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
Federica Agosta

Anno Accademico 2021–2022

Dedica

Indice

Introduzione **1**

1 Background **3**

Background **3**
1.1 Amminoacidi 3
1.1.1 Catena laterale 5
1.2 Proteine 5
1.2.1 Classificazione 6
1.2.2 biochimica 7
1.2.3 Composizione 8
1.2.4 Struttura 8
1.3 Principio di Ramachandran 12
1.4 Ricerca Locale 13
1.4.1 Tipologie di ricerca locale 15
1.4.2 Euristiche 16
1.5 Hydropathic INTeractions HINT 17
1.6 Shortest Path 18
1.6.1 Algoritmo A* 19
1.6.2 Esempio 19

2 Covid **21**

Covid **21**
2.1 Covid-19 21
2.1.1 Storia 22
2.1.2 Caratteristiche genetiche 23
2.1.3 RNA polimerasi 24
2.2 Glicoproteina Spike 25
2.2.1 Struttura della proteina e funzione 26
2.2.2 Scudo di glicani della glicoproteina spike 28

2.3 RBD 28

3 Obbiettivi **31**

4 Progettazione **33**

5 Risultati **51**

Conclusione **53**

Bibliografia **55**

A Appendice di Esempio **59**

Elenco delle figure

1.1 Esempio di amminoacido 4
1.2 Struttura dell'amminoacido 4
1.3 Esempio del grafico di Ramachandran 12

2.1 Molecola di un coronavirus 23
2.2 Principio del RNA polimerasi 25
2.3 Struttura di massima della glicoproteina spike 27
2.4 Struttura del dominio di legame del recettore SARS-CoV-2, nella conformazione aperta (A) e chiusa (B). (C) rappresenta la struttura legata ad ACE2 29

Elenco degli algoritmi

Elenco delle tabelle

1.1 Classi di amminoacidi 5

Introduzione

L'introduzione deve contenere un riassunto del lavoro di Tesi. In particolare bisogna esprimere chiaramente e molto sinteticamente: contesto dello studio, motivazioni, contributo e conclusioni. Bisogna quindi fare un sommario dello studio ad alto livello, fornendo le intuizioni senza ricadere in dettagli tecnici. Anche lo stile dovrebbe essere più discorsivo rispetto alle parti tecniche della tesi.

Capitolo 1

Background

In questo capitolo verranno introdotti i concetti di base utili alla comprensione del contesto. Andremo ad introdurre cosa sono le proteine e quali sono i loro componenti principali.

1.1 Amminoacidi

Gli amminoacidi sono una categoria di composti organici che hanno sia il gruppo funzionale amminico (-NH_2), sia quello carbossilico (-COOH). La parola aminoacido deriva quindi proprio dall'unione dei due gruppi funzionali citati prima. Siccome sono presenti contemporaneamente un gruppo acido (carbossilico) e un gruppo basico (amminico), sono definite molecole anfotere. Anfotere sono sostanze chimiche che possono manifestare sia un comportamento acido che uno basico.

In biochimica, ci si riferisce di solito ad un sottogruppo dei seguenti, ovvero gli $L-\alpha-amminoacidi$, ovvero amminoacidi il cui gruppo amminico e carbossilico sono legati allo stesso atomo di carbonio, chiamato appunto α e la loro configurazione è ad L, ovvero il gruppo amminico si troverà sempre alla sinistra del carbonio α. Sono presenti 22 $L-\alpha-amminoacidi$ che costituiscono la struttura delle proteine, anche detti amminoacidi proteinogenici. Oltre hai al gruppo carbosillico e al gruppo aminico, ogni amminoacido si contraddistingue dagli altri per la presenza di un residuo R, conosciuto anche con il nome di catena laterale.

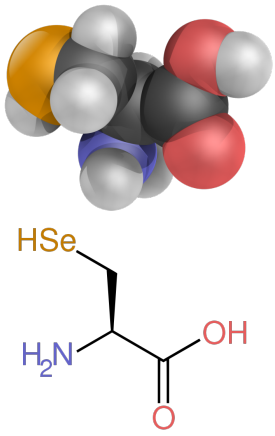

Figura 1.1: Esempio di amminoacido

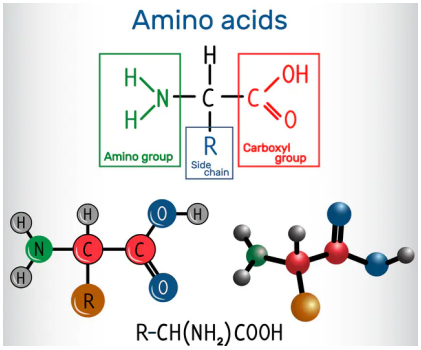

Figura 1.2: Struttura dell'amminoacido

1.1.1 Catena laterale

La catena laterale gioca degli amminoacidi gioca un ruolo importante per la determinazione delle proprietà delle proteine. Esiste una vasta diversità nelle proprietà chimiche delle catene laterali degli amminoacidi, tuttavia essi possono essere raggruppati in 6 classi differenti.

Tipo di catena laterale	`Amminoacidi`
Alifatica	`Glicina, alanina, valina, leucina, isoleucina`
Contenente idrossile o solfuro	`Serina, cisteina, treonina, metionina`
Aromatica	`Fenilalanina, tiroxina, triptofano`
Basica	`Istidina, lisina, arginina`
Acido e la sua ammide	`Acido aspartico, acido glutammico, asparagina, glutammina`
Ciclica	`Prolina`

Tabella 1.1: Classi di amminoacidi

La prolina non può essere inserita in una qualsiasi classe perché è ciclica. La prolina condivide la maggior parte delle proprietà con i gruppi alifatici. La rigidità dell'anello gioca un ruolo cruciale nella struttura delle proteine. Come già detto, gli amminoacidi sono i mattoni di costruzione delle proteine e la metà di questi sono anche essenziali per l'essere umano, poiché non è in grado di produrseli da soli.

Gli amminoacidi in azione combinata o in azione singola sono alla base di molte attività presenti nel nostro corpo.

1.2 Proteine

A livello chimico, le proteine non sono altro che macromolecole biologiche costituite da catene di amminoacidi legate insieme da un legame peptidico. Il legame peptidico o giunto peptidico è un legame covalente che unisce il gruppo (-NH_2) 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.

A differenza di altre macromolecole biologiche come i polisaccaridi e gli acidi nucleici, le proteine sono essenziali negli organismi viventi perché prendono parte a praticamente tutti i processi che avvengono nelle cellule. La maggior parte appartiene alla categoria degli enzimi, che caratterizzano le reazioni biochimiche vitali per il metabolismo degli organismi. Hanno anche funzioni strutturali o meccaniche nei muscoli e che costituiscono il citoscheletro. Alcune sono fondamentali per l'invio di segnali inter ed intracellulari e nella difesa immunitaria. Una volta che sono sintetizzate all'interno del organismo, esistono per un periodo di tempo limitato per poi esser degradate e riciclate attraverso meccanismi cellulari.

Le proteine possono essere purificate da altri componenti cellulari e utilizzando tecniche come: l'ultracentrifugazione; la precipitazione; l'elettroforesi; la cromatografia. L'avvento dell'ingengneria genetica ha portato a nuove tecniche che ne facilitano la purificazione.

Una catena lineare di residui amminoacidi è chiamata polipeptide, ed una proteina generalmente è costituita da uno o più polipeptidi lunghi eventualmente coordinati a gruppi non peptidici, chiamati prostetici o cofattori. I polipeptidi che contengono meno di 20/30 amminoacidi non vengono quasi mai considerati proteine, ma più spesso chiamati peptidi. La sequenza degli amminoacidi in una proteina è definita della sequenza presente nel gene a sua volta codificata nel codice genetico, solitamente ne sono specificati 20 ma possono essere di più in alcuni organismi.

Le proteine che hanno stesso numero e tipo di amminoacidi possono differire per come vengono posti all'interno della struttura, anche una singola variazione può portare ad una variazione nella struttura tridimensionale della macromolecola che può rendere la proteina non funzionale.

1.2.1 Classificazione

Durante l'evoluzione ci sono stati duplicamenti di geni e alterazioni della funzione di una proteina che portano tutt'ora ad avere circa 500 famiglie proteiche. All'interno della stessa famiglia le proteine svolgono funzioni leggermente diverse, ma per quanto riguarda la loro composizione a livello di sequenza di amminoacidi è quasi identica. Chiaramente ci sono anche casi in cui le proteine all'interno della stessa famiglia si differiscono a livello di sequenza di amminoacidi, ma hanno una conformazione tridimensionale molto simile.

Si afferma quindi che nel corso dell'evoluzione si sia più conservata la conformazione tridimensionale, piuttosto che la sequenza. Si può dire che due proteine hanno la stessa struttura generale quando almeno un quarto della loro sequenza amminoacidi corrisponde. Si dice invece che due proteine hanno un qualche grado di parentela, se almeno il 30% degli amminoacidi corrisponde. Alcune proteine possono anche essersi formate per rimescolamento dei domini proteici o per la duplicazione all'interno della proteina stessa con unioni accidentali di DNA.

La classificazione può essere ottenuta grazie alla composizione chimica, alla configurazione molecolare o alla solubilità. Ci sono quindi proteine semplici che sono costituite da soli amminoacidi e proteine coniugate composte dalla proteina semplice e da un gruppo prostetico non proteico.

Tra le proteine semplici vi sono le proteine fibrose, tendenzialmente non solubili nei solventi acquosi e poco attaccabili dagli enzimi proteolitici, inoltre sono presenti le proteine globulari. Mentre per quanto riguarda le proteine coniugate troviamo l'emoglobina, le clorofille e le opsine.

Si possono poi classificare le proteine in base alla funzione che compiono, ci sono le proteine strutturali che sono componenti delle strutture permanenti e che hanno funzione meccanica, poi trovano posto le proteine di trasporto che prendono le sostanza poco idrosolubili e ne consentono il trasporto nell'organismo e poi trovano posto gli enzimi che sono proteine catalitiche. A queste funzioni che abbiamo brevemente descritto si aggiugono la regolazione dell'espressione dei geni, la duplicazione, trascrizione e traduzione del DNA, la regolazione delle reazioni metaboliche, la generazione e la ricezione degli impulsi nervosi.

1.2.2 biochimica

La stra grande maggioranza delle proteine sono costituite da polimeri lineari combinando 20 diversi $L - \alpha - amminoacidi$. Tutti gli aminoacidi che possono essere impiegati nella costruzione di proteine hanno una struttura comune, un carbonio α con un gruppo amminico, un gruppo carbosillico e una catena laterale variabile a seconda dell'aminoacido, l'unica che si distingue è la prolina che contiene un anello insolito al gruppo amminico. Le catene laterali degli amminoacidi hanno ognuna la loro struttura e le loro proprietà chimiche, l'effetto combinato delle catene laterali determina la struttura tridimensionale e la reattività chimica della proteina. Una volta collegati tramite legame peptidico gli amminoacidi sono chiamati residui e la serie di carbonio, azoto e atomi d'ossigeno è nota come catena principale o backbone.

Il legame peptidico ha due forme di risonanza che contribuiscono al doppio legame e non permettono la rotazione attorno al suo asse, in modo tale che i carbonio α dei vari residui siano pressoche complanari. Ci sono però altri due angoli nel legame peptidico che ne determinano la forma assunta.

1.2.3 Composizione

Uno dei dogmi fondamentali della biologia è che ad ogni struttura tridimensionale di una proteina via sia associata una specifica funzione biochimica. In questo modo le proteine possono essere classificate in due famiglie: le proteine globulari e le proteine a struttura estesa o fibrosa. Questa separazione riflette anche una separazione funzionale:

- le proteine estese o fibrose svolgono funzioni biomeccaniche quindi fornendo sostegno strutturale;
- le proteine globulari sono coinvolte in molteplici e specifiche funzioni biologiche e sono di fondamentale importanza per l'economia cellulare.

Una proteina è formata da uno o più polipeptidi, che sono molecole con più di 10 unità di amminoacidi, eventualmente accompagnati o legati da uno o più gruppi prostetici. Una proteina attiva può esistere solo in soluzioni salina diluita e la sua struttura dipendera esclusivamente dalle caratteristiche chimico-fisiche della soluzione in cui è inserita, che può anche determinare modifiche strutturali e alterare le sue propieta funzionali.

La molecola proteica risulta costituita da atomi di carbonio, ossigeno, idrogeno e azoto, può contenere anche zolfo e, talvolta, fosforo e/o metalli.

1.2.4 Struttura

Mettiamo un sunto sensato delle sottosottosezioni....

Ripiegamento

Prendendo una proteina, ovvero una macromolecola formata da decina di migliaia di atomi, potrebbe assumere un numero di ripiegamenti elevati; tuttavia, non è cosi perché ci sono considerazioni fisiche che limitano la maggior parte dei ripiegamenti. Gli atomi non possono sovrapporsi e il loro comportamento è da immaginarsi come delle semplici sfere, con un certo raggio detto raggio di van der Waals. Ciascun amminoacido contribuisce alla formazione della catena con tre possibili legami:

- legame peptidico (C-N) tra il carbonio di un amminoacido e l'azoto di un amminoacido adiacente;

- legame convezionalmente chiamato Cα-C che è presente nei due carboni della catena principale del singolo amminoacido;

- legame Cα-N all'interno dello stesso amminoacido.

Il legame peptidico è planare e non consente alcuna rotazione, mentre gli altri due legami possono ruotare e definiscono due angoli: l'angolo di rotazione del legame Cα-C è detto ψ; l'angolo di rotazione del legame Cα-N è φ. La conformazione degli atomi che fanno parte della catena principale è determinata dagli angoli descritti in precedenza. Non è pero possibile ruotare come si vuole questi angoli dato che non sono possibili collisioni steriche tra gli amminoacidi. Ramachandran come vedremo nella prossima sezione nel dettaglio ha individuato e rappresentato in un grafico le coppie di angoli di rotazione a seconda delle coppie di atomi. Dal grafico si può vedere che le proteine assumono due grandi tipologie di conformazione: l'$\alpha - elica$ e il $\beta - foglietto$.

Tra gli atomi che sono all'interno di una proteina si stabiliscono dei legami che possono essere covalenti o non covalenti, i legami non covalenti sono sicuramente meno potenti, tuttavia il numero all'internoo di una proteina li rende fondamentali per comprendere il ripiegamento. Ci sono tre tipi di legami non covalenti:

- il legame idrogeno, che si effettua tra un atomo di ossigeno e uno vicino ad idrogeno;

- le attrazzioni elettrostatiche che avvengono tra gruppi laterali con cariche periferiche opposte;

- le attrazzioni di van der Waals si verificano tra dipoli molecola istantanei indotti, tra dipoli permanenti o tra un dipolo permanente e uno corrisponde indotto, nelle quali entrano in gioco forze diverse.

Vanno aggiunte alle precendenti interazioni la tendenza dei gruppo di amminoacidi idrofobici ad avvicinarsi e unirsi tra loro, formando delle tasche idrofobiche che sono però lontane dai legami idrogeno sempre presenti in un ambiente acquoso. Tendenzialmente questi gruppi sono posti all'interno della proteina, mentre i suoi amminoacidi idrofili (polari e con carica) saranno tendenzialmente all'esterno, poiché essa si trova tipicamente in un

ambiente acquoso. Di solito la proteina tende poi ad assumere la struttura tridimensionale che ha la più bassa energia libera.

Le conformazioni più comuni

L'$\alpha - elica$ e il $\beta - foglietto$ sono le conformazioni più comuni riscontrabili nelle catene polipeptidiche di una proteina. Una singola proteina può prevedere sia $\alpha - elica$ che $\beta - foglietto$ in numero variabile.

L'$\alpha - elica$ è la più comune nelle proteine, in particolare nei recettori cellulari, e si possono trovare più $\alpha - elica$ per singola proteina. L'elica è una delle conformazioni più favorevoli perché riduce al minimo l'energia libera. Essa si forma quando una catena polipeptidica si ripiega su se stessa con formazione di legami idrogeno tra un legame peptidico e il quarto successivo, nel dettaglio tra il gruppo chetonico C=O dell'uno e il gruppo N-H dell'altro, e il legame è tra O e H. Tutti i gruppi amminici di un'elica sono rivolti verso l'N-terminale della proteina, tutti quelli chetonici verso il C-terminale, così l'elica assume parziale carica positiva all'N-terminale e parziale carica negativa al C-terminale.

Il $\beta - foglio$ pieghettato è la seconda conformazione più comune nelle proteine, si trova maggiormente in alcuni enzimi e nelle proteine coinvolte nella difesa immunitaria. Esso consiste in numerose catene polipeptidiche che si dispongono l'una adiacente all'altra, collegate in una struttura continua da brevi sequenze ad U.

Livelli di organizzazione

All'interno della proteina si possono distinguere vari livelli di organizzazione, che possono essere tre o quattro a seconda della tipologia della stessa. I livelli d'organizzazione sono: vspace10pt

- la struttura primaria è formata dalla sequenza specifica di amminoacidi, dalla catena peptidica e il numero delle catene determina anche il ripiegamento della stessa;

- la struttura secondaria consiste nella conformazione delle catene ($\alpha - elica$, $\beta - foglietto$, etc..). All'interno di una singola proteina vi può essere una combinazione una combinazione di varie tipologie di sequenze;

- il dominio è un'unità globulare o fibrosa formata da catene polipeptidiche ripiegate in più regioni compatte, costituiscono divisioni della

struttura terziaria, ha la caratteristica di ripiegarsi più o meno indipendentemente rispetto al resto della proteina. Molte delle proteine più complesse sono aggregazioni modulari di numerosi dominii proteici;

- la struttura terziaria, che dal punto di vista termodinamo è quella ccon la più bassa energia libera, è rappresentata dalla configurazione tridimensionale completa che la catena polipeptidica assume nell'ambiente in cui si trova;

- la struttura quaternaria deriva dall'associazione di due o più unità polipeptidiche, unite tra loro da legami deboli.

Le proteine contenenti una parte non polipeptidica sono anche dette coniugate. Due proteine possono essere definite isoforme se a parità di struttura primaria ci sono differenze in uno degli altri livelli di struttura. Quando si dice denaturare una proteina significa distruggerne la sua conformazione spaziale, anche se viene mantenuta intatta la struttura primaria, essa non è più in grado di esplicare la sua funzione.

Proteine complesse

Le proteine prese singolarmente sono si complesse, ma all'intermo degli organismi possono aggregarsi ad altre proteine identiche e non creando cosi dei complessi proteici. Tutto ciò avviene grazie ai legami non covalenti che permettono ad una proteina di assumere una determinata configurazione. Nella proteina sono presenti una o più zone capaci di interazioni covalenti e vengono chiamate sidi di legame. Quando le proteine si uniscono insieme in un complesso proteico, le singole unità vengono chiamate subunità proteiche.

Ci sono complessi proteici formati da molte subunità che permettono di realizzare filamenti, poiché in un polo possiedono un sito di legame e dall'altro una struttura proteica complementare allo stesso sito. Vi sono poi proteine la cui funzione è resa possibile proprio dalla loro struttura poco caratterizzabile e quasi casuale; esse principalmente hanno molte funzioni all'interno della cellula. La loro caratteristica è quella di avere ridondanza di amminoacidi e bassa presenza di amminoacidi idrofobici.

Ci possono essere casi in cui la proteina è esposta ad un alto livello di degradazione, esse sono quindi stabilizzate da legami di solfuro, che permettono di mantenere la sua conformazione.

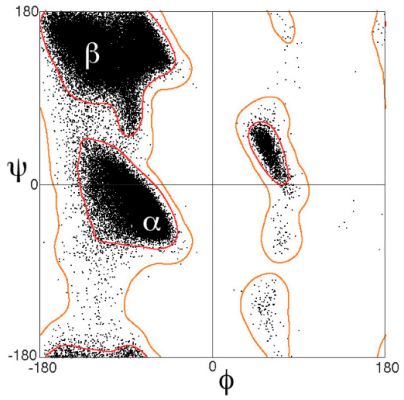

Figura 1.3: Esempio del grafico di Ramachandran

1.3 Principio di Ramachandran

Come detto nella sezione precedente Phi φ e Psi ψ determinano la conformazione di un polipeptide. Il principio di Ramachandran ci dice quali $\alpha - elica$, $\beta - foglietto$ e spire sono le conformazioni più probabili da adottare per una catena polipeptidica, poiché la maggior parte delle configurazioni sono impossibili a causa delle collisioni steriche tra gli atomi.

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

Il principio di Ramachandran viene utilizzato nella biologia strutturale per verificare la qualità delle predizioni di struttura proteica e per identificare eventuali errore nella modellizzazione. Viene anche utilizzato nella progettazione di proteine artificiali e nella modifica della struttura delle proteine esistenti per migliorare le proprietà terapeutiche o funzionali.

Non tutte le coppie di angoli teoricamente possibili sono realmente ottenibili all'interno delle strutture peptidiche in quanto impedite dalla presenza di ingombri sterici fra i gruppi delle catene laterali dei residui amminoacidici; le zone del grafico in cui tali contatti sterici sfavorevoli non sono presenti possono essere delimitate da linee di contorno, e possono essere associate ai motivi secondari assunti dalla catena. La conformazione complessiva del peptide è quindi definita assegnando i valori a ciascuna coppia di angoli ψi, φi per ogni amminoacido.

Nel caso della glicina, le zone di conformazioni possibili nel grafico presentano una simmetria centrale e sono molto più ampie rispetto a quelle degli

altri amminoacidi grazie alla simmetria del residuo ed alle piccole dimensioni dell'atomo di idrogeno legato in posizione α. Gli altri amminoacidi presentano invece un diagramma asimmetrico, con un insieme di zone indicanti le conformazioni ammesse meno esteso, e la cui ampiezza dipende soprattutto dalla presenza di gruppi stericamente ingombranti in posizione β. La presenza di catene laterali, anche lunghe, che non siano ramificate al carbonio β, come nel caso della leucina, ha invece una scarsa influenza e riduce solo di poco il dominio delle conformazioni permesse.

Un caso a parte si ha per la prolina, la cui struttura rigida rende possibili solo poche conformazioni in una porzione limitata del grafico.

1.4 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 migliori. 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 sistematica-

mente 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 mi-

gliore 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.

1.4.1 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.

1.4.2 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.

1.5 Hydropathic INTeractions HINT

La valutazione della stabilità intramolecolare di una proteina gioca un ruolo fondamentale nella comprensione del loro comportamento e nel 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 predirre 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 mutata può essere molto costoso in termini di tempo. La stabilità delle proteine mutate è influenzata dall'interazione intramolecolare, ovvero sono forze che permettono di creare e tenere insieme una struttura. Nel tentativo di analizzare la stabilità proteica, sono stati sviluppati molti metodi che svariano in diversi approcci dai meccanismi molecolari basati su force field fino a tecniche di machine learning. Tuttavia, questa pratica risulta essere molto costosa, specialmente quando sono presenti un elevato numero di mutazioni. Uno degli aspetti fondamentali su

cui è basato HINT è dovuta al fatto che la stabilità delle proteine mutate è dimostrato essere influenzata dalle interazioni intramolecolari. Le strutture native delle proteine sono relativamente 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). A differenza dei legami covalenti che legano gli atomi tra loro, le interazioni chimiche deboli non sono forti abbastanza per legare tra loro atomi isolati. Per questo 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.

HINT è stato sviluppato sulla base dei lavoro svolto da (indica il personaggio) che ha esteso il metodo fragment per la predizione dei $\log P\_o/v$ (coefficiente di partizione per il trasferimento di soluti in 1-ottanolo/acqua), cio permette di quantificare le interazioni idrofobiche e polari tra o all'interno dei molecole. 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.

1.6 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.

1.6.1 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.

1.6.2 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*

Capitolo 2

Covid

In questo capitolo verrà introdotto il virus covid, la funzione della proteina spike e vedremo una conformazione della stessa.

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*. Le specie patogene individuate nel tempo sono SARS-CoV, MERS-CoV e appunto SARS-CoV-2 di ordine nidovirale della famiglia coronaviridae e sotto famiglia ortocoronavirinae, e tutte hanno un aspetto a forma di corona solare. Dei 4 generi di coronavirus $(\alpha, \beta, \gamma, \delta)$ fa parte dei $\beta - CoV$ e mostra molte somiglianze con 2 coronavirus derivati dai pipistrelli.

SARS-CoV e MERS-CoV hanno avuto origine nei pipistrelli, e sembra che sia così anche per SARS-CoV-2. Era gia stata dimostrata in precedenza la possibilità che un host intermedio facilitasse l'emergere del virus negli umani; dopodiché il contagio tra uomo e uomo può avvenire attraverso contatti ravvicinati con goccioline respiratorie, oppure con con diretto con infetti e o per contatto con oggetti e superfici contaminate.

Il genoma del virus contiene 4 strutture proteiche: la proteina spike; la membrana; l'envelope; la nucleocapside. 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 envelope è la più piccola proteina appartenente alla struttura virilica, partecipa all'assemblaggio virale e al gemogliamento.

La proteina nucleocapside è l'unica che si lega al genoma del RNA ed è coinvolta nel assemblaggio virale e nel germogliamento.

La replicazione del virus inizia con l'attaccamento e l'ingresso; l'attaccamento del virus alla cellula ospite avviene tra la proteina spike e il recettore specifico. 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

Ha avuto inizio 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 sviluppo poi 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%.

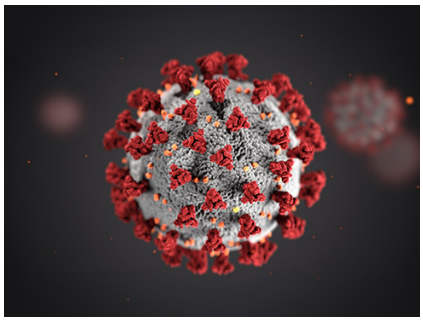

Figura 2.1: Molecola di un coronavirus

2.1.2 Caratteristiche genetiche

I coronavirus sono sferici con un diametro di circa 125nm con punte a forma di clava che sporgono dalla superficie del virus che danno l'aspetto di una corona solare.

All'interno dell'involucro troviamo una simmetria elicoidale dei nucleocapsidi, che in realtà è molto rara tra i virus a RNA a senso positivo. I CoV sono classificati nell'ordine Nidovirales, famiglia Coronaviriade e sotto famiglia Orthocoronavirinae. Nidovirales è un ordine di virus a RNA a singolo filamento positivo. Essi si distinguono da altri virus a RNA per la loro lunghezza e complessità del genoma e per la presenza di una struttura a corona di proteine sulla superficie virale. Il loro genoma contiene diversi geni che codificano proteine non strutturali coinvolte nella replicazione e nella trascrizione del virus nonché geni che codificano le proteine strutturali che costituiscono il virus stesso. Gli Nidovirales includono quattro famiglie virali:

- Coronaviridae,
- Arteriviridae,
- Roniviridae,
- Mesoniviridae.

La famiglia Coronaviridae è una famiglia di virus a RNA a singolo filamento positivo che causano malattie respiratorie e gastroenteriti in animali e in alcuni casi anche nell'uomo. I coronavirus sono stati scoperti per la prima volta negli anni '60, e devono il loro nome alla loro forma a corona, che deriva

dalle protuberanze sulla loro superficie virale. La famiglia Coronaviridae è suddivisa in quattro generi:

- Alphacoronavirus,
- Betacoronavirus,
- Gammacoronavirus,
- Deltacoronavirus.

Un'analisi filogenetica ha inserito SARS-CoV2 sotto il sottogenere Sarbecovirus del genere Betacoronavirus.

Le 4 proteine strutturali, citate in precedenza, sono richieste dalla maggior parte dei CoV per produrre una particella virale strutturalmente completa suggerendo che alcuni Cov possono codificare proteine aggiuntive con funzioni in sovrapposizione compensative.

2.1.3 RNA polimerasi

La fase di ingresso del virus viene approfondita nella prossima sezione, ora pensiamo a cosa succede quando è all'interno. Una volta all'interno della parte acquosa della cellula ospite, chiamata citosol, il virus si "scompone" rilasciando il suo contenuto; un insieme di proteine e materiale genetico. L'uomo conserva l'informazione genetica, ovvero quella che ci permette di costruire le cellule, nelle molecole di DNA, il virus utilizza una singola molecola di RNA. L'RNA è presente anche nell'uomo, ma che principalmente viene utilizzato per la costruzione delle proteine.

Una volta che il coronavirus ha infettato una cellula, il suo scopo è quello di far si che si costruiscano nuove copie dello stesso in modo da avere una discendenza in grado di espandere la specie. Tutto quello che è presente nella singola cellula infetta deve essere "copiato" e assemblato per formare nuovi vironi. Per ottenere la "copia" è necessario replicare il materiale genetico "RNA" del virus stesso. Il meccanismo che permette di far ciò si chiama RNA polimerasi, ovvero un enzima che è in grado di formare lunghe catene di RNA. In modo specifico l'RNA polimerasi di SARS-CoV-2 si chiama RdRP (RNA polimerasi RNA-dipendente).

Esso viene però definito "distratto", infatti ogni volta che effettua la "copia" del materiale genetico originale commette degli errori, di conseguenza la copia presenta delle differenze. Le seguenti differenze non sono altro che mutazioni che si ripercuotono sulla struttura delle proteine del virus e sulla

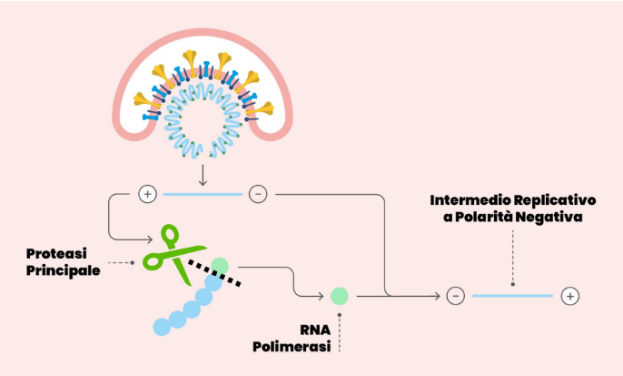

Figura 2.2: Principio del RNA polimerasi

loro funzione. Non a caso si sono sentite varie varianti presenti come Delta, Omicron, etc... Quello che succede è che l'RNA polimerasi ha fatto una copia del genoma virale e ha cambiato una base di RNA. In ogni caso le mutazioni hanno conseguenze sul virus, possono essere neutre, vantaggiose o svantaggiose. Una mutazione può iniziare a circolare se essa da un vantaggio per il suo ciclo vitale, questo può significare essere più abile nell'infettare, ma provocare meno danni nell'organismo ospite, dato che un virus altamente letale è destinato a scomparire per mancanza di ospiti.

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. La fusione della membrana viene mediata dalla membrana di tipo 1 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.

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. Essa è un trimero fortemente glicosilato. La subunità S1 è composta da 672 amminoacidi ed organizzata in 4 domini: un dominio N-terminale; un dominio C-terminale, noto anche come dominio di legame; due sottodomini SD1 e SD2. Mentre la subunità S2 è composta da 588 amminoacidi e contiene un peptide di fusione idrofobica N-terminale, due ripetizioni eptade, un dominio transmembrana e una coda citoplasmatica.

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 coformazionale 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.

L'architettura di massima dell'ectodominio pre fusione della spike è stabilizzato da due mutazioni consecutive della prolina in due conformazioni determinate dalla microscopia crioelettronica a singola particella è un trimero con

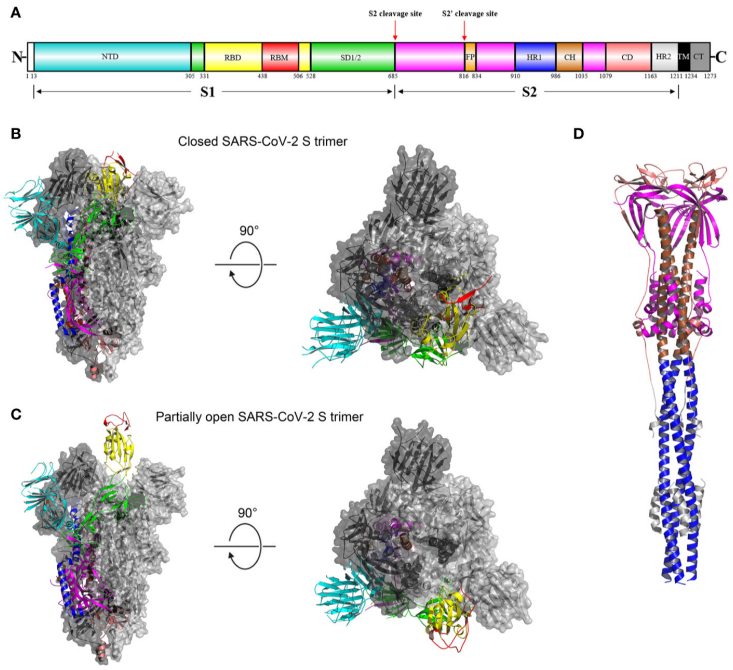

Figura 2.3: Struttura di massima della glicoproteina spike

una sezione trasversale triangolare. 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. **??**B). 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. **??**C). 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.

Le strutture di pre fusione del coronavirus umano senza due mutazioni consecutive della prolina rivelano solo una conformazione completamente chiusa. 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 pensa-

re 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 incontrati 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.

Quando i glicani specifici sono mappati sulla struttura di pre fusione dell'ectodominio della spike del SARS-CoV-2 il modello ha mostrato livelli sostanzialmente più elevati di superficie priva di glicani. Questo ci porta alla considerazione che la proteina spike del SARS-CoV-2 è ricoperta da uno scudo meno denso e quindi risulta essere una buona notizia per i potenziali vaccini.

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

Diverse linee di ricerca hanno stabilito che l'enzima di conversione dell'angiotesina 2 (ACE2) è un recettore di ingresso per SARS-CoV-2. Interazioni

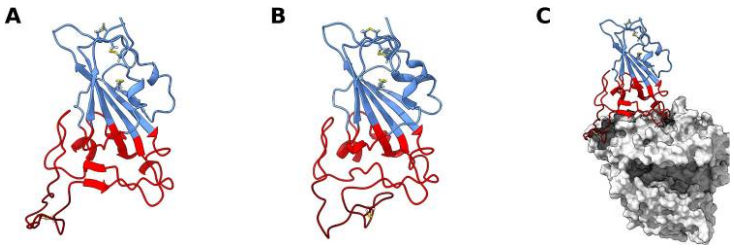

Figura 2.4: Struttura del dominio di legame del recettore SARS-CoV-2, nella conformazione aperta (A) e chiusa (B). (C) rappresenta la struttura legata ad ACE2

dettagliate tra il SARS-CoV-2 RBD e il suo recettore sono state rivelate da diverse strutture in ACE2. Come detto nella sezione precedente le subunità S1 e S2 sono responsabili del legame del recettore e della fusione della membrana. La 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.

Questa struttura centrale del foglio β è ulteriormente stabilizzata da 3 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 interazioni 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.

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

Capitolo 4

Progettazione

Il seguente è lo pseudo codice che guida l'algoritmo della mia ricerca locale

```
parser = PDBParser(PERMISSIVE = True, QUIET = True)
title = '6vxx_chiusa.pdb'
chain_of_interest = 'B'
idx_chain_of_interest = -1
covid_chiusa = parser.get_structure('STS', './covid_pdb/'+title)
covid_loop = parser.get_structure('STS', './covid_pdb/6vxx_chiusa_loop.pdb')
model = covid_loop.get_models()
models = list(model)

### Parte di identificazione dei loop
for model_chain in models:
    list_chain_loop = list(model_chain.get_chains())
    for id in range(len(list_chain_loop)):
        if (list_chain_loop[id].id == chain_of_interest):
            idx_chain_of_interest = id
    coil = []
    loop = identify_loop(list_chain_loop, idx_chain_of_interest, costanti_loopA)
    coil.append(loop)
    loop = identify_loop(list_chain_loop, idx_chain_of_interest, costanti_loopB)
    loop_r = loop[::-1]
    coil.append(loop_r)


coil_dict_coordinates = [{"loop": x, "residui":[{"residuo":y, "coordinate": []}
            for y in range(len(coil[coil_name.index(x)]))]}for x in coil_name]
for loop,idx in zip(coil, range(len(coil))):
```

```
set_coordinate_loop(loop, idx)
```

Innanzitutto viene creato attreverso la funzione messa a disposizione dalla libreria biopython il parser per poter effettivamente eseguire il parsing del file pdb. Il file pdb è un formato particolare che viene utilizzato per il salvataggio su file delle strutture proteiche. Il parser generato in precedenza ci permette di utilizzare alcuni metodi specifici come quello per ottenere la struttare proteica contenuta all'interno del file. La struttura a sua volta ha una composizione pressoche ad albero e ci sono vari livelli al suo interno, noi prendiamo il modello e le catene all'interno del modello. La catena di nostro interesse è la B, una volta identificato il suo id possiamo farci accesso come se stessimo indicizzando un array.

Il prossimo step è quello di andare a recuperare i loop che collegano la parte fissa alla parte mobile utilizzando la procedura identify_loop. I loop all'interno di questa struttura sono due quindi abbiamo due costanti che contengono appunto le costanti di partenza e fine dei due loop. I due loop contengono due residui in più; questo, perché il loop A continente l'ultimo residuo della parte fissa all'inizio e il primo residuo della parte mobile alla fine, mentre il loop B contiene l'ultimo residuo della parte mobile all'inizio e il primo residuo della parte fissa alla fine. Essi hanno senso contrario, questo comporta che il loop B una volta individuato venga invertito, ulteriormente invertito una volta che deve essere caricato come risultato. Vengono conservati due residui in più perché vogliamo essere certi di attaccarci nel posto corretto e senza sbagliare ne l'aggancio con il punto di partenza nel con quello di arrivo per non rischiare di modificare la struttura.

```
def identify_loop(list_chain, chain_of_interest, costanti):
    loop = []
    residue = list(list_chain[chain_of_interest].get_residues())
    for r in residue:
        if (r.id >= costanti[0]) and (r.id <= costanti[1]):
            loop.append(r)
    return loop
```

La procedura identify_loop si occupa appunto di andare ad individuare i loop all'interno della catena. Vengono presi come parametro la lista delle catene, l'idx della catena di interesse e le costanti. Una volta presa la catena di interesse abbiamo un metodo che ci permette di ottenere i residui all'interno della stessa e scorrendo i residui troviamo quelli che hanno l'id compreso tra le costanti. Viene restituita quindi la lista contenenti i residui che soddisfano la condizione.

Una volta individuati i loop, salviamo le coordinate dei singoli atomi in una struttura creata apposta in modo tale da non aver problemi di condivisione della memoria, ovvero non vogliamo che una modifica alla coordinata di un atomo sia condivisa con tutti gli altri atomi.

```
def set_coordinate_loop(loop, idx):
    appoggio = coil_dict_coordinates[idx]['residui']
    idx_res = 0
    for residue in loop:
        appoggio[idx_res]['coordinate'] = []
        for atom in residue.get_atoms():
            array = []
            c = atom.get_vector()
            for el in c.get_array():
                array.append(el)
            appoggio[idx_res]['coordinate'].append(array)
        idx_res += 1
```

La procedura set_coordinate_loop è molto semplice ovvero prende tutti i residui presenti all'interno dell loop, di ogni residuo prendiamo gli atomi e le relative coordinate [x, y, z] e le salviamo all'interno dell'apposito spazio creato per loro all'interno del dizionario.

```
list_chain = list(models[0].get_chains())
residui_parte_mobile = identificazione_parte_mobile(list_chain,
                        idx_chain_of_interest)
residui_parte_mobile_after_rotation = apply_rotation_on_mobil_part
                            (residui_parte_mobile)
if (transition_step != 0):
    print("Applico la funzione di traslazione")
    applytransitionmove(residui_parte_mobile_after_rotation)
res_pt_mob_copy = deepcopy(residui_parte_mobile_after_rotation)
insert_new_mobil_part(list_chain, idx_chain_of_interest,
            residui_parte_mobile_after_rotation)
printpdb("Rotazione_parte_mobile.pdb")
for loop, idx in zip(coil, range(len(coil))):
    print("Loop-",coil_name[idx])
    distanza_iniziale = objective_function(loop,
        residui_parte_mobile_after_rotation, coil_name[idx])
    distanza_migliore.append(distanza_iniziale)
    distanza_raggiunta.append(distanza_iniziale)
    angolo_positivo = False
```

```
residuo_piccolo = False
angolo_positivo_residuo = 0
swap_residuo = 0
loop_new = []
print("Initial distance: ", distanza_iniziale)

print("Rotazioni canoniche sulla meta del loop")
for a in angle_to_direction:
    for j in [0,1]:
        if (coil_name[idx] == 'A'):
            prova = hillclimbing(loop, [7, 8, 9, 10], a,
                                 int(len(loop)/2)+1, j)
            residui_usati.append(int(len(loop)/2)+1)
        else:
            prova = hillclimbing(loop, [], a,
                                 int(len(loop)/2), j)
            residui_usati.append(int(len(loop)/2))
        distance = objective_function(prova,
                residui_parte_mobile_after_rotation, coil_name[idx])
        distanza_raggiunta.append(distance)
        angoli_usati.append(a)
        if (distance < distanza_iniziale):
            distanza_iniziale = distance
            distanza_migliore.append(distanza_iniziale)
            set_coordinate_loop(prova, idx)
        get_coordinate_loop(loop, idx)
print("Distanza dopo prova delle rotazioni sulla meta del loop: ",
                    distanza_iniziale)

if (coil_name[idx] == 'A'):
    insert_new_loop(list_chain, idx_chain_of_interest, loop,
            costanti_loopA, coil_name[idx])
else:
    loop_af = loop[::-1]
    insert_new_loop(list_chain, idx_chain_of_interest, loop_af,
            costanti_loopB, coil_name[idx])
title = "Canonial_rotation" + coil_name[idx] +".pdb"
printpdb(title)

for i in range(iterazioni_massime):
    if (coil_name[idx] == 'A'):
```

```
            n_res = suggestedresidue(len(loop), distanza_iniziale,
                                distanza_raggiunta)
            angle = suggestedangle(distanza_iniziale, distanza_raggiunta)
            swap = suggestedchange(distanza_iniziale, distanza_raggiunta)
            if(swap):
                contatore_swap_effettuati += 1
                if angolo_positivo:
                    angle = 0.5
                else:
                    angle = -0.5
                loop_new = hillclimbing(loop, [7, 8, 9, 10], angle, 1)
                residui_usati.append(1)
                angoli_usati.append(angle)
            else:
                loop_new = hillclimbing(loop, [7, 8, 9, 10], angle, n_res)
                residui_usati.append(n_res)
                angoli_usati.append(angle)
            start = distancefromstarting(loop_new, list_chain,
                                         idx_chain_of_interest, coil_name[idx])
        else:
            n_res = suggestedresidue(len(loop), distanza_iniziale,
                                         distanza_raggiunta)
            angle = suggestedangle(distanza_iniziale, distanza_raggiunta)
            swap = suggestedchange(distanza_iniziale, distanza_raggiunta)
            if(swap):
                contatore_swap_effettuati += 1
                if angolo_positivo:
                    angle = 0.5
                else:
                    angle = -0.5
                loop_new = hillclimbing(loop, [], angle, 1)
                angoli_usati.append(angle)
                residui_usati.append(1)
            else:
                loop_new = hillclimbing(loop, [], angle, n_res)
                residui_usati.append(n_res)
                angoli_usati.append(angle)
            start = distancefromstarting(loop_new, list_chain,
                                         idx_chain_of_interest, coil_name[idx])
        if (start):
            distance = objective_function(loop_new,
```

```
                        residui_parte_mobile_after_rotation, coil_name[id
            distanza_raggiunta.append(distance)
            if (distance <= distanza_iniziale):
                distanza_iniziale = distance
                set_coordinate_loop(loop_new, idx)
                distanza_migliore.append(distanza_iniziale)
            if (distanza_iniziale <= 0.1):
                print("Ho raggiunto la distanza minima")
                print("Mancavano tot iterazioni: ", iterazioni_massime-i)
                break

        get_coordinate_loop(loop, idx)

        if (i % print_pdb_timeout == 0):
            if (coil_name[idx] == 'A'):
                insert_new_loop(list_chain, idx_chain_of_interest, loop,
                                costanti_loopA, coil_name[idx])
            else:
                loop_af = loop[::-1]
                insert_new_loop(list_chain, idx_chain_of_interest, loop_a
                                costanti_loopB, coil_name[idx])
            title = "Situazione-"+coil_name[idx]+"-iter"+str(i)+".pdb"
            printpdb(title)

        if (stopsearch(distanza_iniziale, distanza_raggiunta)):
            print("Ho fermato la ricerca perchè non stavi
                    migliorando abbastanza")
            #print(distanza_raggiunta)
            break

    print("Final distance: ", distanza_iniziale)
    ultime = distanza_raggiunta[len(distanza_raggiunta)-5:
                                len(distanza_raggiunta)]
    print("Ultime 5 distanze: ", ultime)
    print("Swap effettuati: ", contatore_swap_effettuati)
    print('\n')

    title = "Loop-" + coil_name[idx] + ".csv"
    with open(title, 'w') as file:
        # 2. Create a CSV writer1
        writer = csv.writer(file)
```

```
            # 3. Write data to the file
            writer.writerow(header)
            for n in range(len(distanza_raggiunta)):
                if (n == 0):
                    data = [coil_name[idx], distanzeCA[n], scalar_product[n],
                            distanza_raggiunta[n], usedweight[n], "None",
                            "None", "None"]
                else:
                    data = [coil_name[idx], distanzeCA[n], scalar_product[n],
                            distanza_raggiunta[n], usedweight[n],
                            loop[residui_usati[n-1]].id[1], torsion_used[n-1],
                            angoli_usati[n-1]]
                writer.writerow(data)

        if (coil_name[idx] == 'A'):
            insert_new_loop(list_chain, idx_chain_of_interest, loop,
                            costanti_loopA, coil_name[idx])
            visualize_image(distanza_raggiunta, distanza_migliore, angoli_usati,
                            residui_usati, "LoopA-distance")
        else:
            loop_af = loop[::-1]
            insert_new_loop(list_chain, idx_chain_of_interest, loop_af,
                            costanti_loopB, coil_name[idx])
            visualize_image(distanza_raggiunta, distanza_migliore,
                        angoli_usati, residui_usati, "LoopB-distance")
```

In questo secondo blocco di codice andiamo ad effettuare le operazioni per portare a convergenza i loop. La prima operazione da fare è quella di andare ad individuare la parte mobile, mediante una procedura che funziona in modo simile a quella descritta in precedenza per i loop.

```
    def identificazione_parte_mobile(list_chain, chain_of_interest):
        mobil_part = []
        residue = list(list_chain[chain_of_interest].get_residues())
        for r in residue:
            if (r.id >= costanti_parte_mobile[0]) and
                (r.id <= costanti_parte_mobile[1]):
                mobil_part.append(r)
        return mobil_part
```

La procedura identificazione_parte_mobile si occupa di andare ad identificare la parte mobile soggetta poi a rotazione. Si sviluppa nello stesso modo rispet-

to alla procedura per l'identificazione dei loop. In questo caso non vengono passate costanti alla procedura perché la parte mobile è soltanto una quindi vengono prese dalle variabili globali. In questo caso non ci sono aggiunte di residui in più per controllare la convergenza, ma solo ed esclusivamente i residui della parte mobile. L'operazione successiva è quella di applicare una rotazione alla parte mobile.

```
def apply_rotation_on_mobil_part(residui):
    rotation_radians = np.radians(rotation_angle)
    subtrac_coord, r_matrix = calcolocentroide(residui, rotation_radians)
    after_rotation = []
    for residuo in residui:
        for atom in residuo.get_atoms():
            atom_coord = atom.get_coord()
            ### Sottraggo le coordinate da sottrarre
            for k in range(len(atom_coord)):
                atom_coord[k] = atom_coord[k] - subtrac_coord[k]
                set_element(k, atom_coord[k])
            ### Dobbiamo moltiplicare
            for j in range(len(r_matrix)):
                val = 0
                for c1 in range(len(r_matrix[j])):
                    val += r_matrix[j][c1] * get_element(c1)
                atom_coord[j] = val
            ### Sommiamo di nuovo le coordinate sottratte in precedenza
            for k in range(len(atom_coord)):
                atom_coord[k] = atom_coord[k] + subtrac_coord[k]
            atom.set_coord(atom_coord)
        after_rotation.append(residuo)
    return after_rotation
```

La procedura apply_rotation_on_mobil_part ci permette di andare a ruotare di x gradi la parte mobile. Vengono trasformati in radianti i gradi di rotazione;viene calcolato il centroide della parte mobile come somma di tutte le coordinate diviso il numero di punti sommati viene normalizzato e viene la matrice di rotazione. Dopodiché per rendere effettiva la rotazione si prendono le coordinate dei singoli atomi vi si sottraggono le coordinate del centroide, il risultato viene moltiplicato per la matrice di rotazione e infine al risultato vengono sommate le coordinate sottratte in precedenza. Per evitare il problema di condivisione della memoria citato in precedenza mi appoggio ad un array globale con dei metodi di set e get riportati qui sotto.

```
def set_element(index, coord):
    array_appoggio[index] = 0
    array_appoggio[index] = coord

def get_element(index):
    return array_appoggio[index]
```

Oltre alla rotazione della parte mobile è possibile anche traslare la stessa secondo una direzione, il coefficente di traslazione e uno dei parametri presi da riga di comando e la traslazione avviene se e soltanto se il parametro è diverso da zero.

```
def applytransitionmove(residui):
    for residuo in residui:
        for atom in residuo.get_atoms():
            atom_coord = atom.get_coord()
            for i in range(len(atom_coord)):
                if (transition_axis[i] == 1):
                    atom_coord[i] += transition_step
            atom.set_coord(atom_coord)
```

La procedura applytransitionmove si occupa come detto in precendenza di applicare una traslazione secondo gli assi [x, y, z]. Viene preso ogni singolo atomo e per ciascuna delle sue coordinate ci si domanda se è necessario traslarla, se necessario si aggiunge quindi lo step di traslazione.

Dopo aver completato la rotazione e traslazione della parte mobile è necessario fare una copia della stessa per essere sicuri di non modificarla e in caso di modifica abbiamo sempre una copia della stessa. Dopodichè andiamo a reinserire la parte mobile modificata all'interno del file pdb attraverso la seguente procedura.

```
def insert_new_mobil_part(list_chain, chain_of_interest,
    residui_parte_mobile_after_rotation):
    residue = list(list_chain[chain_of_interest].get_residues())
    idx = 0
    for r in residue:
        if (r.id >= costanti_parte_mobile[0])
            and (r.id <= costanti_parte_mobile[1]):
            for atomr, atoml in zip(r.get_atoms(),
                residui_parte_mobile_after_rotation[idx].get_atoms()):
                atomr.set_coord(atoml.get_coord())
            idx += 1
```

La procedura insert_new_mobil_part si occupa di andare ad inserire all'interno della struttura della proteina la parte mobile precedentemente modificata per fare ciò è necessario andare a scorrere di nuovo tutta la nostra catena di interesse e una volta trovati i residui corretti sostituire le loro coordinate con quelle nuove generate dopo la rotazione. Dopo questa fase di lavoro sulla parte mobile si apre il capitolo sulla convergenza dei singoli loop, come detto in precedenza essi sono due e vengono allineati in modo tale da non dover duplicare le operazioni. Come prima cosa dobbiamo andare a misurare quelle che sono le distanze di partenza.

```
def objective_function(loop, mobil_part, loop_name):
    residuo_loop = None
    residuo_parte_mobile = None
    if (loop_name == 'A'):
        ### Devo calcolare la distanza tra l'ultimo elemento
        ### del loop e il primo elemento della parte mobile
        residuo_loop = loop[len(loop)-1]
        residuo_parte_mobile = mobil_part[0]
    else:
        ### Devo calcolare la distanza tra l'ultimo elemento
        ### del loop e il primo elemento della parte mobile
        residuo_loop = loop[len(loop)-1]
        residuo_parte_mobile = mobil_part[len(mobil_part)-1]
    atom_loop = residuo_loop['CA']
    atom_ptmob = residuo_parte_mobile['CA']
    distanzaCA = atom_loop - atom_ptmob
    atomC_loop = residuo_loop['C'].get_coord()
    atomN_loop = residuo_loop['N'].get_coord()
    axis_coord_loop = atomN_loop - atomC_loop
    lenght = math.sqrt((axis_coord_loop[0] * axis_coord_loop[0]) +
        (axis_coord_loop[1] * axis_coord_loop[1]) +
        (axis_coord_loop[2] * axis_coord_loop[2]))
    for i in range(len(axis_coord_loop)):
        axis_coord_loop[i] = axis_coord_loop[i]/lenght

    atomC_ptmob = residuo_parte_mobile['C'].get_coord()
    atomN_ptmob = residuo_parte_mobile['N'].get_coord()
    axis_coord_ptmob = atomN_ptmob - atomC_ptmob
    lenght = math.sqrt((axis_coord_ptmob[0] * axis_coord_ptmob[0]) +
        (axis_coord_ptmob[1] * axis_coord_ptmob[1]) +
        (axis_coord_ptmob[2] * axis_coord_ptmob[2]))
```

```
    for i in range(len(axis_coord_ptmob)):
        axis_coord_ptmob[i] = axis_coord_ptmob[i]/lenght

    prodotto_scalare = 0
    for i, j in zip(range(len(axis_coord_ptmob)), range(len(axis_coord_loop))):
        prodotto_scalare += axis_coord_ptmob[i] * axis_coord_loop[j]
    weight = suggestedweight(distanzaCA)
    usedweight.append(weight)
    distanzeCA.append(distanzaCA)
    scalar_product.append(prodotto_scalare)
    distanza = distanzaCA + weight*(1-prodotto_scalare)
    return distanza
```

La procedura objective_function si occupa di calcolare la distanza tra due residui, ovvero nel caso del loopA l'ultimo del loop A il primo della parte mobile, mentre nel caso del loop B l'ultimo della parte mobile. La distanza viene calcolata nel seguente modo ovvero, viene presa la distanza tra gli atomi Cα dei due residui e poi ci viene sommato il prodotto scalare generato dall'asse tra gli atomi N e C dei due residui, moltiplicato per un coefficente detto peso in modo da rendererlo influente. Il prodotto scalare viene inserito per fare in modo che si converga nel modo corretto con la direzione corretta, altrimenti si rischia di non convergere perchè è possibile che N e C siano opposti rispetto all'altro residuo. Il peso che viene moltiplicato al prodotto scalare viene suggerito dalla procedura suggestedweight che ci restituisce un peso in base alla distanza dei CA. Calcolata la distanza iniziale, ho pensato di applicare delle rotazioni "canoniche" dalla metà del loop in avanti sia su phi che su psi, questo ci garantisce di colmare grandi distanze immediatamente per poi lavorare di fino nella ricerca successiva. Le rotazioni "canoniche" si basano sui seguenti angoli: [-180, -90, -60, -30, -15, -5, -3, -1, -0.5, 0.5, 1, 3, 5, 15, 30, 60, 90, 180]. In precedenza ho nominato phi e psi che sono angoli torsionali, in verità applichiamo la rotazione sull'asse dell'angolo torsionale, ovvero per quanto riguarda phi l'asse è tra CαN, mentre psi l'asse è tra CαC. La procedura che si occupa di applicare la rotazione è la seguente.

```
def hillclimbing(loop, constat_to_not_touch, angle,
                    n_res, direction = -1):
    loop_new = []
    ### Scegliamo a caso un atomo su cui agire
    idx_res_start = n_res
    ### Scegliamo se ruotare su phi o psi
    if (direction == -1):
```

```
            torsional_angle = random.randint(0,1)
        else:
            torsional_angle = direction
        torsion_used.append(torsional_angle)

        if(idx_res_start not in constat_to_not_touch):
            r_matrix =  []
            subtrac_coord = []
            if (torsional_angle == 0):
                atomN_coord = None
                atomCA_coord = None
                for atom in loop[idx_res_start].get_atoms():
                    if atom.id == 'N':
                        atomN_coord = atom.get_coord()
                    if atom.id == 'CA':
                        atomCA_coord = atom.get_coord()
                ### Vettore u su cui dobbiamo ruotare
                subtrac_coord = atomN_coord
                axis_coord = atomCA_coord - atomN_coord
                r_matrix = rotation_matrix(axis_coord, angle)
            else:
                ### Dobbiamo Calpha - C
                atomC_coord = None
                atomCA_coord = None
                for atom in loop[idx_res_start].get_atoms():
                    if atom.id == 'C':
                        atomC_coord = atom.get_coord()
                    if atom.id == 'CA':
                        atomCA_coord = atom.get_coord()
                ### Vettore u su cui dobbiamo ruotare
                subtrac_coord = atomC_coord
                axis_coord = atomCA_coord - atomC_coord
                r_matrix = rotation_matrix(axis_coord, angle)

            ### Con questo primo ciclo copio semplicemente
            i vecchi residui che non verranno modificati
            for i in range(0, idx_res_start):
                loop_new.append(loop[i])

            ### Con questo secondo ciclo applico la rotazione
            sugli atomi
```

```
for i in range(idx_res_start, len(loop)):
    if (torsional_angle == 0):
        if i == idx_res_start:
            for atom in loop[i].get_atoms():
                if (atom.id != 'N' and atom.id != 'CA'):
                    atom_coord = atom.get_coord()
                    ### Sottraggo le coordinate da sottrarre
                    for k in range(len(atom_coord)):
                        atom_coord[k] = atom_coord[k] -
                                    subtrac_coord[k]
                        set_element(k, atom_coord[k])
                    ### Dobbiamo moltiplicare
                    for j in range(len(r_matrix)):
                        val = 0
                        for c1 in range(len(r_matrix[j])):
                            val += r_matrix[j][c1] *
                                get_element(c1)
                        atom_coord[j] = val
                    ### Sommiamo di nuovo le coordinate sottratte
                    ### in precedenza
                    for k in range(len(atom_coord)):
                        atom_coord[k] = atom_coord[k] +
                            subtrac_coord[k]
                    atom.set_coord(atom_coord)
        else:
            for atom in loop[i].get_atoms():
                atom_coord = atom.get_coord()
                ### Sottraggo le coordinate da sottrarre
                for k in range(len(atom_coord)):
                        atom_coord[k] = atom_coord[k] -
                            subtrac_coord[k]
                        set_element(k, atom_coord[k])
                ### Dobbiamo moltiplicare
                for j in range(len(r_matrix)):
                    val = 0
                    for c1 in range(len(r_matrix[j])):
                        val += r_matrix[j][c1] *
                            get_element(c1)
                    atom_coord[j] = val
                ### Sommiamo di nuovo le coordinate sottratte
```

```
                    ### in precedenza
                    for k in range(len(atom_coord)):
                        atom_coord[k] = atom_coord[k] +
                            subtrac_coord[k]
                    atom.set_coord(atom_coord)
    if (torsional_angle == 1):
        if i == idx_res_start:
            for atom in loop[i].get_atoms():
                if (atom.id != 'N' and atom.id != 'C'
                    and atom.id != 'CA'):
                    atom_coord = atom.get_coord()
                    ### Sottraggo le coordinate da sottrarre
                    for k in range(len(atom_coord)):
                        atom_coord[k] = atom_coord[k] -
                            subtrac_coord[k]
                        set_element(k, atom_coord[k])
                    ### Dobbiamo moltiplicare
                    for j in range(len(r_matrix)):
                        val = 0
                        for c1 in range(len(r_matrix[j])):
                            val += r_matrix[j][c1] *
                                get_element(c1)
                        atom_coord[j] = val
                    ### Sommiamo di nuovo le coordinate sottratte
                    ### in precedenza
                    for k in range(len(atom_coord)):
                        atom_coord[k] = atom_coord[k] +
                            subtrac_coord[k]
                    atom.set_coord(atom_coord)
        else:
            for atom in loop[i].get_atoms():
                atom_coord = atom.get_coord()
                ### Sottraggo le coordinate da sottrarre
                for k in range(len(atom_coord)):
                    atom_coord[k] = atom_coord[k] -
                        subtrac_coord[k]
                    set_element(k, atom_coord[k])
                ### Dobbiamo moltiplicare
                for j in range(len(r_matrix)):
                    val = 0
                    for c1 in range(len(r_matrix[j])):
```

```
                                    val += r_matrix[j][c1] *
                                        get_element(c1)
                                atom_coord[j] = val
                            ### Sommiamo di nuovo le coordinate sottratte
                            ### in precedenza
                            for k in range(len(atom_coord)):
                                atom_coord[k] = atom_coord[k] +
                                    subtrac_coord[k]
                            atom.set_coord(atom_coord)
                loop_new.append(loop[i])
        else:
            loop_new = loop
        return loop_new
```

La procedura hillclimbing si occupa di andare ad applicare una rotazione su uno degli assi degli angoli torsionali (phi, psi). Alcuni parametri hanno dei valori di default che cambiano quello che è il comportamento della procedura, ovvero l'angolo torsionale può essere deciso sia randomicamente all'interno della procedura oppure può essere passato dall'esterno come nel caso delle rotazioni "canoniche" in cui ogni singolo angolo viene provato sia per phi che psi. Una volta deciso l'angolo torsionale si verifica che la scelta del residuo su cui vogliamo cominciare ad effettuare le rotazioni non compaia dentro a quei residui esplicitati nella variabile constat_to_not_touch e poi si compie la rotazione. Ci sono residui che non devono essere toccati ovvero, che da quei residui non può partire nessuna rotazione perché sono una parte fissa, mentre se la rotazione parte prima loro possono comunque essere coinvolti. La rotazione su phi o psi non cambia il metodo, ma cambio solo gli interpreti della rotazione.

Terminate le rotazioni "canoniche" si cominciano le iterazioni specificate come parametro del programma e in queste rotazioni il residuo e l'angolo su cui si va a lavorare vengono suggeriti dalle seguenti procedure.

```
    def suggestedangle(distanza, dist_ragg):
        global angolo_positivo
        global angolo_positivo_residuo
        if (len(dist_ragg)-angolo_positivo_residuo > 49):
            change = False
            for i in range(len(dist_ragg)-50, len(dist_ragg)):
                if(dist_ragg[i] > distanza):
                    change = True
                else:
```

```
                change = False
                break
        if (change):
            angolo_positivo_residuo = len(dist_ragg)
            angolo_positivo = not(angolo_positivo)
    angle = 0

    if (distanza >= 20):
        angle = 5
    elif (distanza >= 15):
        angle = 4
    elif (distanza >= 10):
        angle = 3
    elif (distanza >= 5):
        angle = 2
    elif (distanza >= 2):
        angle = 1
    elif (distanza >= 1):
        angle = 0.7
    elif (distanza >= 0.5):
        angle = 0.5
    else:
        angle = 0.1

    if angolo_positivo:
        return angle
    else:
        return -angle

def suggestedresidue(lunghezza, distanza, dist_ragg):
    n_res = lunghezza-1
    n_res = random.randint(1, lunghezza-1)
    return n_res
```

La procedura suggestedangle mi suggerisce un angolo e il segno dell'angolo ovvero l'angolo viene deciso mediante confronto della distanza, mentre il segno viene deciso in base ai risultati dell'esecuzione. La distanza considerata è la migliore ottenuta fino in questo momento, mentre vengono prese le ultime 50 distanze e sono tutte maggiori rispetto alla distanza migliore si inverte il precedente segno dell'angolo, in modo alternare.

La procedura suggestedresidue mi suggerisce in modo randomico un re-

siduo su cui andare a lavorare, viene escluso il primo perchè è per entrambi quello di partenza. Durante l'esecuzione delle iterazioni potrebbe essere necessario eseguire una grossa rivoluzione, ovvero ho pensato ad una tecnica per ribaltare il risultato.

```
def suggestedchange(distanza, dist_ragg):
    swap = False
    global swap_residuo
    if (len(dist_ragg)-swap_residuo  > 10):
        for i in range(len(dist_ragg)-11, len(dist_ragg)):
            if (dist_ragg[i] > distanza and dist_ragg[i] <= distanza+0.5):
                swap = True
            else:
                swap = False
                break
    if (swap):
        swap_residuo = len(dist_ragg)
    return swap
```

La procedura suggestedchange è usata per suggerire al programma principale di applicare un cambiamento ovvero se per 10 risultati consecutivi dell'esecuzione otteniamo una distanza risultante compresa tra la distanza migliore e la distanza migliore a cui viene aggiunto 0.5 significa che non risciamo a trovare una via per avvicinarci di più quindi applicare una rotazione alla base del loop di mezzo grado potrebbe darci una mano per ottenere il nostro risultato.

Una volta terminata la rotazione verifichiamo che non ci siamo discostati dal punto di partenza mediante la funzione distancefromstarting, che non fa altro che verificare che le distanze presenti in precedenza siano ancora invariate, misuriamo quindi la distanza del loop e se ci siamo avvicinati ci teniamo il nuovo risultato e il nuovo loop altrimenti torniamo alla versione precedente e ricominciamo da li.

Ci sono due modi per interrompere preventivamente l'esecuzione del programma: raggiungere la distanza minima di 0.1; oppure essere fermati da una procedura che verifica che negli ultimi 1000 passaggi abbiamo fatto progressi per almeno 0.01.

Capitolo 5

Risultati

Conclusione

Conclusione che riassume il lavoro svolto ed eventuali lavori futuri.

Bibliografia

[Allen and Zilberstein, 2009] Allen, M. and Zilberstein, S. (2009). Complexity of decentralized control: Special cases. In *23rd Annual Conference on Neural Information Processing Systems 2009, 7-10 December , Vancouver, British Columbia, Canada*, pages 19–27. Curran Associates, Inc.

[Bernstein et al., 2002] Bernstein, D. S., Givan, R., Immerman, N., and Zilberstein, S. (2002). The complexity of decentralized control of markov decision processes. *Mathematics of operations research*, 27(4):819–840.

[De Weerdt et al., 2003] De Weerdt, M., Bos, A., Tonino, H., and Witteveen, C. (2003). A resource logic for multi-agent plan merging. *Annals of Mathematics and Artificial Intelligence*, 37(1-2):93–130.

[Durfee, 2001] Durfee, E. H. (2001). *Distributed Problem Solving and Planning*, pages 118–149. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Gelfond and Lifschitz, 1998] Gelfond, M. and Lifschitz, V. (1998). Action languages. *Electron. Trans. Artif. Intell.*, 2:193–210.

[Russell and Norvig, 2010] Russell, S. J. and Norvig, P. (2010). *Artificial Intelligence - A Modern Approach, Third International Edition*. Pearson Education.

Ringraziamenti

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus

a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Appendice A

Appendice di Esempio

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tri-

stique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.