Universitá degli Studi di Napoli "Parthenope"

Scuola interdipartimentale delle Scienze, dell'Ingegneria e della Salute

Dipartimento di Scienze e Tecnologie

Corso di laurea in Informatica

Tesi di laurea triennale

Analisi delle interazioni molecolari e sviluppo di un software per l'automatizzazione del docking molecolare

Preparazione dei ligandi e dei recettori, docking ed estrazione dei legami

Relatore
Ch.mo
Prof. Angelo Ciaramella

Correlatore
Dott. Ferdinando Febbraio

Laureando
Alfredo Mungari
Matr. 0124002134

Anno Accademico 2021-2022

Introduzione

Negli ultimi anni l'informatica sta ricoprendo un ruolo sempre più preponderante nel campo della "Life Science", attraverso lo sviluppo di applicazioni che permettono l'analisi o la predizione di fenomeni biologici. Partendo da un problema biologico come la crescente mortalità delle api e da uno studio biochimico e di risk assessment, che cerca di mettere in relazione questa mortalità con la predizione dell'interazione dei pesticidi con le proteine delle api, si è cercato di realizzare/progettare un software in grado di automatizzare una serie di procedure per rendere più accessibili questo tipo di analisi a ricercatori con background informatici molto diversi. Il software realizzato prende il nome di AUtomated DOcking 4 RIsk ASsessment (AUDO4RIAS), e riunisce in una sola applicazione diverse funzioni e strumenti di bioinformatica. In particolare, questo software è stato sviluppato con l'obiettivo di creare un applicativo capace di automatizzare: (i) i processi di preparazione degli input per l'analisi di docking molecolare, inclusi la ricerca e il download delle strutture 3D delle molecole (ligandi e recettori), (ii) l'esecuzione dell'analisi di docking molecolare e (iii) l'analisi dei risultati della predizione dell'interazione proteina/ligando. Il software è disponibile su GitHub, rilasciato sotto licenza Creative Commons.

Abstract

In recent years, information technology is playing an increasingly prominent role in the field of "Life Science," through the development of applications that enable the analysis or prediction of phenomena biology. Starting from a biological problem such as increasing bee mortality and a biochemical and risk assessment study, which seeks to relate relate this mortality to the prediction of the interaction of pesticides with bee proteins, an attempt was made to implement/design software capable of capable of automating a series of procedures to make this type of analysis more accessible to researchers with very different computer science backgrounds. The software created is called AUtomated DOcking 4 RIsk ASsessment (AUDO4RIAS), and it brings together in a single application different bioinformatics functions and tools. In particular, this software was developed with the the goal of creating an application capable of automating: (i) the processes of preparing inputs for molecular docking analysis, including searching and downloading the 3D structures of molecules (ligands and receptors), (ii) the execution of the analysis of molecular docking, and (iii) the analysis of the results of the protein/ligand interaction prediction. The software is available on GitHub, released under a license Creative Commons.

"Ogni ape porta in sé il meccanismo dell'universo: ognuna riassume il segreto del mondo."

Michel Onfray

Indice

Introduzione **I**

Abstract **II**

Elenco delle figure **VII**

Elenco delle tabelle **IX**

Elenco dei listati **X**

1 Docking Molecolare **1**
1.1 I pesticidi e gli insetti impollinatori 1
1.2 Cos'è il Docking Molecolare 4
1.3 Simulazione 8
1.4 Software per il Docking Molecolare 9
1.5 Gestione del lavoro 10
1.6 Idea e sviluppo 11
1.7 Descrizione dell'applicazione 12
1.8 Contenuto della tesi 13

2 Tecnologie e piattaforme **16**
2.1 Python 16
2.1.1 PubChemPy 18
2.1.2 ProDy 19
2.1.3 Pandas 19
2.1.4 Tkinter 19
2.1.5 Matplotlib 20

2.1.6 NumPy 20
2.2 ADFRsuite 21
2.2.1 MGLTools 22
2.2.2 Open Babel 23
2.3 Database 24
2.3.1 RCSB 25
2.3.2 Pubchem 25
2.4 Autodock 26
2.5 Autodock Vina 27
2.5.1 Funzione di scoring 28

3 Applicazione realizzata 33
3.1 Installazione 33
3.2 Modalità di utilizzo 34
3.3 Dati in input 37
3.4 Preparazione dei ligandi e recettori 41
3.4.1 Preparazione dei ligandi tramite script 42
3.4.2 Preparazione dei ligandi tramite GUI 47
3.4.3 Preparazione dei recettori tramite script 52
3.4.4 Preparazione dei recettori tramite GUI 61
3.5 Docking 65
3.5.1 Docking tramite script 65
3.5.2 Docking tramite GUI 68
3.6 Analisi 72
3.6.1 Analisi tramite script 72
3.7 Analisi tramite GUI 74

4 Risultati sperimentali 77
4.1 I casi di analisi 77
4.2 GNINA 78
4.3 Visualizzazione della predizione della posa 78
4.4 Rilevazione delle interazioni molecolari 82
4.5 Analisi dei tempi di esecuzione 88
4.6 Comparazione con altri tools 90

Conclusioni e sviluppi futuri **91**

A Tabella dei ligandi **95**

B Tabella dei recettori **99**

Bibliografia **100**

Ringraziamenti **101**

Elenco delle figure

1.1 Esemplare adulto di Apis Mellifera 3
1.2 Rappresentazione del docking tra il ligando acrinathrin e la proteina 2h8v 8
1.3 Processo del docking molecolare 10
1.4 QR code della repository di GitHub di AUDO4RIAS . 13
1.5 Diagramma implementativo del software 15

2.1 Logo di Python 18
2.2 Logo di Open Babel 24
2.3 Logo di RCSB 25
2.4 Logo di PubChem 26
2.5 Logo di Autodock 27
2.6 Logo di Autodock Vina 28
2.7 Funzione di scoring pesata 32

3.1 Schema del paradigma model-view-controller 35
3.2 Sezione Home Page della GUI 36
3.3 Tema scuro della GUI 37
3.4 Query dei recettori 39
3.5 Proteine dei recettori dell'apis mellifera 40
3.6 Sezione Help della GUI 41
3.7 Sezione Preparation della GUI 42
3.8 Sezione Ligands della GUI 48
3.9 Browse files 49
3.10 Browse folders 50
3.11 Pannello dell'esecuzione dei ligandi 51

3.12 Messaggio di completamento della preparazione dei ligandi 51
3.13 Messaggio di errore 52
3.14 Gridbox mostrata all'interno della GUI di AutodockVina 60
3.15 Sezione Receptors della GUI 62
3.16 Browse folders 63
3.17 Pannello dell'esecuzione dei recettori 64
3.18 Messaggio di completamento della preparazione dei ligandi 64
3.19 Messaggio di errore 65
3.20 File di log del docking per la coppia proteina-ligando 1fcu-bentazone 68
3.21 Sezione Docking della GUI 69
3.22 Browse folder 70
3.23 Pannello dell'esecuzione del docking 71
3.24 Messaggio di completamento dell' esecuzione del docking 71
3.25 Messaggio di errore 72
3.26 Heatmap relativa al residuo 1cfu 74
3.27 Grafico a barre relativo al residuo 1cfu 74
3.28 Sezione Analysis della GUI 75
3.29 Pannello dell'esecuzione del analisi 76
3.30 Messaggio di completamento del processo dell'analisi . 76

4.1 Conformazioni proteina-ligando per la proteina 2H8V . 79
4.2 Conformazioni proteina-ligando per la proteina 4E81. . 80
4.3 Conformazioni proteina-ligando per la proteina 6LQK_B 81
4.4 Visualizzazione delle heatmaps per la proteina 1FCU. . 84
4.5 Visualizzazione delle heatmaps per la proteina 5XZ3_A. 85
4.6 Visualizzazione degli istogrammi per la proteina 1FCU. 86
4.7 Visualizzazione degli istogrammi per la proteina 5XZ3_A. 87
4.8 Tempi di preparazione dei ligandi 89
4.9 Tempi di preparazione dei recettori 89
4.10 Tempi di esecuzione dell'analisi 90
4.11 AUDO4RIAS vs other combined tools 91

Elenco delle tabelle

2.1 Funzione di scoring pesi e termini 30

3.1 Ligandi non scaricati 45

4.1 Interazioni nel complesso proteina-ligando generato da GNINA e AutoDock Vina. 82

A.1 Tabella dei ligandi in input 98

B.1 Tabella dei recettori in input 99

Elenco dei listati

3.1 Comando per scaricare la repository 33
3.2 Comando per avviare la GUI 36
3.3 Comando per scaricare la repository 42
3.4 funzione selectLigands 44
3.5 funzione pubchempy.get_compounds 44
3.6 pubchempy.download 45
3.7 funzione sdf2pdb 46
3.8 Comando per la conversione da .sdf a .pdb 46
3.9 funzione prepareLigands 47
3.10 Comando per la conversione da .pdb a .pdbqt 47
3.11 funzione prepare_receptors 52
3.12 funzione selectReceptors 53
3.13 funzione RestApiSelection 53
3.14 funzione json.loads 54
3.15 funzione json.dump 54
3.16 funzione requests.get 54
3.17 funzione downloadPdbs 54
3.18 funzione fetchPDB 55
3.19 funzione splitRepeatedResidues 55
3.20 funzione PandasPdb.read_pdb 55
3.21 funzione deleteHeteroatomsChains 56
3.22 funzione splitChains 56
3.23 funzione parsePDB 57
3.24 atoms.getHierVieW 57
3.25 writePDB 57
3.26 prepareReceptors 58
3.27 comando per convertire file da .pdb a .pdbqt 58

3.28 createGridboxes 59
3.29 Comando per eseguire performDocking 65
3.30 Comando per eseguire il docking con Autodock Vina . 66
3.31 Comando per eseguire l'analisi 72

Capitolo 1

Docking Molecolare

Questo capitolo tratta dell'idea che ha ispirato questa tesi, ossia gli effetti negativi dei pesticidi sulle api da miele e del processo che ha portato allo sviluppo del software realizzato che visualizza come le molecole di specifici pesticidi si dispongono, in maniera spaziale, quando sono legate ai recettori delle api, questo processo viene definito docking molecolare e successivamente il software estrae i legami che si vengono a formare.

1.1 I pesticidi e gli insetti impollinatori

Le api recano importanti benefici e servizi ecologici per la società. Con l'impollinazione le api svolgono una funzione strategica per la conservazione della flora, contribuendo al miglioramento ed al mantenimento della biodiversità.
In botanica, l'impollinazione è quel processo che consiste nel trasporto dei pollini dalla parte maschile e quella femminile dell'apparato riproduttivo delle piante. Grazie ad agenti atmosferici e sopratutto al lavoro incessante degli insetti impollinatori, soprattutto le api, il polline viene trasportato da una pianta all'altra rendendo possibile la fecondazione di un organismo vegetale della stessa specie e la conseguente produzione di semi e frutti. Una diminuzione delle api può quindi rappresentare una importante minaccia per gli ecosistemi naturali in cui esse vivono. L'agricoltura, d'altro canto, ha un enorme interesse a mantenere le api quali efficaci agenti impollinatori. La

Food and Agriculture Organization - FAO ha informato la comunità internazionale dell'allarmante riduzione a livello mondiale di insetti impollinatori, tra cui *Apis mellifera*, le api da miele. Circa l'84% delle specie di piante e l'80% della produzione alimentare in Europa dipendono in larga misura dall'impollinazione ad opera delle api ed altri insetti pronubi [1]. Pertanto, il valore economico del servizio di impollinazione offerto dalle api risulta fino a dieci volte maggiore rispetto al valore del miele prodotto.
Da un rapporto dell'Unione Internazionale per la Conservazione della Natura (LUCN) risulta che il 10% delle specie selvatiche di api (*Apis mellifera*) sarebbe in via di estinzione e un altro 5% sarebbe a rischio. Una delle principali cause sono i pesticidi, i quali influenzano l'apprendimento, la capacità riproduttiva, i comportamenti sociali di questi insetti e l'orientamento.
La mortalità delle api del miele (*Apis mellifera*) è un fenomeno che si acuisce soprattutto in primavera e che rischia di compromettere la fondamentale funzione ecologica di questi insetti impollinatori per l'intero ecosistema.
Un'indagine di campo del Centro di referenza nazionale per l'apicoltura dell'Istituto Zooprofilattico Sperimentale delle Venezie nell'ambito di alcune morie riscontrate ha rilevato la presenza, in campioni di api morte, di residui di pesticidi e di alcuni virus delle api. Le infezioni virali potrebbero peggiorare l'impatto già negativo dei pesticidi sulla salute delle api, mettendo ulteriormente in pericolo la sopravvivenza delle colonie.
Lo studio è stato effettuato su 94 campioni, provenienti dal Nord-est dell'Italia e raccolti durante la primavera 2014, prendendo in considerazione 150 principi attivi e 3 virus delle api. Lo studio è pubblicato su Journal of Apicultural Research. I ricercatori hanno riscontrato la presenza di almeno un principio attivo nel 72,2% dei campioni (api morte). Gli insetticidi sono i più abbondanti (59,4%), principalmente quelli appartenenti alla classe dei neonicotinoidi (41,8%), seguiti da fungicidi (40,6%) e acaricidi (24,1%). Gli insetticidi più frequentemente rilevati sono rappresentati da imidacloprid, chlorpyrifos, tau-

fluvalinate e cyprodinil.
La presenza di una possibile relazione tra la mortalità primaverile delle api e l'impiego di trattamenti antiparassitari in agricoltura potrebbe contribuire a comprendere meglio fenomeni complessi come la moria delle api e lo spopolamento degli alveari, che negli ultimi dieci anni hanno colpito questo settore[2].
Poichè i pesticidi vengono impiegati lontano dagli alveari e dalle zone loro circostanti, diventa importante comprendere quali sono gli effetti dei pesticidi a concentrazioni sub-letali e come questi possono in qualche modo alterare i parametri vitali descritti precedentemente. Sono iniziati quindi studi di predizione usando docking molecolare, che cercano di mettere in relazione le interazioni tra le proteine di api e le molecole di pesticidi con effetti a livello metabolico che possono giustificare l'incremento di mortalità. Partendo da questi studi, lo scopo della presente tesi è stato quello di progettare un software capace di automatizzare i processi di preparazione degli input per l'analisi di docking molecolare, che includono la ricerca nei database online e il download delle strutture 3D delle molecole di ligando (pesticida) e di recettori (proteine). Inoltre, il software è in grado di eseguire l'analisi di docking molecolare richiamando e interagendo con software specializzati per il docking, come Autodock Vina e, soprattutto, di presentare in maniera semplice ed interpretabile l'analisi dei risultati della predizione dell'interazione proteina/ligando.

Figura (1.1): Esemplare adulto di Apis Mellifera

1.2 Cos'è il Docking Molecolare

Il **docking molecolare** è una tecnica computazionale che permette lo studio dell'interazione tra due molecole. Viene utilizzata, in particolare, per studiare l'interazione delle proteine con altre molecole di interesse biomedico quali acidi nucleici, farmaci, e altre proteine. Il docking si interessa del modo in cui le molecole si agganciano l'una all'altra.
Per effettuare il docking tra due molecole occorre conoscerne la struttura tridimensionale: questa può essere ottenuta per via sperimentale tramite cristallografia a raggi X o per risonanza magnetica nucleare (NMR, Nuclear magnetic resonance), oppure può essere costruita teoricamente. Per studiare il docking è necessario modellizzare le interazioni fondamentali tra i costituenti atomici delle molecole. Queste interazioni sono approssimate in modo adeguato da potenziali di interazione classici (interazioni elettrostatiche, di volume escluso, nonché intramolecolari). Tali potenziali sono utilizzati anche per studiare l'evoluzione temporale delle molecole.
Il docking è possibile grazie alla potenza di calcolo dei computer moderni e allo sviluppo di adeguati algoritmi. Per poter studiare efficacemente le interazioni strutturali tra molecole (in particolare tra proteine e ligandi) si fa anche ricorso a un'analisi statistica: si utilizzano cioè le strutture di proteine e ligandi note sperimentalmente e che possono essere usate come modello (*template*). A questo fine è stato necessario, in anni recenti, sviluppare grandi database (per es., la RCSB Protein Data Bank (RCSB-PDB)) e algoritmi e talvolta linguaggi di programmazione adatti per la ricerca rapida nei database.
Nell'ambito delle bioscienze, le proteine sono di gran lunga le molecole più studiate, per l'importanza che ricoprono nei processi biologici. L'interazione tra proteine è però estremamente complessa in quanto sono oggetti flessibili, ossia in grado di assumere nel tempo, grazie alle fluttuazioni termiche, un insieme molto vasto di conformazioni: è proprio questa flessibilità che permette alle proteine di interagire efficacemente tra loro. Dal punto di vista computazionale è tuttavia

difficile tenere conto efficacemente del panorama conformazionale. Le tecniche che simulano la dinamica possono infatti accedere a tempi di scala ancora limitati (nanosecondi) e le tecniche di docking fanno riferimento a strutture cristallografiche fisse che sono ottenute in condizioni sperimentali lontane dall'ambiente funzionale. Le strutture NMR hanno il vantaggio di essere ottenute quasi in *vivo* (ossia con minore impatto della sperimentazione sulla struttura proteica), ma sono difficili da ottenere e hanno risoluzione spaziale inferiore.
Nonostante le difficoltà, il docking è estremamente utile per lo studio dell'interazione tra proteine e piccoli ligandi. Esistono apposite tecniche di drug design in cui si progetta una molecola in funzione della struttura cui si dovrà legare per espletare la funzione voluta. Nel considerare la configurazione ottimale del legame tra proteina e ligando, vanno inoltre considerati non solo i caratteri topologici di complementarietà (del tipo chiave-serratura), ma anche il bilancio energetico della configurazione. Questo andrà ottimizzato ricercando lo stato in cui l'energia libera del sistema proteina-ligando sia minima, rendendo il loro legame più probabile, e quindi maggiormente stabile rispetto ad altre configurazioni. Poiché esiste un gran numero di configurazioni possibili nelle interazioni tra molecole, sono stati lanciati numerosi progetti di calcolo distribuito con collaborazioni internazionali per fare uno screening su larga scala delle molecole in grado di interagire con specifiche proteine di interesse biomedico come, per es., quelle legate al virus HIV o all'infezione malarica. Si pensa in questo modo di riuscire a ottimizzare l'efficacia dei farmaci esistenti, nonché a scoprirne di nuovi, diretti verso specifici bersagli terapeutici molecolari[3].
Gli algoritmi di docking sono formati da due componenti fondamentali:

- L'algoritmo di ricerca o *"search algorithm"*
- La *"scoring function"*.

Il primo si occupa di generare un insieme di 12 conformazioni del ligando all'interno del sito designato del target, mentre la seconda valuta le *poses* generate, assegnando a ciascuna di esse un punteggio

detto *"score"* in base a parametri di tipo geometrico ed energetico. Le migliori conformazioni in uscita da questa valutazione sono passate nuovamente all'algoritmo di ricerca, che andrà a creare una nuova generazione di conformazioni partendo dalle migliori soluzioni dell'esecuzione precedente. Il funzionamento iterativo del *search algorithm* e della *scoring function* permettono di ottenere, alla fine di un determinato numero di cicli, un insieme di *poses* che vengono fornite come output all'utente e che sono ritenute essere le migliori soluzioni per il *binding* delle molecole in esame da parte del programma di docking utilizzato.
In generale, il docking molecolare è eseguibile in tre differenti condizioni, che si differenziano l'una dall'altra per i gradi di libertà tenuti in considerazione dall'algoritmo durante il calcolo:

I. **docking a corpo rigido**, che approssima sia il ligando che la proteina come strutture rigide

II. **docking semi-flessibile**, che considera il target come rigido, tendendo però in considerazione i gradi di libertà conformazionale del ligando

III. **docking flessibile**, in cui vengono considerati i gradi di libertà sia del ligando che dei residui del target nel sito attivo.

Intuitivamente, passando da un approccio a corpo rigido fino ad uno flessibile, la complessità di calcolo aumenta, e proporzionalmente, anche il tempo di esecuzione.
Ad oggi sono disponibili diversi protocolli di docking ognuno dei quali sfrutta una particolare coppia algoritmo di *ricerca-scoring function.*
Il docking molecolare consiste in tre obiettivi principali collegati tra loro:

1. predizione della posa
2. screening virtuale
3. stima dell'affinità di legame.

Una metodologia di docking di successo deve essere in grado di prevedere correttamente la posa nativa del ligando all'interno del sito di legame del recettore, cioè trovare la geometria sperimentale del ligando entro un certo limite di tolleranza, e le interazioni fisico-chimiche molecolari associate.
Quando si analizzano librerie di composti di grandi dimensioni, il metodo deve essere in grado di distinguere con successo le molecole che si legano da quelle che non si legano e di classificare correttamente questi ligandi tra i migliori composti del database.
Un algoritmo di ricerca e una funzione di score energetico sono gli strumenti di base di una metodologia di docking per generare e valutare le conformazioni dei ligandi. La capacità di gestire con successo la flessibilità molecolare intrinseca di un sistema e di descrivere correttamente l'energia delle interazioni recettore-ligando è cruciale per lo sviluppo di metodologie di docking predittivo che sono utili negli studi prospettici di progettazione di farmaci[4].

Figura (1.2): Rappresentazione del docking tra il ligando acrinathrin e la proteina 2h8v

1.3 Simulazione

La **simulazione** di un processo di docking è un processo molto più che complicato. In tale approccio, la proteina e il ligando sono separati fisicamente da una certa distanza, e il ligando trova la sua posizione nel sito attivo della proteina dopo aver compiuto diversi movimenti nello spazio. Tali movimenti includono rotazioni, traslazioni e torsione di alcuni angoli di rotazione degli atomi. Ognuno di questi movimenti ha un determinato costo energetico nel sistema, dato che dopo ogni mossa viene ricalcolata l'energia totale. Questo approccio rappresenta molto bene quello che accade nella realtà. Di contro, il costo richiesto in termini di tempo e prestazioni è molto elevato.

1.4 Software per il Docking Molecolare

I programmi di docking molecolare eseguono un algoritmo di ricerca in cui la conformazione del ligando viene valutata ricorsivamente fino a raggiungere la convergenza all'energia minima. Infine, una funzione di punteggio di affinità, ΔG (Energia potenziale totale in kcal/mol), viene impiegata per classificare le pose candidate come la somma delle energie elettrostatiche e di van der Waals. Le forze trainanti per queste specifiche interazioni nei sistemi biologici mirano alla complementarità tra la forma e l'elettrostatica delle superfici del sito di legame e del ligando o del substrato.
Negli ultimi vent'anni, sono stati sviluppati più di 60 diversi strumenti e programmi di docking sia per uso accademico e commerciali, come DOCK (Venkatachalam et al. 2003), AutoDock (Österberg et al. 2002), FlexX (Rarey et al. 1996), Surflex (Jain 2003), GOLD (Jones et al. 1997), ICM (Schapira et al. 2003), Glide (Friesner et al. 2004), Cdocker, LigandFit (Venkatachalam et al. 2003), MCDock, FRED (McGann et al. 2003), MOE-Dock (Corbeil et al. 2012), LeDock (Zhao e Caflisch 2013), AutoDock Vina (Trott e Olson 2010), rDock (Ruiz-Carmona et al. 2014), UCSF Dock (Allen et al. 2015) e molti altri.
Tra questi programmi, AutoDock Vina, GOLD e MOE-Dock hanno predetto le pose migliori con gli score migliori. AutoDock e LeDock sono stati in grado di identificare i corretti legami dei ligandi nelle pose. Sia Glide (XP) che AutoDock hanno previsto le pose con un'accuratezza del 90,0% (Wang et al. 2016). È stato dimostrato che AutoDock ha prodotto fattori di arricchimento più rispetto a Glide in uno studio di screening virtuale contro il Fattore Xa, mentre Glide ha superato AutoDock contro lo stesso bersaglio in un analogo studio di screening virtuale. Nel complesso, è stato riportato recentemente che questi programmi di docking sono in grado di predire pose sperimentali con deviazioni al quadrato della radice (RMSD) in media (RMSD) in media da 1,5 a 2 Å[5].
Come mostrato nella Figura 1.3, il software di docking molecolare può

aiutarci a individuare la conformazione e l'orientamento ottimali in base alla complementarità e alla pre-organizzazione con un algoritmo specifico, quindi ad applicare una funzione di scoring per prevedere l'affinità del legame e ad analizzare la modalità interattiva[6].

Figura (1.3): Processo del docking molecolare

1.5 Gestione del lavoro

Il software trattato è frutto del lavoro congiunto del sottoscritto e del mio collega di studi Massimiliano Giordano Orsini con il coordinamento del dottor Ferdinando Febbraio, ricercatore presso l'Istituto di Biochimica e Biologia Cellulare del Consiglio Nazionale delle Ricerche (IBBC-CNR), e mio correlatore di tesi. Il lavoro può essere equiparato ad un vero e proprio progetto software, e si è sviluppato in più fasi. In una prima fase attraverso continue discussioni nell'ambito della biologia ed esempi di utilizzo dei software e delle rispettive analisi, sono stati chiariti una serie di concetti poco familiari a me ed al mio

collega. A questa fase sono seguite varie fasi di analisi e di progettazione, per poi passare alla fase di sviluppo, corrispondente alla fase di effettiva implementazione del software. Infine sono state effettuate le fasi di testing funzionale (FAT) e utente (UAT) per individuare e risolvere errori e bug ed approvare il software realizzato. Le varie fasi di sviluppo software non si sono succedute in maniera sequenziale, seguendo quindi i dettami del modello di sviluppo waterfall, bensì le varie fasi sono state inframmezzate l'una con l'altra e, più che come passi di un processo sono state iterazioni cicliche delle stesse, rispettando i paradigmi dello sviluppo agile. Questo processo di sviluppo è stato dettato dalla natura stessa del progetto, poiché a seguito del raggiungimento di un obiettivo, sono state valutate e decise ulteriori implementazioni per dotare di maggiore complementarietà il software prodotto.

1.6 Idea e sviluppo

L'idea nasce dall'attività di tirocinio svolta presso l'IBBC-CNR di Napoli, per un totale di 300 ore, sotto la supervisione del Responsabile del laboratorio di informatica dell'università Parthenope, professor Angelo Ciaramella e del dottor Ferdinando Febbraio dell'IBBC-CNR di Napoli. Il lavoro effettuato è consistito nella realizzazione di un software, del tutto preliminare al progetto di tesi proposto. Lo sviluppo è avvenuto attraverso diverse fasi nelle quali sono stati utilizzati ed implementati i seguenti tools:

- software per l'esecuzione del docking
- funzioni di bioinformatica per la preparazione degli input necessari
- software per l'analisi dei risultati dell'intero processo.

Sono state determinate le componenti software ideali per automatizzare il processo di docking conseguendo risultati efficienti per quanto riguarda l'output e l'analisi dello stesso, offrendo una buona usabilità

del prodotto realizzato mediante una semplice ed intuitiva interfaccia grafica.

1.7 Descrizione dell'applicazione

Il progetto di tesi proposto ha come focus principale la realizzazione di un applicativo che effettua il docking tra i ligandi contenuti in specifici pesticidi e i recettori dell' *Apis mellifera* e l'estrazione dei legami che si vengono a formare.
Il nome scelto per l'applicazione realizzata è **AUtomated DOcking 4 RIsk ASsessment** (**AUDO4RIAS**) in quanto prova della digitalizzazione della disciplina biologica.
L'applicazione può essere scaricata con le configurazioni di default, direttamente dalla repository di github. Il software realizzato può essere utilizzato in due modalità:

- mediante **script python da terminale**
- mediante **interfaccia grafica** realizzata seguendo il paradigma model ViewController.

La GUI usufruirà degli stessi script lanciati da riga di comando garantendo le stesse funzionalità secondo i dettami della OOP, ossia riutilizzo del codice funzionante anziché modifiche dello stesso o scriverne uno nuovo.
L'applicazione è composta da tre **moduli**:

1. Preparazione dei ligandi e dei recettori
2. Esecuzione del docking
3. Analisi dei risultati del docking.

Figura (1.4): QR code della repository di GitHub di AUDO4RIAS

1.8 Contenuto della tesi

La tesi è divisa in tre moduli:

1. nel primo modulo saranno discusse le tecnologie, le piattaforme scelte per la realizzazione del software, i linguaggi di programmazione e gli strumenti di bioinformatica utilizzati
2. nel secondo modulo sarà illustrata l'applicazione realizzata, dalla preparazione dei ligandi e recettori, passando per il docking, finendo con l'estrazione dei legami dall'output ottenuto

3. nell'ultimo modulo saranno trattate le conclusioni e saranno indicati gli sviluppi futuri del software realizzato.

Figura (1.5): Diagramma implementativo del software

Capitolo 2

Tecnologie e piattaforme

In questo capitolo saranno trattate le tecnologie utilizzate, a partire dai linguaggi, i tools impiegati e le piattaforme hardware di lavoro. È stato fondamentale l'utilizzo di macchine in remoto per eseguire le operazioni di training del modello. In questo caso le macchine messe a disposizione dall'Università Parthenope e dal CNR sono state fondamentali in quanto hanno garantito stabilità ed efficienza.

2.1 Python

La scelta del linguaggio di programmazione è stata importante in quanto ci sono tantissimi linguaggi con librerie adatte allo scopo, solo pochi ricevono costante supporto e sono anche utilizzati nel mondo del lavoro.
Il linguaggio scelto per lo sviluppo di **AUDO4RIAS** è il **Python**.
Python è un linguaggio informatico facile da imparare che si sta affermando tra gli scienziati, probabilmente perché è semplice da usare e, sufficientemente potente per raggiungere la maggior parte degli obiettivi di programmazione. Con **Python** si può iniziare a programmare in modo molto rapido. Riviste come Computing in Science and Engineering, Briefings in Bioinformatics e PLOS Computational Biology hanno pubblicato articoli introduttivi sull'impiego del **Python** per lo sviluppo di software di bioInformatica. Gli scienziati utilizzano Python per la visualizzazione molecolare, l'annotazione genomica, la manipolazione dei dati e innumerevoli altre applicazioni. Nel caso par-

ticolare delle scienze della vita, lo sviluppo di **Python** è stato molto importante.[7]
Il linguaggio di programmazione **Python** è dinamico ed orientato agli oggetti, inoltre è utilizzabile per molti tipi di sviluppo software. Offre un forte supporto all'integrazione con altri linguaggi e programmi, è fornito di una estesa libreria standard e può essere imparato in pochi giorni.
Python è dotato delle librerie di bioinformatica adatte allo scopo del software realizzato (**AUDO4RIAS**). Di seguito le principali utilizzate:

- **PubChemPy**, libreria utilizzata nelle ricerche chimiche per nome, sottostruttura e somiglianza, standardizzazione chimica, conversione tra formati di file chimici, rappresentazione e recupero delle proprietà chimiche
- **ProDy**, libreria utilizzata per l'analisi della dinamica strutturale delle proteine
- **Pandas**, libreria utilizzata per la manipolazione ed analisi dei dati
- **Tkinter/customTkinter**, libreria utilizzata per la realizzazione dell'interfaccia grafica
- **MolKit**, pacchetto che fornisce classi per leggere le molecole in diversi formati di file (PDB, Mol2...) e costruire una struttura gerarchica ad albero che riproduce la struttura interna della molecola
- **Matplotlib**, libreria utilizzata per la creazione di visualizzazioni statiche, animate e interattive
- **NumPy**, libreria utilizzata per la gestione delle strutture dati
- **Os**, libreria utilizzata per la gestione dei files

La versione di Python utilizzata è la **2.7** in quanto maggiormente compatibile con le librerie e i pacchetti utilizzati.

Figura (2.1): Logo di Python

2.1.1 PubChemPy

La libreria **PubChemPy** si basa interamente sul database **PubChem** e sui toolkit chimici forniti tramite il servizio web PUG REST. Questo servizio fornisce un'interfaccia ai programmi per svolgere automaticamente le operazioni che altrimenti si potrebbero eseguire manualmente tramite il sito web di **PubChem**. È importante ricordare questo aspetto quando si utilizza **PubChemPy**: ogni richiesta effettuata viene trasmessa ai server di **PubChem**, dove viene valutata e quindi viene inviata una risposta. Ci sono alcuni aspetti negativi in questo processo:

- è meno adatto per lavori riservati
- richiede una connessione costante a Internet
- alcune operazioni saranno più lente rispetto a quelle eseguite localmente sul proprio computer.

D'altra parte, questo significa che si hanno a disposizione le vaste risorse del database **PubChem** e dei toolkit chimici. Di conseguenza, è possibile eseguire in pochi secondi complesse ricerche di somiglianza e di sottostruttura su un database contenente decine di milioni di

composti, senza bisogno dello spazio di memoria o della potenza di calcolo del proprio computer locale [8].

2.1.2 ProDy

ProDy è un pacchetto python per l'analisi della dinamica delle proteine basata sulla struttura. **ProDy** consente di caratterizzare quantitativamente le variazioni strutturali in insiemi eterogenei di strutture risolte sperimentalmente per un dato sistema biomolecolare e di confrontare queste variazioni con le dinamiche di equilibrio previste teoricamente. Gli insiemi di dati comprendono insiemi strutturali per una determinata famiglia o sottofamiglia di proteine, i loro mutanti e omologhi di sequenza, in presenza/assenza dei loro substrati, ligandi o inibitori. Numerose funzioni ausiliarie consentono l'analisi comparativa dei dati sperimentali e teorici e la visualizzazione dei principali cambiamenti nelle conformazioni accessibili nei diversi stati funzionali. L'interfaccia di programmazione delle applicazioni (API) di **ProDy** è stata progettata in modo che gli utenti possano facilmente estendere il software e implementare nuovi metodi[9].

2.1.3 Pandas

Pandas è un pacchetto **Python** che fornisce strutture di dati veloci e flessibili, progettate per rendere facile e intuitivo il lavoro con i dati "relazionali" o "strutturati". **Pandas** è la pietra miliare per l'analisi pratica ad alto livello dei dati del mondo reale in **Python**. Inoltre, ha l'obiettivo più ampio di diventare il più potente e flessibile strumento open source di analisi e manipolazione dei dati disponibile in qualsiasi linguaggio[10].

2.1.4 Tkinter

La libreria **Tkinter** fornisce alle applicazioni **Python** un'interfaccia utente facile da programmare. Tkinter supporta una collezione di widget Tk che supportano la maggior parte delle applicazioni. **Tkinter** è l'interfaccia Python a Tk. Il toolkit GUI per Tcl/Tk. Tcl/Tk è la struttura di scripting e grafica sviluppato da John Ousterhout,

originariamente all'Università della California a Berkeley e successivamente presso Sun Microsystems. Attualmente, Tcl/Tk è sviluppato e supportato dalla Scriptics Corporation, fondata da Ousterhout. Tcl/-Tk gode di un notevole seguito di sviluppatori in diversi campi in diversi campi, prevalentemente su sistemi UNIX, ma più recentemente su sistemi Win32 e MacOS.
Tcl/Tk è stato inizialmente progettato per essere eseguito sotto il sistema X Window e i suoi widget e le sue finestre sono stati realizzati in modo da assomigliare a Motif. Anche il comportamento dei controlli e dei collegamenti è stato progettato per imitare Motif. Nelle versioni recenti di Tcl/Tk (in particolare, la release 8.0 e successive), i widget assomigliano ai widget nativi dell'architettura implementata. In effetti, molti dei widget sono widget nativi e la tendenza ad aggiungerne altri probabilmente continuerà. Come le estensioni di **Python**, Tcl/-Tk è implementato come un pacchetto di librerie C con moduli che supportano script interpretati o applicazioni. L'interfaccia **Tkinter** è implementata come un modulo **Python**[11].

2.1.5 Matplotlib

Matplotlib è un pacchetto per la creazione di diagrammi e immagini in 2D, finalizzato principalmente alla visualizzazione di dati scientifici, ingegneristici e finanziari. **Matplotlib** può essere utilizzato in modo interattivo dalla shell **Python** o richiamato da come funzione **Python** o incorporato in un'applicazione GUI (GTK, Wx, Tk, Windows). Sono supportati molti formati tra cui JPEG, PNG, PostScript e SVG. Le caratteristiche includono la creazione di più assi e figure per pagina, la navigazione interattiva, molte linee predefinite, molti stili e simboli predefiniti, immagini, antialiasing, alpha blending, diagrammi di dati e finanziari, gestione dei font conformi al W3C e supporto di FreeType2, legende e figure, tabelle, grafici, testo matematico e altro ancora[12].

2.1.6 NumPy

Nel mondo **Python**, gli array **NumPy** sono la rappresentazione standard dei dati numerici e consentono un'implementazione efficiente

dei calcoli numerici in un linguaggio di alto livello. Le prestazioni di **NumPy** possono essere migliorate grazie a tre tecniche: vettorializzazione dei calcoli, evitare di copiare i dati in memoria e minimizzare il numero di operazioni[13].

2.2 ADFRsuite

AutoDockFR (o **ADFR** in breve) è un programma per il **docking tra proteine e ligandi** sviluppato nel laboratorio Sanner dello Scripps Research sotto l'egida di *AutoDock*.
ADFR utilizza:

- la funzione di scoring di *AutoDock4*, implementata in una libreria C++ ed inglobata in Python
- un proprio algoritmo genetico in grado di evolvere e ottimizzare più soluzioni simultaneamente, di gestire un gran numero di legami ruotabili e di terminare le iterazioni al momento della convergenza, cioè potenzialmente senza esaurire il numero assegnato di valutazioni della funzione energia
- una rappresentazione generica di flessibilità molecolare, chiamata *Albero di Flessibilità*, che consente di incorporare una varietà di movimenti molecolari sia nel ligando che nel recettore.

Pur supportando le modalità di docking disponibili in *AutoDock4* e *Vina*, è stato progettato specificamente per includere la **flessibilità del recettore** e supporta anche il **docking covalente**. Il suo algoritmo genetico personalizzato consente il docking di ligandi con più legami ruotabili rispetto ad *AutoDock4*.
Viene distribuito come parte della suite del software *ADFR*, che fornisce strumenti aggiuntivi per facilitare il docking automatizzato.
ADFR implementa diverse caratteristiche che aiutano a semplificare la procedura di docking e a supportare la gestione e la riproducibilità degli esperimenti di docking attraverso la prova dei dati. Vengono utilizzati file autodocumentati per memorizzare:

- la rappresentazione del sito di binding (cioè **i file .trg target**)
- i risultati del docking (file **.dro Docking Results Object**)
- I metadati memorizzati in questi file non solo supportano la riproducibilità, ma riducono anche i rischi di errori dell'operatore.

ADFR legge i **ligandi** preparati per il docking con AutoDock, cioè nel formato PDBQT e organizza la flessibilità del ligando in base ai legami ruotabili. Un file PDBQT può essere generato da un file *.pdb* di un ligando utilizzando il comando *prepar_ligand*.
Il **recettore** è specificato come file target, cioè un singolo file che descrive il recettore. I file target possono essere calcolati per un recettore nel formato PDBQT dal comando utilizzando il programma ***agfr*** o l'interfaccia grafica ***agfrgui***. Un file PDBQT può essere generato da un file pdb di un recettore utilizzando il comando *prepare_receptor* della suite *ADFR*.
ADFR è implementato nei moderni linguaggi di programmazione orientati agli oggetti e si basa su componenti software riutilizzabili. I componenti critici per le prestazioni sono implementati in C e C++ (ad esempio ADFRcc), mentre altri sono implementati in Python (ADFR, AutoSite, MolKit2, ProDy). *ADFR* è rilasciato sotto la licenza open source LGPL v2.
In particolare per l'applicazione trattata sono stati utilizzati i due script: **prepare_ligand4** e **prepare_receptors4**, rispettivamente per la preparazione dei ligandi e dei recettori e per effettuare la loro conversione dal formato *.pdb* al formato *.pdbqt* necessario per la procedura di docking.

2.2.1 MGLTools

La suite software **MGLTools** è stata sviluppata nel laboratorio Sanner presso il Center for Computational Structural Biology (CCRB) precedentemente noto come Molecular Graphics Laboratory (MGL) dello Scripps Research Institute per la visualizzazione e l'analisi delle strutture molecolari. MGLTools comprende:

- **Python Molecular Viewer (PMV)**, un visualizzatore molecolare di uso generale
- **AutoDockTools (ADT)**, un insieme di comandi PMV sviluppati specificamente per supportare gli utenti di AutoDock
- **Vision**, un ambiente di programmazione visuale.

Questi strumenti software sono altamente integrati e basati su componenti software riutilizzabili implementati in Python e C++ (con binding Python). Il kit di strumenti grafici sottostante è Tk (Tkinter). L'ultima versione di MGLtools è la 1.5.7 che fornisce:

- un widget della dashboard ridisegnato
- un widget per la visualizzazione delle sequenze
- ottimizzazioni delle prestazioni
- nuovi comandi per i calcoli di sovrapposizione e RMSD

2.2.2 Open Babel

Open Babel è un tool per applicazioni di chimica progettato per interpretare i molteplici formati dei dati chimici e per cercare, convertire, analizzare o archiviare dati da modellistica molecolare, chimica, materiali a stato solido, biochimica o aree correlate.
La versione di OpenBabel 2.3 converte fino a 110 formati di file chimici.
I database sono ampiamente utilizzati per memorizzare le informazioni chimiche soprattutto nell'industria farmaceutica. Un requisito fondamentale di un database di questo tipo è la capacità di indicizzare le strutture strutture chimiche in modo che possano essere recuperate rapidamente, data una query di ricerca. Open Babel offre questa funzionalità utilizzando un'indicizzazione basata sul percorso. Questa indicizzazione, FP2 in Open Babel, identifica tutte le sottostrutture lineari e ad anello della molecola di lunghezza da 1 a 7 (escluse le sottostrutture a 1 atomo C e N) e le mappa in una stringa di bit di lunghezza su una stringa di bit di lunghezza 1024 utilizzando una

funzione di hash. Se una molecola interrogata è una sottostruttura di una molecola di destinazione, allora tutti i bit impostati nella molecola di query saranno impostati anche nella molecola di destinazione. Le indicizzazioni di due molecole possono anche essere utilizzate per calcolare la somiglianza strutturale utilizzando il coefficiente di Tanimoto, il numero di bit in comune diviso per tutti i bit dell'insieme.
Chiaramente, la ricerca iterativa dello stesso insieme di molecole comporterà l'uso ripetuto dello stesso insieme di indicizzazioni. Per evitare la necessità di ricalcolare le indicizzazioni per un particolare file multi-molecola (come un file SDF), Open Babel fornisce un formato fastindex che memorizza esclusivamente un'indicizzazione insieme a un indice nel file originale. Questo indice porta a un rapido aumento della velocità di ricerca di corrispondenze a a fronte di una query: insiemi di dati con diversi milioni di molecole sono facilmente consultabili in modo interattivo. In questo modo un file multi-molecola può essere utilizzato come un'alternativa efficace a un sistema di database chimico[14].

Figura (2.2): Logo di Open Babel

2.3 Database

I dati da cui attinge **AUDO4RIAS** provengono da due database:

- **PubChem** per i ligandi
- **RCSB** per le proteine.

2.3.1 RCSB

Il Research Collaboratory for Structural Bioinformatics Protein Data Bank (RCSB), è il centro dati statunitense per l'archivio di file PDB di uso globale, rendendo i dati PDB liberamente disponibili a tutti gli utenti, dai biologi strutturali a quelli computazionali e non solo[15].
I depositanti di dati PDB comprendono biologi strutturali che utilizzano la cristallografia macromolecolare, la spettroscopia di risonanza magnetica nucleare e la microscopia elettronica 3D. I consumatori di dati PDB sono ricercatori, educatori e studenti che studiano biologia fondamentale, biomedicina, biotecnologia ed energia. La recente riorganizzazione delle attività del PDB dell'RCSB in quattro servizi integrati e interdipendenti è descritta in dettaglio, insieme agli strumenti e alle risorse aggiunti negli ultimi due anni ai portali web del PDB dell'RCSB a sostegno di una "visione strutturale della biologia"[16].

Figura (2.3): Logo di RCSB

2.3.2 Pubchem

PubChem è il più grande database al mondo di informazioni chimiche liberamente accessibili, attraverso il quale è possibile cercare le sostanze chimiche per nome, formula molecolare, struttura e altri identificatori. Inoltre è possibile trovare proprietà chimiche e fisiche, attività biologiche, informazioni sulla sicurezza e sulla tossicità, brevetti, citazioni bibliografiche e altro ancora. L'interfaccia tra l'applicazione ed il database è realizzata mediante la libreria di Python **PubChemPy**.
PubChem è un database di molecole chimiche, gestito dal centro nazionale per l'Informazione biotecnologica statunitense (NCBI), parte della biblioteca nazionale di medicina (NLM) dell'istituto nazionale

della sanità americano (NIH). L'accesso al database PubChem può essere eseguito liberamente attraverso un sito web e possono essere scaricati dati riguardanti milioni di strutture di composti e dati descrittivi tramite il protocollo FTP. PubChem possiede descrizioni di molecole con meno di 1000 atomi e 1000 legami.
PubChem gestisce i dati in tre database interconnessi: **Substance, Compound e BioAssay**. Il database Substance archivia le descrizioni delle sostanze chimiche fornite dai depositanti.
Il database Compound archivia le strutture chimiche uniche estratte dal database Substance attraverso la standardizzazione delle strutture.
Il database BioAssay contiene la descrizione e i risultati degli esperimenti di analisi biologica.

Figura (2.4): Logo di PubChem

2.4 Autodock

AutoDock è un programma di docking che utilizza un algoritmo genetico, il *Lamarckian Genetic Algorithm*, per il calcolo della pose migliore che interagisce con il sito attivo della proteina. Dopo aver calcolato inizialmente una popolazione di possibili soluzioni, l'algoritmo ne selezionerà una parte in base alle funzioni di scoring e darà origine a una nuova popolazione di soluzioni figlie, da cui avrà inizio un secondo ciclo di generazione e così via. In questo modo il "genotipo", ovvero

la stringa binaria a cui corrisponde ciascun ligando, verrà influenzato da fattori esterni, esattamente come nell'ipotesi lamarckiana.
L'interazione tra ligando e recettore è valutata in due fasi, calcolando la variazione di energia intramolecolare del passaggio dalla forma libera a quella legata e la variazione di energia libera intermolecolare implicata nello stesso passaggio.
Per effettuare il docking è necessario per prima cosa preparare le coordinate di ligando e recettore. La preparazione delle coordinate è la fase più importante nella procedura, poiché in esse sono inclusi parametri fondamentali come: idrogeni polari, atom–type e cariche parziali. Le coordinate del ligando originale e della macromolecola sono trattate separatamente ed i loro file sono in un formato particolare, il PDBQT.

Figura (2.5): Logo di Autodock

2.5 Autodock Vina

AutoDock Vina è un nuovo programma per il docking molecolare e lo screening virtuale. Vina rappresenta la nuova versione di AutoDock, infatti presenta molte similitudini con il suo predecessore ma, allo stesso tempo, anche molte differenze. Una differenza importante consiste nella velocità di calcolo, dato che Vina è molto poco dispendioso sotto questo punto di vista; altra differenza fondamentale è rappresentata dal fatto che al momento del calcolo della griglia, Vina calcola internamente ed automaticamente le "grid maps" impiegate in AutoDock. Questo costituisce un grande vantaggio in termini di facili-

tà e velocità di esecuzione. Inoltre, le funzioni di scoring e gli algoritmi utilizzati in questo tipo di analisi risultano essere completamente diversi rispetto al suo predecessore, cosa che porta a considerare Vina quasi come un software a sé stante. Vina migliora al tempo stesso in modo significativo l'accuratezza delle previsioni delle modalità di legame. Un'ulteriore miglioramento è ottenuta grazie al parallelismo, che utilizza il multithreading su macchine multicore. AutoDock Vina calcola automaticamente le mappe della griglia e raggruppa i risultati in modo trasparente per l'utente. Autodock vina all'interno del software prende in input i ligandi ed i recettori in formato *.pdbqt*[17].

Figura (2.6): Logo di Autodock Vina

2.5.1 Funzione di scoring

La funzione di scoring prende come input una posa e restituisce un valore che rappresenta la bontà della posa, intesa in termini energetici favorevoli. La maggior parte delle funzioni di scoring sono basate sui campi di forza delle interazioni molecolari che valutano l'energia di ogni posa: un buon valore (che varia a seconda della funzione usata) indica una posa stabile. Un approccio alternativo è derivare un potenziale statistico per le interazioni sulla base di un database di complessi proteina-ligando, come la Protein Data Bank e valutare il punteggio della posa secondo questi potenziali. Funzioni di scoring basate su questi presupposti riescono a modellare correttamente dei ligandi altamente affini con la proteina sebbene il rischio di ottenere dei falsi positivi in questa fase è molto alto. Un modo per eliminare questi falsi positivi è quello di valutare le migliore pose con ulteriori

funzioni di scoring più accurate.
La funzione di scoring di AutoDock Vina (qui indicata come Vina) può essere rappresentata attraverso la seguente formula:

$$c = \sum_{i<j} f_{t_i t_j}(r_{ij}) \tag{2.1}$$

dove la sommatoria è su tutte le coppie di atomi che possono muoversi l'uno rispetto all'altro, escludendo normalmente le interazioni 1-4, ovvero gli atomi separati da tre legami covalenti consecutivi. Ad ogni atomo i viene assegnato un tipo t_i e un insieme simmetrico di funzioni di interazione $f_{t_i t_j}$ della distanza interatomica r_{ij} da definire.
Questo valore può essere visto come una somma di contributi intermolecolari e intramolecolari:

$$c = c_{inter} + c_{intra} \tag{2.2}$$

L'algoritmo di ottimizzazione, descritto in seguito, cerca di trovare il minimo globale di c e di altre conformazioni a basso punteggio, che poi classifica.
L'energia libera di legame prevista viene calcolata a partire dalla parte intermolecolare della conformazione con il punteggio più basso, designata come:

$$s_1 = g(c_1 - c_{intra1}) = g(c_{inter1}) \tag{2.3}$$

dove la funzione g può essere una funzione arbitraria strettamente crescente possibilmente non lineare.
Nell'output le altre conformazioni a basso punteggio vengono formalmente restituite dai valori di s, ma per preservare il ranking, si utilizza c_{intra} come migliore modalità di legame:

$$s_i = g(c_i - c_{intra1}) \tag{2.4}$$

Per ragioni di modularità, gran parte del programma non fa riferimento ad alcuna forma funzionale delle interazioni $f_{t_i t_j}$ o g. Essenzialmente, queste funzioni vengono passate come parametro per il resto

del codice.
Inoltre, il programma è stato progettato in modo tale da poter utilizzare schemi di tipizzazione degli atomi come la tipizzazione degli atomi di AutoDock4 o SYBIL.

Pesi	**Termini**
-0.0356	$gauss_1$
-0.00516	$gauss_2$
0.840	$repulsion$
-0.0351	$hydrophobic$
-0.587	$hydrogenbonding$
0.0585	N_{rot}

Tabella (2.1): Funzione di scoring pesi e termini

La particolare implementazione della funzione di scoring che verrà presentata è stata ispirata principalmente da X-score e come tale è stato messo a punto utilizzando PDBbind. Tuttavia, alcuni termini sono diversi da X-score e, nel mettere a punto la funzione di scoring, si è andati oltre la regressione lineare. Inoltre, va notato che Vina classifica le conformazioni secondo l'eq. (2.2) o, equivalentemente, eq. (2.4), mentre X-score conta solo i contributi intermolecolari. Per quanto ne sappiamo, X-score non è stato implementato in un programma di docking, ignorare i vincoli interni potrebbe portare l'algoritmo di ottimizzazione a ricercare strutture corrotte all'interno.
La derivazione della nostra funzione di scoring combina alcuni vantaggi tra quelli potenzialmente conosciuti e le funzioni di scoring empiriche: estrae informazioni empiriche da entrambe le preferenze conformazionali del sia dei complessi recettore-ligando sia dalle misure sperimentali affini.
Lo schema di tipizzazione degli atomi segue quello di X-score. Gli atomi di idrogeno non sono considerati esplicitamente, se non per la tipizzazione degli atomi, e sono omessi dall'eq. (2.1).
Le funzioni di interazione $f_{t_i t_j}$ sono definite rispetto alla distanza di superficie $d_{ij} = r_{ij} - R_{ti} - Rtj$:

$$f_{t_i t_j}(r_{ij}) \equiv h_{t_i t_j}(d_{ij}) \tag{2.5}$$

dove R_t è il raggio di van der Waals dell'atomo di tipo t.
Nella nostra funzione di scoring, $h_{t_i t_j}$ è una somma ponderata di interazioni steriche (i primi tre termini nella tabella 2.1), identica per tutte le coppie di atomi, interazione idrofobiche tra atomi idrofobici e, dove possibile, legami a idrogeno. I pesi sono mostrati nella tabella 2.1. I termini sterici sono i seguenti:

$$gauss_1(d) = e^{-(d/0.5\mathring{A})^2} \tag{2.6}$$

$$gauss_2(d) = e^{-((d/0.5\mathring{A})/2\mathring{A})^2} \tag{2.7}$$

$$repulsion(d) = \begin{cases} d^2, se & d < 0 \\ 0, se & d \geq 0 \end{cases} \tag{2.8}$$

Il termine idrofobico è uguale a 1, quando $d < 0,5\mathring{A}$; 0, quando $d > 1,5\mathring{A}$, ed è interpolato linearmente tra questi valori. Il termine di legame a idrogeno è uguale a 1, quando $d < -0,7\mathring{A}$; 0, quando $d < 0,5\mathring{A}$, e viene interpolato linearmente tra questi valori. Seguendo Xscore, trattiamo formalmente i metalli come donatori di legami a idrogeno. In questa in questa implementazione, tutte le funzioni di interazione $f_{t_i t_j}$ sono tagliate a $r_{ij} = 8\mathring{A}$.
La Figura 1 mostra i termini sterici ponderati da soli o combinati con i termini idrofobici o H con le interazioni idrofobiche o di legame H[18].

Figura (2.7): Funzione di scoring pesata

Capitolo 3

Applicazione realizzata

Questo capitolo tratta dei moduli dell'applicazione, come sono stati implementati ed il loro funzionamento, le modalità di installazione e di utilizzo del software, il formato dei dati in input ed output trattati.

3.1 Installazione

L'applicazione realizzata, **AUDO4RIAS**, effettua il docking tra i ligandi contenuti in specifici pesticidi e i recettori dell'*Apis mellifera*, infine effettua l'estrazione dei legami che si vengono a formare.
L'applicazione viene scaricata dalla repository di github mediante il comando:

```
1 git clone https://github.com/mungowz/AUDO4RIAS.git
```

Listato (3.1): Comando per scaricare la repository

Verrà quindi installata nella directory corrente la repository contenente il progetto, all'interno di questa si trovano gli script utilizzati dall'applicazione e le directory ed i file di input di default, tra cui:

- Il file di input dei ligandi, ovvero *ligands_list.txt*
- La directory contenente la lista di default dei ligandi, ovvero */data/files/*
- La directory di default che conterrà i file *.xlsx* di output prodotti, ovvero */output/excel_files*

- La directory di default che conterrà i file *.sdf* prodotti, ovvero */data/ligands/sdf/*
- Le directory di default che conterranno i file *.pdb* prodotti, ovvero */data/ligands/pdb/* per i ligandi e */data/proteins/pdb/* per i recettori
- Le directory di default che conterranno i file *.pdbqt prodotti*, ovvero */data/ligands/pdbqt/* per i ligandi e */data/proteins/pdbqt/* per i recettori.

Sarà necessario installare alcune dipendenze esterne (software esterni) ed interne (librerie e pacchetti di python) per far funzionare l'applicazione, la lista delle dipendenze e la procedura di installazione è specificata nel file *README* della repository di github.

3.2 Modalità di utilizzo

AUDO4RIAS può essere utilizzato in due modalità:

1. Mediante script python da terminale
2. Mediante interfaccia grafica realizzata seguendo il paradigma *model-view-controller*.

Figura (3.1): Schema del paradigma model-view-controller

Non sarà necessario effettuare ulteriori installazioni per usufruire di entrambe le modalità ma bisognerà semplicemente scaricare la repository e seguire le istruzioni del *README*.
Per eseguire l'applicazione da riga di comando devono essere eseguiti separatamente nell'ordine i seguenti script:

- **prepare_ligands.py**
- **prepare_receptors.py**
- **performDocking.sh**
- **detect_interactions.py**

La GUI usufruirà degli stessi script eseguiti da riga di comando, garantendo così le stesse funzionalità in entrambe le modalità di utilizzo e seguendo i dettami della **OOP** secondo i quali bisogna riutilizzare codice funzionante anzichè modificarlo o scriverlo ex-novo.
Per avviare la GUI deve essere digitato nel terminale il comando:

```
python app.py
```

Listato (3.2): Comando per avviare la GUI

Avviata la GUI si aprirà la home page con le seguenti opzioni:

- Opzione **Preparation**: si accede alla sezione relativa alla preparazione degli input
- Opzione **Docking**: sarà possibile effettuare il docking
- Opzione **Analysis**: sarà possibile effettuare l'analisi dei risultati del docking
- Opzione **Help**: si accede alla sezione relativa alle informazioni utili per l'utente per l'uso del software.

Figura (3.2): Sezione Home Page della GUI

Attraverso il menù a tendina in basso a sinistra sarà possibile scegliere il tema dell'applicazione tra:

- Tema chiaro
- Tema scuro
- Tema del sistema.

Figura (3.3): Tema scuro della GUI

3.3 Dati in input

I dati in input all'applicazione trattata sono: ligandi e proteine.
La lista dei ligandi è fornita in input tramite foglio calcolo (*.xlsx*, *.xls*) o mediante file di testo (*.txt*), in entrambi i casi ogni riga corrisponde al nome di un ligando. Essendo l'applicazione incentrata sullo studio

degli effetti dei ligandi dei pesticidi sull'*Apis mellifera*, come dati di esempio sono state utilizzati i ligandi le cui molecole costituiscono i pesticidi maggiormente diffusi sul mercato, per un totale di 297 ligandi. La lista è disponibile nell'appendice A.1.
I recettori vengono selezionati mediante una **WebView** aperta sulla pagina di ricerca del sito del database RCSB, da qui l'utente digiterà la propria query e, dopo aver selezionato il tasto **search** gli verranno mostrati tutti i composti organici relativi alla query digitata, tramite il tasto **Get Query** l'utente andrà a scaricare tutti i file dei composti organici in formato *.pdb*.
Nel caso di esempio sono state scelte le proteine dei recettori dell'*Apis Mellifera* secondo la query mostrata nelle foto: 3.4 e 3.5, per un totale di 11 file *.pdb* contenenti le strutture di determinate proteine. La lista delle proteine scaricate è disponibile nell'appendice: B.1. L'applicazione usata tramite GUI dispone della sezione **Help** (3.6) contenente le informazioni relative alle impostazioni di default e al formato dei dati in input.

Figura (3.4): Query dei recettori

Figura (3.5): Proteine dei recettori dell'apis mellifera

Figura (3.6): Sezione Help della GUI

3.4 Preparazione dei ligandi e recettori

Il primo step propedeutico per il docking è la preparazione dei ligandi e dei recettori, questa fase viene esplicitamente eseguita dal software realizzato. L'intera fase è svolta:

- per i ligandi dallo script python **prepare_ligands.py**
- per i recettori dallo script python **prepare_receptors.py**.

Se l'applicazione viene utilizzata mediante GUI la preparazione dei ligandi e dei recettori può essere effettuata mediante la sezione **Preparation** della pagina principale come mostrato in figura 3.7.

- Selezionando l'opzione **Ligands** si accede alla sezione relativa alla preparazione dei ligandi
- Selezionando l'opzione **Receptors** si accede alla sezione relativa alla preparazione dei recettori
- Selezionando l'opzione **Back** si ritorna alla pagina principale.

Figura (3.7): Sezione Preparation della GUI

3.4.1 Preparazione dei ligandi tramite script

Lo script python **prepare_ligands.py**, che esegue tale fase, viene eseguito da terminale mediante il comando:

```
python prepare_ligands.py
```

Listato (3.3): Comando per scaricare la repository

Questo script può ricevere diversi argomenti in input:

- [$-v$]: serve per attivare il verbose e se non viene specificata tale opzione viene lasciata di default inattivo
- [$-e$]: specifica un nuovo file di input (*.xlsx*, *.xls*) o (*.txt*) da cui prendere i nomi dei ligandi, se non viene specificata tale opzione verrà scelto il file di default scaricato insieme al software
- [$-E$]: specifica la directory dei file *.xlsx* di output, se non viene specificata tale opzione verrà scelta la directory di default
- [$-s$]: specifica la directory dei file *.sdf* di output, se non viene specificata tale opzione verrà scelta la directory di default
- [$-P$]: specifica la directory dei file *.pdb* di output, se non viene specificata tale opzione verrà scelta la directory di default
- [$-p$]: specifica la directory dei file *.pdbqt* di output, se non viene specificata tale opzione verrà scelta la directory di default
- [$-k$]: specificando questa opzione viene scelto di non cancellare i file dei ligandi precedentemente scaricati, se non viene specificata tale opzione i file verranno cancellati
- [$-h$]: stampa la spiegazione degli input per lo script.

Lo script quando eseguito andrà ad inizializzare i vari percorsi, nel caso siano stati inseriti percorsi diversi da quelli di default verrà controllata l'esistenza e la validità degli stessi. Nel caso non siano presenti le directory contenenti i file *.sdf*, *.pdb* e *.pdbqt*, queste verranno create automaticamente dall'applicazione, inoltre verranno cancellati i file precedentemente scaricati a meno di input diversi.
Lo script richiama la funzione **selectLigands** la quale si occupa di scaricare da **Pubchem** i file *.sdf* corrispondenti alla lista dei ligandi in input. La funzione prende in input:

- *input_path*, il path del file di input
- *sdf_folder*, il path della directory di output per i file *.sdf*

- *excel_folder*, il path della directory di output per i file *.xlsx*
- *verbose*, il flag relativo al verbose.

In output sono restituiti i file *.sdf* corrispondenti ai ligandi in input, dove il nome di ogni file è preceduto dal suffisso *ligand_*, e un file *.xlsx* contenente i risultati dell'operazione di download nominato *ligands_sdf_output.xls*.

```
selectLigands(input_path, sdf_folder, excel_folder, verbose)
```

Listato (3.4): funzione selectLigands

L'interfaccia con il database **PubChem** avviene tramite il pacchetto **PubChemPy**, in particolare la funzione **pubchempy.get_compounds** permette di ricercare nel database il record relativo allo specifico ligando il cui nome, a cui viene precedentemente aggiunto il suffisso *ligand_*, viene fornito in input. La funzione prende in input:

- *identifier*, il composto da ricercare nel database
- *namespace*, il parametro in base al quale ricercare il composto, nel nostro caso il parametro scelto è il nome indicato dalla stringa *"name"*
- *record_type* ovvero il tipo di record relativo al composto in questione da scaricare, nel nostro caso il parametro scelto è la stringa *"3d"* che indica la struttura 3D del ligando.

In output sarà prodotto il record della struttura 3D del ligando in questione.

```
pubchempy.get_compounds(identifier, "name", record_type="3d")
```

Listato (3.5): funzione pubchempy.get_compounds

La funzione che si occupa di scaricare la struttura 3D del ligando in formato *.sdf*, richiamata da **selectLigands** è: **pubchempy.download**, la quale prende in input:

- *outformat*, il formato del file di output, nel nostro caso il parametro scelto è il formato *.sdf* indicato dalla stringa *"SDF"*

- *path*, il path della directory in cui vengono scaricati i file *.sdf*
- *identifier*, il composto da scaricare nel database
- *namespace*, il parametro in base al quale ricercare il composto, nel nostro caso il parametro scelto è il nome indicato dalla stringa *"name"*
- *record_type* ovvero il tipo di record relativo al composto in questione da scaricare, nel nostro caso il parametro scelto è la stringa *"3d"* che indica la struttura 3D del ligando.

L'output di tale funzione sarà il file *.sdf* del ligando in questione.

```
1 pubchempy.download("SDF", path, identifier, "name", record_type="3d"
    )
```

Listato (3.6): pubchempy.download

I nomi di alcuni file potrebbero non essere presenti nel database oppure, a causa di spazi presenti nel loro nome, non essere riconosciuti. Nel primo caso i ligandi vengono semplicemente scartati nel secondo caso gli spazi vengono sostituiti da underscore (_), seguendo quindi la nomenclatura del database, e viene effettuata nuovamente la ricerca di questi nel database. Se ancora la ricerca non ha successo il ligando in questione viene definitivamente scartato.
Oltre ai file *.sdf* la **selectLigands** produrrà anche un file *.xlsx* contenente il riepigolo dei risultati ottenuti dalla funzione, ovvero:

- file scaricati
- file scaricati con il nome modificato
- ligandi non trovati all'interno del database.

Nel caso di esempio, dei 297 ligandi in input, 289 sono stati trovati e scaricati con successo, i restanti 8 sono stati scartati: 3.1.

Sodium 4-nitrophenolate	Silthiofa
Sodium 2-methoxy-5-nitrophenolate	Potassium bicarbonate
(E,E)-7,9-Dodecadien-1-yl acetate	Dodine
Sodium 2-nitrophenolate	Ziram

Tabella (3.1): Ligandi non scaricati

Dopo aver scaricato i file *.sdf* questi devono essere convertiti in formato *.pdb*, per fare ciò lo script richiama la funzione **sdf2pdb**, la quale prende in input:

- *sdf_folder*, la directory con i file *.sdf* in input
- *pdb_folder*, la directory con i file *.pdb* in output
- *verbose*, il flag relativo al verbose.

La funzione restituisce in output i file *.pdb* corrispondenti a tutti i file *.sdf* dati in input.

```
sdf2pdb(sdf_folder, pdb_folder, verbose)
```

Listato (3.7): funzione sdf2pdb

La funzione usufruisce del programma **Open babel** che esegue la conversione da *.sdf* a *.pdb* richiamando la sua versione da terminale tramite il comando **obabel**:

```
obabel sdf_file_path -O pdb_path
```

Listato (3.8): Comando per la conversione da .sdf a .pdb

Nell'istruzione sopra. *sdf_file_path* indica la directory con i file *.sdf* in input, la directory con i file *.pdb* in output, *pdb_path*, viene specificata tramite lo switch *-O*. Tramite tale istruzione è possibile effettuare la conversione di un singolo file, la conversione di tutti i *.sdf* avviene ciclando su tutti i file nella directory corrispondente.
L'ultimo step nella preparazione dei ligandi che è effettuato dalla funzione **prepareLigands**, la quale prende in input:

- *pdb_folder*, la directory con i file *.pdb* in input
- *pdbqt_folder*, la directory con i file *.pdbqt* in output
- *verbose*, il flag relativo al verbose.

La funzione restituisce in output la conversione in file *.pdbqt* dei corrispondenti i file *.pdb* dati in input.

```
prepareLigands(pdb_folder, pdbqt_folder, verbose)
```

Listato (3.9): funzione prepareLigands

La funzione **prepareLigands** utilizza lo script **prepare_ligand4** della suite **ADFR** tramite il quale effettua la corretta conversione da *.pdb* a *.pdbqt*. Lo script viene richiamato da riga di comando dalla funzione mediante il comando **prepare_ligand**:

```
prepare_ligand -l pdb_file_path -v -o pdbqt_path
```

Listato (3.10): Comando per la conversione da .pdb a .pdbqt

Nell'istruzione sopra, *pdb_file_path* indica la directory con i file *.pdb* in input ed è specificata tramite lo switch *-l*, lo switch *-v* indica che è attivato il verbose e la directory con i file *.pdbqt* in output, *pdbqt_path*, viene specificata tramite lo switch *-o*. Tramite tale istruzione è possibile effettuare la conversione di un singolo file, la conversione di tutti i *.pdb* avviene ciclando su tutti i file nella directory corrispondente.

3.4.2 Preparazione dei ligandi tramite GUI

La preparazione dei ligandi tramite GUI avviene selezionando il tasto **Ligands** nella sezione **Preparation** del software. All'interno della pagina relativa alla preparazione dei ligandi, premendo il tasto **Execute** è possibile effettuare i procedimenti spiegati nella sezione relativa all'esecuzione tramite script (3.4.1).
Premendo il tasto **Back** è possibile tornare alla sezione **Preparation** relativa alla preparazione degli input. Come si osserva nella figura 3.8 all'interno di tale sezione sono presenti delle **entry** dove è possibile specificare diversi input tra cui:

- un file di input (*.xlsx*, *.xls*) o (*.txt*) da cui prendere i nomi dei ligandi
- la directory dei file *.xlsx* di output
- la directory dei file *.sdf* di output
- la directory dei file *.pdb* di output

- la directory dei file *.pdbqt* di output.

Se queste entry non sono valorizzate, verrano impostate le configurazioni di default. E' presente anche un **checkbox** il quale, se selezionato, permette di mantenere i file precedentemente scaricati che altrimenti verrebbero cancellati.
Per andare a selezionare direttamente nel file system del PC i file e le directory richieste, sono stati implementati rispettivamente i tasti **Browse files** e **Browse folders**, come è visibile nelle figure 3.8, 3.9 e 3.10.

Figura (3.8): Sezione Ligands della GUI

Figura (3.9): Browse files

Figura (3.10): Browse folders

La GUI per la preparazione dei ligandi richiama lo script **prepare_ligands.py** discusso nella precedente sezione, adattando l'interfaccia grafica allo script senza eliminare alcuna funzionalità, precedentemente illustrata, per la versione da riga di comando e rispettando i dettami dell'**OOP**.
Durante l'esecuzione viene mostrato un **pannello** (3.11) in cui vengono descritte:

- le operazioni che si stanno effettuando
- eventuali errori ed avvisi per l'utente, in modo tale che sia restituito un feedback all'utente relativo al progresso della preparazione dei ligandi.

Al completamento di ogni fase sarà restituito un messaggio di avviso (figura: 3.12) ed in caso di input non valido sarà restituito un messaggio di errore (figura: 3.13).

Figura (3.11): Pannello dell'esecuzione dei ligandi

Figura (3.12): Messaggio di completamento della preparazione dei ligandi

Figura (3.13): Messaggio di errore

3.4.3 Preparazione dei recettori tramite script

Lo script python **prepare_receptors.py**, che esegue tale fase, viene eseguito da terminale mediante il comando:

```
python prepare_receptors.py
```

Listato (3.11): funzione prepare_receptors

Questo script può ricevere diversi argomenti in input:

- $[-v]$: serve per attivare il verbose e se non viene specificata tale opzione viene lasciata di default inattivo
- $[-E]$: specifica la directory dei file *.xlsx* di output, se non viene specificata tale opzione verrà scelta la directory di default
- $[-P]$: specifica la directory dei file *.pdb* di output, se non viene specificata tale opzione verrà scelta la directory di default
- $[-p]$: specifica la directory dei file *.pdbqt* di output, se non viene specificata tale opzione verrà scelta la directory di default
- $[-k]$: specificando questa opzione viene scelto di non cancellare i file delle proteine precedentemente scaricati, se non viene specificata tale opzione i file verranno cancellati
- $[-m]$: specificando questa opzione viene fornita in input la dimensione in *angstrom* del margine della *grid box* per le proteine se non viene specificata tale opzione viene impostato il valore di default ovvero 3

- $[-h]$: stampa la spiegazione degli input per lo script.

Lo script quando eseguito andrà ad inizializzare i vari percorsi, nel caso siano stati inseriti percorsi diversi da quelli di default verrà controllata l'esistenza e la validità degli stessi. Nel caso non siano presenti le directory contenenti i file *.txt*, *.pdb* e *.pdbqt*, queste verranno create automaticamente dall'applicazione, inoltre verranno cancellati i file precedentemente scaricati a meno di input diversi.
Lo script richiama la funzione **selectReceptors** la quale si occupa di scaricare da **RCSB** i file *.pdb* corrispondenti alle proteine scelte. La funzione prende in input:

- *pdb_folder*, il path della directory di output per i file *.pdb*
- *excel_folder*, il path della directory di output per i file *.xlsx*
- *verbose*, il flag relativo al verbose.

In output sono restituiti i file *.pdb* corrispondenti ai recettori scelti in input e un file *.xlsx* contenente i risultati dell'operazione di download nominato *info_proteins.xls*.

```
1 selectReceptors(pdb_folder, excel_folder, verbose)
```

Listato (3.12): funzione selectReceptors

All'interno di questa **selectReceptors** viene richiamata la funzione **RestApiSelection** la quale prende in input l'*URL* di RCSB:

```
1 RestApiSelection(URL)
```

Listato (3.13): funzione RestApiSelection

Questa funzione apre una **WebView** sul sito di RCSB come mostrato nella figura 3.4, l'utente potrà digitare la propria query e scaricare i composti scelti selezionando il tasto in alto **Get Query**.
La **web view** restituisce l'**URL** della pagina del database contenente i composti selezionati dalla query, il *JSON* di tale pagina viene estratto e convertito nella struttura dati *dizionario* di python tramite la funzione **loads** della libreria **json**.

```
dictionary = json.loads(data)
```

Listato (3.14): funzione json.loads

Nel *dizionario* restituito dalla funzione sarà contenuto l'elenco dei composti selezionati tramite la query.
Il software rieseguirà direttamente la query tramite la funzione **get** contenuta nella libreria **requests**. Questa funzione permette di inviare una richiesta di tipo *HTTP/1.1* prendendo in input **URL** della query, al quale viene aggiunto l'elenco di composti da scegliere in formato *JSON*, ciò eseguito convertendo il *dizionario* precedentemente ottenuto in **JSON** mediante la funzione **dump** della libreria **json**:

```
dictionary = json.dump(dictionary)
```

Listato (3.15): funzione json.dump

La funzione **get** restituisce la lista di proteine da scaricare:

```
requests.get(f"https://search.rcsb.org/rcsbsearch/v2/query?json={
    dictionary}")
```

Listato (3.16): funzione requests.get

La funzione **selectReceptors** richiama successivamente la funzione **downloadPdbs** che prende in input:

- *pdb_list*, la lista di proteine precedentemente ottenuta
- *outputh_path*, il path della directory di output per i file *.pdb*
- *verbose*, il flag relativo al verbose.

La funzione restituisce in output i file *.pdb* delle proteine contenute nella lista in input:

```
downloadPdbs(pdbs_list, output_path, verbose)
```

Listato (3.17): funzione downloadPdbs

La funzione **downloadPdbs** richiama la funzione **fetchPDB** del pacchetto **ProDy**, che prende in input:

- *protein_code*, la proteina da scaricare

- *folder*, il path della directory di output per i file *.pdb*, nel nostro caso sarà impostato ad *out_path*
- *compressed*, il flag per decidere se il file scaricato dovrà essere compresso oppure no, nel nostro caso il flag sarà impostato a *False* per indicare che i file scaricati debbano essere già decompressi.

Tramite tale istruzione è possibile interfacciarsi con **RCSB** e scaricare un singolo file *.pdb* mediante richiesta *FTP*, il download di tutte le proteine della lista avviene ciclando su tutti gli elementi di essa:

```
fetchPDB(protein_code, folder=output_path, compressed=False)
```

Listato (3.18): funzione fetchPDB

Dopo aver effettuato il download di tutte i file *.pdb* le informazioni relative al download ed ai file scaricati vengono salvati in un file *.xlsx* chiamato *info_proteins.xlsx* che viene salvato nella directory */output/*.
Lo script **prepare_receptors.py** una volta terminata la funzione **selectReceptors** richiama la funzione **splitRepeatedResidues** che prende in input:

- *pdb_folder*, il path della directory di output per i file *.pdb*
- *verbose*, il flag relativo al verbose.

Questa funzione si occupa di rimuovere i residui ripetuti memorizzati nel file *.pdb* sotto la dicitura *alternative location* o *alt_loc*:

```
splitRepeatedResidues(pdb_folder, verbose, output_folder=None)
```

Listato (3.19): funzione splitRepeatedResidues

Per accedere agli attributi dei file *.pdb* utilizziamo la funzione **read_pdb** del pacchetto **PandasPdb**, che prende in input il nome del file *.pdb*:

```
PandasPdb.read_pdb(pdb_file)
```

Listato (3.20): funzione PandasPdb.read_pdb

Gli attributi ripetuti sono denominati A e B, di cui i B saranno eliminati.
Successivamente viene richiamata dallo script la funzione **deleteHeteroatomsChains**, la quale prende in input:

- *pdb_folder*, il path della directory di output per i file *.pdb*
- *verbose*, il flag relativo al verbose.

Questa funzione si occupa di rimuovere le catene di eteroatomi contenute nel file *.pdb*, queste sono evidenziate nel file tramite un loro attributo ovvero la keyword *HETATM*, una volta rimosse, le catene vengono salvate.

```
deleteHeteroatomsChains(pdb_folder, verbose)
```

Listato (3.21): funzione deleteHeteroatomsChains

L'estrazione degli attributi dai file *.pdb* avviene anche in questo caso tramite la funzione **read_pdb** precedentemente descritta.
Lo script **prepare_receptors.py** richiama successivamente la funzione **splitChains** che prende in input:

- *pdb_folder*, il path della directory di output per i file *.pdb*
- *verbose*, il flag relativo al verbose.

Questa funzione si occupa di dividere le catene di residui che formano la struttura della proteina mantenendo soltanto le catene distinte, cioè quelle catene che hanno solo una sequenza amminoacidica distinta e non ripetuta da altre catene.

```
splitChains(pdb_folder, verbose)
```

Listato (3.22): funzione splitChains

Tramite la funzione **parsePDB** della libreria **prody** viene ricostruita la struttura proteica attraverso i seguenti oggetti:

- la lista delle molecole

- la lista degli atomi e dei residui, che viene rappresentata da un oggetto della classe **AtomGroup** restituito dalla funzione **parsePDB**.

,

```
atoms = parsePDB(protein_file)
```

Listato (3.23): funzione parsePDB

La ricostruzione della struttura proteica avviene tramite la vista gerarchica della struttura, ciò viene implementata tramite il metodo **getHierView** della classe **AtomGroup**:

```
atoms.getHierView()
```

Listato (3.24): atoms.getHierVieW

Vengono quindi esaminate tutte le sequenze, sempre tramite la funzione **parsePDB**, e vengono scelte solo le sequenze distinte, infine vengono salvate in un file tutte le catene distinte come se fossero delle proteine attraverso la funzione **writePDB** di **prody**, che prende in input:

- *filename*, il nome del file
- *new_atoms*, il composto da salvare

Questi file prenderanno il nome della proteina principale al quale viene aggiunto un underscore (_) seguito dal nome della catena distinta salvata nel file.

```
writePDB(filename, new_atoms)
```

Listato (3.25): writePDB

Lo step successivo nella preparazione dei recettori è effettuato dalla funzione **prepareReceptors**, che prende in input:

- *pdb_folder*, la directory con i file *.pdb* in input
- *pdbqt_folder*, la directory con i file *.pdbqt* in output
- *verbose*, il flag relativo al verbose

- *charges_to_add*, la carica da aggiungere alle proteine.

La funzione restituisce la conversione in file *.pdbqt* dei corrispondenti i file *.pdb* dati in input.

```
prepareReceptors(pdb_folder, pdbqt_folder, verbose)
```

Listato (3.26): prepareReceptors

La funzione **prepareReceptors** utilizza lo script **prepare_receptors4** della suite **ADFR**. In realtà viene utilizzata una versione modificata di tale script: **replacePrepareReceptor4.sh**. Questa versione modificata tramite bash scripting sostituisce la carica di Gasteiger con la carica in input ovvero *charges_to_add* che nel nostro caso sono cariche di Kollman come specificate dalla stringa *"Kollman"*. Tramite questo script si effettua la corretta conversione da *.pdb* a *.pdbqt*. Lo script viene richiamato da riga di comando dalla funzione mediante il comando **prepare_receptor**:

```
prepare_receptor -r pdb_file_path -A checkhydrogens -C
    charges_to_add -e -o pdbqt_folder
```

Listato (3.27): comando per convertire file da .pdb a .pdbqt

Nell'istruzione sopra:

- *pdb_file_path* indica la directory con i file *.pdb* in input ed è specificata tramite lo switch *-r*
- *checkydrogens* indica l'opzione tramite la quale vengono controllati gli atomi di idrogeno specificata lo switch *-A*
- *charges_to_add* indica la carica usata in input impostata tramite lo switch *-C*
- *pdbqt_folder* indica la directory con i file *.pdbqt* in output specificata tramite lo switch *-o*

Tramite tale istruzione è possibile effettuare la conversione di un singolo file, mentre la conversione di tutti i *.pdb* avviene ciclando su tutti i file nella directory corrispondente.

L'ultima fase nella preparazione dei ligandi consiste nella creazione delle *grid box*, mediante la funzione **createGridboxes** richiamata dallo script **prepare_receptors.py**, la funzione prende in input:

- **pdb_folder**, la directory contenente i file *.pdb*
- **gridbox_outputh_folder**, la directory di output per le *grid box*
- **margin**, la dimensione in *angstrom* del margine della *grid box*
- **verbose**, il flag relativo al verbose.

Per ogni proteina ottenuta deve essere creata una *grid box*, che definisce lo spazio conformazionale dove si colloca la proteina, è definita da:

- Un centro
- Le dimensioni
- L'esaustività, la quale sarà utilizzata da qualsiasi software da **AutoDock Vina** per la ricerca delle pose possibili del ligando.

Una *grid box* definisce la regione delle proteine dove il docking viene effettuato. Qualsiasi altra regione al di fuori dalla *grid box* non verrà presa in considerazione durante il processo di docking.
Le *grid box* sono un input richiesto da **AutoDock Vina** ma non da altri software per il docking.

```
1 createGridboxes(pdb_folder, gridbox_output_folder, margin, verbose)
```

Listato (3.28): createGridboxes

Per ogni proteina vengono estratti gli attributi contenuti nel proprio file e viene richiamata da **createGridboxes** la funzione **createGridbox**, questa prende in input:

- **ppdb**, l'insieme di attributi del file *.pdb* ottenuti come oggetto della classe **AtomGroup** restituito dalla funzione **parsePDB**

Figura (3.14): Gridbox mostrata all'interno della GUI di AutodockVina

- **protein_path**, il file *.pdb* corrispondente ad una singola proteina
- **gridbox_outputh_folder**, la directory di output per le *grid box*
- **margin**, la dimensione in *angstrom* del margine della *grid box*
- **verbose**, il flag relativo al verbose.

La funzione **createGridbox** costruisce una *grid box* per una singola proteina utilizzando gli attributi del file *.pdb* rispettando i parametri precedentemente definiti per una *grid box*.

3.4.4 Preparazione dei recettori tramite GUI

La preparazione dei recettori tramite GUI avviene selezionando il tasto **Receptors** nella sezione **Preparation** del software. All'interno della pagina relativa alla preparazione dei recettori premendo il tasto **Execute** è possibile effettuare i procedimenti spiegati nella sezione relativa all'esecuzione tramite script (3.4.3). Premendo il tasto **Back** è possibile tornare alla sezione **Preparation** relativa alla preparazione degli input. Come si osserva nella figura 3.15 all'interno di tale sezione sono presenti delle **entry** dove è possibile specificare diversi input tra cui:

- la directory dei file *.xlsx* di output
- la directory dei file *.pdb* di output
- la directory dei file *.pdbqt* di output
- il valore del margine delle *grid box*

Se lasciate vuote queste entry verrano impostate le configurazioni di default. E' presente anche un **checkbox** il quale se selezionato permette di mantenere i file precedentemente scaricati che altrimenti verrebbero cancellati.
Per andare a selezionare direttamente nel file system del PC le directory richieste sono stati implementati i tasti **Browse folders**, come è visibile nelle figure 3.8 e 3.16.

Figura (3.15): Sezione Receptors della GUI

Figura (3.16): Browse folders

La GUI per la preparazione dei recettori richiama lo script **prepare_receptors.py** discusso nella precedente sezione, adattando l'interfaccia grafica senza eliminare alcuna funzionalità della versione da riga di comando e rispettando i dettami dell'**OOP**.
Durante l'esecuzione viene mostrato un **pannello** (3.17) in cui vengono descritti:

- le operazioni che si stanno effettuando
- eventuali errori
- avvisi per l'utente.

Tutto ciò per restituire un feedback all'utente relativo al progresso della preparazione dei recettori. Al completamento di ogni fase sarà restituito un messaggio di avviso (figura: 3.18) ed in caso di input non valido sarà restituito un messaggio di errore (figura: 3.19).

Figura (3.17): Pannello dell'esecuzione dei recettori

Figura (3.18): Messaggio di completamento della preparazione dei ligandi

Figura (3.19): Messaggio di errore

3.5 Docking

Il docking può essere considerato come l'operazione principale eseguita da **AUDO4RIAS**. Il software scelto per eseguire il docking come detto in precedenza è **AutoDock Vina**, in particolare è stata scelta la versione 1.1.2 perchè dotata della capacità di produrre i file di log. La procedura del docking prende in input i dati ricavati dalla fase di preparazione precedentemente descritta. Risulta chiaro quindi come sia necessaria la corretta esecuzione della preparazione dei ligandi e delle proteine affinchè il docking vada a buon fine.

3.5.1 Docking tramite script

Dopo aver ottenuto i ligandi ed i recettori può essere effettuato il **Docking**, questo avviene mediante lo script bash **performDocking.sh**, il comando per eseguire lo script è:

```
1 ./performDocking.sh -s vina
```

Listato (3.29): Comando per eseguire performDocking

Lo script quando viene eseguito prende in input il parametro opzionale $-h$, che mostra il prompt con le istruzioni per l'uso dello script.
Lo script prende in input la cartella dove sono presenti:

- le proteine in formato *.pdbqt*
- i ligandi in formato *.pdbqt*

- le gridbox in formato *.txt*

Successivamente verrà creata la cartella */output/docking* dove saranno salvati i risultati del docking e, per ogni coppia proteina-ligando verrà eseguito il seguente comando:

```
1 vina --config gridbox_file.txt --receptor receptor_file.pdbqt --
    ligand ligand_file.pdbqt --out "{
    percorso_directory_software_installato}/output/docking/vina/
    cartella_nome_recettore/cartella_nome_ligando/out.pdbqt" --log "{
    percorso_directory_software_installato}/output/docking/vina/
    cartella_nome_recettore/cartella_nome_ligando/log.txt"
```

Listato (3.30): Comando per eseguire il docking con Autodock Vina

Nell'istruzione sopra:

- *Vina* è il commando per eseguire il docking con **AutoDock Vina**
- *gridbox_file.txt* è la gridbox in input relativa al recettore *receptor_file.pdbqt*, la gridbox è specificata dallo switch *−−config*
- *receptor_file.pdbqt* è il file di input del recettore nel formato *.pdbqt*, specificato dallo switch *−−receptor*
- *ligand_file.pdbqt* è il file di input del ligando nel formato *.pdbqt*, specificato dallo switch *−−ligand*
- *directory di output*, specificata dallo switch *−−out*
- *log di output*, specificato dallo switch *−−log*.

Questo comando è in grado di eseguire il docking per una sola coppia proteina-ligando. Per effettuare il docking su tutte le coppie deve essere eseguito tale comando per tutte le proteine su ogni ligando. Tale istruzione creerà una cartella per ogni proteina il cui nome corrisponde a quello della proteina (*cartella_nome_recettore*), all'interno di tale cartella verranno create tante cartelle quanti sono i ligandi con cui vengono studiate le pose, il nome delle sottocartelle è quello dei rispettivi ligandi (*cartella_nome_ligando*). All'interno di ogni sottocartella sono contenuti due file:

- Un file *.pdbqt* nominato out.pdbqt, il corrispondente output per la coppia proteina-ligando
- Un file *.txt* nominato *log.txt*, contenente le informazioni relative alla posa ottenuta.

Se si considera la proteina **1fcu** ed il ligando **bentazone**, entrambi utilizzati nel caso di esempio, verrà creata una cartella nominata **1cfu** contenente una sottocartella nominata **bentazone** contenente a sua volta un file *log.txt* e *out.pdbqt*.
All'interno del file di log è possibile ricavare le seguenti informazioni:

- *mode*, il numero della posa calcolata in modo randomico
- *affinity*, indica la stabilità del legame, misurata in kcal/mol, della coppia proteina-ligando ed è calcolata sul valore dell'energia di legame, più il valore è negativo o l'affinità di legame è bassa, più il ligando-recettore è stabile
- *dist from best mode* indica la *root-mean-square deviation (RMSD)* ossia la distanza media tra gli atomi, questo parametro si divide in due valori: *lower bound (l. b)* distanza minima ed *upper bound (u. p.)* distanza massima.

È possibile selezionare la migliore conformazione tra i risultati di **autoDock Vina**, ma non la migliore conformazione per una particolare proteina. Durante la ricerca delle pose il programma mantiene la conformazione data e inizia la selezione delle pose da lì. Per questo motivo si ottiene sempre un valore RMSD pari a 0. Di seguito il file di log del docking per la proteina **1fcu** combinata con il ligando **bentazone**:

```
##################################################################
# If you used AutoDock Vina in your work, please cite:          #
#                                                                #
# O. Trott, A. J. Olson,                                         #
# AutoDock Vina: improving the speed and accuracy of docking    #
# with a new scoring function, efficient optimization and        #
# multithreading, Journal of Computational Chemistry 31 (2010)  #
# 455-461                                                        #
#                                                                #
# DOI 10.1002/jcc.21334                                          #
#                                                                #
# Please see http://vina.scripps.edu for more information.      #
##################################################################

WARNING: The search space volume > 27000 Angstrom^3 (See FAQ)
Detected 8 CPUs
Reading input ... done.
Setting up the scoring function ... done.
Analyzing the binding site ... done.
Using random seed: -2136627046
Performing search ... done.
Refining results ... done.

mode |   affinity | dist from best mode
     | (kcal/mol) | rmsd l.b.| rmsd u.b.
-----+------------+----------+----------
   1         -6.6      0.000      0.000
   2         -6.0      1.874      3.171
   3         -6.0     31.288     32.713
   4         -5.9     28.684     30.487
   5         -5.9     31.907     33.061
   6         -5.9     30.972     32.563
   7         -5.7      3.854      6.764
   8         -5.7      2.808      5.241
   9         -5.7     28.294     29.592
Writing output ... done.
```

Figura (3.20): File di log del docking per la coppia proteina-ligando 1fcu-bentazone

3.5.2 Docking tramite GUI

Il docking tramite GUI avviene selezionando il tasto **Docking** nella sezione **Home page** del software. All'inteno della pagina relativa al docking premendo il tasto **Execute** è possibile effettuare il docking, premendo il tasto **Back** è possibile tornare alla sezione **Home Page**. Come si osserva nella figura 3.21 all'interno di tale sezione sono presenti delle **entry** dove è possibile specificare diversi input tra cui:

- la directory dei file *.pdbqt* delle proteine di input
- la directory dei file *.pdbqt* dei ligandi di input
- la directory dei file *.txt* delle gridbox di input
- la directory dei log e dell'output del docking

Se lasciate vuote queste entry verrano impostate le configurazioni di default. E' presente anche un **checkbox** il quale se selezionato permette di mantenere i file precedentemente scaricati che altrimenti verrebbero cancellati.
Per andare a selezionare direttamente nel file system del PC le directory richieste sono stati implementati i tasti **Browse folders**, come è visibile nelle figure 3.21 e 3.22.

Figura (3.21): Sezione Docking della GUI

Figura (3.22): Browse folder

La GUI per il **docking** richiama lo script **performDocking.sh** discusso nella precedente sezione, adattando l'interfaccia grafica ad esso senza eliminare alcuna funzionalità precedentemente illustrata per la versione da riga di comando e rispettando i dettami dell'**OOP**.
Durante l'esecuzione viene mostrato un **pannello** (3.23) in cui vengono descritti:

- le operazioni che si stanno effettuando
- eventuali errori
- avvisi per l'utente.

Tutto questo per restituire un feedback all'utente relativo al progresso del docking. Al completamento di ogni fase sarà restituito un messaggio di avviso (figura: 3.24) ed in caso di input non valido sarà restituito un messaggio di errore (figura: 3.25).

Figura (3.23): Pannello dell'esecuzione del docking

Figura (3.24): Messaggio di completamento dell' esecuzione del docking

Figura (3.25): Messaggio di errore

3.6 Analisi

Il processo di analisi consiste nell'analizzare l'output del docking per andare a determinare le interazioni (legami) presenti nelle pose ottenute. Le interazioni ricercate dal software sono:

- I close-contacts
- Legami ad idrogeno

3.6.1 Analisi tramite script

L'analisi si divide in un primo step fondamentale che consiste nella rilevazione legami presenti all'interno delle pose risultanti dal docking, per implementare ciò è stato preso spunto dalla funzionalità messa a disposizione da **AutodockTools**. A seguito di un'ispezione fatta sul suo codice sorgente si è determinato che non è possibile richiamare le funzioni capaci di identificare i legami delle pose, per cui il loro codice sorgente è stato importato ed adattato al software trattato, lo script contenente queste funzioni e che si occupa della rilevazione delle interazioni è lo script python **detect_interactions.py**.
Il comando da digitare per eseguire l'analisi è:

```
python3 detect_interactions.py
```

Listato (3.31): Comando per eseguire l'analisi

Lo script prende in input ciascun risultato del docking e restituisce:

- Un dizionario contenente le interazioni per ogni coppia-proteina ligando
- Il dizionario delle proteine dove per ciascun residuo e per ciascuna proteina coinvolti nelle interazioni, vengono memorizzati il numero di close-contacts ed il numero di legami ad idrogeno, questo dizionario è il risultato della fusione di tutti i dizionari delle coppie proteina-ligando
- Un dizionario contenente l'informazione relativa ai ligandi coinvolti nel calcolo dei residui, nello specifico viene salvato il tipo di ligando ed il tipo di interazione che si viene a formare nella coppia proteina-ligando.

Il dizionario delle proteine e dei ligandi vengono serializzati e memorizzati come file binari.
Le informazioni contenute in questi file vengono mostrate tramite heatmap e grafici a barre, questi vengono costruite tramite le funzioni di **numpy**.
Le heatmap contengono sull'asse delle ascisse i ligandi e sull'asse delle ordinate i residui, ciascuna entry ligando-residuo può assumere tre stati:

- Nessun legame, in viola
- Close-contact, in azzurro
- Legame ad idrogeno, in giallo

Per mostrare l'heatmap di una singola proteina, salviamo solo il dizionario di una singola proteina anzichè il dizionario completo.

Figura (3.26): Heatmap relativa al residuo 1cfu

I grafici a barre presentano sull'asse delle ascisse i residui e sull'asse delle ordinate il numero di legami e per ciascun residuo viene creata una barra stacked rappresentante il numero di close-contacts in blu ed il numero di legami ad idrogeno in giallo.

Figura (3.27): Grafico a barre relativo al residuo 1cfu

3.7 Analisi tramite GUI

l'Analisi tramite GUI avviene selezionando il tasto **Analysis** nella sezione **Home page** del software. All'inteno della pagina relativa all'analisi premendo il tasto **Execute** è possibile effettuare l'analisi, premendo il tasto **Back** è possibile tornare alla sezione **Home Page**.

Figura (3.28): Sezione Analysis della GUI

La GUI per l' **analisi** richiama lo script **detetct_interactions.py** discusso nella precedente sezione, adattando l'interfaccia grafica ad esso senza eliminare alcuna funzionalità precedentemente illustrata per la versione da riga di comando e rispettando i dettami dell'**OOP**. Durante l'esecuzione viene mostrato un **pannello** (3.29) in cui vengono descritti:

- le operazioni che si stanno effettuando
- eventuali errori
- avvisi per l'utente.

Tutto questo per restituire un feedback all'utente relativo al progresso del analisi. Al completamento di ogni fase sarà restituito un messaggio di avviso (figura: 3.30).

Figura (3.29): Pannello dell'esecuzione del analisi

Figura (3.30): Messaggio di completamento del processo dell'analisi

Capitolo 4

Risultati sperimentali

In questo capitolo verrà analizzata l'applicazione ponendo l'attenzione sul software usato per eseguire il docking e successivamente saranno anilzzate le prestazioni dell'applicazione nel complessivo prendendo in esame i tempi di esecuzione dei moduli che compongono l'applicativo.

4.1 I casi di analisi

I dati prodotti dall'applicazione soddisfano gli obbiettivi preposti, per effettuare un'analisi migliore delle prestazioni e delle capacità di **AUDO4RIAS**, è stato stressato il modulo principale del software ovvero il modulo del docking, confrontando lo strumento software utilizzato dall'applicativo trattato, **AutoDock Vina**, con un altro strumento software per eseguire il docking, **GNINA**.
I dati e parte dei risultati sperimentali sono stati ottenuti dallo studio effettuato dal mio collega Orsini, parte importante della sua tesi riguardava proprio il confronto dei due software. A tal riguardo è possibile ottenere maggiori informazioni tramite il seguente link: `https://github.com/gomax22/bachelor-thesis`.
Le prestazioni sono state riportate a vari livelli di profondità, partendo da una prima considerazione visuale della posa del ligando predetta, arrivando a motivare e valutare i valori osservati per le caratteristiche riportate. Le proteine selezionate per l'analisi dei risultati del docking effettuato tra i due software sopracitati sono le proteine: *1FCU, 2H8V, 3FE9, 4E81, 5XZ3_A e 6LQK_B*, per queste proteine sono

state scelte le pose con i ligandi: *Acrinathrin, Dimethomorph, Glyphosate e Oxyfluorfen.*
Le proteine precedentemente citate sono state scelte in quanto si differenziano per dimensione e conformazione, offrendo maggiore varietà dei casi di esame.
Per fornire un'analisi complessiva di **AUDO4RIAS**, infine, sono stati analizzati i tempi di esecuzione dei moduli che compongono il software.

4.2 GNINA

GNINA è un software per l'esecuzione del docking, che utilizza un insieme di reti neurali convoluzionali (CNN) come funzione di scoring. Le prestazioni del docking, sono valutate confrontando lo scoring ottenuto da GNINA con quello di AutoDock Vina quando si utilizzano tasche di legame esplicitamente definite o proteine intere. GNINA, utilizzando una funzione di punteggio CNN per ricalibrare le pose in uscita, supera il punteggio di AutoDock Vina nei compiti di redocking e cross-docking quando la tasca di legame è definita (aumenta dal 58% al 73% e dal 27% al 37%, rispettivamente) e quando è l'intera proteina a definire la tasca di legame (aumenta dal 31% al 38% e dal 12% al 16%, rispettivamente). Di seguito il repository di GNINA con licenza open source per l'uso come strumento di docking molecolare: `https://github.com/gnina/gnina`.[19]

4.3 Visualizzazione della predizione della posa

Il primo termine di confronto tra AutoDock Vina e GNINA sono le pose predette, sono state fatte calcolare ad entrambi le pose per ogni proteina sopracitata, queste sono poi state analizzate utilizzando la suite **AutoDockTools** trattata nella sezione 2.2.1.
Solo le pose migliori predette da entrambi i software sono state analizzate, per cui le differenze riscontrate possono essere imputate alla randomicità del calcolo delle pose, ai diversi criteri adottati da ciascun software e da eventuali filtri applicati.

(a) Acrinathrin

(b) Dimethomorph

(c) Glyphosate

(d) Oxyfluorfen

Figura (4.1): Visualizzazione delle migliori pose generate da GNINA (in giallo) ed AutoDock Vina (in rosso) relativamente alla proteina 2H8V ed alcuni dei ligandi del dataset iniziale (Acrinathrin, Dimethomorph, Glyphosate, Oxyfluorfen).

Nella foto 4.1 sono mostrate le migliori pose calcolate da AutoDock Vina e da GNINA per la proteina *2h8v*, questa è caratterizzata da piccole dimensioni, osservando il sito di legame relativo alle migliori pose predette si può concludere come il sito di legame individuato dai due software sia sostanzialmente lo stesso, a meno di orientazioni e torsioni del ligando coinvolto. Possiamo concludere che per recettori di piccole dimensioni le predizioni di AutoDock e di GNINA sono piuttosto simili, sebbene abbiano configurazioni differenti.

(a) Acrinathrin

(b) Dimethomorph

(c) Glyphosate

(d) Oxyfluorfen

Figura (4.2): Visualizzazione delle migliori pose generate da GNINA (in giallo) ed AutoDock Vina (in rosso) relativamente alla proteina 4E81 ed alcuni dei ligandi del dataset iniziale (Acrinathrin, Dimethomorph, Glyphosate, Oxyfluorfen).

Se si analizza un caso differente, quale le pose calcolate per la proteina *4E81*, mostrate nella figura 4.2, si nota una notevole differenza nelle predizioni: nel caso di GNINA, il ligando risulta occupare una posizione quasi esterna alla proteina mentre è tendenzialmente interna nel caso di AutoDock Vina.

(a) Acrinathrin

(b) Dimethomorph

(c) Glyphosate

(d) Oxyfluorfen

Figura (4.3): Visualizzazione delle migliori pose generate da GNINA (in giallo) ed AutoDock Vina (in rosso) relativamente alla proteina 6LQK_B ed alcuni dei ligandi del dataset iniziale (Acrinathrin, Dimethomorph, Glyphosate, Oxyfluorfen).

E' possibile illustrare un ulteriore caso di analisi in merito alla previsione delle pose, ossia il residuo *6LQK_B*, ovvero la proteina avente maggiore dimensione tra quelle analizzate, le cui pose sono mostrate nella figura 4.3. Dall'immagine si può osservare come alcune predizioni coincidano ed altre siano nettamente differenti, con una prevalenza per la collocazione esterna della posa. Il motivo di questo può essere dovuto al fatto che la struttura proteica in questione non presenta cavità per legami interni coi ligandi.

4.4 Rilevazione delle interazioni molecolari

Un altro tipo di analisi condotta sulle pose predette da Autodock Vina e GNINA è quella relativa alla rilevazione delle interazioni molecolari, essendo questo un modulo di **AUDO4RIAS**.
A tale scopo, sono state rilevate le interazioni molecolari interessanti per ciascuna coppia proteina-ligando e poi sintetizzate graficamente per interpretare i risultati.
Per ciascuna delle proteine selezionate dal dataset iniziale sono riportati in tabella il numero ed il tipo delle interazioni interessanti risultanti dal processo di rilevazione sia per GNINA che per AutoDock Vina. Inoltre, viene riportato anche il rapporto tra il numero di legami ad idrogeno rispetto al numero di contatti totali, corrispondente alla somma tra il numero di close contacts ed il numero di legami ad idrogeno (*HBR, Hydrogen Bonds Ratio*).

$$HBR = \frac{hydrogen\,bonds}{close\,contacts + hydrogen\,bonds} \tag{4.1}$$

	GNINA			AutoDock Vina		
Protein (code)	**Close contacts (int)**	**Hydrogen bonds (int)**	**Hydrogen Bonds Ratio (float)**	**Close contacts (int)**	**Hydrogen bonds (int)**	**Hydrogen Bonds Ratio (float)**
1FCU	1207	137	0.101	1621	185	0.102
2H8V	1112	52	0.044	1267	45	0.034
3FE9	1497	30	0.019	1684	34	0.019
4E81	1326	147	0.099	1664	125	0.069
5XZ3_A	1243	169	0.119	1472	203	0.121
6LQK_B	1139	119	0.094	1356	163	0.107

Tabella (4.1): Close contacts e legami a idrogeno prodotti da ciascuna delle proteine selezionate rispetto alle pose generate da GNINA ed AutoDock Vina. Per ciascun software e per ciascuna proteina, viene riportato l'Hydrogen Bonds Ratio, illustrato nell'Equazione 4.1.

Dalla Tabella 4.1, si evince che tendenzialmente il numero totale di contatti rilevati è decisamente inferiore quando viene utilizzato GNINA rispetto ad AutoDock Vina. Tuttavia, andando a rapportare il numero di legami ad idrogeno rispetto alla somma dei contatti totali, si osserva che i rapporti sono circa gli stessi. Ciò significa che in

proporzione, le pose generate da GNINA permettono maggiormente la formazione di legami a idrogeno.
Nonostante tutto però è stato analizzato empiricamente e sperimentalmente che AutoDock Vina genera pose di ligandi che risultano avere un'affinità migliore ma che nel complesso recettore-ligando creano nella maggior parte dei casi interazioni meno efficaci, sebbene in quantità maggiori rispetto a GNINA. Questo porta ad dire che la predizione effettuata da GNINA è certamente più predittiva rispetto ad AutoDock Vina. E' ipotizzabile che questo tipo di comportamento sia accentuato ed ampiamente confermato per esperimenti di docking su sacche, cavità o siti di legame specifici.
A questo punto dell'analisi vengono mostrate le heatmap più significative per dare ancora più enfasi ai risultati sperimentali ed alle considerazioni effettuate finora. Tra le varie heatmap realizzate per le proteine selezionate, risultano essere particolarmente contrastanti quelle relative alle proteine *1FCU* e *5XZ3_A*. I tipi di legame possibili sono stati codificati come segue: nessun legame (viola), close contacts (azurrino/verde) e legami a idrogeno (giallo). Sull'asse delle ordinate sono collocati i residui della proteina e sull'asse delle ascisse sono collocati i ligandi coinvolti.
La Figura 4.4 riporta le heatmaps prodotte dall'analisi delle interazioni molecolari relativamente alla proteina 1FCU per il dataset dei ligandi utilizzato (vedi Figura 4.4a per GNINA, vedi 4.4b per AutoDock Vina).
Si può osservare dalla tabella 4.1 che GNINA ed AutoDock Vina operino sugli stessi dati, ma per quanto detto prima il riscontro è che le interazioni molecolari prodotte all'interno del complesso sono profondamente diverse. La visualizzazione delle heatmaps permette una conferma di quanto detto. Analizzando la Figura 4.4a che è relativa ai risultati di GNINA, è possibile notare come i residui coinvolti nell'intero processo superino le 100 unità, presentando inoltre una distribuzione più equa dei legami all'interno del complesso.
AutoDock Vina produce risultati differenti per numero, in quanto i residui coinvolti sono circa 60 unità, e perchè le interazioni sono

(a) Heatmap delle interazioni per la proteina 1FCU prodotta a partire dalle pose predette dei ligandi da GNINA.

(b) Heatmap delle interazioni per la proteina 1FCU prodotta a partire dalle pose predette dei ligandi da AutoDock Vina.

Figura (4.4): Visualizzazione delle interazioni per la proteina 1FCU attraverso heatmaps. In alto, è riportata l'heatmap risultante dalle pose predette da GNINA mentre in basso, quella risultante dalle pose predette da AutoDock Vina. Sull'asse delle ascisse sono riportati i ligandi, sulle ordinate i residui coinvolti per la proteina. In giallo sono riportati i legami a idrogeno, in azzurro/verde i close contacts, in viola nessun legame.

(a) Heatmap delle interazioni per la proteina 5XZ3_A prodotta a partire dalle pose predette dei ligandi da GNINA.

(b) Heatmap delle interazioni per la proteina 5XZ3_A prodotta a partire dalle pose predette dei ligandi da AutoDock Vina.

Figura (4.5): Visualizzazione delle interazioni per la proteina 5XZ3_A attraverso heatmaps. In alto, è riportata l'heatmap risultante dalle pose predette da GNINA mentre in basso, quella risultante dalle pose predette da AutoDock Vina. Sull'asse delle ascisse sono riportati i ligandi, sulle ordinate i residui coinvolti per la proteina. In giallo sono riportati i legami a idrogeno, in azzurro/verde i close contacts, in viola nessun legame.

maggiormente concentrate in determinati residui: *ARG244, SER225, TYR188, TYR190.*
Si può concludere che AutoDock Vina permette di distinguere più facilmente i residui che forniscono un apporto significativo da quelli il cui contributo è minimo, GNINA invece fa si che ogni residuo fornisca il proprio contributo in termini di interazioni.
Questo tipo di situazione appare ancora più netta nel caso della proteina 5XZ3_A, prendendo visione della Figura 4.5.
In questo caso però, la particolarità delle relative heatmaps sta nel fatto che nel caso della Figura 4.5b, corrispondente ai risultati di AutoDock Vina, è possibile sintetizzare non solo la maggior parte delle interazioni in un sotto-insieme dei residui coinvolti, comunque minore

(a) Heatmap delle interazioni per la proteina 5XZ3_A prodotta a partire dalle pose predette dei ligandi da GNINA.

(b) Istogramma delle interazioni per la proteina 1FCU prodotta a partire dalle pose predette dei ligandi da AutoDock Vina.

Figura (4.6): Visualizzazione delle interazioni per la proteina 1FCU attraverso istogrammi. In alto, è riportato l'istogramma risultante dalle pose predette da GNINA mentre in basso, quello risultante dalle pose predette da AutoDock Vina. Sull'asse delle ordinate è riportato il numero di contatti, sulle ascisse i residui coinvolti per ciascuna proteina. In arancione sono riportati i legami a idrogeno, in blu i close contacts.

in numero rispetto a quello evidenziato dalla Figura 4.5a relativa ai risultati di GNINA, ma anche la maggior parte dei legami a idrogeno prodotti in un sotto-insieme ancor più ridotto dei residui che compongono la proteina: *HIS61, TYR72*.
Le medesime considerazioni possono essere fatte ancor più intuitivamente visualizzando gli istogrammi relativi alle interazioni prodotte, i cui picchi individuano i residui maggiormente presenti nei legami. In basso sono riportati i residui che effettuano almeno un'interazione rilevanti; in blu è riportato il numero di close-contacts mentre in arancione il numero di legami a idrogeno per ciascun residuo proteico.
In conclusione possiamo affermare che dal punto di vista delle interazioni molecolari, quelle prodotte da GNINA, sono quantitativamente

(a) Istogramma delle interazioni per la proteina 5XZ3_A prodotta a partire dalle pose predette dei ligandi da GNINA.

(b) Istogramma delle interazioni per la proteina 5XZ3_A prodotta a partire dalle pose predette dei ligandi da AutoDock Vina.

Figura (4.7): Visualizzazione delle interazioni per la proteina 5XZ3_A attraverso istogrammi. In alto, è riportato l'istogramma risultante dalle pose predette da GNINA mentre in basso, quello risultante dalle pose predette da AutoDock Vina. Sull'asse delle ordinate è riportato il numero di contatti, sulle ascisse i residui coinvolti per ciascuna proteina. In arancione sono riportati i legami a idrogeno, in blu i close contacts.

inferiori ma qualitativamente migliori, rispetto a quelle prodotte da AutoDock Vina. Nonostante il numero di legami sia inferiore, è emerso che il numero di residui coinvolti in legami è notevolmente maggiore, producendo quindi heatmaps più ampie e sparse in cui le interazioni sono maggiormente distribuite, rispetto a quelle ridotte e fitte di AutoDock Vina, in cui è maggiormente possibile approssimare l'insieme dei residui coinvolti in interazioni ad un sotto-insieme ridotto di residui interessanti.

4.5 Analisi dei tempi di esecuzione

Per valutare le performance delle funzioni sviluppate, sono stati eseguiti 10 cicli incrementali per ciascuna funzionalità sviluppata, eccezion fatta per il docking la cui esecuzione con tutti i dati di input (289 ligandi, 11 recettori) presi in esame ha richiesto circa 3 giorni di esecuzione.
Tutti i test sono stati eseguiti su Google Colaboratory (Colab). Colab è una piattaforma gratuita che permette a chiunque di scrivere ed eseguire codice Python attraverso un browser. L'unico requisito è possedere un account Google (ad esempio Gmail). Colab è basato su un progetto Open Source chiamato Jupyter. I documenti/programmi scritti su Colab sono chiamati Notebook e verranno salvati automaticamente sul Google Drive associato al proprio account.
Le macchine virtuali messe a disposizione in Google Colab ospitano un ambiente configurato che consente di concentrarsi sin da subito sui progetti di Data Science: sono presenti numerose librerie Python, tra cui moltissime di Data Science, si può usufruire di GPU e TPU per dare boost computazionali importanti ai lavori.
Per facilitare la comprensione dei risultati ottenuti sono stati realizzati dei grafici, uno per ciascuna funzionalità testata.
Ogni grafico riporta l'andamento dei tempi in funzione dell'incremento degli input relativi.
Di seguito sono riportati i grafici relativi ai test effettuati per le funzioni:

- Preparazione dei ligandi
- Preparazione dei recettori
- Esecuzione dell'analisi.

Figura (4.8): Tempi di preparazione dei ligandi

Figura (4.9): Tempi di preparazione dei recettori

Figura (4.10): Tempi di esecuzione dell'analisi

Osservando e comparando i dati dei grafici sopra, si può notare che:

- le prestazioni delle funzioni, preparazione ligandi, preparazione recettori e analisi rispecchiano la stessa complessità di tempo
- le prestazioni del sistema in generale non degradano con il crescere dei dati in input
- l'andamento dei tempi si incrementa linearmente con l'aumentare dell'input.

4.6 Comparazione con altri tools

AUDO4RIAS è stato confrontato da un punto di vista di tempi di esecuzioni con un insiem di sofotware che eseguono le stesse funzionalità, quali:

- **ADFR suite**, per la preparazione dei ligandi e dei recettori
- **GNINA**, per il Docking
- **AUTODOCKTOOLS**, per l'analisi dei risultati del dockig.

Nell'analisi comparativa effettuata, i tempi degli altri tools sono stati rilevati senza tener conto del tempo impiegato dall'utente per l'inserimento degli input per le integrazioni dei vari software, in quanto variabile da individuo ad individuo.
I risultati ottenuti sono stati rappresentati nel seguente grafico:

Figura (4.11): AUDO4RIAS vs other combined tools

Si può osservare che i tempi di AUDO4RIAS sono abbastanza in linea con quelli degli altri tool combinati, al netto del tempo impiegato dall'utente per le operazioni manuali intermedie necessarie necessarie durante l'utilizzo degli altri tools.
Nell'analisi i tempi sono calcolate in ore e le elaborazioni per entrambi sono state effettuate su Google Colab.

Conclusioni e sviluppi futuri

Nell'idea iniziale del progetto AUDO4RIAS gli obiettivi erano quelli di creare un software a riga di comando capace di automatizzare i processi di preparazione degli input per il docking, di esecuzione del docking e l'analisi dei suoi risultati, riunendo in una sola applicazione diverse funzioni e strumenti di bioinformatica. Successivamente si è deciso di dotare il software di un'interfaccia grafica, in modo da permetterne un utilizzo più intuitivo e user frendly anche a chi non abbia particolari conoscenze informatiche e familiarità con l'uso della riga di comando. Inoltre, questo ci ha permesso di aggiungere ulteriori funzionalità al software, quali:

- casi di analisi aggiuntivi
- altre opzioni di esecuzione.

Il software sviluppato durante il mio lavoro di tesi può avere interessanti applicazioni nel campo della ricerca scientifica, in particolare negli studi di risk assessment, infatti è stato inizialmente progettato con lo scopo di analizzare l'effetto dei pesticidi sulle proteine da Apis mellifera, tuttavia può essere impiegato in altri approcci sperimentali. Più nello specifico l'applicazione sviluppata ha più scopi:

- Riunire in un unico software tutti i tools e strumenti per effettuare un analisi di docking
- Fornire strumenti e procedure per l'analisi dei risultati del docking
- Automatizzare l'intero processo di docking e di analisi

- Dare la possibilità di sfruttare un'applicazione semplice, che non richiede particolari conoscenze in ambito informatico per l'utente finale.

Conclusioni

Il software AUDO4RIAS automatizza l'intero processo di virtual screening partendo dalla creazione delle library alla preparazione delle strutture dei ligandi e dei recettori, passando per il processo di docking e concludendo con l'analisi dell'output derivato dalla fase precedente. Inoltre, viene concessa ampia libertà all'utente per quanto riguarda la scelta degli input e le impostazioni d'uso del software, in modo da non limitare l'utente esperto. Dal punto di vista funzionale, il software è stato in grado di fornire risultati qualitativamente soddisfacenti, poichè sia i risultati del docking che dell'analisi sono stati comparabili, se non identici, a quelli ottenuti manualmente con un livello di difficoltà e con tempi, nell'ordine di giorni, significativamente superiori. Da un punto di vista quantitativo sono stati provati diversi output sia per tipo che per numero e l'applicazione è sempre stata in grado produrre risultati con tempistiche equiparabili ad altri software simili (Gold, Gnina, Keplero, ecc...). Un altro aspetto di cui si è tenuto conto è stata l'usabilità dell'applicazione. Infatti AUDO4RIAS è dotato di una GUI semplice da usare e che si avvicina non solo ai modelli di applicazione maggiormente diffusi sul mercato, ma tiene l'utente sempre al corrente di ciò che il software sta facendo, facilitandone la comprensione da parte dell'utente.Per quanto riguarda i tempi di esecuzione, il software gode di una buona responsività nei confronti di chi la usa, infatti tramite l'uso del parallelismo e della concorrenza sviluppati attraverso i thread, è possibile effettuare diversi compiti eseguiti dal software in parallelo e con tempi di risposta contenuti. Inoltre, va ricordato che, per andare incontro a macchine che non dispongono di molte risorse hardware, è sempre ossibile utilizzare la versione da riga di comando veloce e leggera del software. Infine, l'applicazione è scalabile considerata la facilità di evoluzione per sviluppi futuri. In conclusione, Essendo in grado di eseguire il docking, preparare i suoi

input ed analizzarne gli output in maniera efficiente ed efficace, l'applicativo AUDO4RIAS si propone di essere utilizzata nel campo della ricerca e diventare oggetto di publicazioni scientifiche.

Sviluppi futuri

Lo scopo alla base dello sviluppo di AUDO4RIAS è stato quello di realizzare un software in grado di analizzare i possibili effetti dei pesticidi più diffusi sulle proteine di Apis mellifera in maniera rapida ed automatizzata. L'obiettivo a lungo termine dell'applicazione consiste nell'ampliare il suo dominio di analisi ad altre specie e casi, come ad esempio il topo oppure l'uomo. Da un punto di vista puramente pratico una delle finalità principali dell'applicazione è quella di concedere sempre più libertà per quanto riguarda la modalità di input dei dati, nel caso specifico l'obiettivo è quello di usare come banca dati non solo i database di Pubchem ed RCSB, ma usufruire anche degli altri archivi di composti chimici o organici e di macromolecole biologiche presenti in internet. Per quanto riguarda l'aspetto degli input sarebbe utile trovare una modalità di input per i ligandi simile a quella sfruttata per i recettori, garantendo una maggiore quantità di input da poter fornire. Da un punto di vista computazionale si mirerà ad usare pattern, strategie, algoritmi e schemi di programmazione che mireranno a ridurre sempre di più la complessità di tempo e spazio di AUDO4RIAS. Alla luce di quanto detto, risulta chiaro come questo non sia un punto di arrivo ma il punto di partenza del progetto AUDO4RIAS.

Appendice A

Tabella dei ligandi

(E,E)-7,9-Dodecadien-1-yl acetate	Gibberellins
(E,E)-8,10-Dodecadien-1-ol	Glyphosate
(3E,8Z,11Z)-Tetradeca-3,8,11-trienyl acetate	Halauxifen-methyl
3E,8Z-Tetradecadienyl acetate	Halosulfuron-methyl
(E)-11-Tetradecen-1-yl acetate	Hexythiazox
(E)-5-Decen-1-ol	Hymexazol
(E)-5-Decen-1-yl acetate	Imazalil
(E)-8-Dodecen-1-yl acetate	Imazamox
(Z,E)-7,11-Hexadecadien-1-yl acetate	Indolylbutyric acid
(Z,E)-Tetradeca-9,11-dienyl acetate	Indoxacarb
(Z,E)-9,12-Tetradecadien-1-yl acetate	Iodosulfuron
(Z)-11-Hexadecen-1-ol	Ipconazole
(Z)-11-Hexadecen-1-yl acetate	Iprovalicarb
(Z)-11-Hexadecenal	Isofetamid
(Z)-11-Tetradecen-1-yl acetate	Isopyrazam
(Z)-13-Octadecenal	Isoxaben
(Z)-7-Tetradecenal	Isoxaflutole
(Z)-8-Dodecen-1-ol	Kresoxim-methyl
(Z)-8-Dodecen-1-yl acetate	L-Ascorbic acid
(Z)-8-Tetradecen-1-ol	lambda-Cyhalothrin
z-8-Tetradecenyl acetate	Laminarin
(Z)-9-Dodecen-1-yl acetate	Lauric acid
(Z)-9-Hexadecenal	Lavandulyl senecioate
(Z)-9-Tetradecen-1-yl acetate	Lenacil
1-Decanol	Malathion
1-methylcyclopropene	Maleic hydrazide
1-Naphthylacetamide	Mandestrobin
1-Naphthylacetic acid	Mandipropamid
	Continua nella pagina successiva
1,4-Dimethylnaphthalene	MCPA
2-Phenylphenol	MCPB
2,4-D	Mecoprop-P
Methyl 2,5-dichlorobenzoate	Mefentrifluconazole
24-Epibrassinolide	Mepanipyrim

4-(2,4-Dichlorophenoxy)butanoic acid	Mepiquat
6-Benzyladenine	Meptyldinocap
8-Hydroxyquinoline	Mesosulfuron
Acequinocyl	Mesotrione
Acetamiprid	Metaflumizone
Acetic acid	Metalaxyl
Acibenzolar-S-methyl	Metalaxyl-M
Aclonifen	Metaldehyde
Acrinathrin	Metam
Ametoctradin	Metamitron
Amidosulfuron	Metazachlor
Aminopyralid	Metconazole
Amisulbrom	Methoxyfenozide
Azimsulfuron	Methyl decanoate
Azoxystrobin	Methyl octanoate
Beflubutamid	Zineb
Benalaxyl-M	Metobromuron
Benfluralin	Metrafenone
Bensulfuron	Metribuzin
Bentazone	Metsulfuron-methyl
Benthiavalicarb	Milbemectin
Benzoic acid	Tetradecyl acetate
Benzovindiflupyr	Napropamide
Bifenazate	Nicosulfuron
Bifenox	Oleic acid
Bispyribac	Orange oil
Bixafen	Oxamyl
Boscalid	Oxathiapiprolin
Bromuconazole	Oxyfluorfen
Bupirimate	Paclobutrazol
Buprofezin	Pelargonic acid
Capric acid	Penconazole
Caprylic acid	Pendimethalin
Captan	Penflufen
Carfentrazone-ethyl	Penoxsulam
Carvone	Penthiopyrad
Chlorantraniliprole	Pethoxamid
	Continua nella pagina successiva
Chlormequat	Phenmedipham
Chlorotoluron	Phosmet
Chromafenozide	Phosphane
Clethodim	Picloram
Clodinafop	Picolinafen
Clofentezine	Pinoxaden
Clomazone	Pirimicarb
Clopyralid	Pirimiphos-methyl
Cyantraniliprole	Potassium bicarbonate
Cyazofamid	Prochloraz

Cycloxydim	Prohexadione
Cyflufenamid	Propamocarb
Cyflumetofen	Propaquizafop
Cyhalofop-butyl	Propoxycarbazone
Cymoxanil	Propyzamide
Cypermethrin	Proquinazid
Cyprodinil	Prosulfocarb
Daminozide	Prosulfuron
Dazomet	Prothioconazole
Deltamethrin	Pyraclostrobin
Dicamba	Pyraflufen-ethyl
Dichlorprop-P	Pyridaben
Diclofop	Pyridalyl
Difenoconazole	Pyridate
Diflufenican	Pyrimethanil
Dimethachlor	Pyriofenone
Dimethenamid-P	Pyriproxyfen
Dimethomorph	Pyroxsulam
Dimoxystrobin	Quinmerac
Dithianon	Quizalofop-P
Dodecan-1-ol	Quizalofop-P-ethyl
Dodecyl acetate	Quizalofop-P-tefuryl
Dodemorph	Rescalure
Dodine	Rimsulfuron
Ethephon	S-Metolachlor
Ethofumesate	Sedaxane
Etofenprox	Sodium 2-methoxy-5-nitrophenolate
Etoxazole	Sodium 2-nitrophenolate
Eugenol	Sodium 4-nitrophenolate
Fenazaquin	Spiromesifen
Fenhexamid	Spirotetramat
Fenoxaprop-P	Spiroxamine
	Continua nella pagina successiva
Fenpicoxamid	Sulcotrione
Fenpropidin	Sulfosulfuron
Fenpyrazamine	Sulfoxaflor
Fenpyroximate	Sulfuryl fluoride
Flazasulfuron	tau-Fluvalinate
Flonicamid	Tebuconazole
Florasulam	Tebufenozide
Florpyrauxifen-benzyl	Tebufenpyrad
Fluazifop-P	Tefluthrin
Fluazinam	Tembotrione
Flubendiamide	Terbuthylazine
Fludioxonil	Tetraconazole
Flufenacet	Tetradecan-1-ol
Flumetralin	Thiabendazole
Flumioxazin	Thiencarbazone-methyl

Fluometuron	Thifensulfuron-methyl
Fluopicolide	Thymol
Fluopyram	Tolclofos-methyl
Fluoxastrobin	Tri-allate
Flupyradifurone	Tribenuron
Fluquinconazole	Triclopyr
Flurochloridone	Trifloxystrobin
Fluroxypyr	Triflusulfuron
Flutianil	Trinexapac
Flutolanil	Triticonazole
Fluxapyroxad	Tritosulfuron
Folpet	Urea
Foramsulfuron	Valifenalate
Forchlorfenuron	Ziram
Formetanate	Zoxamide
Fosetyl	Quartz sand
Fosthiazate	Silthiofam
Gamma-cyhalothrin	Esfenvalerate
Garlic extract	Ethylene
Geraniol	Gibberellic acid
(2Z,4E)-5-[(1S)-1-Hydroxy-2,6,6-trimethyl-4-oxocyclohex-2-en-1-yl]-3-methylpenta-2,4-dienoic acid	
1-(4-Chlorophenyl)-5-(2-methoxyethoxy)-4-oxo-1,4-dihydrocinnoline-3-carboxylic acid	

Tabella (A.1): Tabella dei ligandi in input

Appendice B

Tabella dei recettori

6GXN	6YSS
6GXP	6YSU
6GXO	5O2R
6GWT	2N8V
6YST	3QRX
1CCV	

Tabella (B.1): Tabella dei recettori in input

Bibliografia

[1] V Bellucci, P Bianco e V Silli. *Le api, sentinelle dell'inquinamento ambientale.*

[2] Marianna Martinello et al. «Spring mortality in honey bees in northeastern Italy: detection of pesticides and viruses in dead honey bees and other matrices». In: *Journal of Apicultural Research* 56.3 (2017), pp. 239–254.

[3] Mauro Capocci. *Enciclopedia della Scienza e della Tecnica.* `https://www.treccani.it/enciclopedia/docking-molecolare_28Enciclopedia-della-Scienza-e-della-Tecnica29/`. 2008.

[4] Isabella A Guedes, Camila S de Magalhães e Laurent E Dardenne. «Receptor–ligand molecular docking». In: *Biophysical reviews* 6.1 (2014), pp. 75–87.

[5] Nataraj S Pagadala, Khajamohiddin Syed e Jack Tuszynski. «Software for molecular docking: a review». In: *Biophysical reviews* 9.2 (2017), pp. 91–102.

[6] Jiyu Fan, Ailing Fu e Le Zhang. «Progress in molecular docking». In: *Quantitative Biology* 7.2 (2019), pp. 83–89.

[7] Sebastian Bassi. *Python for bioinformatics.* Chapman e Hall/CRC, 2016.

[8] Matt Swain. *PubChemPy documentation.* `https://pubchempy.readthedocs.io/en/latest/guide/introduction.html`. 2014.

[9] Ahmet Bakan, Lidio M. Meireles e Ivet Bahar. «ProDy: Protein Dynamics Inferred from Theory and Experiments». In: *Bioinformatics* 27.11 (apr. 2011), pp. 1575–1577. ISSN: 1367-4803. DOI: `10.1093/bioinformatics/btr168`. eprint: `https://academic.oup.com/bioinformatics/article-pdf/27/11/1575/5904480/btr168.pdf`. URL: `https://doi.org/10.1093/bioinformatics/btr168`.

[10] The pandas development team. *pandas-dev/pandas: Pandas.* Ver. latest. Feb. 2020. DOI: `10.5281/zenodo.3509134`. URL: `https://doi.org/10.5281/zenodo.3509134`.

[11] John E Grayson. *Python and Tkinter programming.* Manning Publications Co. Greenwich, 2000.

[12] Paul Barrett et al. «matplotlib–A Portable Python Plotting Package». In: *Astronomical data analysis software and systems XIV.* Vol. 347. 2005, p. 91.

[13] Stefan Van Der Walt, S Chris Colbert e Gael Varoquaux. «The NumPy array: a structure for efficient numerical computation». In: *Computing in science & engineering* 13.2 (2011), pp. 22–30.

[14] Noel M O'Boyle et al. «Open Babel: An open chemical toolbox». In: *Journal of cheminformatics* 3.1 (2011), pp. 1–14.

[15] Peter W. Rose et al. «The RCSB protein data bank: integrative view of protein, gene and 3D structural information». In: *Nucleic Acids Research* 45.D1 (ott. 2016), pp. D271–D281. ISSN: 0305-1048. DOI: 10.1093/nar/gkw1000. eprint: https://academic.oup.com/nar/article-pdf/45/D1/D271/8846530/gkw1000.pdf. URL: https://doi.org/10.1093/nar/gkw1000.

[16] Stephen K Burley et al. «RCSB Protein Data Bank: biological macromolecular structures enabling research and education in fundamental biology, biomedicine, biotechnology and energy». In: *Nucleic Acids Research* 47.D1 (ott. 2018), pp. D464–D474. ISSN: 0305-1048. DOI: 10.1093/nar/gky1004. eprint: https://academic.oup.com/nar/article-pdf/47/D1/D464/27436259/gky1004.pdf. URL: https://doi.org/10.1093/nar/gky1004.

[17] Ruth Huey, Garrett M Morris e Stefano Forli. «Using AutoDock 4 and AutoDock vina with AutoDockTools: a tutorial». In: *The Scripps Research Institute Molecular Graphics Laboratory* 10550 (2012), p. 92037.

[18] Oleg Trott e Arthur J Olson. «AutoDock Vina: improving the speed and accuracy of docking with a new scoring function, efficient optimization, and multithreading». In: *Journal of computational chemistry* 31.2 (2010), pp. 455–461.

[19] Andrew T McNutt et al. «GNINA 1.0: molecular docking with deep learning». In: *Journal of cheminformatics* 13.1 (2021), pp. 1–20.

Ringraziamenti

La fine di un ciclo, questa è la fine di un ciclo.
Terminato il liceo non avevo la minima idea di che percorso intraprendere, le superiori non sono riuscite a farmi appassionare o propendere per qualche percorso particolare, decisi quindi di seguire il consiglio di mio padre, studiare informatica, una materia che non conoscevo minimamente ma che prometteva tanti sbocchi e grossa richiesta nel mondo del lavoro.
Al primo anno di università ho scritto per la prima volta un programma, il classico main.c che stampava nel terminale la scritta "Hello, World!", rimasi affascinato da ciò, finalmente qualcosa di pratico, iniziai ad appassionarmi dell'informatica.
Tre anni fa circa, ho iniziato un nuovo percorso, durante questi anni sono cambiato, ho vissuto una pandemia, ho conosciuto nuove persone e ne ho perse altre, si sono alternati momenti di grande gioia e momenti di tristezza, ma in entrambi i casi non ero solo, vicino a me c'erano sempre i miei genitori.
Mia mamma Assunta, che quando mi vedeva triste o pensieroso, cercava sempre di capire cosa stesse succedendo nonostante la scacciassi, e che trovava sempre una parola o un gesto di conforto durante i miei momenti difficili.
Mio padre Franco, che spera sempre il meglio per me e che mi ha sempre accompagnato in qualsiasi cosa facessi, aiutandomi sempre anche fino a notte tarda.
Grazie ad entrambi ed ai loro ho avuto la possibilità di trascorrere questi tre anni senza dovermi occupare di problemi che non fossero strettamente legati al mondo universitario, ma questa è solo l'ennesima dimostrazione di quanto mi vogliano bene.

Ho avuto molti compagni con cui affrontare questa esperienza, in ordine del tutto casuale, grazie a : Denny, Max, Dom e Kekko i quali si sono sacrificati nel fare progetti con me sopportando il mio carattere difficile, spero di avervi regalato qualche sorriso durante il lavoro.
Vorrei ringraziare altri dei miei compagni universitari, con i quali non avrò fatto progetti insieme, ma che non si sono mai rifiutati di aiutarmi o semplicemente di essermi amici, anche loro in ordine totalmente casuale: Gianfranco, Francesco, Gianluca, Aniello, Luigi, Ilaria, Simone, Mariano e Giuseppe.
Un grazie a tutti i miei zii i quali mi hanno regalato sempre tanto affetto