando di incastrare correttamente tutto. L'altro obiettivo è quello di trovare un cammino minimo tra le due conformazioni aperta e chiusa, in modo da "simulare" il processo naturale di evoluzione delle conformazioni. Il framework si pone quindi al di sotto di un sistema di dinamica molecolare, ma che consente una panoramica di studio del comportamento della proteina.

Dal titolo della tesi si evince che il framework è basato sulla ricerca locale, che è una famiglia di algoritmi di ottimizzazione che cercano di migliorare iterativamente una soluzione corrente mediante la modifica di componenti o variabili della soluzione stessa. All'interno di questo sistema viene utilizzata in due ambiti: il primo è quello di ricerca di convergenza tra le due conformazioni; mentre il secondo è utilizzate sempre in ambito di convergenza ma questa volta tra il loop e la nuova conformazione della parte mobile. Nel primo caso si tratta di un algoritmo di shortest-path che utilizza la ricerca locale per migliorare la soluzione corrente.

SPIEGARE L'IMPORTANZA DELLA MIA RICERCA

La tesi è organizzata nel seguente modo:

- nella Sezione 1 si introducono quelle che sono le nozioni preliminari per contestualizzare il lavoro svolto;
- nella Sezione 2 si introduce il covid-19 e in modo dettagliato il caso di studio della glicoproteina Spike;
- nella Sezione 3 si introducono gli obbiettivi dell'elaborato;
- nella Sezione 4 si mostra lo pseudo-codice che guida il framework;
- nella Sezione 5 si mostrano i risultati conseguiti;
- nella Conclusione oltre a riassumere brevemente il lavoro svolto, vengono evidenziati alcuni possibili sviluppi futuri.

INTRODUCI I DIFETTI DEL SISTEMA
INTRODUCI LE CONCLUSIONI

Capitolo 1

Background

1.1 Proteine

Le proteine sono considerate il più importante strumento molecolare con le quali è possibile esprimere l'informazione genetica.

Come riporta [David L. Nelson, 2014], le proteine di qualsiasi organismo, dai batterei più semplici agli essere umani, sono formate sempre dalla stessa serie di venti amminoacidi. Poiché ogni amminoacido ha una catena laterale con specifiche proprietà chimiche, questo gruppo di 20 precursori può essere considerato come una forma di alfabeto con cui è scritto il linguaggio della struttura delle proteine.

Gli amminoacidi, in modo retorico, possono essere considerati dei mattoncini che costruiscono le proteine e giocando con questi mattoncini possiamo costruire diverse tipologie di proteine. Come riporta [David L. Nelson, 2014], le proteine sono polimeri di amminoacidi in cui ogni residuo amminoacidico è unito a quello vicino da uno specifico legame covalente. Il legame covalente in questione è il legame peptidico o giunto peptidico che unisce il gruppo (-NH2) di un amminoacido con il gruppo (-COOH) di un altro amminoacido. Negli organismi viventi le proteine svolgono innumerevoli funzioni, tra cui la catalisi delle reazioni metaboliche, funzione di sintesi come replicazione del DNA, la risposta a stimoli e il trasporto di molecole da un luogo ad un altro. Le proteine in generale si differiscono nella sequenza degli amminoacidi, che viene conservata nei geni e che si traduce in un particolare ripiegamento della stessa e una struttura tridimensionale specifica che caratterizza la sua attività.

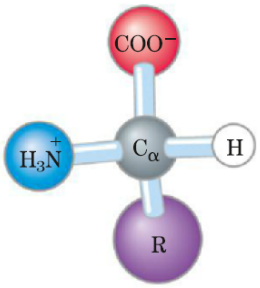


Figura 1.1: La struttura è comune a tutti gli amminoacidi. Il gruppo R è legato al carbonio α ed è diverso per ogni amminoacido. Solo nel caso della glicina il gruppo R è un atomo di idrogeno

1.1.1 Amminoacidi

Tutti gli amminoacidi presenti nelle proteine sono α-amminoacidi. Essi sono composti da un gruppo carbosillico e un gruppo amminico legati allo stesso atomo di carbonio, denominato carbonio α, e si distinguono l'uno dall'altro mediante il gruppo R, che si differenzia non solo per la struttura, ma anche per proprietà chimico-fisiche, Fig.1.1.

Nei comuni amminoacidi, il carbonio α è legato a quattro gruppi differenti: il gruppo carbosillico; il gruppo amminico; il gruppo R; un'atomo di idrogeno. Il carbonio α è detto centro chirale, ovvero ha quattro sostituenti diversi ed è asimmetrico.

Classificazione degli amminoacidi

Gli amminoacidi possono essere classificati sulla base della caratteristiche chimico-fisiche del gruppo R legato al carbonio α:

- Gruppi R alifatici, non polari: in questa classe gli amminoacidi non sono polari e quindi idrofobici; le catene laterali tendono a interagire all'interno della proteina tramite interazioni idrofobiche;

- Gruppi R aromatici: in questa classe gli amminoacidi sono tutti non polari (idrofobici) e tutti possono intervenire nelle interazioni idrofobiche;

- Gruppi R polari, non carichi: in questa classe gli amminoacidi sono più solubili in acqua (o più idrofilici), rispetto ai non polari, perché contengono gruppi funzionali che creano legami idrogeno;

- Gruppi R carichi positivamente: in questa classe gli amminoacidi, sono quelli più idrofilici poichè contengono una carica netta positiva;

- Gruppi R carichi negativamente: in questa classe gli amminoacidi, sono quelli più idrofilici poichè contengono una carica netta negativa e sono caratterizzati dall'avere un secondo gruppo carbosillico.

Amminoacido	**Simbolo**	$\mathbf{M_r}$	$\mathbf{pK_1}$	$\mathbf{pK_2}$	$\mathbf{pK_r}$	**pI**	**Indice idropatico**	**Presenza nelle proteine %**
Gruppi R alifatici, non polari								
Glicina	Gly	75	2.34	9.60		5.97	-0.4	7.2
Alanina	Ala	89	2.34	9.69		6.01	1.8	7.8
Prolina	Pro	115	1.99	10.96		6.48	-1.6	5.2
Valina	Val	117	2.32	9.62		5.97	4.2	6.6
Leucina	Leu	131	2.36	9.60		5.98	3.8	9.1
Isoleucina	Ile	131	2.36	9.68		6.02	4.5	5.3
Metionina	Met	149	2.28	9.21		5.74	1.9	2.3
Gruppi R aromatici								
Fenilalanina	Phe	165	1.83	9.13		5.48	2.8	3.9
Tirosina	Tyr	181	2.20	9.11	10.07	5.66	-1.3	3.2
Triptofano	Trp	204	2.38	9.39		5.89	-0.9	1.4
Gruppi R polari, non carichi								
Serina	Ser	105	2.21	9.15		5.68	-0.8	6.8
Treonina	Thr	119	2.11	9.62		5.87	-0.7	5.9
Cisteina	Cys	121	1.96	10.28	8.18	5.07	2.5	1.9
Asparagina	Asn	132	2.02	8.80		5.41	-3.5	4.3
Glutammina	Gln	146	2.17	9.13		5.65	-3.5	4.2
Gruppi R carichi positivamente								
Lisina	Lys	146	2.18	8.95	10.53	9.74	-3.9	5.9
Istidina	His	155	1.82	9.17	6.00	7.59	-3.2	2.3
Arginina	Arg	174	2.17	9.04	12.48	10.76	-4.5	5.1
Gruppi R carichi negativamente								
Aspartato	Asp	133	1.88	9.60	3.65	2.77	-3.5	5.3
Glutammato	Glu	147	2.19	9.67	4.25	3.22	-3.5	6.3

Tabella 1.1: Proprietà e nomenclatura degli amminoacidi comuni presenti nelle proteine.

Indice idropatico è un valore che combina idrofobicità e idrofilicità dei gruppi R e si riferisce all'energia libera di trasferimento (δG) della catena laterale dell'amminoacido da un solvente idrofobico all'acqua. Il trasferimento è favorito ($\delta G < 0$, valore dell'indice negativo) per le catene laterali degli amminoacidi carichi o polari; ed è sfavorito ($\delta G > 0$, valore dell'indice positivo) per le catene laterali degli amminoacidi non polari o idrofobici.

Presenza nelle proteine % rappresenta la presenza media in più di 1150 proteine.

1.1.2 Polimeri degli amminoacidi

Le proteine differiscono nel numero di amminoacidi che li compongono. Due molecole di amminoacidi come detto in precedenza possono unirsi covalentemente mediante un legame ammidico, chiamato appunto legame peptidico, formando un dipeptide. Come dice [David L. Nelson, 2014], questo tipo di legame si genera per eliminazione di una molecola d'acqua dal gruppo α-carbosillico di un amminoacido e dal gruppo α-amminico dell'altro (Fig. 1.2).

Oligopeptidi e Polipeptidi differiscono per il numero di amminoacidi costituenti. I polipeptidi o proteine sono delle macromolecole a struttura complessa costituite da un numero di unità monometriche convenzionalmente superiore a 50. Alcune proteine sono costituite da una singola catena polipetidica, mentre altre hanno più polipetidi associati in modo non covalente e sono chiamate proteine multisubunità. All'interno delle proteine multisubunità, se almeno due dei polipeptidi presenti sono uguali, allora la proteina viene detta oligomerica, e le subunità identiche sono dette protomeri. Come riporta [David L. Nelson, 2014], la composizione amminoacidica delle proteine è molto variabile. Infatti i 20 amminoacidi comuni non sono quasi mai distribuiti in egual proporzione nella proteina.

Una volta espresse, alcune proteine possono subire delle modifiche chimico-strutturali essenziali per la loro funzione. Tra queste ricordiamo la glicosilazione, una modifica post-tradizionale tale per cui alcuni residui aminoacidici sono non-covalentemente legati a zuccheri, molecole formate da un anello a cinque o sei atomi di carbonio legati a gruppi ossidrilici

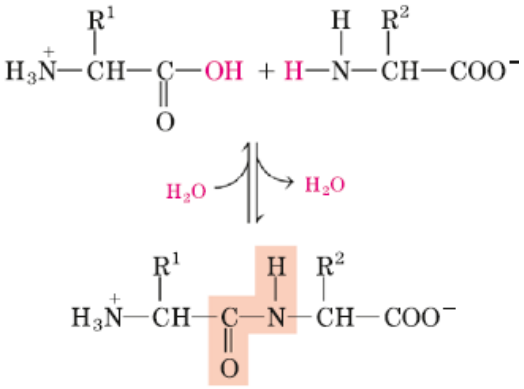


Figura 1.2: Formazione di un legame peptidico per condensazione, li gruppo α-amminico di un amminoacido (con ilgruppo R2) agisce da nucleofilo e reagisce con Il gruppo ossidrilico del carbossile di un altro amminoacido (con Il gruppo R1), formando un legame peptidico (ombreggiato in rosa).

1.1.3 Struttura delle proteine

Le caratteristiche strutturali conferiscono alla proteina stessa la sua funzione, ma la struttura di una proteina di grandi dimensioni è complessa e può essere descritta secondo vari livelli di complessità. In generale vengono descritti quattro livelli di struttura delle proteine, come si può vedere nella Fig. 1.3.

La struttura primaria è costituita da tutti i legami covalenti che legano tra loro i vari amminoacidi della catena polipeptidica. La struttura secondaria contiene organizzazioni particolari e stabili di brevi sequenze di amminoacidi, che danno il via a profili strutturali ricorrenti. La struttura terziaria descrive l'aspetto tridimensionale del polipeptidico, mentre se sono presenti più unità allora è presente anche la struttura quaternaria. In particolar modo la struttura primaria di una proteina determina la disposizione degli atomi nello spazio tridimensionale che ne caratterizzano la funzione.

Come riporta [David L. Nelson, 2014], la disposizione spaziale degli atomi di una proteina, o di una porzione di proteina è detta conformazione. Le conformazioni possibili di una proteina, o di una parte di essa, corrispondono a tutte le strutture che la proteina può assumere senza rottura di legami covalenti. I legami covalenti perciò hanno un ruolo importante nel determinare la conformazione di un polipeptide. Gli atomi di carbonio α di residui amminoacidici adiacenti sono separati da tre legami covalenti che si susseguono in questo modo: C_α—C—N—C_α. Facendo analisi su questi atomi del gruppo peptidico, si evince che i legami C—N, a causa del loro parziale carattere di doppio legame non può ruotare liberamente; è invece concessa la rotazione tra i legami N—C_α e C_α—C. Lo scheletro della catena polipeptidica può

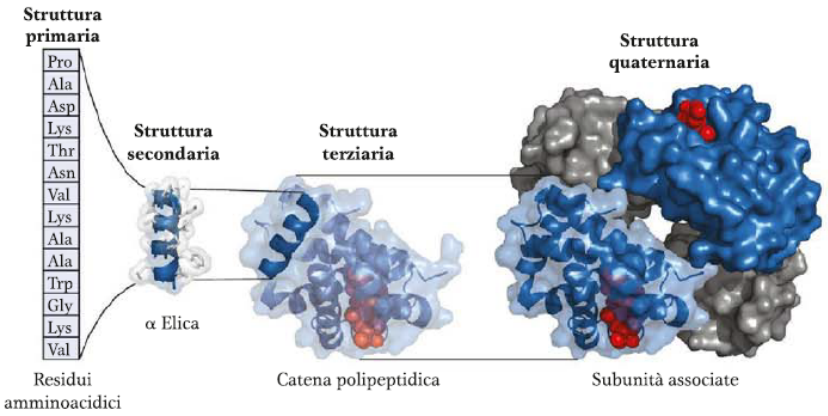


Figura 1.3: Livelli di struttura delle proteine. La struttura primaria è una sequenza di amminoacidi unita da legame peptidico. I polipeptidi risultanti possono disporsi in una struttura regolare ricorrente che viene definita struttura secondaria, come l'α elica. Essa costituisce una delle parti della struttura terziaria, che può essere a sua volta una delle subunità della struttura quaternaria.

quindi essere considerato una serie di piani rigidi, in cui i piani consecutivi hanno come punto di rotazione in comune C_α (Fig. 1.4). La rigidità del legame limita considerevolmente la possibilità di movimento e di conseguenza il numero delle conformazioni che possono essere assunte. La conformazione è poi specificata dai tre angoli diedrici (o angoli torsionali) chiamati ϕ (phi), ψ (psi) e ω (omega).

La struttura secondaria si riferisce ad un segmento polipeptidico della proteina e descrive l'organizzazione spaziale della catena principale senza considera le catene laterali. Le strutture secondarie regolari più comuni sono l'α elica, la conformazione β foglietto e il ripiegamento β. La struttura terziaria corrisponde alla struttura tridimensionale completa di una catena polipeptidica e in base ad essa si possono distinguere due classi di proteine: le fibrose e le globulari.La struttura quaternaria deriva dalle interazioni tra le subunità che costituiscono una proteina multimerica.

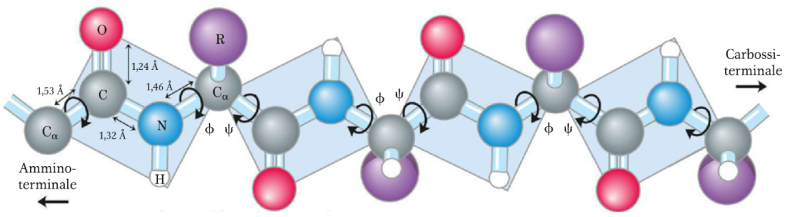


Figura 1.4: Tre legami separano due atomi di carbonio α in successione in una catena polipeptidica. I legaml N—C_α e C_α—C possono ruotare, descrivendo due angoli diedrici chiamati rispettivamente ϕ e ψ. Il legame C—N non è libero di ruotare. La rotazione degll altri legami singoli dello scheletro peptidico è resa difficoltosa dalle dimensioni e dalle cariche del gruppi R.

1.1.4 Classificazione delle proteine

Le proteine hanno molteplici funzioni, sono coinvolte nel movimento, nella trasmissione dei segnali (ormoni), proteggono l'organismo (anticorpi), mantengono le sostanze, le trasportano, fungono da enzima, hanno funzione recettoriale e anche strutturale.E' possibile classificare le proteine sulla base della loro funzione come segue:

- proteine di trasporto: facilitano il trasporto passivo (per diffusione) o attivo (attraverso legame chimico e consumo di energia) di sostanze attraverso le membrane biologiche e nei vari tessuti. Ne è un esempio l'emoglobina, che trasporta ossigeno dai polmoni ai tessuti, e la mioglobina che permette di veicolare l'ossigeno ai mitocondri presenti nelle cellule del tessuto muscolare;

- proteine implicate nella segnalazione cellulare: permettono la trasmissione di segnali chimici o elettrici essenziali per la vita cellulare. Appartengono a questa categoria anche gli anticorpi prodotti dai linfociti B attivati in risposta agli antigeni, come batteri, virus e altre sostanze riconosciute come microorganismi e agenti estranei pericolosi;

- enzimi: gli enzimi sono le proteine responsabili della catalisi delle reazioni chimiche all'interno del nostro organismo. Di fatto accelerano una reazione chimica specifica senza essere consumati e senza entrare nei prodotti finali della reazione;

- proteine strutturali: tipicamente formano delle strutture di sostegno, si tratta infatti di molecole caratterizzate da durezza e resistenza, un

esempio può essere l'α-cheratina che forma le strutture interne dell'organismo.

1.1.5 Principio di Ramachandran

Il principio di Ramachandran ha stabilito che solo alcune conformazioni di angoli di torsione sono stabili e presenti naturalmente nelle proteine. Queste conformazioni stabili sono descritte come regioni favorevoli o regioni accettabili nel grafico di Ramachandran. Le conformazioni non favorevoli o non accettabili sono associate con strutture instabili o anomale.

Il principio di Ramachandran è spesso molto utile per saggiare la qualità e l'attendibilità delle strutture tridimensionali delle proteine.

Le conformazioni dei peptidi sono definite come descritto in precedenza da ϕ e ψ, che possono assumere qualsiasi valore compreso tra -180 e $+180$. Non tutte le coppie di angoli torsionali teoricamente possibili sono realmente ottenibili all'interno delle strutture peptidiche in quanto impedite dalla presenza di ingombri sterici fra i gruppi delle catene laterali, basati su calcoli in cui sono utilizzati il raggio di van der Walls e gli angoli di legame. Come si può vedere nella Fig. 1.5, le aree colorate in blu scuro, contengono le conformazioni che non presentano sovrapposizioni steriche, ovvero quando ogni atomo può essere mostrato come una sfera solida, di conseguenza sono zone completamente consentite. Le zone evidenziate con un colore intermedio sono zone in cui è presente un piccolo scontro; quindi, se viene concesso uno scontro di 0.1 nm sono conformazioni concesse. Le zone evidenziate in azzurro sono consentito a patto che sia consentita una leggera flessibilità dell'angolo ω. Le regioni non evidenziate sono zone non consentite. Si può notare la poca simmetria presente nel grafico, a causa della stereochimica dei residui amminoacidici.

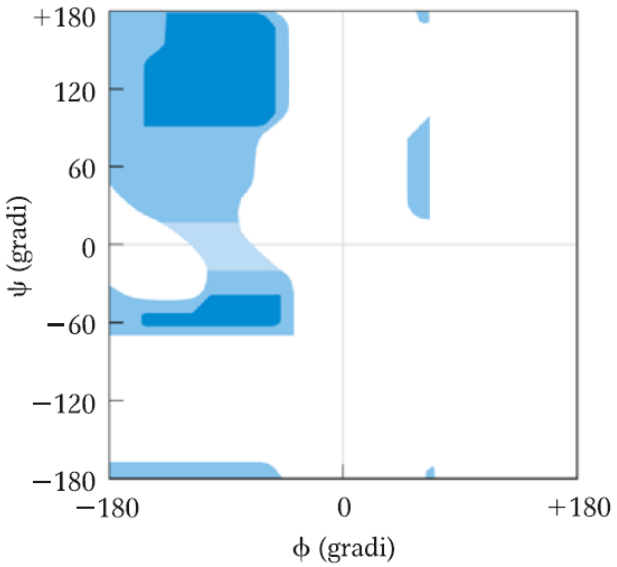


Figura 1.5: Esempio grafico di Ramachandran

1.2 Hydropathic INTeractions HINT

Come riportato da [Federica Agosta,], la valutazione della stabilità intramolecolare di una proteina gioca un ruolo fondamentale nella comprensione del comportamento e del meccanismo di azione. Piccole alterazioni strutturali possono impattare l'attività biologica e di conseguenza la modulazione farmacologica. L'analisi della struttura tridimensionale della proteina e la risultante stabilità permettono di predire un possibile meccanismo di attivazione e rivelano nuove strategie per scoprire nuovi farmaci. Ogni modifica apportata alla struttura di una proteina porta quasi sicuramente ad una variazione del suo meccanismo di azione, e valutare la stabilità di una proteina può essere molto costoso in termini di tempo. La stabilità delle proteine è influenzata dalle interazioni intramolecolare, ovvero sono forze che permettono di creare e tenere insieme una struttura. Le strutture native delle proteine sono stabili poiché si formano come risultato di un equilibrio tra le varie forze non covalenti a cui sono soggette: i legami idrogeno, i legami ionici, le forze di Wan der Walls e le interazioni idrofobiche. In generale le interazioni chimiche deboli sono attrazzioni tra atomi appartenenti o alla stessa molecola (intramolecolari) o a molecole diverse (intermolecolari). Le interazioni chimiche deboli si formano e si rompono in continuazione alla temperatura fisiologica dell'organismo. a meno che, cumulandosi in gran numero, esse non diano collettivamente stabilità alle strutture che contribuiscono a generare.

La valutazione della stabilità intramolecolare di una proteina è particolarmente complessa e ha portato allo sviluppo e ottimizzazione di metodi computazionali differenti che vanno dalla meccanica classica ai più recenti algoritmi di machine Learning

HINT (Hydrophatic INTeractions) è un programma designato per quantificare le interazioni inter e intramolecolari, come citato in [Glen E. Kellogg, 1991] e in [Eugene Kellogg and Abraham, 2000]. In breve, sulla base di valori di LogPo/W (coefficiente di ripartizione ottanolo/acqua di una sostanza, indice della sua polarità/idrofobicità) calcolati per tutti gli amminoacidi del sistema è possibile classificare e quantificare le interazioni calcolati come il prodotto tra la superficie accessibile al solvente e la costante idrofobica di ogni atomo e la loro distanza, come descritto in [Abraham and Leo, 1987]. La funzione di energia intramolecolare di HINT può essere calcolata rapidamente dalla struttura in termini di somma di punteggi d'interazione atomo-atomo. Il punteggio intramolecolare viene calcolato come somma delle interazioni idrofobiche e polari tra tutte le coppie di atomi considerando l'area accessibile al solvente. Le interazioni idrofobiche si verificano quando le molecole non polari si avvicinano tra di loro in un solvente polare, come l'acqua. Le molecole non polari tendono a respingersi reciprocamente e ad attirarsi con le molecole del solvente. Il processo anche noto come esclusione dei solventi, porta alla formazioni di molecole non polari all'interno del solvente. Le molecole non polari quindi cercano di organizzarsi in modo tale da minimizzare il contatto con il solvente e massimizzare le interazioni tra di loro.

Il punteggio energetico di HINT intramolecolare consente quindi di calcolare tutte le interazioni idropatiche (idrofobiche e polari) che si verificano nella molecola e ne consentono di valutare la stabilità termodinamica. Dato il suo livello di sensibilità nell'individuare piccole differenze di energie verrà poi utilizzato per stimare la stabilità della Glicoproteina spike del SARS-CoV-2.

1.3 Ricerca Locale

La ricerca locale è una tecnica euristica di ottimizzazione e risoluzione dei problemi che mira a trovare una soluzione che sia vicina alla soluzione attuale e la migliore. Funziona apportando piccole modifiche alla soluzione corrente e valutando i risultati, con l'obiettivo di trovare una soluzione ottimale in uno spazio di ricerca limitato. La ricerca locale viene spesso utilizzata nei problemi di ottimizzazione combinatoria, come il problema del commesso viaggiatore, e nell'apprendimento automatico, dove può essere utilizzata per ottimizzare algoritmi complessi come le reti neurali. L'idea chiave alla base della ricerca locale è evitare di rimanere bloccati in soluzioni non ottimali

apportando una serie di piccole modifiche informate alla soluzione corrente fino a quando non viene trovata una soluzione ottimale.

La ricerca locale è un sottocampo di:

- Metaheuristic: è una procedura o euristica di livello superiore progettata per trovare, generare o selezionare un'euristica che può fornire una soluzione sufficientemente buona a un problema di ottimizzazione, in particolari con informazioni incomplete o limitate capacità di calcolo;

- Stochastic optimization: sono metodi di ottimizzazione che generano e utilizzano variabili casuali; I metodi di ottimizzazione stocastica generalizzano metodi deterministici per problemi deterministici;

- Mathematical optimization: è la selezione di un elemento migliore rispetto ad un qualche criterio, da un insieme di alternative disponibili; Nell'approccio più generale, un problema di ottimizzazione consiste nel massimizzare o minimizzare una funzione reale scegliendo sistematicamente i valori di input all'interno di un insieme consentito e calcolando il valore della funzione.

La maggior parte dei problemi può essere formulata in termini di spazio di ricerca e target in diversi modi. Ad esempio, per il problema del commesso viaggiatore una soluzione può essere un percorso che tocca tutte le città e l'obiettivo è trovare il percorso più breve. Ma una soluzione può anche essere un percorso, che può anche contenere un ciclo.

Un algoritmo di ricerca locale parte da una soluzione candidata e quindi si sposta iterativamente verso una soluzione vicina; un quartiere è l'insieme di tutte le possibili soluzioni che differiscono dalla soluzione attuale per la minima misura possibile. Ciò richiede la definizione di una relazione di vicinato nello spazio di ricerca. Ad esempio, l'intorno della copertura del vertice è un'altra copertura del vertice che differisce solo per un nodo. Per la soddisfacibilità booleana, i vicini di un'assegnazione booleana sono quelli che hanno una singola variabile in uno stato opposto. Lo stesso problema può avere più quartieri distinti definiti su di esso; l'ottimizzazione locale con quartieri che implicano la modifica di fino a k componenti della soluzione viene spesso definita k-opt.

Tipicamente, ogni soluzione candidata ha più di una soluzione vicina; la scelta di quale selezionare viene presa utilizzando solo le informazioni sulle soluzioni nelle vicinanze dell'assegnazione corrente, da cui il nome ricerca locale. Quando la scelta della soluzione del vicino è fatta prendendo quella che massimizza localmente il criterio, cioè: una ricerca avida, la metaeuristica

prende il nome di hill climbing. Quando non sono presenti vicini in miglioramento, la ricerca locale è bloccata in un punto localmente ottimale. Questo problema di ottimo locale può essere risolto utilizzando i riavvii (ricerca locale ripetuta con diverse condizioni iniziali), la randomizzazione o schemi più complessi basati su iterazioni, come la ricerca locale iterata, sulla memoria, come l'ottimizzazione della ricerca reattiva, su modifiche stocastiche senza memoria, come la simulated annealing.

La ricerca locale non fornisce una garanzia che una determinata soluzione sia ottimale. La ricerca può terminare dopo un determinato periodo di tempo o quando la migliore soluzione trovata fino a quel momento non è migliorata in un determinato numero di passaggi. La ricerca locale è un algoritmo anytime: può restituire una soluzione valida anche se viene interrotta in qualsiasi momento dopo aver trovato la prima soluzione valida. La ricerca locale è in genere un'approssimazione o un algoritmo incompleto, poiché la ricerca potrebbe interrompersi anche se la migliore soluzione corrente trovata non è ottimale. Ciò può accadere anche se la terminazione avviene perché la migliore soluzione attuale non può essere migliorata, poiché la soluzione ottima può trovarsi lontano dall'intorno delle soluzioni attraversate dall'algoritmo.

Schuurman & Southey propongono tre misure di efficacia per la ricerca locale (profondità, mobilità e copertura):

- Profondità: il costo dell'attuale (migliore) soluzione;
- Mobilità: la capacità di spostarsi rapidamente in diverse aree dello spazio di ricerca (mantenendo bassi i costi);
- Copertura: quanto sistematicamente la ricerca copre lo spazio di ricerca, la distanza massima tra qualsiasi incarico inesplorato e tutti gli incarichi visitati.

Ipotizzano che gli algoritmi di ricerca locale funzionino bene, non perché abbiano una certa comprensione dello spazio di ricerca, ma perché si spostano rapidamente in regioni promettenti ed esplorano lo spazio di ricerca a basse profondità nel modo più rapido, ampio e sistematico possibile.

1.3.1 Tipologie di ricerca locale

La ricerca locale si basa sul presupposto che il miglioramento di una soluzione che si trovi nelle immediate vicinanze di quest'ultima (vicinato). Data una qualsiasi soluzione euristica (nodo corrente) la ricerca locale verifica l'esistenza di soluzioni migliori nell'insieme delle soluzioni più vicine (nodi vicini). Ad

esempio, un algoritmo di ricerca locale può essere utilizzato per risolvere il problema del commesso viaggiatore. Le tecniche di ricerca locale consentono di raggiungere risultati sia nella ricerca delle soluzioni a un problema (problem solving) e sia nell'ottimizzazione. Appartengono alla categoria degli algoritmi di ricerca locale i seguenti algoritmi:

- Ricerca hill climbing: L'algoritmo di ricerca hill climbing è una tecnica di ricerca locale in cui il nodo corrente è sostituito con il nodo migliore;

- Simulated annealing: L'algoritmo di Simulated annealing o Annealing simulato (SA) è un algoritmo probabilistico-metaeuristico di ottimizzazione della ricerca in uno spazio di ricerca di grandi dimensioni. Il suo nome prende spunto dal processo metallurgico che consente di riaggregare la materia fusa dei metalli in base a determinate caratteristiche comuni. Nell'algoritmo di simulated annealing un processo di selezione consente l'eventuale sostituzione della soluzione del nodo corrente con quella di un nodo selezionato casualmente da una lista di soluzioni candidate;

- Beam Search. La beam search è una ricerca locale basata su tecniche euristiche. La beam search (ricerca di fascio) esplora un fascio limitato di nodi promettenti (soluzioni parziali) dello spazio di ricerca e li analizza con una ricerca locale. Utilizzando le tecniche euristiche l'algoritmo beam search consente di riconoscere quando una soluzione parziale è anche una soluzione completa.

1.3.2 Euristiche

Un algoritmo di tipo Ricerca Locale appartengono alla classe delle euristiche di miglioramento. Sono euristici quegli algoritmi che non garantiscono di fornire la soluzione ottima. Ovvero, data una soluzione iniziale ammissibile s, essa è migliorata attraverso successive trasformazioni di s.

Le euristiche sono quindi strategie che aiutano a trovare una soluzione ottimale in una ricerca locale. Ecco alcune delle eurisiche più comuni utilizzate nella ricerca locale:

- First-Improvement: Questa euristica cerca sempre la prima soluzione migliore rispetto alla soluzione corrente. Il vantaggio di questa euristica è che è molto veloce, poiché interrompe la ricerca non appena viene trovata una soluzione migliore. Tuttavia, il suo limite è che potrebbe

non essere in grado di trovare la soluzione ottimale, poiché potrebbe fermarsi presto;

- Best-Improvement: Questa euristica cerca sempre la migliore soluzione possibile rispetto alla soluzione corrente. Il vantaggio di questa euristica è che è molto precisa, poiché cerca sempre la soluzione migliore. Tuttavia, il suo limite è che potrebbe essere molto lenta, poiché deve valutare tutte le soluzioni possibili prima di scegliere la migliore;

- Stochastic Hill Climbing: Questa euristica cerca una soluzione migliore casualmente. Il vantaggio di questa euristica è che è molto flessibile, poiché cerca soluzioni in modo casuale. Tuttavia, il suo limite è che potrebbe essere impreciso, poiché potrebbe scegliere soluzioni subottimali;

- Simulated Annealing: Questa euristica cerca una soluzione migliore basata sulla probabilità. Il vantaggio di questa euristica è che è molto precisa, poiché cerca soluzioni basate sulla probabilità. Tuttavia, il suo limite è che potrebbe essere molto lenta, poiché deve valutare molte soluzioni prima di scegliere la soluzione migliore;

- Variable Neighborhood Search: Questa euristica cerca soluzioni migliori cambiando continuamente il vicinato della soluzione corrente. Il vantaggio di questa euristica è che è molto flessibile, poiché cerca soluzioni in molti vicinati diversi. Tuttavia, il suo limite è che potrebbe essere molto lenta, poiché deve valutare molte soluzioni prima di scegliere la soluzione migliore;

- Greedy Algorithm: Questa euristica seleziona sempre la soluzione che sembra essere la migliore in un determinato momento. Il vantaggio di questa euristica è che è molto veloce, poiché seleziona sempre la soluzione migliore. Tuttavia, il suo limite è che potrebbe non essere in grado di trovare la soluzione ottimale, poiché seleziona sempre la soluzione migliore in un dato momento, senza preoccuparsi delle conseguenze future. Pertanto, è importante utilizzare questo algoritmo con cautela e verificare che la soluzione ottenuta soddisfi i requisiti del problema.

1.4 Shortest Path

Lo shortest path è un importante concetto nell'ambito dell'informatica cosi come nell'ingegneria e nella matematica applicata. Essa si riferisce al per-

corso più breve tra due punti in un grafo, ovvero insieme di nodi collegati da archi.

Esso è molto importante per esempio nel campo della robotica, viene utilizzato per pianificare il percorso del robot, consentendogli di evitare ostacoli e raggiungere il suo obiettivo in modo efficiente. Possiamo trovare lo stesso concetto applicato nelle reti di telecomunicazioni, poiché viene utilizzato per instradare il traffico tra due nodi garantendo una comunicazione efficiente e affidabile. Viene anche usato in algoritmi di routing per cercare di minimizzare il tempo di ritardo. Chiaramente esso si adatta bene nel campo della logistica dove è necessario ottimizzare il trasporto delle merci o delle persone. Infine, trova applicazione anche nel campo della progettazione di circuiti elettronici.

1.4.1 Algoritmo A*

L'algoritmo A* è spesso utilizzato nella risoluzione del problema dello shortest path, ovvero per trovare il percorso più breve tra due nodi in un grafo pesato. L'algoritmo A* utilizza una funzione di stima euristica per guidare la ricerca verso il percorso più breve, permettendo di ottenere un'efficace combinazione di completezza e velocità nell'individuare la soluzione.

Nella versione dello shortest path risolto con l'algoritmo A*, ogni nodo del grafo viene valutato in base alla sua distanza dal nodo di partenza, la stima della distanza dal nodo di arrivo e il costo totale del percorso, ovvero la somma della distanza dal nodo di partenza e della stima della distanza dal nodo di arrivo. In questo modo, l'algoritmo A* tiene traccia del percorso più breve scoperto fino a quel momento e lo utilizza per determinare quale nodo espandere successivamente.

L'algoritmo A* può essere ulteriormente ottimizzato attraverso l'utilizzo di tecniche come la memorizzazione della distanza minima calcolata per ogni nodo, in modo da ridurre il numero di espansioni necessarie. Inoltre, l'algoritmo A* può essere utilizzato con successo in grafi di grandi dimensioni, grazie alla sua capacità di selezionare le espansioni più promettenti e di escludere le aree meno promettenti del grafo.

1.4.2 Esempio

In questo esempio, la funzione A_star prende in input la cella di partenza 'start', la cella di arrivo 'goal' e una rappresentazione del labirinto come 'graph'. La funzione A_star esplora le celle adiacenti alla cella corrente, calcola il costo del percorso per ogni cella e aggiunge le celle alla coda di priorità. La coda di priorità è ordinata in base al costo totale, che è la

somma del costo del percorso e della stima euristica. La funzione A_star restituisce il percorso più breve dal punto di partenza al punto di arrivo.

```
    # Definizione della funzione di costo
    def cost(current, next):
      if current[0] == next[0] or current[1] == next[1]:
        return 1
      else:
        return math.sqrt(2)

    # Definizione della funzione euristica
    def heuristic(current, goal):
      return math.sqrt((current[0]-goal[0])**2 + (current
    [1]-goal[1])**2)

    # Definizione dell'algoritmo A*
    def A_star(start, goal, graph):
      queue = []
      heapq.heappush(queue, (0, start, []))
      visited = set()

      while queue:
        _, current, path = heapq.heappop(queue)
        if current == goal:
        return path + [current]
        if current in visited:
          continue
        visited.add(current)
        for neighbor in graph[current]:
          if neighbor in visited:
            continue
          new_cost = cost(current, neighbor)
          total_cost = new_cost + heuristic(neighbor, goal)
          heapq.heappush(queue, (total_cost, neighbor, path
    + [current]))
      return None
```

Codice 1.1: Algoritmo A*

1.5 Algebra e Geometria

L'algebra e la geometria sono due aree fondamentali della matematica che sono strettamente correlate e si influenzano a vicenda. La geometria si occupa dello studio delle figure geometriche nello spazio, come punti, linee, piani, poligoni, solidi e curve.

L'algebra, come accennato in precedenza, si occupa dello studio delle proprietà delle operazioni aritmetiche e delle relazioni tra le variabili. L'algebra fornisce uno strumento matematico molto potente per la risoluzione di problemi e l'espressione di relazioni matematiche in modo astratto.

L'algebra e la geometria sono strettamente correlate, poiché l'algebra può essere utilizzata per risolvere problemi geometrici, come ad esempio la determinazione di equazioni di rette e di superfici, la risoluzione di equazioni di secondo grado che descrivono curve, e la risoluzione di problemi di trigonometria.

D'altra parte, la geometria può essere utilizzata per visualizzare le relazioni algebriche, ad esempio rappresentando graficamente le funzioni algebriche o le equazioni differenziali.

1.5.1 Spazio vettoriale

Gli spazi vettoriali sono una struttura algebrica complessa, composta da: un campo, in cui gli elementi sono detti scalari; un insieme, i cui elementi sono detti vettori; due operazioni binarie, la somma e la moltiplicazione scalare con determinate proprietà.

Per ogni numero naturale $n \geq 1$ consideriamo l'insieme $\mathbf{R}^n$ delle n-uple ordinate di numeri reali:

$$\mathbf{R}^n = \{v = (x_1, .., x_n) | x_i \in \mathbf{R}\}$$

.

Viene chiamato $\mathbf{R}^n$ lo spazio n-dimensionale e i suoi elementi vettori o punti, come descritto in [L. alessandrini, 2004]. Presi due vettori

$$v = (x_1, .., x_n) e w = (y_1, .., y_n)$$

si possono sommare componente per componente per ottenere un nuovo vettore:

$$v + w = (x_1 + y_1, .., x_n + y_n)$$

.

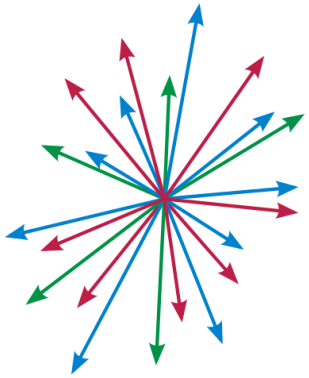


Figura 1.6: Uno spazio vettoriale è una collezione di oggetti, chiamati "vettori", che possono essere sommati e riscalati.

Un numero $c \in \mathbf{R}$ si può moltiplicare con un vettore $v = (x_1, .., x_n)$ in modo da ottenere un nuovo vettore:

$$cv = (cx_1, .., cx_n)$$

.

Le operazioni di somma e moltiplicazione per scalare sono anche chiamate operazioni di spazio vettoriale.

Per la somma di vettori $v, w, u \in \mathbf{R}$ vengono verificate le seguenti proprietà:

- $v + w = w + v$;
- $(v + w) + u = v + (w + u)$;
- $v + \mathcal{O} = v, v + (-v) = \mathcal{O}$, dove $\mathcal{O} = (0, .., 0)$.

Per la moltiplicazione per scalare di $a, b \in \mathbf{R}$ con $v, w \in \mathbf{R}^n$ valgono invece:

- $a(bv) = (ab)v$;
- $a(v + w) = av + aw, (a + b)v = av + bv$;
- $1v = v$.

Si osserva che le operazioni di somma e di moltiplicazione per scalare si possono definire non solo nell'insieme dei numeri reali o delle n-uple di numeri, ma anche in altri insiemi (di funzioni, di matrici, ...). Un insieme con somma e moltiplicazione per scalare che verificano tutte le proprietà precedentemente descritte si dice spazio vettoriale, come descritto in [L. alessandrini, 2004].

1.5.2 Prodotto scalare

Il prodotto scalare di $v = (x_1, .., x_n) e w = (y_1, .., y_n)$ è il numero

$$\langle v, w \rangle := x_1 y_1 + x_2 y_2 + ... + x_n y_n = \sum_{i=1}^{n} x_i y_i$$

talvolta viene anche denotato come $v \cdot w$.

Ci sono importanti proprietà alla base del prodotto scalare, ovvero per $v, w, u \in \mathbf{R}^n e c \in \mathbf{R}$ valgono:

- $\langle v, w \rangle = \langle w, v \rangle$;
- $\langle v, w + u \rangle = \langle v, w \rangle + \langle v, u \rangle$;
- $\langle cv, w \rangle = c\langle v, w \rangle = \langle v, cw \rangle$;
- $\langle v, v \rangle \geq 0$, e $\langle v, v \rangle = 0 \iff v = \mathcal{O}$ (positività di $\langle , \rangle$).

Nel piano cartesiano il prodotto scalare permette di definire e trattare la nozione geometrica di lunghezza di un vettore. Tale concetto può essere esteso ad uno spazio vettoriale di dimensione arbitraria introducendo un concetto analogo: la norma.

1.5.3 Matrici di rotazione

In matematica, e in particolare in geometria, una rotazione è una trasformazione del piano o dello spazio euclideo che sposta gli oggetti in modo rigido e che lascia fisso almeno un punto, nel caso del piano, o una retta, nel caso dello spazio. I punti che restano fissi nella trasformazione formano più in generale un sottospazio: quando questo insieme è un punto o una retta, si chiama rispettivamente il centro e l'asse della rotazione.

Più precisamente, una rotazione è una isometria di uno spazio euclideo che ne preserva l'orientazione, ed è descritta da una matrice ortogonale speciale. Qualunque sia il numero delle dimensioni dello spazio di rotazione, gli elementi della rotazione sono:

- il verso (orario-antiorario);
- l'ampiezza dell'angolo di rotazione;
- l centro di rotazione (il punto attorno a cui avviene il movimento rotatorio).

Nel nostro caso ci concentriamo nello spazio a 3 dimensioni; in questo caso la rotazione è determinata da un asse, dato da una retta r passante per l'origine, e da un angolo θ di rotazione. Per evitare ambiguità, si fissa una direzione dell'asse, e si considera la rotazione di angolo θ effettuata in senso antiorario rispetto all'asse orientato. Senza cambiare base, la rotazione di un angolo θ intorno ad un asse determinato dal versore (x, y, z) (ossia un vettore di modulo unitario) è descritta dalla matrice seguente:

$$\begin{bmatrix} x^2 + (1 - x^2)\cos(\theta) & xy(1 - \cos(\theta)) - z\sin\theta & xz(1 - \cos\theta) + y\sin\theta \\ xy(1 - \cos(\theta)) - z\sin\theta & y^2 + (1 - y^2)\cos(\theta) & yz(1 - \cos\theta) - x\sin\theta \\ xz(1 - \cos(\theta)) - y\sin\theta & yz(1 - \cos\theta) + x\sin\theta & z^2 + (1 - z^2)\cos(\theta) \end{bmatrix}$$

Ponendo $(x, y, z) = (1, 0, 0)$ oppure $(x, y, z) = (0, 1, 0)$ oppure $(x, y, z) = (0, 0, 1)$ si ottiene rispettivamente la rotazione attorno all'asse x, all'asse y e all'asse z. Tale matrice è stata ottenuta scrivendo la matrice associata alla trasformazione lineare (rispetto alle basi canoniche nel dominio e codominio) della formula di Rodrigues.

1.5.4 Super Fibonacci Spirals

Le super spirali di Fibonacci sono un'estensione delle spirali di Fibonacci che consentono la generazione rapida di un arbitrario ma fisso numero di orientamenti 3D. Una valutazione completa rispetto ad altri metodi mostra che gli insiemi di orientamenti generati hanno una bassa discrepanza, componenti spuri minimi nello spettro di potenza e volumi Voronoi quasi identici, come descritto in [Alexa, 2022]. L'ottimizzazione degli insiemi per discrepanze basse è difficile. Inoltre, l'ottimizzazione $\mathcal{SO}(3)$ è scomoda a causa della geometria sottostante. [1]

Lo strumento principale per usare il campionamento di Fibonacci su $\mathcal{S}^3$ è una mappatura che preserva il volume di un cilindro solido in $\mathcal{R}^3$ alla 3-sfera.

[1] $\mathcal{SO}(3)$ è il gruppo delle rotazioni tridimensionali che preservano la norma e il prodotto vettoriale, e che hanno determinante 1. Sono tutte le possibili rotazioni attorno ad un punto nello spazio tridimensionale, senza alcuna traslazione.

Considerando il cilindro $(h, y = (y_0, y_1))| - \pi < h \leq \pi, y^T y \leq 1$. Allora

$$x(h, y) = \begin{pmatrix} z \cos h \\ z \sin h \\ y_0 \\ y_1 \end{pmatrix}, z = \sqrt{1 - y^T y}$$

mappa punti nel cilindro della sfera $\mathcal{R}^4$. La mappatura inversa di $x = (x_0, x_1, x_2, x_3)$ è data da $(h, y_0, y_1) = (\arctan 2(x_1, x_0), x_2, x_3)$. Questo mostra che la mappatura è una biezione tra l'interno relativo del cilindro e la sfera senza equatore $x_0 = x_1 = 0$. La linea $-\pi < h \leq \pi, y^T y \leq 1$ sulla superficie del cilindro sono mappate ai punti $(0, 0, y_0, y_1)$ sull'equatore della sfera. All'interno del cilindro $y^T y < 1$ dove la mappatura è biettiva, possiamo calcolare la Jacobians come:

$$\mathcal{J}_{h,j} = \begin{pmatrix} -z \sin h - \frac{y_0}{z} \cos h - \frac{y_1}{z} \cos h \\ z \cos h - \frac{y_0}{z} \sin h - \frac{y_1}{z} \sin h \\ 010 \\ 001 \end{pmatrix}$$

Usando quindi Jacobians possiamo analizzare il cambio di volume e reclamare che la mappatura $x(h, y) = (h, y)| - \pi < h \leq \pi, y^T y < 1 \mapsto \mathcal{S}^3 \subset \mathcal{R}^4$ preserva il volume.

Si può dimostrare ciò che viene reclamato in precedenza come $det(x(h, y), J_(h, y))$, perché $x(h, y)$ è ortogonale alla tangente al piano e ha lunghezza unitaria. Sviluppando si ottiene

$$\begin{aligned} &+z \cos h(z \cos h) - z \sin h(z \sin h) \\ &+y_0(y_0 \sin^2 h + y_0 \cos^2 h) \\ &-y_1(y_1 \cos^2 h - y_1 \sin^2 h) \\ &= (1 - y^T y)(\cos^2 h + \sin^2 h) + y^T y = 1 \end{aligned}$$

.

Data la mappatura, l'idea è quella di utilizzare il campionamento di Fibonacci 2 volte per generare punti sul cilindro: la prima lungo l'asse principale h del cilindro; la seconda sul cerchio (y_0, y_1) ortogonale all'asse. Usando due differenti costanti ϕ e ψ, per i due campionamenti otteniamo:

$$y(t) = \left(\sqrt{t} \sin \frac{2\pi n t}{\phi}, \sqrt{t} \cos \frac{2\pi n t}{\phi} \right)$$

$$z(t) = \left(\frac{nt}{\psi} - \left\lfloor \frac{nt}{\psi} \right\rfloor, \sqrt{t} \sin \frac{2\pi n t}{\phi}, \sqrt{t} \cos \frac{2\pi n t}{\phi} \right)^T$$

.

Inserendo questo campione nella mappatura della 3-sfera otteniamo la seguente semplice curva che esibisce la simmetria aspettata:

$$w(t) = \begin{pmatrix} \sqrt{t}\sin\frac{2\pi nt}{\phi} \\ \sqrt{t}\cos\frac{2\pi nt}{\phi} \\ \sqrt{t-1}\sin\frac{2\pi nt}{\psi} \\ \sqrt{t-1}\cos\frac{2\pi nt}{\psi} \end{pmatrix}$$

.

Il campionamento di questa curva a valori regolari di t_i è un metodo naturale per la generazione di campioni di orientamento. L'implementazione algoritmica è la seguente 1.

Algoritmo 1 Generazione di n campioni in $\mathcal{SO}(3)$

function SUPER-FIBONACCI(n, ϕ, ψ)
  for $i \in 0, .., n-1$ **do**
    $s \leftarrow i + \frac{1}{2}$
    $t \leftarrow \frac{s}{n}$, $d \leftarrow 2\pi s$
    $r \leftarrow \sqrt{t}$, $R \leftarrow \sqrt{1-t}$
    $\alpha \leftarrow \frac{d}{\phi}$, $\beta \leftarrow \frac{d}{\psi}$
    $q_i \leftarrow (r\sin\alpha, r\cos\alpha, R\sin\beta, R\cos\beta)$

Questo algoritmo mostra che un insieme di kn campioni contiene l'insieme generato per n campioni o, più generalmente, insieme con m e n campioni condividono ogni k-esimo campione, dove k è il minimo comune diviso tra m e n. Giocano un ruolo fondamentale anche gli altri due parametri di questo algoritmo, ovvero ϕ e ψ che devono essere irrazionali, ma anche la loro relazione è importante. Non ci sono però teorie matematiche a supporto quindi è necessario adattarli alla situazione d'uso.

Capitolo 2

Covid

2.1 Covid-19

I coronavirus sono stati scoperti negli anni 60 e da allora i coronavirus sugli umani sono stati identificati a partire dal SARS-CoV nel 2002. La pandemia da Covid-19 è causata da un nuovo ceppo virale chiamato SARS-CoV-2, un virus a singola elica di RNA della famiglia *coronaviridae* (Fig. 2.1). Dei 4 generi di coronavirus $(\alpha, \beta, \gamma, \delta)$ fa parte dei $\beta - CoV$.

Il genoma del virus, come citato in [Salleh et al., 2021], contiene quattro strutture proteiche: la proteina spike; la membrana; l'envelope; la nucleocapside; e da 16 proteine non strutturali (Nsp1-16). La proteina nucleocapside avvolge il genoma del RNA, incapsulato all'interno di un involucro associato alle proteine spike, envelope e membrana. La proteina spike media l'attaccamento del virus ai recettori dell'ospite, ma la approfondiremo nella prossima sezione. La membrana è la proteina più abbondante e definisce la forma dell'involucro virale. La proteina spike media l'interazione con i recettori della cellula ospite, che approfondiremo nella prossima sezione. La proteina

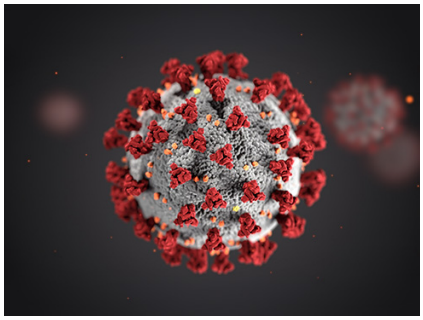


Figura 2.1: Struttura di un Coronavirus

membrana che è la più abbondante e definisce la forma dell'involucro virale. La proteina envelope si attiva durante il ciclo di replicazione e viene rilasciata nella cellula infetta in grandi quantità, ma solo una piccola parte viene incorporata all'interno del involucro virale.

La replicazione del virus comincia con l'interazione della glicoproteina Spike con ACE2, il target espresso nelle cellule umane. Una volta che si viene a creare il legame, il virus entra nel citosol della cellula ospite. Dopodiché avviene la traduzione del gene per la replicazione dal RNA genomico e quindi poi la traduzione e l'assemblaggio dei processi di replicazione virale. Una volta conclusa questa fase avviene l'incapsulamento che porta alla formazione del virus maturo. Una volta completato il processo di assemblaggio viene trasportato sulla superficie cellulare e rilasciato per esocitosi.

Prima dell'epidemia del SARS-CoV erano già stati indivuduati due tipologie di coronavirus nell'uomo che però erano visti come le cause del raffreddore. Con la comparsa nel 2012 di MERS-CoV e l'attuale SARS-CoV-2 è necessario studiare e comprendere appieno quelle che sono le proprietà e le caratteristiche di questo virus.

2.1.1 Storia

Come riportato da [Tinelli,] e da [di Sanità,], si è scoperta l'esistenza di questo virus il 31 Dicembre del 2019 quando l'OMS (Organizzazione Mondiale della Sanità) è stata informata di casi di polmonite di eziologia sconosciuta nella città di Wuhan provincia di Hubei Cina. Il nuovo corona virus è stato ufficialmente annunciato il 7 gennaio del 2020 e tre giorni dopo è stata resa pubblica la sequenza genomica. Sono state poi rilasciate altre sequenze genomiche, le quali tutte suggerivano la presenza di un virus strettamente legato al SARS-CoV.

L'11 Febbraio del 2020 l'OMS ha definito la nuova polmonite indotta da coronavirus come malattia da corona virus 2019. Allo stesso tempo la Commissione internazionale di classificazione dei virus ha annunciato che il virus nominato provvisoriamente come 2019-nCoV veniva nominato come grave sindrome respiratoria acuta SARS-CoV2. Dopo che il patogeno è stato valutato sulla base della filogenesi, della tassonomia e della pratica consolidata, è stato definito un forte legame con il precedente SARS-CoV.

L'inizio della pandemia è avvenuto quindi a Wuhan in Cina. In Italia si è sviluppato un focolaio autoctono che poi si è diffuso progressivamente in tutto il paese e in particolare nelle regioni del nord. Successivamente il virus si espanse in Europa e nel resto del mondo. L'OMS dichiarò l'inizio della pandemia l'11 Marzo del 2020, è poi storia di ogni giorno della pandemia che

ha raggiunto milioni di persone. Si ritiene che comunque il tasso di mortalità del virus sia di circa il 3.5%.

2.2 Glicoproteina Spike

La glicoproteina spike svolge un ruolo essenziale nell'attaccamento, nella fusione e nell'ingresso del virus nella cellula ospite. Una caratteristica dei coronavirus è quella di accedere alle cellule ospiti e poi dare inizio all'infezione attraverso la fusione delle membrane virali alle cellule. Come descritto in [Duan Liangwei,], la fusione della membrana viene mediata dalla membrana della glicoproteina spike e dal recettore affine della cellula ospite. Essendo in una posizione superficiale nella struttura del virus, ciò la rende un bersaglio diretto per le risposte immunitarie dell'ospite rendendola anche il principale bersaglio degli anticorpi. Data la sua importanza nella replicazione e fusione virale è al centro della maggior parte delle strategie vaccinali e degli interventi terapeutici.

La glicoproteina Spike viene sintetizzata come precursore di una poliproteina sul reticolo endoplasmatico ruvido (RER). Il precursore non processato ospita una sequenza segnale del reticolo endoplasmatico (ER) situato nel terminale N, che indirizza la glicoproteina alla membrana RER. Durante la sintesi vengono aggiunte catene laterali di oligosaccaridi ad alto contenuto di mannosio. Poco dopo la sintesi i monomeri della glicoproteina trimerizzano, il che può facilitare il trasporto dall'ER al complesso di Golgi. Il complesso di Golgi è un organulo di composizione lipo-proteica con una delicata struttura nella cellula in posizione paranucleare che si occupa di rielabolare, selezionare ed esportare i prodotti del reticolo endoplasmatico. All'interno del complesso di Golgi la glicoproteina spike viene scissa proteoliticamente dalla furina cellulare o da proteasi simili in S1 e S2. La subunità di superficie S1, che attacca il virus al recettore della superficie della cellula ospite e la subunità S2 che media la fusione delle membrane cellulari alla cellula ospite. Anche dopo la fase di scissione le subunità S1 e S2 rimangono associate attraverso interazioni non covalenti in uno stato di profusione metastabile. La scissione è però necessaria per l'infettività virale ed è anche necessaria per un efficacie infezione delle cellule polmonari. Un segnale di recupero dell'ER costituito da un motivo conservato KxHxx assicura che la proteina matura si accumuli vicino al compartimento intermedio di Golgi dove guidata dall'interazione con la proteina di membrana (M) partecipano all'assemblaggio delle particelle virali. Una frazione delle proteine mature viaggia attraverso via secretoria fino alla membrana plasmatica, dove può mediare la fusione di cellule infette con cellule non infette per formare cellule giganti multinucleate.

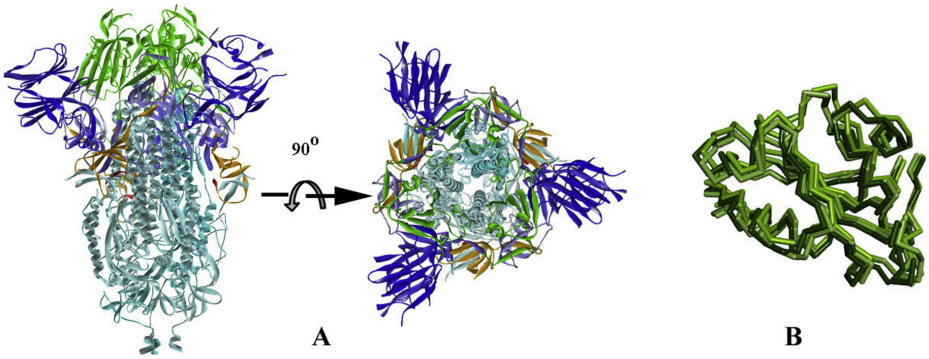


Figura 2.2: Proteina spike: A, in due viste: NTD blu, RBD verde, CTD2 azzurro, CTD3 arancione, S1/S2 link rosso e S2 celeste. B, sovrapposizione degli RBD in conformazione chiusa e aperta; vi è anche lo stato di pre-fusione in colore verde sbiadito.

2.2.1 Struttura della proteina e funzione

Come accenato nei precedenti paragrafi la glicoproteina spike svolge un ruolo fondamentale nell'infezione virale e nella patogenesi. Come descritto in [Laha et al., 2020], la glicoproteina spike è una proteina omo-trimerica con due subunità S1 e S2. La subunità S1 è responsabile del legame ospite-recettore, mentre la subunità S2 aiuta nelle fusione delle membrane del virus e dell'ospite (Fig. 2.2A). La proteina spike è divisa nelle due subunità, ma rimangono comunque legate tra di loro in modo non covalente nello stato di pre-fusione. É possibile dividere ulteriormente la subunità S1 in due sottodomini, ovvero il dominio N-terminale e in tre dominii C-terminale (Fig. 2.2A).

Il CDT1, (Dominio C-terminale 1), è la regione principale della proteina spike per l'interazione ospite-recettore, ed è quindi definita Receptor Binding Domain (RBD). L'RBD subisce dei cambiamenti conformazionali durante il legame con il recettore umano che porta S1 in una configurazione aperta favorevole all'interazione. Confrontando poi le due conformazioni, quella inerte chiusa e quella attiva aperta, si scopre che l'RBD si muove come un corpo rigido attorno alla sua regione di collegamento con NTD e CTD2 (Fig. 2.2B). Un simile cambiamento è visto anche nello stato di pre-fusione.

Ci sono mutazioni senza senso nella proteina spike che vengono classificate come stabilizzanti e destabilizzanti in base ai cambiamenti di energia libera.

Come una tipica proteina di fusione di classe I la glicoproteina spike condivide caratteristiche strutturali topologiche e meccaniche comuni con altre proteina di fusione di classe I come la glicoproteina dell'involucro dell'HIV e l'emoagglutinina del virus dell'influenza. Essa è una macchina coformazio-

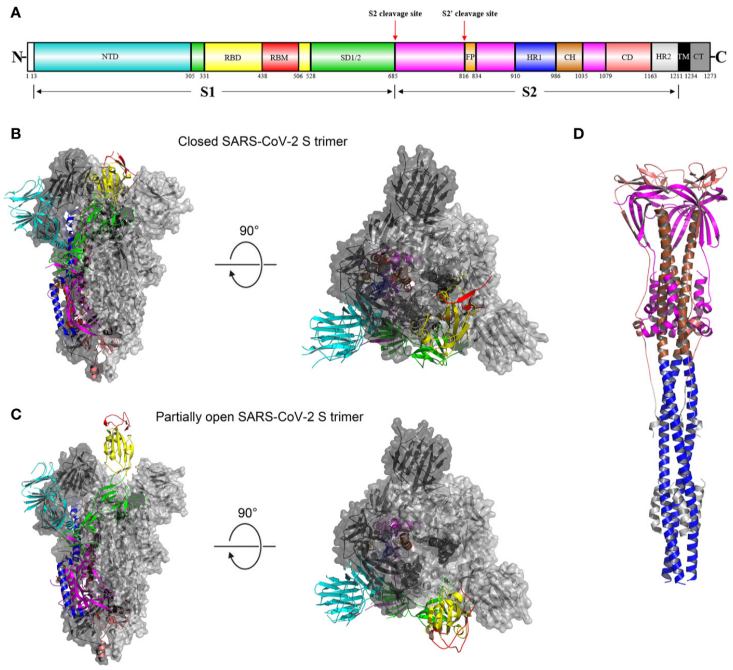


Figura 2.3: Struttura della glicoproteina spike

nale che media l'ingresso virale riorganizzando da uno stato non unliganded metastabile, attraverso uno stato intermedio ad uno stato post fusione stabile. Da quando è stata resa pubblica la struttura sono state scoperte numerose strutture per i frammenti di trimero della glicoproteina spike negli stati pre e post fusione.

La differenza strutturale tra queste due conformazioni risiede solo nella posizione di uno dei tre RDB. Quando tutti e tre gli RBD sono nella posizione giù il trimero risultante di ectodominio S assume una configurazione chiusa, in cui la superficie di legame del recettore dell'RBD S1 è sepolta tra i protomeri e non può essere accessibile dal suo recettore (Fig. 2.3B). Il trimero di ectodominio S con un singolo RBD nella posizione "up" assume una conformazione parzialmente aperta e rappresenta lo stato funzionale poiché la superficie di legame del recettore del RBD "up" può essere completamente esposta (Fig. 2.3C). La subunità S1 riposa mentre il trimero S2 stabilizzano quest'ultimo nella conformazione di pre fusione. Quando il trimero di ectodomionio adotta una conformazione parzialmente aperta l'RBD nella posizione "su" abolirà i contatti con la subunità S2 di un protomero adiacente, destabilizzando la conformazione parzialmente aperta. Ciò sarà vantaggioso per la dissociazione e faciliterà i riagganciamenti subiti per mediare l'ingresso virale.

In particolare è noto che la pre fusione trimerica risiede principalmente in una configurazione chiusa che è conformazionalmente mascherata per eludere le neutralizzazioni mediate dagli anticorpi. Si può quindi pensare che le glicoproteine spike del covid-19 native su virioni maturi e infettivi che condividano una simile caratteristica di mascheramento conformazionale, nascondendo la superficie di legame del recettore.

2.2.2 Scudo di glicani della glicoproteina spike

Come nominato in precedenza la proteina spike del SARS-CoV-2 è fortemente circondata da glicani N-legati che sporgono dalla superficie del trimero. Sono stati visionati fino a 22 glicani N-legati che probabilmente svolgono un ruolo importante nel ripiegamento delle proteine e nell'invasione immunitaria dell'ospite come scudo glicano. Dei 22 potenziali disponibili per la glicosilazione, 14 vengono identificati come prevalentemente occupati da glicani di tipo complesso. I restanti invece risultano dominati da glicani di tipo oligomannosio che sono diversi da quelli fondati sulle glicoproteine dell'ospite. Per glicosilazione si intende una modifica della struttura della proteina da parte del complesso di Goigi, durante o in seguito ad un processo di sintesi proteica. Essa avviene per più motivi, uno dei quali è il raggiungimento del ripiegamento corretto, la può proteggere dall'attaccco di proteasi e aumenta la solubilità della molecola che viene dunque stabilizzata in tutti gli aspetti. Si può anche affermare che l'affinità di legame tra la proteina spike del SARS-CoV-2 e ACE2 non dipendono dalla glicosalinzione della stessa.

Nel caso di SARS-CoV-2, più recentemente è stato dimostrato che un potente anticorpo neutralizzante sia contro SARS-CoV che SARS-CoV-2, S309, riconosce un epitopo RBD contenente glicano altamente conservato. Queste osservazioni suggeriscono che le frazioni di carboidrati potrebberò essere immunogeniche ed evidenziano la necessità per gli immunogeni di mostrare i glicani importanti per il riconoscimento degli anticorpi neutralizzanti. Di conseguenza anche in questo caso è diventa fondamentale la ricerca in questo campo per i vaccini.

2.3 RBD

Come descritto in [Valério M,], l'enzima di conversione dell'angiotesina 2 (ACE2) è un recettore di ingresso per SARS-CoV-2. Interazioni dettagliate tra il SARS-CoV-2 RBD e il suo recettore sono state rivelate da diverse strutture con ACE2. Come detto nella sezione precedente le subunità S1 e S2 sono responsabili del legame del recettore e della fusione della membrana. La

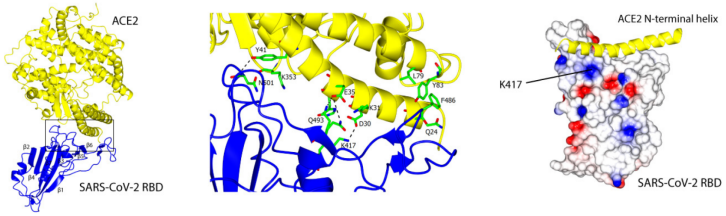


Figura 2.4: SARS-CoV-2 RBD/ACE2. L'immagine a sinistra mostra la struttura complessiva del SARS-CoV-2(blu) legata a ACE2(giallo). L'immagine centrale mostra in dettaglio alcuni legame idrogeno e un ponte salino tra SARS-CoV-2 e ACE2. Nell'immagine sulla destra viene mostrata la mappa del potenziale elettrostatico del SARS-CoV-2 RBD con l'elica N-terminale di ACE2. Il dettaglio lo si può trovare nella Tab. 2.1

subunità S1 è costituita da un dominio N-terminale e un dominio di legame o RBD. Nello stato di pre fusione, la proteina S esiste come omotrimero e subisce grandi cambiamenti conformazionali per controllare l'esposizione e l'accessibilità del RBD. Tutto questo avviene mediante un meccanismo di "su" e "giù", la differenza sta nel rendere accessibile o inaccessibile il recettore. La struttura del nucleo RBD quando è lagata ad ACE2 è costituita da un foglio β antiparallelo a cinque filamenti intrecciati con eliche e anelli di collegamento corti.

hACE2	Q24	D30	E35	E37	D38	Y41	Q42
SARS-CoV-2	N487	K417*	Q493	Y505	Y449	T500/ N501	G446/ Y449
hACE2	Y83	Q325	E329	N330	K353	R393	
SARS-CoV-2	Y489/ N487				G502	Y505	

Tabella 2.1: Tabella che riassume i principali residui che formano legami idrogeno e ponti salini tra ACE2 e SARS-CoV-2.

Il residuo che forma ponti salini è contrassegnato con un asterisco.

Questa struttura centrale del foglio β è ulteriormente stabilizzata da 4 legami di di solfuro. Tra i filamenti centrali c'è una regione estesa contenente 2 filamenti β corti, le eliche e gli anelli. Questa regione è il motivo legante il recettore (RBM) che contiene la maggior parte dei residui responsabili dell'interazione con ACE2. Quando complessato con ACE2, l'RBM si ripiega in una superficie concava che ospita l'α-elica N-terinale di ACE2. E' proprio in questa superficie che diversi residui di RBM stabiliscono intera-

zioni specifiche e non specifiche con i residui di ACE2. Dai dati disponibili riguardo alla struttura sembrerebbe che la struttura centrale sia abbastanza stabile, mentre l'RBM risulta molto dinamico e non definito strutturalmente, a meno che non sia legato ad altre proteine come ACE2. Come citato in [Salleh et al., 2021], c'è un'elevata somiglianza tra l'RBD del SARS-CoV e l'RBD del SARS-CoV-2. Comparando gli RBD di SARS-CoV-2 e SARS-CoV, si nota che un numero maggiore di residui dell'RBD di SARS-CoV-2 interagiscono con ACE2, il che potrebbe spiegare la sua affinità di legame 10 volte superiore. Come si può notare in Fig. 2.4, K417 si trova al di fuori dell'RBM e forma una connessione a ponte salino con D30 di ACE2. K417 si può vedere come un cerotto carico positivamente sulla superficie di RBD che si collega all'Asp caricato negativamente di ACE2, per ottenere un legame più forte.

Durante il corso della pandemia sono state segnalate un numero significativo di mutazioni naturali della proteina Spike. Molte delle mutazioni sono state identificate nel RBD, alcune delle quali hanno dato origine a varianti virali. Si ritiene che molte di queste mutazioni RBD aumentino l'affinità di legame per ACE2 o riescono ad ingannare in modo migliore gli anticorpi monoclonali.

I vaccini che sono stati elaborati nel corso della pandemia vanno proprio ad agire in questa zona tra RBD e ACE2, cercando di impedire che avvenga il contatto e che quindi impedire di conseguenza l'ingresso del virus all'interno dell'ospite.

Capitolo 3

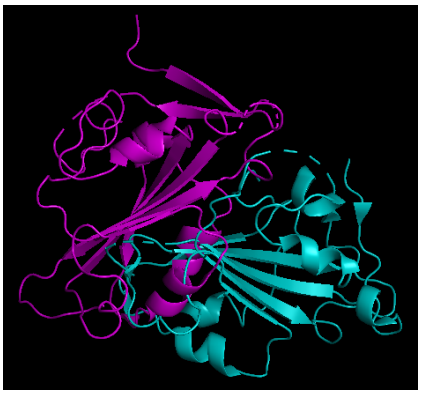
Obbiettivi

Come spiegato nel precedente capitolo il caso di studio su cui ci focalizziamo e incentriamo il nostro lavoro è la glicoproteina spike del SARS-CoV-2. Focalizziamo la nostra attenzione su questa proteina per studiarne il comportamento attraverso l'uso di tecniche di ricerca locale, avvicinandosi molto alle tecniche di dinamica e modellistica molecolare. Queste due tecniche introdotte si basano su metodi teorici o tecniche computazionali utilizzate per simulare il comportamento delle molecole. Queste tecniche vengono utilizzate per studiare la dinamica di evoluzione nel tempo di un sistema chimico e svolgere il processo mediante l'utilizzo della tecnologia permette l'applicazione della modellistica a sistemi relativamente complessi. Il livello di dettaglio che si può ottenere utilizzando questi sistemi è il livello atomistico; il più piccolo livello di informazione rappresentato dagli atomi individuali (o piccoli gruppi di atomi). Queste tecniche vengono utilizzate per svariati compiti dal folding proteico, la catalisi enzimatica, la stabilità delle proteine, i cambiamenti conformazionali associati alla funzione biomolecolare e il riconoscimento molecolare delle proteine. Le tecniche descritte in precedenza hanno però delle problematiche a partire dalle risorse necessarie a compiere questo tipo di compiti, il tempo computazionale necessario e il fatto che si possa lavorare in modo completo solo su piccole strutture oppure è necessario limitare di molto lo spazio di lavoro.

Ricollegandoci alla Glicoproteina spike del SARS-CoV-2 prendiamo in considerazione entrambe le configurazioni sia quello in stato chiuso che quella in stato aperto; esse differiscono per lo stato in cui si trova la parte mobile: nella chiusa si trova in uno stato di pre fusione; nel momento in cui il virus raggiunge i tessuti in cui è espressa ACE2 (il suo target), il microambiente cellulare, attraverso meccanismi biochimici complessi che interessano lo scudo glicidico della Spike, innesca la variazione conformazione dell'RBD che passa in conformazione aperta. Quando l'RBD è in conformazione aperta

lega ACE2. Possiamo vedere realmente la differenza delle due configurazioni guardando la Fig. 3.1.

(a) *Struttura intera*



(b) *Dettaglio*

Figura 3.1: La Glicoproteina spike nelle due configurazioni: contraddistinta dal colore magenta troviamo la configurazione chiusa; contraddistinta dal colore ciano troviamo la configurazione aperta.

L'obbiettivo di questo framework lavorando sulle due configurazione della glicoproteina spike del SARS-CoV-2 è quello di trovare uno shortest path tra

le due configurazioni utilizzando due principi differenti per il costo del singolo movimento.

- calcolo del costo mediante la sola geometria, ovvero somma delle componenti di traslazione e angolo di rotazione

- calcolo del costo mediante non solo la geometria, ma l'utilizzo della funzione d'energia per minimizzare le variazioni di energia.

Il percorso più breve porta con se la necessità di ottenere delle configurazioni intermedie per raggiungere l'obbiettivo preposto e questo ci porta a dover affrontare il problema della fattibilità della connessione tra parte mobile nella nuova configurazione e la parte fissa [Fig. 3.2]. In questo caso si va effettivamente ad agire sugli amminoacidi che fanno parte dei due loop che collegano appunto le parti e in questo caso possiamo affermare che il framework proposto si occupa non solo del movimento della backbone (catena principale), ma tiene in considerazione anche la presenza delle catene laterali. Le catene laterali non vengono coinvolte attivamente nel movimento, ma viene considerato l'ingombro nei confronti e nel rispetto delle proprietà chimico-fisiche. Il framework quindi muove la catena principale rispettando la struttura dell'amminoacido e poi nel rispetto della struttura si preoccupa che la catena laterale non effettui clash con nessun amminoacido nelle vicinanze.

Il motivo per cui è stato effettuato questo lavoro è quello di proporre un framework che fosse in grado di fornire un metodo per lo studio del movimento di una proteina in modo più ampio rispettando comunque tutte le regole necessarie in questo tipo di movimenti. Questo framework consente quindi di effettuare studi di più larga scala rispetto a quanto fornito dai metodi di dinamica molecolare e giocando con i parametri che tengono in considerazione il clash possiamo fornire uno strumento con complessità computazionale minore.

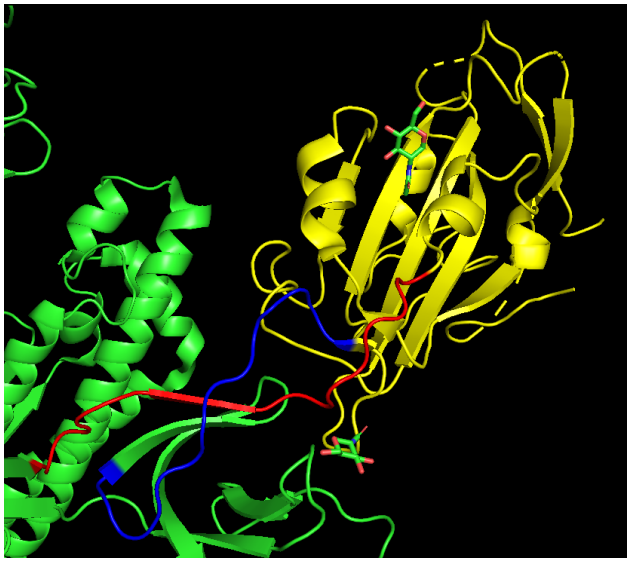


Figura 3.2: Nell'immagine dettagliata notiamo i due loop che collegano la parte mobile alla parte fissa colorati rispettivamente di rosso e di blu.

Capitolo 4

Progettazione

In questo capitolo viene affrontata la parte di implementazione e progettazione, vedremo le strategie di ricerca scelte, l'organizzazione delle strutture dati a supporto dell'esecuzione e poi avremo un approccio top-down verso il codice scritto. Introduco anche la libreria di supporto che ci ha permesso di svolgere il lavoro sul python.

4.1 Librerie a supporto

In questo framework mi sono avvalso della libreria opensource [Jeff Chang,]. É una libreria di strumenti per la biologia computazionale e la bioinformatica. Essa contiene classi per rappresentare sequenze biologiche e annotazioni di sequenze ed è in grado di leggere e scrivere file provenienti da diversi formati. Questa libreria mi ha permesso di utilizzare il parser per i file pdb (Protein Data Bank) al cui interno sono codificate le informazioni riguardanti gli atomi come nome dell'atomo posizione all'interno dello spazio tridimensionale ed altre informazioni. Oltre a fornirmi un parser, al suo interno sono presenti dei moduli che permettono di generare oggetti come le catene, i residui e gli atomi stessi. Questo da la possibilità di chiamare metodi specifici che semplificano il lavoro di gestione delle entità atomo, catena e residui. Viene fornita anche la possibilità di scrivere file in output nel formato desiderato (pdb), il che ci porta al secondo strumento utile il PyMOL.

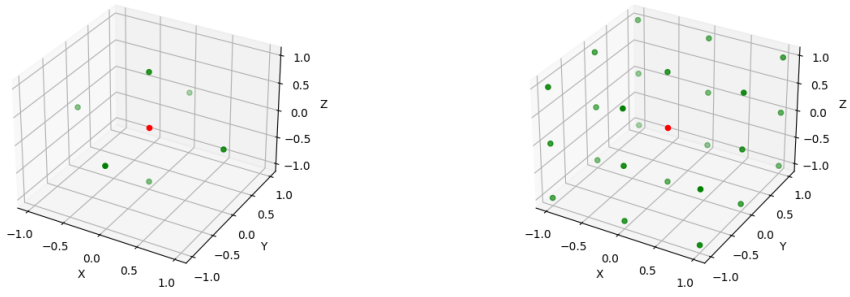
Il PyMOL è un opensuorce software di grafica 3D, che viene utilizzato per la rappresentazione di biomolecole. Come si può dedurre dalla parte iniziale del nomem si riferisce al linguaggio python. Risulta molto utile nella fase di verifica, ovvero quando si deve controllare ciò che è stato compiuto su una molecola. Fornisce un linguaggio di programmazione molto simile al

python che permette di creare script per maneggiare e visionare le modifiche effettuate.

4.2 Strategie di ricerca

All'interno di questo framework si è reso necessario utilizzare tecniche di ricerca locale nella fase in cui si cerca di andare a ristabilire il collegamento dei due loop alla parte mobile una volta effettuata una rotazione/traslazione su di essa. In questo caso la tecnica utilizzata è la hill-climbing, in cui si cerca di trovare il massimo (o il minimo) di una funzione di costo attraverso la ricerca di soluzioni vicine a quella corrente. In questo caso specifico, la funzione di ricerca cerca di ottimizzare la conformazione di un loop di collegamento alla volta attraverso la rotazione degli angoli torsionali. Si parte da una configurazione iniziale del loop rappresentata mediante ad una lista di residui, e cerco di migliorarla ruotando uno dei due angoli torsionali a disposizione, partendo da un residuo specificato. L'angolo torsionale viene scelto casualmente tra ϕ o ψ. Una volta applicata la rotazione viene restituita la nuova conformazione della proteina e si procede in questo modo fin tanto che non viene raggiunto il risultato desiderato.

Viene utilizzata anche una tecnica di ricerca per guidare il processo che porta al completamento del percorso più breve tra le due configurazioni. La strategia di ricerca, in questo caso, è una forma di shortest path con una coda di priorità implementata tramite heap, albero binario ordinato. Essa viene utilizzata per selezionare il nodo con il costo totale (cioè il costo del percorso finora più la stima del costo rimanente) più basso in ogni iterazione. In questo modo si cerca di espandere i nodi che hanno costo totale più basso per primi. Ad ogni iterazione viene preso il vicinato del punto analizzato, si trasla in questi nuovi punti, come si può vedere in Fig. 4.1. Per ogni transizione mediante un sistema di indici si risale ad un insieme di possibili rotazioni amiche, concetto che approfondiremo nelle sezioni successive, e dopodiché si prova la convergenza.



(a) *6 vicini* (b) *26 vicini*

Figura 4.1: In (**a**) vengono mostrati i 6 vicini come intorno del punto (0,0,0) contrassegnato di rosso; mentre in (**b**) vengono mostrati i 26 vicini di (0,0,0) contrassegnato sempre in rosso.

4.3 Strutture dati a supporto

Le strutture dati sono importanti all'interno di un algoritmo, non solo per permettere di conservare informazioni, ma incidono anche sul tempo di esecuzione del singolo programma. Organizzare quindi l'accesso alle strutture dati nel modo migliore permette di risparmiare tempo d'esecuzione oltre a semplificare nella maggior parte dei casi la comprensione della stessa.

Il linguaggio di programmazione scelto, python, necessità di studiare nel modo corretto come accedere alle strutture dati per evitare di trovarsi con dati inconsistenti poiché sono frutto di più modifiche. Per questo motivo e per quanto detto in precedenza io ho previsto 8 strutture di dati importanti: due dedicate a contenere la parte mobile e i loop, due dedicate al salvataggio delle posizioni acquisite da parte mobile e loop durante il cammino da una configurazione all'altra, due dedicate al salvataggio temporaneo e al ripristino in caso di clash delle coordinate dei loop pre e post rotazione, due dedicate alla gestione dei clash e due dedicate al concetto di amicizia delle matrici di rotazione pre-calcolate.

Le strutture dati sono state quasi tutte sviluppate come dei dizionari ad accesso chiave valore. Per quanto riguarda la struttura dati dedicata al controllo della presenza dei clash, la struttura contiene tuple di coordinate tridimensionali e per ciascuna tupla è contenuto la catena il residuo e l'atomo che si trovano in quella posizione. Questo è stato possibile mediante una discretizzazione di tutti gli atomi all'interno di una matrice tridimensiona-

le. Chiaramente il passo di campionamento può essere impostato mediante costante, di default è impostato a 5 Ångström. [1]

Per quanto riguarda le strutture dati per mantenere salvate le coordinate dei loop e della parte mobile anch'esse sono organizzate come dizionari la cui chiave indicizza l'insieme delle varie coordinate. Le chiavi che indicizzano la struttura sono degli interi calcolati nel seguente modo:

$$key = x + y \cdot dx + z \cdot dx \cdot dy + r \cdot dx \cdot dy \cdot dz$$

In questo caso x, y, z identificano la traslazione che abbiamo effettuato sulla parte mobile, r identifica univocamente la rotazione applicata alla parte mobile, mentre dx, dy, dz sono le grandezze che su ogni asse separano la configurazione aperta da quella chiusa. Attraverso metodi di get e set possiamo settarli nelle variabili coinvolte nel processo evitando problemi di condivisione della memoria.

Abbiamo due dizionari dedicati al salvataggio temporaneo e al ripristino delle coordinate nel caso la rotazione non sia buona. Questi dizionari sono solamente per i due loop all'interno della funzione che si occupa di farli convergere alla parte mobile. La struttura dedicata ai loop e alla parte mobile invece sono due liste che a loro volta contengono residui a cui applicare le varie rotazioni.

Le ultime due strutture dati nominate, utilizzate per il concetto di amicizia, sono a loro volta dei dizionari indicizzati per chiave, dove la chiave è appunto l'indice r nominato in precedenza che identifica univocamente all'interno di queste strutture una matrice di rotazione e i suoi amici visti come una lista di indici.

Voglio comunque porre enfasi sullo studio necessario per progettare la singola struttura che mi ha concesso non solo di risolvere il problema della condivisione della memoria, ma anche di velocizzare il processo di accesso alla struttura dati e quindi portare più velocità nell'esecuzione della ricerca locale.

4.4 Approccio Top-Down al codice

Analizziamo lo pseudo-codice che permette di compiere il lavoro preposto dall'obbiettivo.

Come si può vedere in 2 si vanno ad inizializzare le componenti principali del sistema. Innanzitutto viene creato un parser che ha il compito di parsare il file pdb in ingresso. Mediante il parser vengono estratte le configurazioni

[1] Ångström, sono un'unità di misura, non SI, pari a $10^{-}10m$.

Algoritmo 2 Inizializzazione framework

$parser \leftarrow PDBParser(PERMISSIVE = True, QUIET = True)$
$covidchiusa \leftarrow parser.getstructure(titleclosed)$
$covidaperta \leftarrow parser.getstructure(titleopen)$
$calcoloamici(DIV)$
$ptmob \leftarrow partemobile(listacatene, catenainteresse)$
$centroide \leftarrow calcolocentroide(ptmob)$
$normalizzazionestruttura(covidchiusa, covidaperta, centroide)$
$calcoloparallelepipedo(covidchiusa, covidaperta)$
$loop \leftarrow identifyloop(listacatene, catenainteresse, costantiA)$
$coil \leftarrow [loop]$
$loop \leftarrow identifyloop(listacatene, catenainteresse, costantiB)$
$loopr \leftarrow loop[:: -1]$
$coil.append(loopr)$
$salviamo(coildictcoordinates)$
$key = x + y \cdot dx + z \cdot dx \cdot dy + r \cdot dx \cdot dy \cdot dz$
$ptmob \leftarrow partemobile(listacatene, catenainteresse)$
$storedptmob \leftarrow salviamo(ptmob)$
if $key \notin posizioniacquisiteptmob$ **then**
    $insert(posizioniacquisite, key)$
$residuiparte fissa \leftarrow parte fissa(listacatene, catenainteresse)$
$coordinate \leftarrow salviamo(coildictcoordinates)$
if $key \notin posizioniacquisiteloop$ **then**
    $insert(posizioniacquisiteloop, key)$
$nuova\_chiave, prev \leftarrow shortest\_path(key)$
$printpdb(configurazioni)$

dai file pdb. Dopodiché vengono calcolate le matrici di rotazione e il concetto di amicizia tra matrici di rotazione, come riportato in 4.4.1. Una volta fatto, calcoliamo il centroide della parte mobile aperta e normalizziamo la struttura chiusa e aperta, in modo da averle centrate in (0,0,0). Dopodiché si restringe lo spazio di ricerca, come mostrato in 4.4.2.

Una volta terminata la fase di inizializzazione, vengono presi i loop normalizzati, la parte mobile della configurazione chiusa normalizzata e la parte fissa utile per la gestione dei clash. Generiamo la chiave che ci permette di salvare le informazioni e cominciamo a cercare convergenza tra le due configurazioni mediante 4.5.

4.4.1 Amicizia tra matrici di rotazione

Utilizzando il metodo descritto in 1.5.4, si possono generare un insieme di vettori x che rappresentano i vettori x delle matrici di rotazione tridimensionale che verranno poi utilizzate per ruotare la parte mobile.

Il concetto di amicizia tra due matrici è relativo al prodotto scalare tra i vettori x e y di due matrici. In questo caso è necessario definire delle soglie per discriminare una matrice piuttosto che un altra, le soglie impostate sono per quanto riguarda il prodotto scalare dei vettori x 0.94, mentre per i vettori y 0.87. Vengono impostate queste soglie poiché nel nostro caso il prodotto scalare dei vettori è compreso tra $[-1, +1]$, dove $+1$ rappresenta l'ugualianza dei due vettori, mentre -1 rappresenta la completa estraneità dei due vettori. Vengono scelte le seguenti soglie, perché vogliamo rendere piccolo lo spostamento da una matrice all'altra quando vengono applicate alla parte mobile; infatti, movimenti più piccoli garantiscono una migliore probabilità di convergenza tra i loop e la parte mobile.

L'algoritmo proposto 3 permette di ottenere il risultato mostrato in Fig. 4.2, il parametro scelto è 20, che permette di generare 400 diversi vettori x che a loro volta danno vita a 8000 matrici, in questo modo attraverso le soglie impostate in precedenza possiamo ottenere circa 3 o 4 migliaia di matrici amiche.

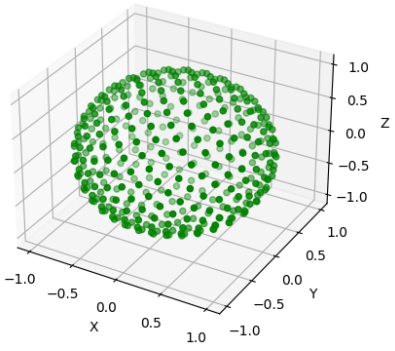


Figura 4.2: Punti equidistanti sulla superficie di una sfera, creando una distribuzione uniforme degli stessi.

Algoritmo 3 Procedura che permette il calcolo degli amici.

procedure CALCOLOAMICI(DIV)
  $sphere \leftarrow []$
  $n1 \leftarrow DIV$
  $n \leftarrow n1 * n1$
  $golden\_angle \leftarrow 3.1415 \cdot (3 - \sqrt{5})$
  for i **in** $range(n)$ **do**
    $\theta \leftarrow golden\_angle \cdot i$
    $pos \leftarrow float(i/(n-1))$
    $z \leftarrow (1 - 1.0/n - (1.0/n - 1)) \cdot pos + (1.0/n - 1)$
    $radius \leftarrow \sqrt{1 - z \cdot z}$
    $temp \leftarrow [0, 0, 0]$
    $temp[2] \leftarrow radius \cdot \cos(\theta)$
    $temp[1] \leftarrow radius \cdot \sin(\theta)$
    $temp[0] \leftarrow -z$
    $sphere.append(temp)$
  for x, j **in** $zip(sphere, range(len(sphere)))$ **do**
    $multiplier \leftarrow [0, 0, 1]$
    if $j == 0$ **then**
      $multiplier \leftarrow [0, 0, -1]$
    $y \leftarrow \langle x, multiplier \rangle$
    $y\_norm \leftarrow normalizza(y)$
    $z \leftarrow \langle x, y \rangle$
    $z\_norm \leftarrow normalizza(z)$
    for i **in** $range(DIV)$ **do**
      $angle \leftarrow radians((360/DIV * i))$
      $y_new \leftarrow y_norm \cdot \cos(angle) - z \cdot \sin(angle)$
      $z_new \leftarrow y_norm \cdot \sin(angle) + z \cdot \cos(angle)$
  $amici \leftarrow amici(0.94, 0.87)$

4.4.2 Riduzione area di ricerca

Restringere l'area di ricerca è fondamentale per non perdere tempo computazionale in zone dello spazio tridimensionale che non sono utili ai fini della ricerca. Per questo ho sviluppato la seguente funzione 4. In questo modo riu-

Algoritmo 4 Calcolo del parallelepipedo

procedure CALCOLOPARALLELEPIPEDO($chiusa, aperta$)
$mobilclosedpart \leftarrow partemobile(listacatene, catenainteresse)$
$mobilopenedpart \leftarrow partemobile(listacatene, catenainteresse)$
$ca \leftarrow$ CALCOLOCENTOIDE($mobilopenedpart$)
$cc \leftarrow$ CALCOLOCENTOIDE($mobilclosedpart$)
$x\_parallelepipedo\_min \leftarrow int(min(0, (ca[0] - cc[0]) - 1))$
$x\_parallelepipedo\_max \leftarrow int(max(0, (ca[0] - cc[0]) + 1))$
$y\_parallelepipedo\_min \leftarrow int(min(0, (ca[1] - cc[1]) - 1))$
$y\_parallelepipedo\_max \leftarrow int(max(0, (ca[1] - cc[1]) + 1))$
$z\_parallelepipedo\_min \leftarrow int(min(0, (ca[2] - cc[2]) - 1))$
$z\_parallelepipedo\_max \leftarrow int(max(0, (ca[2] - cc[2]) + 1))$
$dx \leftarrow x\_parallelepipedo\_min - x\_parallelepipedo\_max$
$dy \leftarrow y\_parallelepipedo\_min - y\_parallelepipedo\_max$
$dz \leftarrow z\_parallelepipedo\_min - z\_parallelepipedo\_max$

sciamo a restringere il campo di lavoro, ad una sorta di parallelepipedo che contiene entrambe le configurazioni. Il parallelepipedo è individuato dalle coordinate per ogni asse massime e minime e dopodiché possiamo verificare l'appartenenza all'interno dello stesso mediante una semplice funzione che verifica l'appartenza. All'interno di questa funzione vengono anche calcolati gli indici che permettono di calcolare la chiave di indicizzazione di ogni nodo generato.

4.5 Shortest-Path

Come descritto in 3, l'obbiettivo è quello di convergere tra la conformazione chiusa e quella aperta mediante il cammino con il minor costo possibile. Per fare ciò lavoriamo e ci concentriamo esclusivamente sulla conformazione chiusa, l'utilizzo della conformazione aperta è esclusivamente necessario per verificare il raggiungimento dell'obiettivo.

Terminando la fase di inizializzazione in 2, si procede nel individuare e salvare i loop, che mettono in collegamento la parte mobile con la parte fissa. I due loop sono simili, ma diversi nella direzione da loro presa, ovvero il

loop-A si dirige dalla parte fissa alla parte mobile, mentre il loop-B lavora al contrario quindi per rendere omogeneo il comportamento del programma nei confronti dei due loop è necessario capovolgere il secondo. Questo non comporta nessun altra modifica, è però necessario ricordarsi di capovolgere il loop nel momento in cui si effettua il processo di scrittura file. Le costanti sul posizionamento dei loop sono state fornite insieme alle configurazioni ed è importante sottolineare che è stato preso un residuo in più in capo e in coda al loop, per verificare che ogni modifica si adatti bene alla struttura a cui si deve attaccare. Una volta che abbiamo terminato l'attività di salvataggio dei loop e della parte mobile avviamo shortest-path 5 e una volta concluso stampiamo passo passo le configurazioni intermedie.

Algoritmo 5 Shortest-Path

```
procedure SHORTEST-PATH(start_idx)
    heap ← [(0, start_idx)]
    prev ← {}, costo ← {}
    prev[start_idx] ← −1, costo[start_idx] ← 0
    while heap do
        cost, indice_current ← heap
        movimento, r ← indice_current
        ptmob ← load(indice_current), loop ← load(indice_current)
        friends ← amici(r)
        for traslazione in traslazioni do
            for r2 in friend do
                ptmob ← load(r2), ptmob ← applytransition(traslazione)
                if not is_allowed(ptmob) then
                    continue
                pre − processing(ptmob)
                convergenza ← esecuzione_rotazioni()
                if convergenza then
                    nuovo_indice ← indice_current
                    nuovo_costo ← costo + costo(r, r2)
                    if nuovo_costo < costo[indice_current] then
                        prev[nuovo_indice] ← indice_current,
                        costo[nuovo_indice] ← nuovo_costo
                        heap ← (nuovo_indice, nuovo_costo)
                        insert(posizioniacquisiteloop, nuovo_indice)
                        insert(posizioniacquisiteloop, nuovo_indice)
                        if dist(ptmob, covid_aperta) < 0.1 then
                            return nuovo_indice, prev
```

Come descritto in 5, viene utilizzata una coda di priorità implementata tramite heap, albero binario ordinato, in cui passo passo vengono inseriti i nuovi nodi scoperti. Si inizia iterando sulla struttura dati, e si estrae il nodo che ha costo minore. Una volta estratto il nodo, prendiamo l'indice e ne ricaviamo la traslazione che fino a quel momento ci ha permesso di arrivare in quella posizione e la rotazione della parte mobile. Recuperiamo e settiamo nuovamente le coordinate alla parte mobile e ai loop, in modo tale da ripartire da dove eravamo rimasti. Una volta estratte le rotazioni amiche dell'attuale, si cercano nuove soluzioni.

Per ogni traslazione elencate nell'array traslazioni, come in Fig. 4.1, e per ogni rotazione amica si cerca convergenza. Prima di raggiungere se possibile la convergenza è necessario effettuare delle operazioni, ovvero dobbiamo caricare la parte mobile nella determinata rotazione $r2$ poiché tutte le rotazioni sono pre-calcolate in modo da ridurre il tempo di esecuzione; dobbiamo traslare la parte mobile nella corretta posizione fornita dagli indici. Una volta posizionata correttamente la parte mobile è necessario effettuare il controllo che la parte mobile sia all'interno del parallelepipedo, per i motivi descritti in 4.4.2, dopodiché se la rotazione è consentita dobbiamo effettuare il pre-processing per determinare le strutture che permettono di comprendere la presenza di clash quando si ruotano i residui che compongono i loop.

Terminata la fase di pre-processing necessaria all'individuazione di clash, si procede alla convergenza, ovvero si cerca di far convergere i loop alla nuova posizione della parte mobile, vedremo più nel dettaglio questa funzione nella prossima sottosezione. Se viene raggiunta convergenza, nella maggior parte dei casi è cosi perché gli spostamenti sono relativamente piccoli a meno di clash insormontabili, va inserito all'interno dell'heap il nuovo nodo se e soltanto se il nuovo costo è migliore del costo già presente per quel nuovo indice nel dizionario dei costi. Il nuovo costo viene calcolato come descritto in 4.5.1. Se effettivamente minore inseriamo il nuovo costo ed è anche necessario aggiornare anche il nodo precedente che ci ha permesso di arrivare a questo nodo nella struttura prev.

Se abbiamo raggiunto convergenza è necessario poi salvare le nuove coordinate sia dei loop che della parte mobile. É importante anche capire se abbiamo ottenuto il nostro obbiettivo principale ovvero se siamo arrivati a convergenza tra le due configurazioni, quella attuale e quella aperta, per fare ciò utilizziamo semplicemente la distanza euclidea tra la configurazione attuale e quella di arrivo. La soglia impostata per terminare il lavoro è 0.1 A che garantisce l'effettiva convergenza.

4.5.1 Calcolo costo arco

La parte più importante all'interno dell'algoritmo di shortest-path è quello di calcolare il costo del nodo che si sta analizzando. Il nostro costo viene calcolato mediante 3 contributi: il primo è il costo del movimento delle coordinate; il secondo è costo dell'amicizia tra le matrici di rotazione; il terzo è lo spostamento dall'asse centrale che supponiamo essere il percorso migliore.

Analizzando nel dettaglio le componenti troviamo che il costo del puro movimento è costante a 1, poiché che ci si muova in negativo o in positivo il delta di spostamento è 1 a patto che venga utilizzato il vicinato di transizione semplice (Fig. 4.1 **a**).

Il costo dell'amicizia tra due matrici è estratto come prodotto scalare dei due vettori x e y, però preso in questo modo, ovvero $(1-\langle x, x2\rangle)+(1-\langle y, y2\rangle)$. In questo il costo di amicizia ci permette di pagare poco se ci muoviamo poco, mentre pagare un costo elevato se decidiamo di effettuare un movimento elevato.

Il terzo componente è dedicato alla distanza del nuovo centroide generato dal segmento, ovvero se ci allontaniamo dall'ipotetico percorso migliore paghiamo una penalità. Questo coefficente viene calcolato come la distanza tra un punto e un segmento usando la proiezione ortogonale del punto sul segmento e la distanza euclidea tra il punto originale e il punto proiettato sul segmento.

Queste tre componenti determinano il costo del nuovo nodo generato, privilegiando i nodi più vicini al percorso ottimale e lasciando lo sviluppo dei nodi pù marginali in fondo.

4.5.2 Convergenza tra loop e parte mobile

La parte più importante è appunto questa che si occupa della convergenza dei loop, verso la nuova conformazione della parte mobile. Innanzitutto viene calcolata la distanza che i loop devono colmare fra se stessi e i punti di aggancio della parte mobile, che avviene mediante la funzione $objective\_function$(4.5.2).

Una volta misurata la distanza da colmare, effettuando vari test, ho riscontrato che è possibile dare una mano alla convergenza dei loop, praticando delle rotazioni preliminari su un ampio spettro di angoli da $-180°$ a $180°$ con passo di campionamento 1. Ogni angolo viene provato per ciascun angolo torsionale. Per effettuare la singola rotazione utilizziamo la funzione $hillclimbing$(4.5.2). Una volta effettuata la singola rotazione e propagata all'interno del loop è necessario, se la distanza risultante è minore della pre-

Algoritmo 6 Esecuzione rotazioni tra loop e parte mobile

```
procedure ESECUZIONE_ROTAZIONI(ptmob)
    distanza_iniziale ← [0, 0]
    for loop in coil do
        distanza_iniziale[idx] ← OBJECTIVE_FUNCTION(loop, ptmob)
    angle_to_direction ← SUGGESTEDSETOFANGLE(180, 1)
    for a in angle_to_direction do
        for loop in coil do
            if (distanza_iniziale[idx] ≥ 0.1000) then
                res ← int(len(loop)) − 2
                prova ← HILLCLIMBING(loop, residui_costanti[idx], a, res)
                distance ← OBJECTIVE_FUNCTION(prova, ptmob)
                if (distance < distanza_iniziale[idx]) then
                    clash ← SEARCHNEIGHBOR(prova)
                    if not (clash) then
                        distanza_iniziale[idx] ← distance
                        storage(prova, idx)
            load(loop, idx)
    for i in range(iterazioni_massime) do
        for loop in coil do
            if not stop_loop[idx] then
                stop_loop[idx] ← STOPSEARCH
                if (distanza_iniziale[idx] ≥ 0.1000) then
                    n_res ← SUGGESTEDRESIDUE(len(loop))
                    angle ← SUGGESTEDANGLE(distanza_iniziale[idx])
                    swap ← SUGGESTEDCHANGE(distanza_iniziale[idx])
                    loop_new ← HILLCLIMBING(loop, residui_costanti[idx],
angle, n_res)
                    distance ← OBJECTIVE_FUNCTION(loop_new, ptmob)
                    if (distance < distanza_iniziale[idx]) then
                        clash ← SEARCHNEIGHBOR(prova)
                        if not (clash) then
                            distanza_iniziale[idx] ← distance
                            storage(prova, idx)
                    load(loop, idx)
    if distanza_iniziale[0] ≤ 0.1 and distanza_iniziale[1] ≤ 0.1 then
        return True
    return False
```

cedente si procede a verificare la presenza di clash per garantire la stabilità della proteina.

Tutto ciò avviene grazie alla funzione *searchneighbor*(4.5.2), che si avvale di ciò che è stato fatto durante il pre-processing descritto in precedenza. Se non vengono riscontrati clash, la rotazione viene considerata valida e vengono aggiornate le coordinate nelle strutture dati a supporto dell'esecuzione.

Una volta terminata la fase di applicazione di queste rotazioni canoniche, si passa alle iterazioni che lavorano di fino in base alla distanza del loop. Come prima cosa ci si chiede se questo loop è stato fermato nella sua progressione perché, se è cosi non ha senso continuare a iterare se almeno uno dei loop non può raggiungere la meta. Se il loop non è ancora stato fermato verifichiamo che al giro successivo non debba essere fermato mediante la funzione *stopsearch*. Questa funzione agisce sulle distanze raggiunte nel corso delle iterazioni dal loop e calcola un indice di incremento tra le ultime 20 distanze raggiunte. Questo indice è sicuramente negativo perché la distanza decresce, se però è maggiore di -0.01 significa che la distanza raggiunta nel corso delle 20 distanze ha una differenza che è troppo piccola per proseguire il lavoro. Ovviamente ho eseguito vari test per trovare la soglia e il numero di distanze da prendere in considerazione ed ho cercato il miglior compromesso per evitare di fermare troppo presto il cammino del loop.

Se il loop non è stato fermato allora devono essere determinati alcuni parametri: il residuo su cui lavorare, che viene determinato in modo casuale; l'angolo con cui provare la successiva rotazione determinato dalla funzione *suggestedangle*(4.5.2); lo swap che interviene per cercare di smuovere il processo di convergenza del loop nel caso si raggiunga un "asintoto". Nel caso in cui venga richiesto uno swap, si effettuano delle rotazioni di $-0.5°$ e $0.5°$ sulla base del loop per smuovere il processo. Una volta effettuata la singola rotazione e propagata all'interno del loop è necessario, se la distanza risultante è minore della precedente, verificare la presenza di clash per garantire la stabilità della proteina.

Le iterazioni possono terminare per i seguenti motivi:

- entrambi i loop raggiungono la distanza minima quindi la funzione restituisce l'avvenuta convergenza;
- uno dei loop è stato fermato durante il processo di convergenza, quindi la funzione restituisce che la convergenza non è avvenuta;
- uno dei due loop non ha colmato la distanza, quindi la convergenza non è avvenuta.

Funzione obbiettivo

La funzione obiettivo, 7, viene utilizzata per calcolare la distanza tra l'ultimo residuo del loop e la parte di attacco della parte mobile corrispondente. La distanza viene misurata mediante due contributi: il primo deriva dalla distanza euclidea dei due atomi carbonio α; il secondo contributo è dato dal prodotto scalare dei due vettori tra gli atomi carbonio e azoto dell'ultimo residuo della parte mobile e del loop.

É necessario considerare il prodotto scalare tra i due vettori, poiché è necessario che i due residui interagiscano nel modo corretto; infatti, i due atomi carbonio α potrebbero essere a contatto, ma i due atomi di carbonio e azoto potrebbero essere opposti e non favorire il legame tra i due residui.

Algoritmo 7 Objective Function

function OBJECTIVE_FUNCTION($loop, ptmob$)
$residuo\_loop \leftarrow loop[resiude]$
$residuo\_ptmob \leftarrow ptmob[residue]$
$distanzaCA \leftarrow residuo\_loop[CA] - residuo\_ptmob[CA]$
$asse\_loop \leftarrow residuo\_loop[N] - residuo\_loop[C]$
$asse\_loop \leftarrow normalizza(asse\_loop)$
$asse\_ptmob \leftarrow residuo\_ptmob[N] - residuo\_ptmob[C]$
$asse\_ptmob \leftarrow normalizza(asse\_ptmob)$
$prodotto\_scalare \leftarrow \langle asse\_loop, asse\_ptmob \rangle$
$product \leftarrow (\sqrt{1 - prodotto\_scalare}/2)$
return $distanzaCA \cdot (1 + product)$

Hillclimbing

La funzione *hillclimbing*, 8, viene utilizzata per applicare il movimento ad un residuo e poi propagarlo al resto del loop. A seconda dell'angolo torsionale estratto necessitiamo di generare una matrice di rotazione differente. Nel caso 0, ovvero ϕ, costruiamo la matrice di rotazione sull'asse degli atomi Cα-N; nel caso 1, ovvero ψ, costruiamo la matrice di rotazione sull'asse degli atomi Cα-C. Una volta calcolata la matrice la applichiamo ai residui a partire da quello scelto in modo randomico in precedenza. E' necessario controllare nel caso del loop-A di non applicare la rotazione in alcuni particolari residui che fanno parte del β-foglietto rigido.

Algoritmo 8 Hillclimbing

```
function HILLCLIMBING(loop, angolo, idx_rex)
    torsional_angle ← random(0, 1)
    loop_new ← []
    if torsional_angle == 0 then
        asse ← loop[idx_rex][CA] − loop[idx_rex][N]
        asse ← normalizza(asse)
        rotation_matrix ← matrice(asse, angolo)
    else
        asse ← loop[idx_rex][CA] − loop[idx_rex][C]
        asse ← normalizza(asse)
        rotation_matrix ← matrice(asse, angolo)
    for i in range(0, idx_rex) do
        loop_new ← loop[i]
    for i in range(idx_rex, len(loop)) do
        loop_new ← ruota(loop[i])
    return loop_new
```

SearchNeighbor

La funzione *searchneighbor*, 9, si occupa di verificare la presenza di clash, ovvero interazioni tra atomi come descritto in 1.1.5. In questo caso è necessario verificare sia che siano presenti clash tra i due loop, poiché sono a stretto contatto, e sia che siano presenti con gli altri atomi presenti nella proteina. L'unico modo per verificare la presenza di un clash è quella di controllare che la distanza euclidea tra due atomi sia maggiore rispetto alla somma dei rispettivi raggi di van der Walls.

Algoritmo 9 Searchneighbor

```
1: procedure SEARCHNEIGHBOR(residuo)
2:     res_linked ← []
3:     if residue.id[1] ∈ side_chain_processed[idx_loop] then
4:         res_linked ← side_chain_processed[idx_loop][residue.id[1]]
5:     for atom in residue do
6:         if atom.id in res_linked then
7:             for atom_check in res_linked[atom.id] do
8:                 d ← (atom_check − atom)
9:                 r ← raggio(atom_check) + raggio(atom)
10:                if d < r then
11:                    return True
12:        coordinate ← atom.get_coord()
13:        key ← (mx, my, mz)
14:        six_coord ← catch_coordinates(key)
15:        for coord in six_coord do
16:            for element in tridimensional_clash[coord] do
17:                chain ← element['chain']
18:                atom_to_analize ← element['atom']
19:                if not same(atom, chain, residue) then
20:                    d ← (atom_to_analize − atom)
21:                    r ← (raggio(atom_to_analize) + raggio(atom)
22:                    if d < r then
23:                        return True
24:    return False
```

Suggestedangle

La funzione *suggestedangle*, 10, mi permette di determinare un angolo a partire dalla distanza residua. Questa funzione mi permette di avere un angolo piccolo quando la distanza è molto piccola e un angolo abbastanza grande quando la distanza è elevata(Fig. 4.3).

Algoritmo 10 SuggestedAngle

```
function SUGGESTEDANGLE(distanza)
    angle ← 0.1
    if distanza ≥ 0.5 then
        angle ← (2^(distanza^−0.1) · distanza/6)
    return angle
```

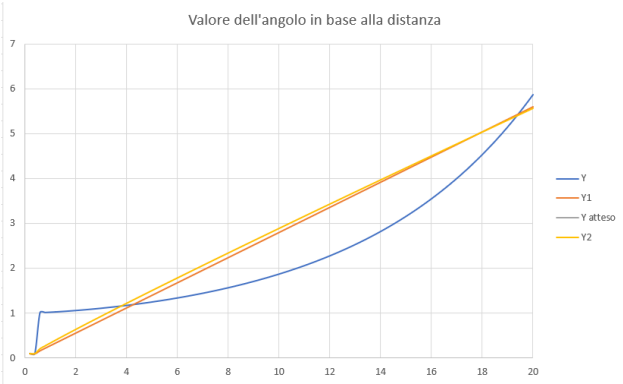


Figura 4.3: La funzione Y2 è quella che mi permette di descrivere meglio il comportamento richiesto.

Capitolo 5

Risultati

I risultati ottenuti utilizzando questo framework ci portano ad effettuare alcune riflessioni.

Innanzitutto, analizziamo lo spazio di ricerca considerato. Come si può dedurre da Fig. 5.1 viene ristretto lo spazio di ricerca delle possibile nuove soluzioni il più possibile per non cadere in percorsi troppo complessi. I due centroidi, colorati di rosso, ci dicono dove sono posizionate le parti mobili nello spazio centrate intorno al punto (0,0,0). Tutto sommato lo spazio di ricerca rimane ampio, però per come vengono penalizzati i singoli archi quando cercano di prendere un percorso che si allontana dal segmento ideale che passa in mezzo ai due centroidi riusciamo a mantenere lo spazio esplorato contenuto.

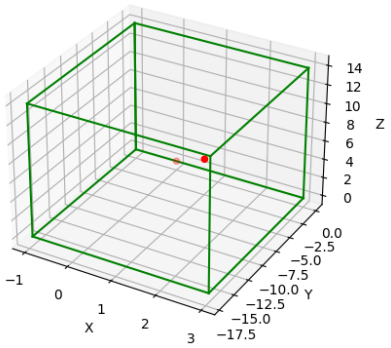
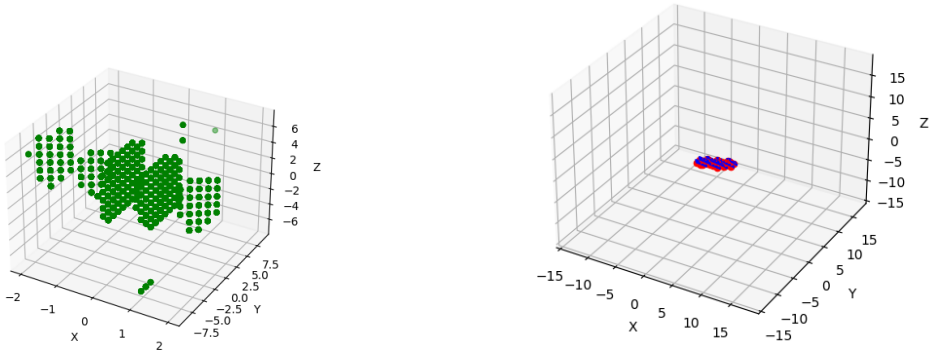


Figura 5.1: In rosso sono riportati i centroidi delle due configurazioni che stiamo analizzando, mentre in verde evidenziamo il parallelepipedo che racchiude lo spazio di ricerca.

Idealmente esiste come percorso ottimo quello che appunto segue il segmento tra i due centroidi. Infatti testando la fattibilità del percorso abbiamo ottenuto il seguente risultato Fig. 5.2.



(a) *Vicinato d'esplorazione* (b) *Percorso minimo pratico*

Figura 5.2: In (**a**) viene mostrato il vicinato d'esplorazione; mentre in (**b**) viene mostrato il percorso pratico minimo.

Come è possibile notare in effetti non ci si allontana molto dalla segmento ideale che connette i due centroidi, l'unica differenza presente è l'evoluzione a scaletta data dall'uso del vicinato semplice, come visto in Fig. 4.1(**a**).

Si può notare anche l'evoluzione del vicinato di esplorazione, prodotto da shortest path, si aggira intorno a quello che è il segmento teorico che unisce i due centroidi a riprova della corretta valutazione del costo per ogni nodo.

Finora è stata dimostrata la fattibilità teorica e pratica di convergere dalla conformazione chiusa alla conformazione aperta, il tutto senza verificare che ogni nuova parte mobile generata fosse raggiungibile dai loop. Prima di verificare come cambiano i risultati descritti in precedenza cercando convergenza anche con i loop è necessario esaminare come ogni loop crei un collegamento stabile con la nuova configurazione intermedia.

Conclusione

Conclusione che riassume il lavoro svolto ed eventuali lavori futuri.

Bibliografia

[Abraham and Leo, 1987] Abraham, D. J. and Leo, A. J. (1987). Extension of the fragment method to calculate amino acid zwitterion and side chain partition coefficients. *Proteins: Structure, Function, and Bioinformatics*, 2(2):130–152.

[Alexa, 2022] Alexa, M. (2022). Super-fibonacci spirals: Fast, low-discrepancy sampling of so(3). In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8291–8300.

[David L. Nelson, 2014] David L. Nelson, M. M. C. (2014). *I principi di biochimica di Lehninger*. Zanichelli.

[di Sanità,] di Sanità, I. S. Tutto sulla pandemia di sars-cov-2. Technical report.

[Duan Liangwei,] Duan Liangwei, Zheng Qianqian, Z. H. N. Y. L. Y. W. H. The sars-cov-2 spike glycoprotein biosynthesis, structure, function, and antigenicity: Implications for the design of spike-based vaccine immunogens.

[Eugene Kellogg and Abraham, 2000] Eugene Kellogg, G. and Abraham, D. J. (2000). Hydrophobicity: is logpo/w more than the sum of its parts? *European Journal of Medicinal Chemistry*, 35(7):651–661.

[Federica Agosta,] Federica Agosta, Glen E. Kellogg, P. C. From oncoproteins to spike proteins: the evaluation of intramolecular stability using hydropathic force field.

[Glen E. Kellogg, 1991] Glen E. Kellogg, Simon F. Semus, D. J. A. (1991). Hint: A new method of empirical hydrophobic field calculation for comfa. *Journal of Computer-Aided Molecular Design*, 5(6):545–552.

[Jeff Chang,] Jeff Chang, Brad Chapman, I. F. T. H. M. d. H. P. C. T. A. E. T. B. W. *Biopython Tutorial and Cookbook*. Open Bioinformatics Foundation (OBF), Lyon, France.

[L. alessandrini, 2004] L. alessandrini, L. (2004). *Geometria A*. Uninova.

[Laha et al., 2020] Laha, S., Chakraborty, J., Das, S., Manna, S. K., Biswas, S., and Chatterjee, R. (2020). Characterizations of sars-cov-2 mutational profile, spike protein stability and viral transmission. *Infection, Genetics and Evolution*, 85:104445.

[Salleh et al., 2021] Salleh, M. Z., Derrick, J. P., and Deris, Z. Z. (2021). Structural evaluation of the spike glycoprotein variants on sars-cov-2 transmission and immune evasion. *International Journal of Molecular Sciences*, 22(14).

[Tinelli,] Tinelli, M. La storia del sars-cov-2.

[Valério M,] Valério M, Borges-Araújo L, M. M. L. D. S. C. Sars-cov-2 variants impact rbd conformational dynamics and ace2 accessibility.

Ringraziamenti

In conclusione, voglio dedicare un pensiero a tutti coloro che mi hanno accompagnato in questo percorso.

Un ringraziamento speciale al Professor Alessandro Dal palù che mi ha assistito durante tutto questo percorso e mi ha permesso di raggiungere questo traguardo. La ringrazio non solo per la cordialità e la pazienza che ha dimostrato, ma anche per avermi concesso la possibilità di mettere in pratica ciò che ho studiato in questi anni lavorando al progetto "Carriere studenti". Voglio ringraziare anche il Professor Pietro Cozzini e la Dott.sa Federica Agosta per la disponibilità e l'impegno messo in questo piccolo progetto.

Un grazie immenso alla mia famiglia perché se tutto ciò sta avvenendo è anche grazie a voi. Mamma scusami per tutte le volte che ti risposto male e trattato peggio forse non sono molto bravo a gestire la pressione, però senza di te non ce l'avrei potuta fare. Papà se sono un minimo determinato a raggiungere i miei obbiettivi è grazie a te che mi hai insegnato a non mollare mai e non dare nulla per scontato, spero di continuare cosi e non mollare mai come fai tu. Let'z ci sarebbero troppe cose da dire, ma per qualsiasi cosa ti servirà ci sarò sempre, in fin dei conti siamo fratelli, però quanta pazienza che ci vuole.

Alexa, da un anno e mezzo a questa parte i nostri cammini si sono intrecciati e fino ad ora tra alti e bassi è stato bellissimo; speriamo di poter condividere ancora tante altre avventure insieme. Grazie per essermi stata accanto, anche se so che non è stato facile, e cercherò a mia volta di sostenerti in qualsiasi percorso tu intraprenderai come hai fatto con me.

Grazie a tutti i miei amici, quelle persone che ci sono sempre e che sempre ci saranno. Voglio però citare alcune persone in particolare con cui sono più legato nell'ultimo periodo, nonostante siate tutti molto importanti. Partendo da questi sette scappati di casa (Francesco, Nicolò, Mirko, Diego, Michele, Gigi e Mirco) il cui unico obbiettivo è vandalizzarmi casa se salto un uscita. Siete però degli amici, nel vero senso della parola, che non ti fanno mai mancare il supporto anche nei momenti più infelici. Ci sono però anche tanti bei momenti e notti magiche che non si possono dimenticare.

Fede, a te va un ringraziamento speciale per avermi ascoltato e tante volte risolto problemi che dal mio punto di vista sembravano insormontabili, la mia collega preferita.

Martina, la mia seconda sorella, sempre pronta a dedicarti un sorriso e che porta sempre un pizzico di allegria in più. Poi che dire hai passato un intero pomeriggio con me a guardare i legami covalenti questo dice già tutto, per non parlare di quella bellissima paperella gialla.

Un ringraziamento speciale va anche a tutte quelle persone con cui ho condiviso questo percorso sia la triennale che la magistrale, abbiamo condiviso tanto tempo insieme e tanta sofferenza, manteniamo i rapporti perché siamo veramente un bel gruppo. Ci sono poi gli amici di una vita per cui non ci sono parole e anche se ultimamente passiamo meno tempo insieme, non siete assolutamente meno importanti.

Voglio ringraziare infine anche "the best English class" (Angy, Sofy, Giulia, Alice e Diego) che mi avete sempre ascoltato durante tutte le lezioni e un grazie speciale anche alla mia "favorite teacher" Kim che mi sta insegnando l'inglese con tanta pazienza perché diciamocelo non sono uno studente modello e si sa i compiti di inglese non vorrebbe farli nessuno.

Grazie!!.