

Università 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

Laureando

Alfredo Mungari

Matr. 0124002134

Correlatore

Dott. Ferdinando Febbraio

Anno Accademico 2021-2022

Indice

Elenco delle figure	IV
Elenco delle tabelle	VI
Elenco dei listati	VII
1 Introduzione	1
1.1 Docking Molecolare	4
1.2 Simulazione	7
1.3 Software per il docking molecolare	7
1.4 Gestione del lavoro	9
1.5 Idea e sviluppo	10
1.6 Contenuto della tesi	11
2 Tecnologie e piattaforme	12
2.1 Linguaggi e tools	12
2.1.1 Python	12
2.1.1.1 PubChemPy	14
2.1.1.2 ProDy	14
2.1.1.3 Pandas	15
2.1.1.4 Tkinter	15
2.1.1.5 Matplotlib	16

2.1.2	NumPy	17
2.1.3	ADFRsuite	17
2.1.4	MGLTools	20
2.1.5	Open Babel	21
2.2	Database	22
2.3	Autodock	23
2.4	Autodock Vina	24
2.4.1	Funzione di scoring	25
3	Applicazione realizzata	31
3.1	Installazione	31
3.2	Modalità di utilizzo	32
3.3	Dati in input	34
3.4	Preparazione dei ligandi e dei recettori . . .	38
3.4.1	Preparazione dei ligandi tramite script	40
3.4.2	Preparazione dei ligandi tramite GUI	46
3.4.3	Preparazione dei recettori tramite script	50
3.4.4	Preparazione dei recettori tramite GUI	60
3.5	Docking tramite script	66
3.6	Docking tramite GUI	71
4	Risultati sperimentali	76
5	Conclusioni e sviluppi futuri	77
5.1	Conclusioni	78
5.2	Sviluppi futuri	79
A	Tabella dei ligandi	81

B Tabella dei recettori	85
Bibliografia	86

Elenco delle figure

1.1	Esemplare adulto di Apis Mellifera	3
1.2	Rappresentazione dei due ligandi ibuprofe- ne a sinistra e celecoxib a destra che hanno effettuato il docking con un enzima di COX2	6
1.3	Processo del docking molecolare	9
2.1	Funzione di scoring pesata	30
3.1	Sezione Home Page della GUI	34
3.2	Query dei recettori	36
3.3	Proteine dei recettori dell'apis mellifera . .	37
3.4	Sezione Help della GUI	38
3.5	Sezione Preparation della GUI	39
3.6	Sezione Ligands della GUI	47
3.7	Browse folders	48
3.8	Pannello dell'esecuzione dei ligandi	49
3.9	Messaggio processo completato con successo	49
3.10	Messaggio di errore	50
3.11	Sezione Receptors della GUI	62
3.12	Browse files	63
3.13	Browse folders	64
3.14	Pannello dell'esecuzione dei recettori . . .	65

3.15	Messaggio processo completato con successo	65
3.16	Messaggio di errore	66
3.17	File di log del docking per la coppia proteina- ligando 1fcu-bentazone	70
3.18	Sezione Receptors della GUI	72
3.19	Browse files	73
3.20	Pannello dell'esecuzione del docking . . .	74
3.21	Messaggio processo completato con successo	74
3.22	Messaggio di errore	75

Elenco delle tabelle

2.1	Funzione di scoring pesi e termini	27
3.1	Ligandi non scaricati	44
A.1	Tabella dei ligandi in input	84
B.1	Tabella dei recettori in input	85

Elenco dei listati

3.1	Comando per scaricare la repository	31
3.2	Comando per avviare la GUI	33
3.3	Comando per scaricare la repository	40
3.4	funzione selectLigands	42
3.5	funzione pubchempy.get_compounds	42
3.6	pubchempy.download	43
3.7	funzione sdf2pdb	44
3.8	Comando per la conversione da <i>.sdf</i> a <i>.pdb</i> .	45
3.9	funzione prepareLigands	45
3.10	Comando per la conversione da <i>.pdb</i> a <i>.pdbqt</i>	45
3.11	funzione prepare_receptors	50
3.12	funzione selectReceptors	52
3.13	funzione RestApiSelection	52
3.14	funzione json.loads	52
3.15	funzione json.dump	53
3.16	funzione requests.get	53
3.17	funzione downloadPdb	53
3.18	funzione fetchPDB	54
3.19	funzione splitRepeatedResidues	55
3.20	funzione PandasPdb.read_pdb	55
3.21	funzione deleteHeteroatomsChains	55
3.22	funzione splitChains	56

3.23	funzione parsePDB	56
3.24	atoms.getHierVieW	56
3.25	writePDB	57
3.26	prepareReceptors	57
3.27	comando per convertire file da <i>.pdb</i> a <i>.pdbqt</i>	58
3.28	createGridboxes	60
3.29	Comando per eseguire performDocking . . .	66
3.30	Comando per eseguire il docking con Auto- dock Vina	67

Capitolo 1

Introduzione

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 soprattutto al lavoro incessante degli insetti impollinatori, soprattutto le api, il polline viene trasportato da una pianta all'altra rendendo possibile la fecondazione di un'essenza 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. Cir-

ca 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 (IUCN) 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 (*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].

Lo scopo della presente tesi è quello di illustrare la progettazione di un software 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.



Figura (1.1): Esemplare adulto di Apis Mellifera

1.1 Docking Molecolare

Il docking molecolare è una tecnica computazionale che mira a determinare le migliori conformazioni adottate da una molecola per legarsi ad un'altra al fine di formare un complesso stabile. Il docking molecolare viene fondamentalemente utilizzato per la valutazione delle interazioni ligando-target. A partire quindi da un ligando e dalla struttura nota del suo target, il docking molecolare permette di generare una serie di conformazioni possibili del ligando stesso localizzato all'interno del sito attivo della proteina. Esse sono denominate "*binding poses*" e sono valutate da particolari funzioni chiamate "*scoring functions*", che creano quindi un vero e proprio ranking. Le *migliori poses* rappresentano quella che viene identificata come la miglior soluzione proposta dall'algoritmo per l'interazione tra il ligando e il target[3].

Gli algoritmi di docking sono formati da due componenti fondamentali: l'algoritmo di ricerca (o "*search algorithm*") e 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 della run 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:

- docking a corpo rigido, che approssima sia il ligando che la proteina come strutture rigide
- docking semi-flessibile, che considera il target come rigido, tendendo però in considerazione i gradi di libertà conformazionale del ligando
- 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, e ognuno sfrutta una particolare coppia algoritmo di *ricerca-scoring function*.

Il docking molecolare consiste in tre obiettivi principali collegati tra loro: predizione della posa, screening virtuale e

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 di trovare la geometria sperimentale del ligando entro un certo limite di tolleranza) e le interazioni fisico-chimiche molecolari associate. Inoltre, 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].

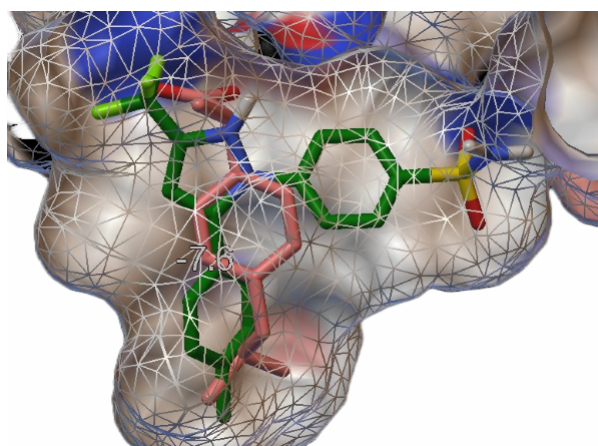


Figura (1.2): Rappresentazione dei due ligandi ibuprofene a sinistra e celecoxib a destra che hanno effettuato il docking con un enzima di COX2

1.2 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, quindi dopo ogni mossa viene ricalcolata l'energia totale del sistema. Questo approccio modella molto precisamente quello che accade nella realtà. Di contro, il costo richiesto in termini di tempo e prestazioni è molto elevato.

1.3 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].

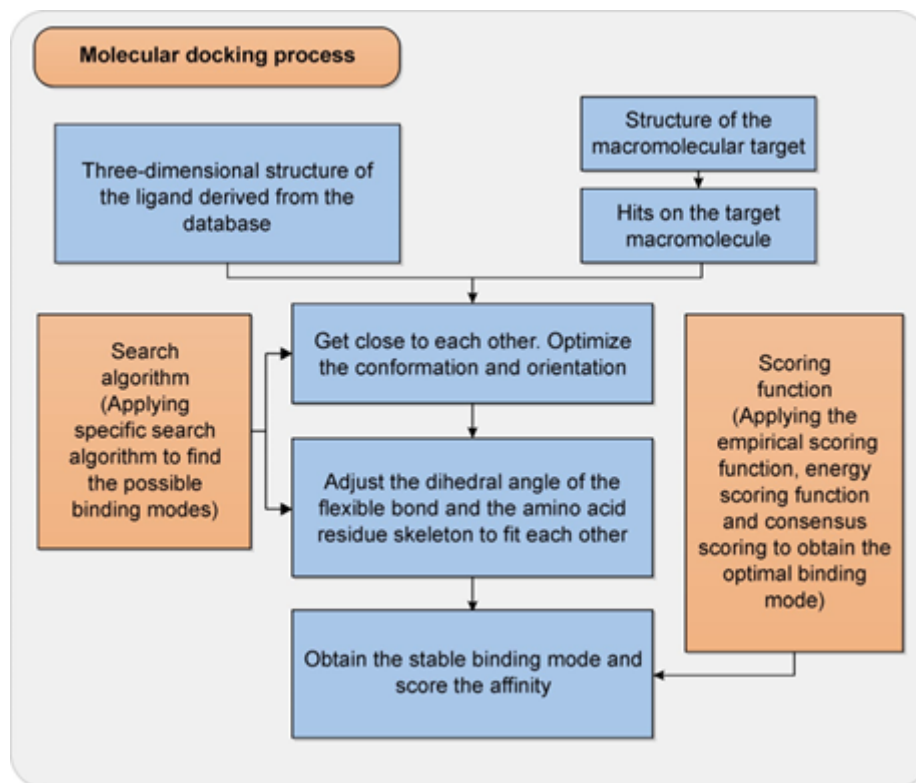


Figura (1.3): Processo del docking molecolare

1.4 Gestione del lavoro

Il software trattato è frutto del lavoro del lavoro congiunto del sottoscritto, del mio collega e laureando Massimiliano Giordano Orsini e del ricercatore del CNR dottor Ferdinando Febbraio. Il lavoro per quanto prodotto e per il suo svolgimento può essere considerato alla stregua di un progetto software vero e proprio. Prima di metterci all'opera io ed il mio collega abbiamo dovuto tenere diversi incontri con il dottor Febbraio in quanto esperto del **dominio applicativo** per chiarire zone a noi oscure dell'ambito in

cui l'applicazione andava sviluppata: la biologia. Dopo varie fasi di **analisi** e di **progettazione** si è arrivati alla fase di **sviluppo** corrispondente alla fase di effettiva implementazione del software, ovviamente è stata effettuata una fase di **testing**, alla quale ha partecipato anche il dottor Febbraio, per individuare e risolvere errori e bug all'interno del software. E' bene chiarire che le varie fasi di sviluppo software non si sono succedute in maniera sequenziale, seguendo quindi i dettami del modello di sviluppo **waterfall** (a cascata), ma le varie fasi sono state inframmezzate l'una con l'altra e più che come passi di un processo è possibile vederle come iterazioni cicliche di esse, risulta chiaro che il modello di sviluppo software di riferimento è quello dello sviluppo **agile**.

Questo processo di sviluppo è stato dettato, non solo dalla scelta a seguito di riflessioni mie e del mio collega, ma anche a causa della natura stessa del progetto, poichè a seguito del repentino raggiungimento con successo degli obiettivi prefissati, sono state aggiunte ulteriori richieste per dotare di maggiore complementarietà il software prodotto.

1.5 Idea e sviluppo

L'**idea** nasce dall'attività di tirocinio svolta presso il "Centro nazionale di ricerca" di Napoli, per un totale di 300 ore (12 CFU), sotto la supervisione del Responsabile del laboratorio professore Angelo Ciaramella e del dottore Ferdinando Febbraio del CNR di Napoli. Il lavoro effettuato è consistito nella realizzazione di un software, del tutto pre-

liminare 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.

1.6 Contenuto della tesi

La tesi è divisa in tre moduli:

- nella prima parte verranno discusse le tecnologie, le piattaforme scelte per la realizzazione del software, i linguaggi di programmazione e gli strumenti di bioinformatica utilizzati
- nella seconda verrà illustrata l'applicazione realizzata, dalla preparazione dei ligandi e recettori, passando per il docking, finendo con l'estrazione dei legami dall'output ottenuto
- nell'ultima parte verranno trattate le conclusioni e saranno indicati gli sviluppi futuri del software realizzato.

Capitolo 2

Tecnologie e piattaforme

In questo capitolo verranno trattate le tecnologie utilizzate, a partire dai linguaggi, i tools utilizzate 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 Linguaggi e tools

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.

2.1.1 Python

Python è un linguaggio di programmazione dinamico orientato agli oggetti utilizzabile per molti tipi di sviluppo software. Offre un forte supporto all'integrazione con altri lin-

guaggi e programmi, è fornito di una estesa libreria standard e può essere imparato in pochi giorni. Python inoltre è fornito delle librerie di bioinformatica adatte allo scopo del software realizzato. Di seguito le principali utilizzate:

- **PubChemPy**, per le ricerche chimiche per nome, sottostruttura e somiglianza, standardizzazione chimica, conversione tra formati di file chimici, rappresentazione e recupero delle proprietà chimiche
- **ProDy**, per l'analisi della dinamica strutturale delle proteine
- **pandas**, per la manipolazione ed analisi dei dati
- **Tkinter/customTkinter**, 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**, per la creazione di visualizzazioni statiche, animate e interattive
- **NumPy**, per la gestione delle strutture dati
- **os**, 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.

2.1.1.1 PubChemPy

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**, viene valutata e quindi viene inviata una risposta. Ci sono alcuni aspetti negativi: È meno adatto per lavori riservati, richiede una connessione costante a Internet e alcune operazioni saranno più lente rispetto a quelle eseguite localmente sul proprio computer. D'altra parte, questo significa che abbiamo 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 [7].

2.1.1.2 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 teorica-

mente. 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[8].

2.1.1.3 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[9].

2.1.1.4 Tkinter

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**[\[10\]](#).

2.1.1.5 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 incorpo-

rato 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[11].

2.1.2 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[12].

2.1.3 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*. 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 recet-

tori e per effettuare la loro conversione dal formato *.pdb* al formato *.pdbqt* necessario per la procedura di docking.

2.1.4 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

2.1.5 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 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, tutti i bit 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 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[13].

2.2 Database

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.

2.3 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.

Le popolazioni di soluzioni sono ottenute tramite operatori genetici (mutazioni, crossover e migrazioni) che imitano quelli biologici. I gradi di libertà sono codificati in geni o

stringhe binarie, e a geni e cromosomi è assegnato un valore basato sulla fitness della scoring function. Le operazioni di mutazione causano cambiamenti nel valore di un gene, mentre il crossover muove un set di geni da un cromosoma “genitore” ad un altro; la migrazione invece muove singoli geni da una sottopopolazione ad un'altra.

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.

2.4 Autodock Vina

AutoDock Vina è un nuovo programma per il docking molecolare e lo screening virtuale. Vina rappresenta la nuova versione di AutoDock, ed 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 facilità 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*[\[14\]](#).

2.4.1 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 po-

tenziali. 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 migliori 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:

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

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:

$$c = c_{inter} + c_{intra} \quad (2.2)$$

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:

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

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:

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

Per ragioni di modularità, gran parte del programma non fa riferimento ad alcuna forma funzionale delle interazioni $f_{t_it_j}$ 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	<i>gauss₁</i>
-0.00516	<i>gauss₂</i>
0.840	<i>repulsion</i>
-0.0351	<i>hydrophobic</i>
-0.587	<i>hydrogenbonding</i>
0.0585	<i>N_{rot}</i>

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 PDB-

bind. 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 $f_{t_it_j}$ sono definite rispetto alla distanza di superficie $d_{ij} = r_{ij} - R_{ti} - R_{tj}$:

$$f_{t_it_j}(r_{ij}) \equiv h_{t_it_j}(d_{ij}) \quad (2.5)$$

dove R_t è il raggio di van der Waals dell’atomo di tipo t . Nella nostra funzione di scoring, $h_{t_it_j}$ 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:

$$gauss_1(d) = e^{-(d/0.5\text{\AA})^2} \quad (2.6)$$

$$gauss_2(d) = e^{-((d/0.5\text{\AA})/2\text{\AA})^2} \quad (2.7)$$

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

Il termine idrofobico è uguale a 1, quando $d < 0,5\text{\AA}$; 0, quando $d > 1,5\text{\AA}$, ed è interpolato linearmente tra questi valori. Il termine di legame a idrogeno è uguale a 1, quando $d < -0,7\text{\AA}$; 0, quando $d < 0,5\text{\AA}$, e viene interpolato linearmente tra questi valori. Seguendo Xscore, trattiamo formalmente i metalli come donatori di legami a idrogeno. In questa implementazione, tutte le funzioni di interazione $f_{t_it_j}$ sono tagliate a $r_{ij} = 8\text{\AA}$.

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[15].

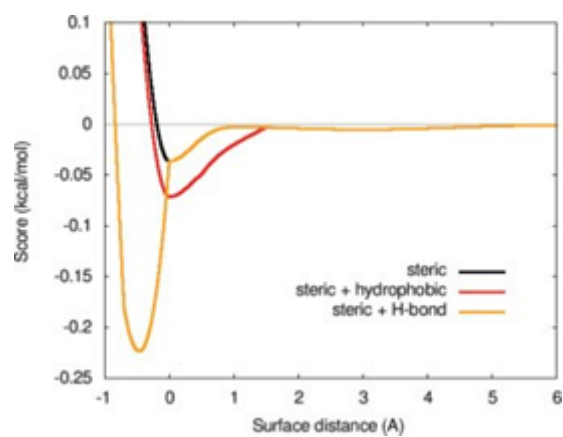


Figura (2.1): Funzione di scoring pesata

Capitolo 3

Applicazione realizzata

Il progetto di tesi proposto ha come focus principale la realizzazione del docking tra i ligandi contenuti in specifici pesticidi e i recettori dell'apis mellifera e l'estrazione dei legami che si vengono a formare.

3.1 Installazione

L'applicazione viene scaricata dalla [repository di github](#) mediante il comando:

```
1 git clone https://github.com/mungowz/Computational-Docking.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 inoltre 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

Il software realizzato può essere utilizzato in due modalità: mediante script python da terminale o mediante interfaccia grafica realizzata seguendo il paradigma *model-view-controller*.

Non sarà necessario effettuare ulteriori installazione 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**

La GUI usufruirà degli stessi lanciati da riga di comando il che garantisce 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 scriverne di nuovo.

Per avviare la GUI deve essere digitato nel terminale il comando:

```
1 python app.py
```

Listato (3.2): Comando per avviare la GUI

Avviando 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.

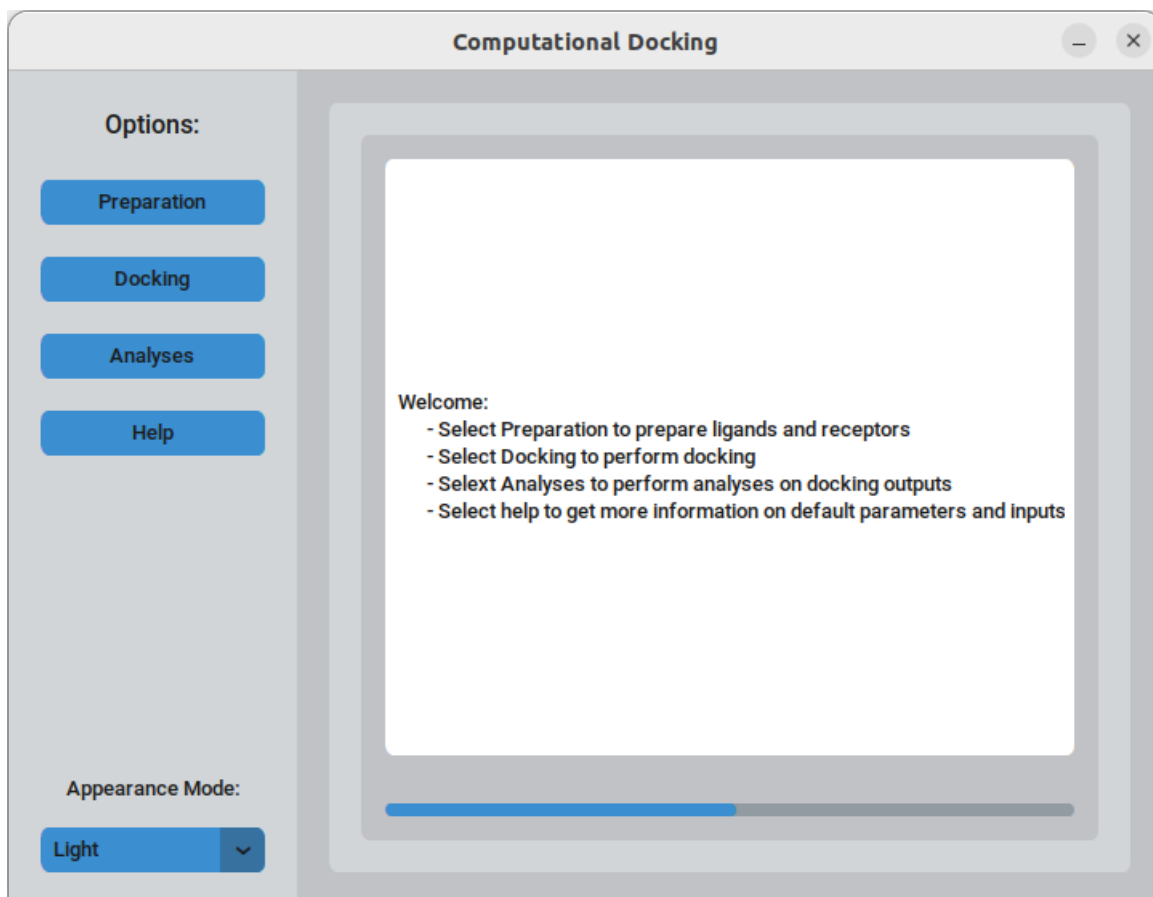


Figura (3.1): Sezione **Home Page** della GUI

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'appen-

dice A.1.

I recettori vengono selezionati mediante una **web view** aperta sulla pagina di ricerca del sito di [PubChem](#), quivi l'utente digiterà la propria query e dopo aver selezionato il tasto **research** 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 tutte le proteine dei recettori dell'Apis Mellifera come mostrato nelle foto: 3.2 e 3.3, per un totale di 60 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** (3.4) contenente le informazioni relative alle impostazioni di default e al formato dei dati in input.

Query

Get Query

RCSB PDB
Deposit
Search
Visualize
Analyze
Download
Learn
More
Documentation
Careers
MyPDB
Contact us

RCSB PDB
PROTEIN DATA BANK

197,512 Structures from the PDB
1,000,361 Computed Structure Models (CSM)

3D Structures
Enter search term(s), Entry ID(s), or sequence
Include CSM

Advanced Search | Browse Annotations

PDB-101
PDB
EMDataResource
NUCLEIC ACID DATABASE
wwPDB Foundation

f t y d

Search
Query History
Browse Annotations
MyPDB

Use the **Advanced Search Query Builder** tool to create composite boolean queries. See the [Help](#) page for more detailed information.

Advanced Search Query Builder
Help

Full Text

Structure Attributes

Scientific Name of the Source Organism
x
has exact phrase
Apis mellifera
+ NOT
Count
x

Add Attribute
Add Subquery
Remove Subquery

Add Subquery

Chemical Attributes

Sequence Similarity

Sequence Motif

Structure Similarity

Structure Motif

Chemical Similarity

Return
Structures
grouped by
No Grouping
Include Computed Structure Models (CSM)
Count
Clear
Search

Figura (3.2): Query dei recettori

36

Query

Get Query

RCSB PDB
Deposit
Search
Visualize
Analyze
Download
Learn
More
Documentation
Careers
MyPDB
Contact us

Return
Structures
grouped by
No Grouping
Include Computed Structure Models (CSM)
Count
Clear
Search

Search Summary

This query matches 60 Structures.

Refinements

Structure Determination Methodology

☐ experimental (60)

Scientific Name of Source Organism

☐ Apis mellifera (60)
☐ Escherichia coli (7)
☐ Escherichia coli K-12 (2)
☐ Chlamydomonas reinhardtii (1)
☐ Mus musculus (1)
☐ Staphylococcus aureus (1)

Taxonomy

☐ Eukaryota (60)
☐ Bacteria (9)

Experimental Method

☐ X-RAY DIFFRACTION (42)
☐ ELECTRON MICROSCOPY (9)
☐ SOLUTION NMR (9)

Polymer Entity Type

☐ Protein (59)
☐ RNA (9)

1 to 25 of 60 Structures

5YYL

Structure of Major Royal Jelly Protein 1 Oligomer

Tian, W., Chen, Z.

(2018) Nat Commun 9: 3373-3373

Released2018-08-08

MethodX-RAY DIFFRACTION 2.65 Å

OrganismsApis mellifera

MacromoleculeApisimin (protein)
Major royal jelly protein 1 (protein)

Unique Ligands94R, NAG

Unique branched monosaccharidesNAG

7ASD

Structure of native royal jelly filaments

Mattei, S., Ban, A., Piconi, A., Leibundgut, M., Glockshuber, R., Boehringer, D.

(2020) Nat Commun 11: 6267-6267

Released2020-12-30

MethodELECTRON MICROSCOPY 3.5 Å

OrganismsApis mellifera

MacromoleculeApisimin (protein)

Figura (3.3): Proteine dei recettori dell'apis mellifera

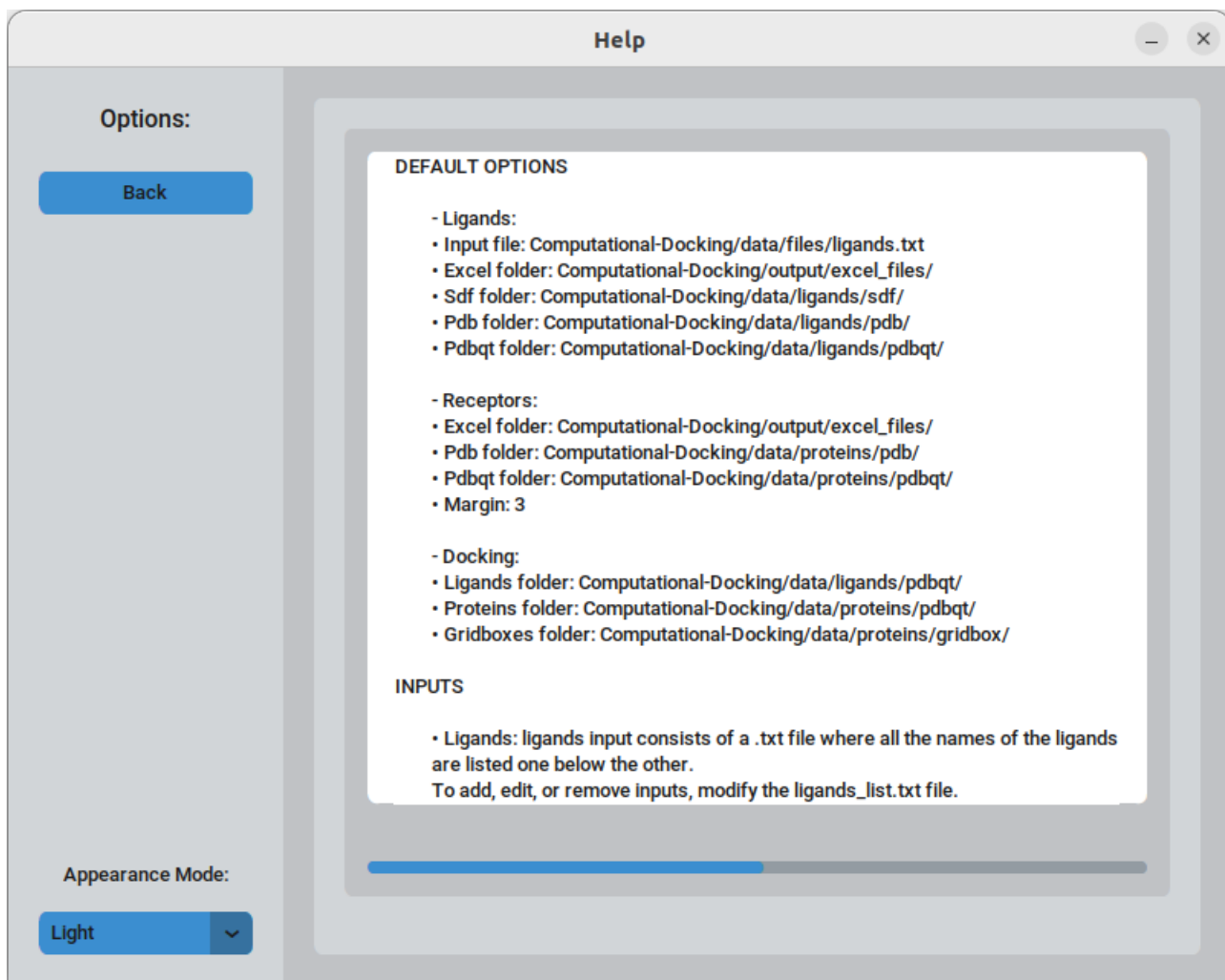


Figura (3.4): Sezione **Help** della GUI

3.4 Preparazione dei ligandi e dei 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** mentre, 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 3.5.

- 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.

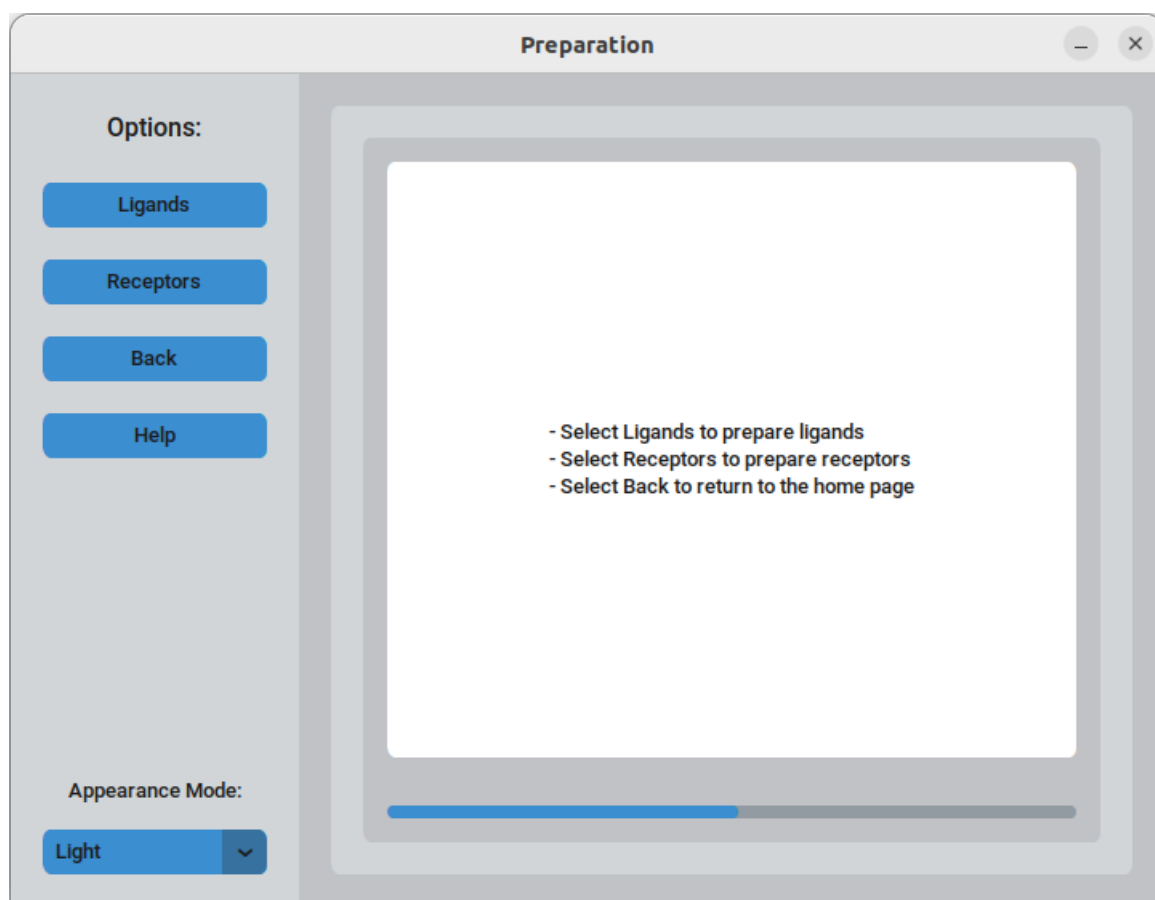


Figura (3.5): Sezione **Preparation** della GUI

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:

```
1 python prepare_ligands.py
```

Listato (3.3): Comando per scaricare la repository

Questo script può ricevere diversi argomenti in input:

- `[-v][--verbose]`: serve per attivare il verbose e se non viene specificata tale opzione viene lasciata di default inattivo
- `[-e][--input - file]`: 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][--excel - folder]`: specifica la directory dei file *.xlsx* di output, se non viene specificata tale opzione verrà scelta la directory di default
- `[-s][--sdf - folder]`: specifica la directory dei file *.sdf* di output, se non viene specificata tale opzione verrà scelta la directory di default
- `[-P][--pdb - folder]`: specifica la directory dei file *.pdb* di output, se non viene specificata tale opzione verrà scelta la directory di default
- `[-p][--pdbqt - folder]`: specifica la directory dei file *.pdbqt* di output, se non viene specificata tale opzione verrà scelta la directory di default

- `[-k][--keep-ligands]`: 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][--help]`: stampa la spiegazione degli input per lo script.

Lo script quando eseguito andrà ad inizializzare i vari percorsi, e nel caso siano stati inseriti in input 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 a meno di input inseriti dall'utente e saranno cancellati i file precedentemente scaricati a meno di input contrario.

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 ci vengono dati 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*.

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

Listato (3.4): funzione selectLigands

L'interfaccia con il database 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_*, gli viene fornito in input. Questa 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 verrà fornito il record della struttura 3D del ligando in questione.

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

Listato (3.5): funzione pubchempy.get_compounds

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

- *outformat*, il formato in cui deve essere scaricato il file, 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.

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

Listato (3.7): funzione sdf2pdb

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

```
1 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 è effettuato dalla funzione **prepareLigands**, questa funzione 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 la conversione in file *.pdbqt* dei corrispondenti i file *.pdb* dati in input.

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

Listato (3.9): funzione prepareLigands

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

```
1 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 precedentemente 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 3.6 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, verranno 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 in figura 3.6 e 3.7.

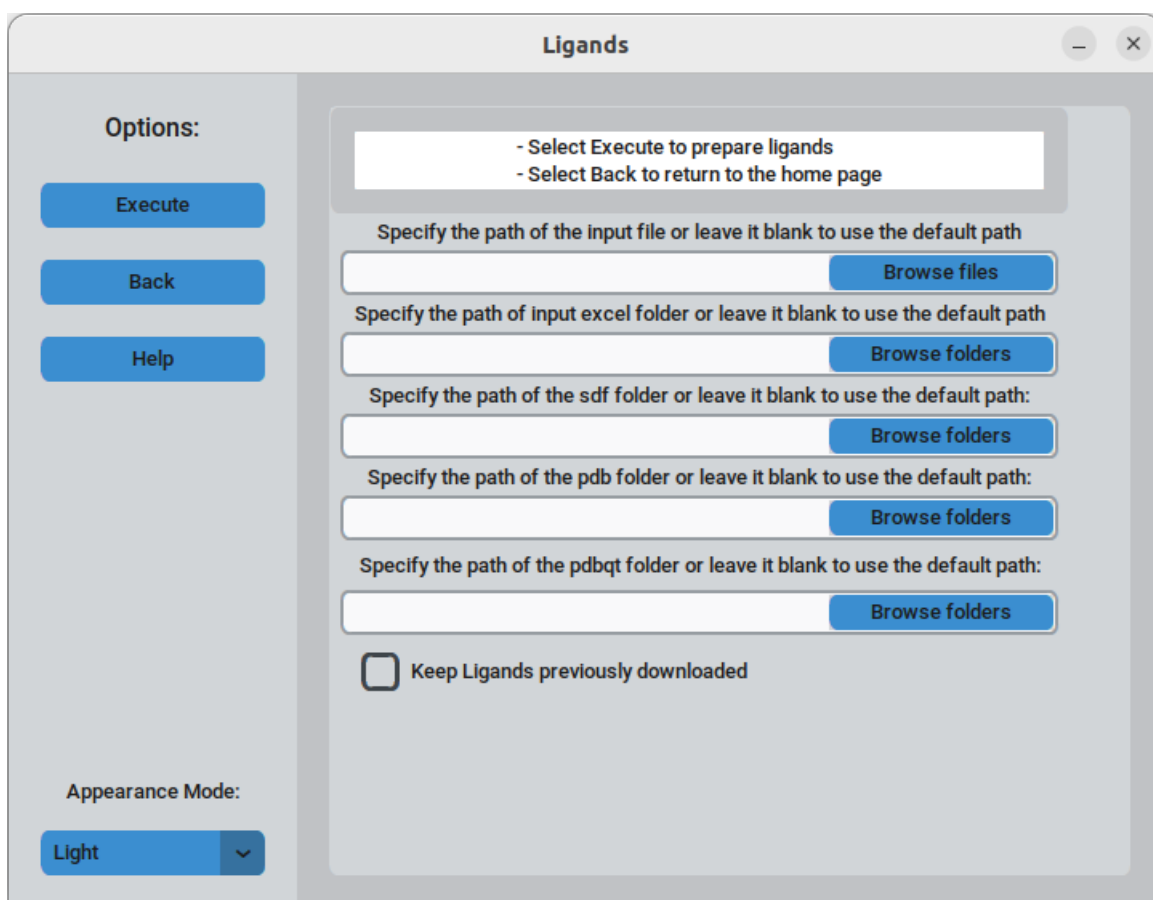


Figura (3.6): Sezione **Ligands** della GUI

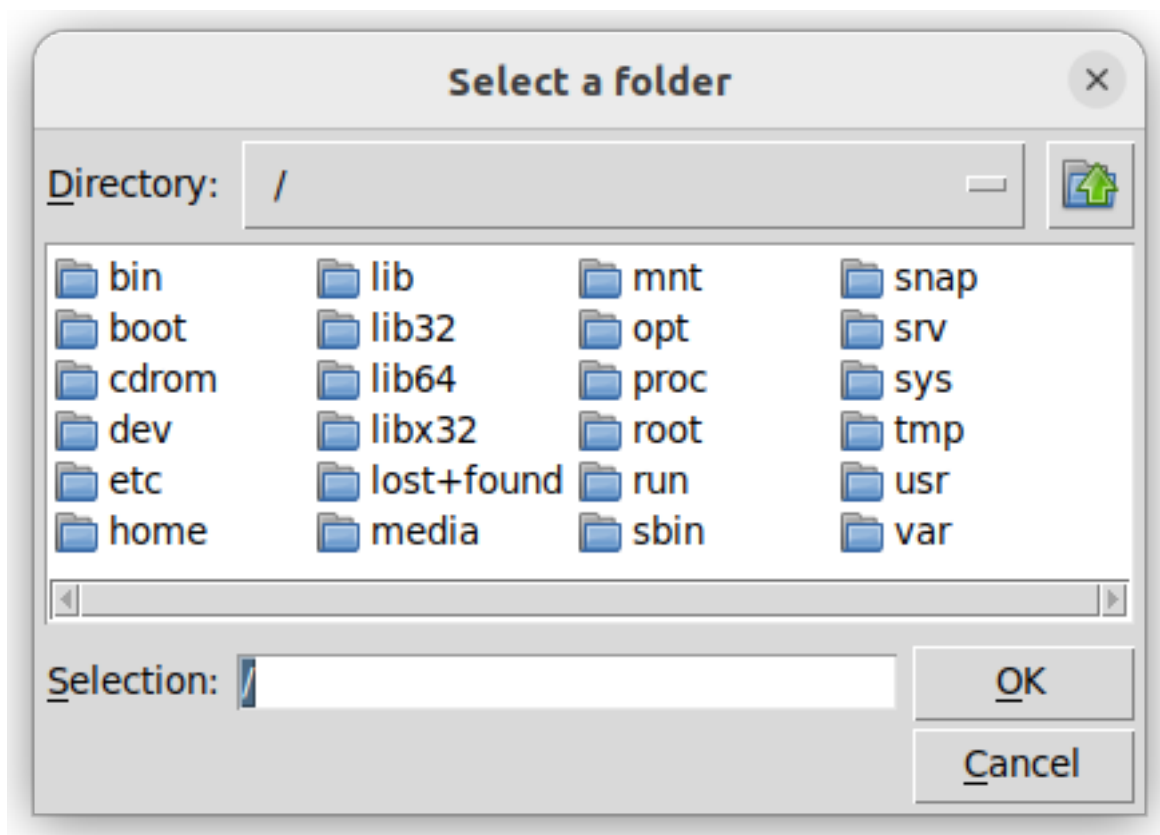
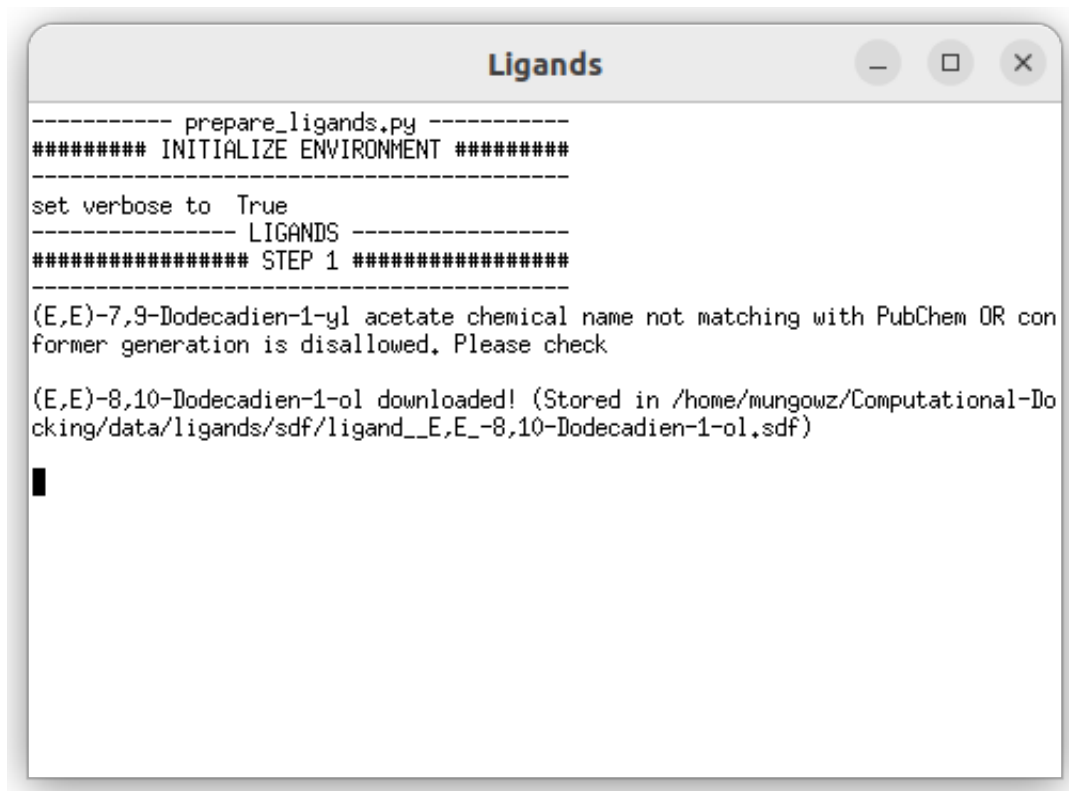


Figura (3.7): Browse folders

La GUI per la preparazione dei ligandi richiama lo script **prepare_ligands.py** discusso nella precedente sezioni, 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** (3.8) 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: 3.9) ed in caso di input non valido sarà restituito un messaggio di

errore (figura: 3.10).



```
----- prepare_ligands.py -----
##### INITIALIZE ENVIRONMENT #####
-----
set verbose to True
----- LIGANDS -----
##### STEP 1 #####
-----
(E,E)-7,9-Dodecadien-1-yl acetate chemical name not matching with PubChem OR con
former generation is disallowed. Please check

(E,E)-8,10-Dodecadien-1-ol downloaded! (Stored in /home/mungowz/Computational-Dock
ing/data/ligands/sdf/ligand__E,E_-8,10-Dodecadien-1-ol.sdf)

█
```

Figura (3.8): **Pannello** dell'esecuzione dei ligandi



Figura (3.9): Messaggio processo completato con successo



Figura (3.10): 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:

```
1 python prepare_receptors.py
```

Listato (3.11): funzione prepare_receptors

Questo script può ricevere diversi argomenti in input:

- `[-v][--verbose]`: serve per attivare il verbose e se non viene specificata tale opzione viene lasciata di default inattivo
- `[-E][--excel-folder]`: specifica la directory dei file `.xlsx` di output, se non viene specificata tale opzione verrà scelta la directory di default
- `[-P][--pdb-folder]`: specifica la directory dei file `.pdb` di output, se non viene specificata tale opzione verrà scelta la directory di default
- `[-p][--pdbqt-folder]`: specifica la directory dei file `.pdbqt` di output, se non viene specificata tale opzione verrà scelta la directory di default

- `[-k][--keep-pdb-files]`: 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][--margin]`: 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][--help]`: stampa la spiegazione degli input per lo script.

Lo script quando eseguito andrà ad inizializzare i vari percorsi, e nel caso siano stati inseriti in input 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 a meno di input inseriti dall'utente e saranno cancellati i file precedentemente scaricati a meno di input contrario.

Lo script richiama la funzione **selectReceptors** la quale si occupa di scaricare da **Pubchem** 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 ci vengono dati 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 [PubChem](#):

```
1 RestApiSelection(URL)
```

Listato (3.13): funzione RestApiSelection

Questa funzione apre una **web view** sul sito di [PubChem](#) come mostrato nella figura 3.2, 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**.

```
1 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**:

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

Listato (3.15): funzione json.dump

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

```
1 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 **downloadPdb** che prende in input:

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

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

```
1 downloadPdb(pdb_list, output_path, verbose)
```

Listato (3.17): funzione downloadPdb

La funzione **downloadPdb** richiama la funzione **fetchPDB** del pacchetto **ProDy**, questa 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 **PubChem** e scaricare di un singolo file *.pdb* mediante richiesta *FTP*, il download di tutte le proteine della lista avviene ciclando su tutti gli elementi di essa:

```
1 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 **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*:

```
1 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**, questa prende in input il nome del file *.pdb*:

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

Listato (3.20): funzione `PandasPdb.read_pdb`

In particolare gli attributi *A* e *B* sono quelli ripetuti e i *B* saranno quelli eliminati nella colonna *alt_loc*.

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.

```
1 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 aminoacidica distinta e non ripetuta da altre catene.

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

Listato (3.22): funzione splitChains

Tramite la funzione **parsePDB** della libreria **prody** viene ricostruita la struttura proteica attraverso oggetti come: la lista delle molecole, la lista degli atomi e dei residui, questa viene rappresentata da un oggetto della classe **AtomGroup** restituito dalla funzione **parsePDB**.

```
1 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**:

```
1 atoms.getHierView()
```

Listato (3.24): atoms.getHierView

Vengono quindi esaminate tutte le sequenze, sempre tramite la funzione **parsePDB**, e scelte solo le sequenze distinte, infine vengono salvate in un file tutte le catene di-

stinte come se fossero delle proteine attraverso la funzione **writePDB** di **prody**, questa funzione 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.

```
1 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.

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

Listato (3.26): prepareReceptors

La funzione **prepareReceptors** utilizza lo script **prepare_receptors4** offerto dalla suite **ADFR**. In realtà viene

utilizzata una versione modificata di tale script: **replace-PrepareReceptor4.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 tramite riga di comando dalla funzione mediante il comando **prepare_receptor**:

```
1 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*
- *checkhydrogens* 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 **createGrid-boxes** richiamata dallo script **prepare_receptors.py**, la funzione prende in input:

- **pdb_folder**, la directory contenente i file *.pdb*
- **gridbox_output_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*, questa definisce lo spazio conformazionale dove si colloca la proteina, è definita da:

- Un centro
- Le dimensioni
- L'eshaustività, 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**
- **protein_path**, il file *.pdb* corrispondente ad una singola proteina
- **gridbox_output_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 precedentemente 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 3.11 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 verranno 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 in figura 3.6, 3.12 e 3.13.

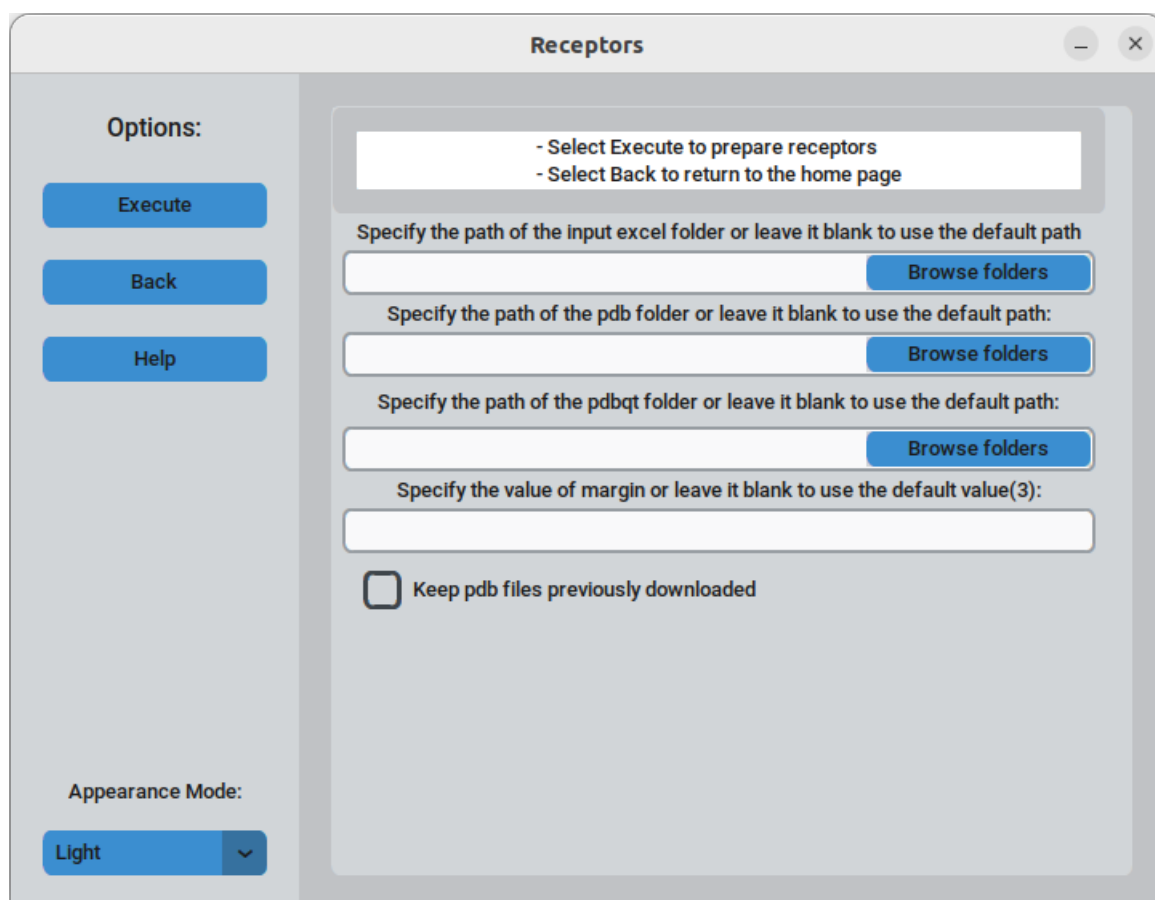


Figura (3.11): Sezione **Receptors** della GUI

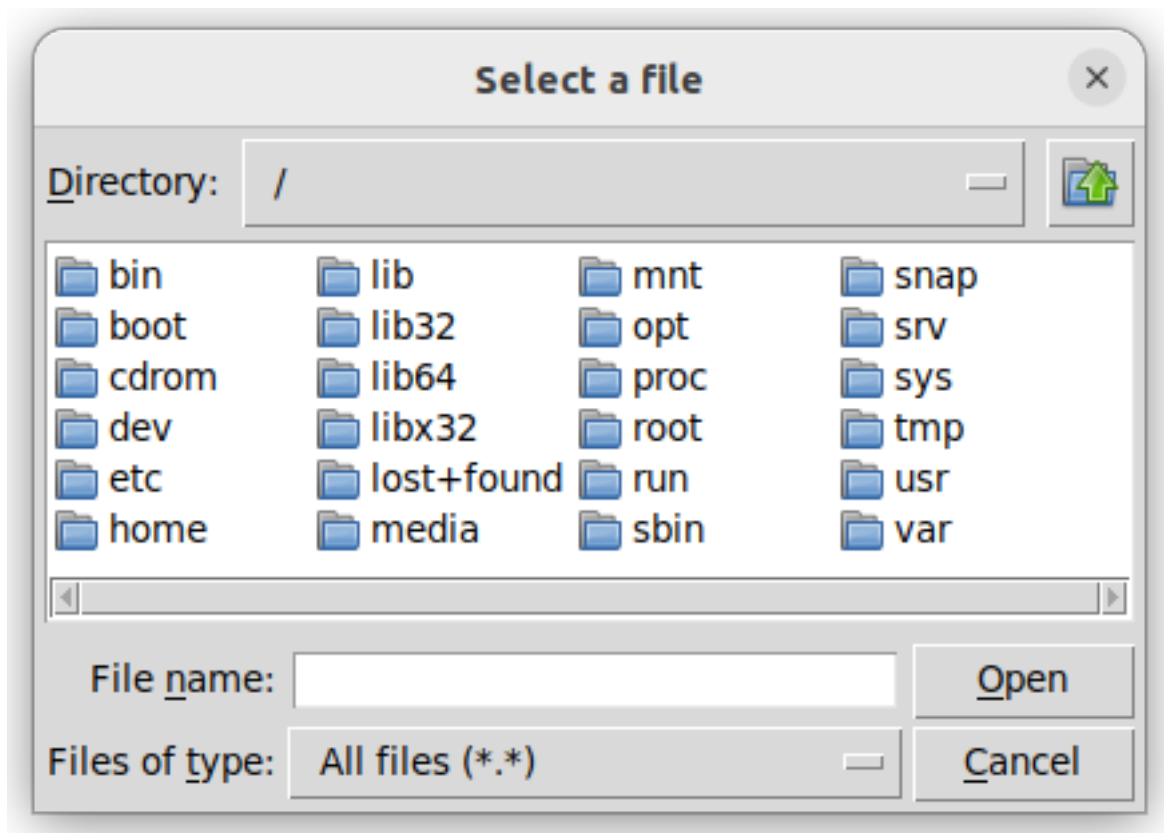


Figura (3.12): Browse files

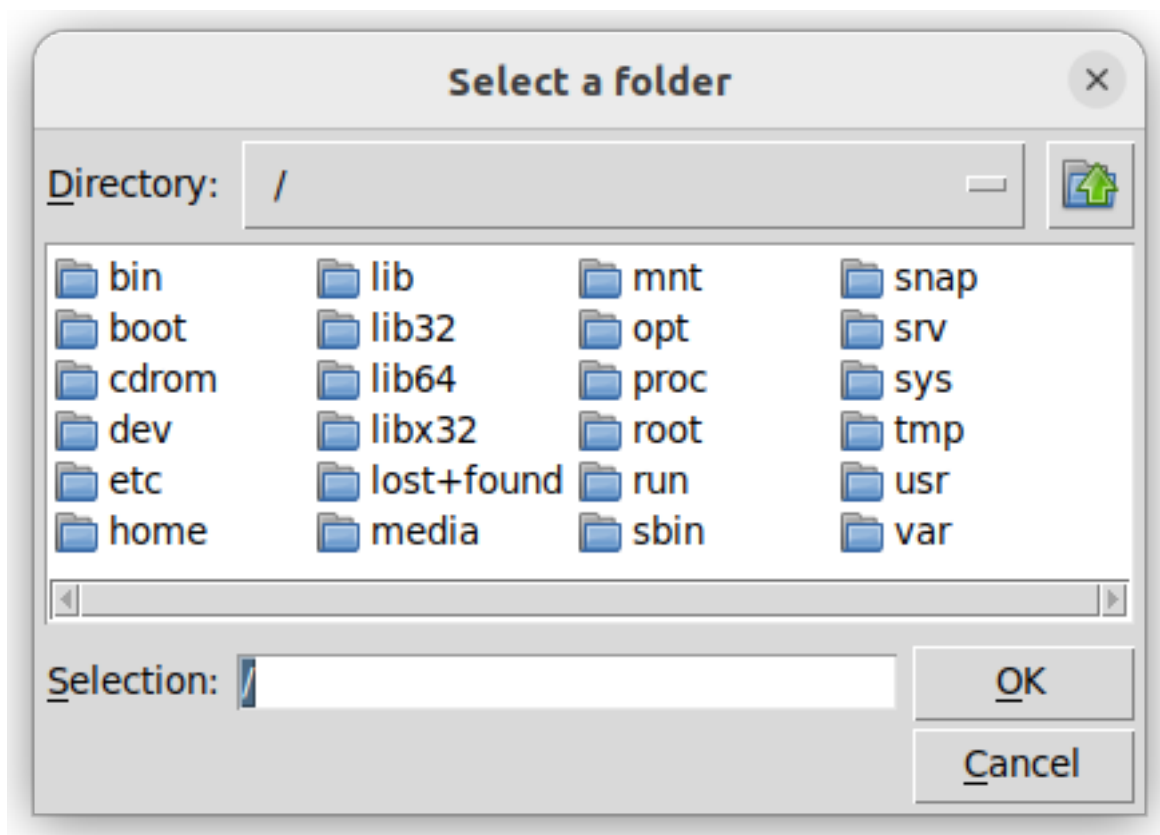
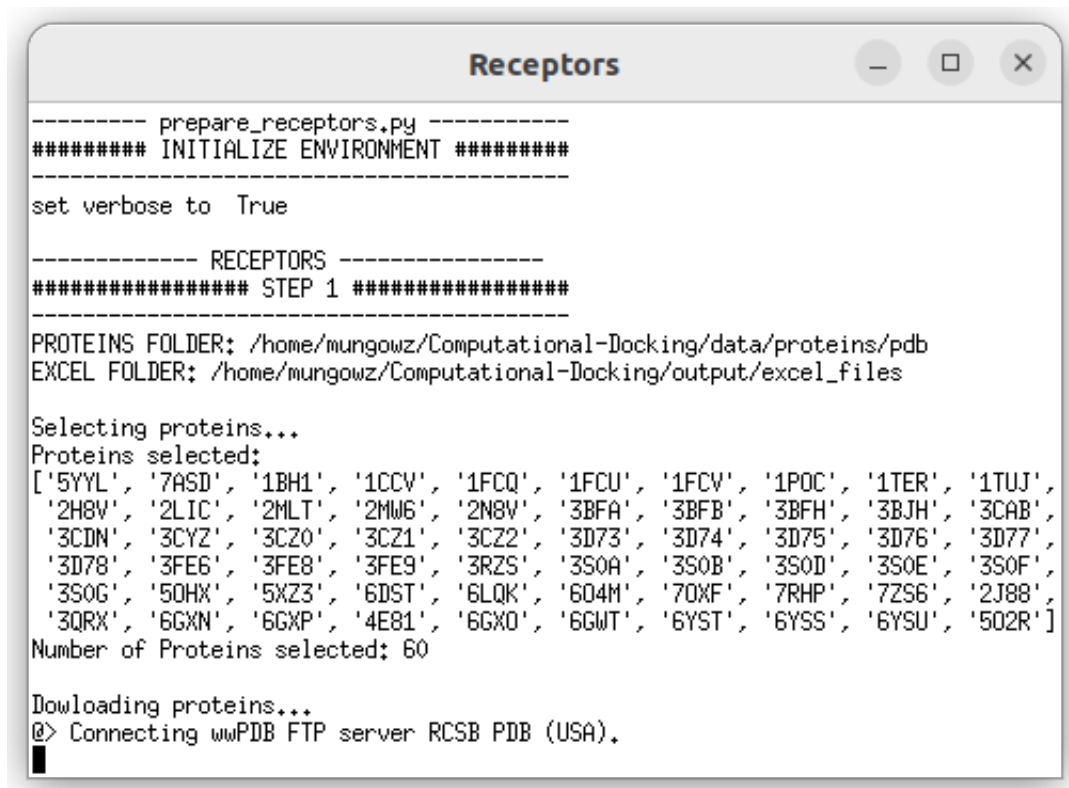


Figura (3.13): **Browse folders**

La GUI per la preparazione dei ligandi richiama lo script **prepare_ligands.py** discusso nella precedente sezioni, 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** (3.14) 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: 3.15) ed in caso di input non valido sarà restituito un messaggio di

errore (figura: 3.16).



```
----- prepare_receptors.py -----
##### INITIALIZE ENVIRONMENT #####
-----
set verbose to True

----- RECEPTORS -----
##### STEP 1 #####
-----
PROTEINS FOLDER: /home/mungowz/Computational-Docking/data/proteins/pdb
EXCEL FOLDER: /home/mungowz/Computational-Docking/output/excel_files

Selecting proteins...
Proteins selected:
['5YYL', '7ASD', '1BH1', '1CCV', '1FCQ', '1FCU', '1FCV', '1POC', '1TER', '1TUJ',
'2H8V', '2LIC', '2MLT', '2MM6', '2N8V', '3BFA', '3BFB', '3BFH', '3BJH', '3CAB',
'3CDN', '3CYZ', '3CZ0', '3CZ1', '3CZ2', '3D73', '3D74', '3D75', '3D76', '3D77',
'3D78', '3FE6', '3FE8', '3FE9', '3RZS', '3S0A', '3S0B', '3S0D', '3S0E', '3S0F',
'3S0G', '5OHX', '5XZ3', '6DST', '6LQK', '6O4M', '7OXF', '7RHP', '7ZS6', '2J88',
'3QRX', '6GXN', '6GXP', '4E81', '6GXO', '6GWT', '6YST', '6YSS', '6YSU', '502R']
Number of Proteins selected: 60

Downloading proteins...
@> Connecting wwPDB FTP server RCSB PDB (USA).
█
```

Figura (3.14): **Pannello** dell'esecuzione dei recettori



Figura (3.15): Messaggio processo completato con successo



Figura (3.16): Messaggio di errore

3.5 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 lanciato 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 creerà la cartella */output/docking* dove verranno salvati i risultati del docking, 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 comando 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 **1fcu** 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 (3.17): File di log del docking per la coppia proteina-ligando **1fcu-bentazone**

3.6 Docking tramite GUI

Il docking tramite GUI avviene selezionando il tasto **Docking** nella sezione **Home page** del software. All'interno 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 3.18 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 verranno 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 in figura 3.18 e 3.19.

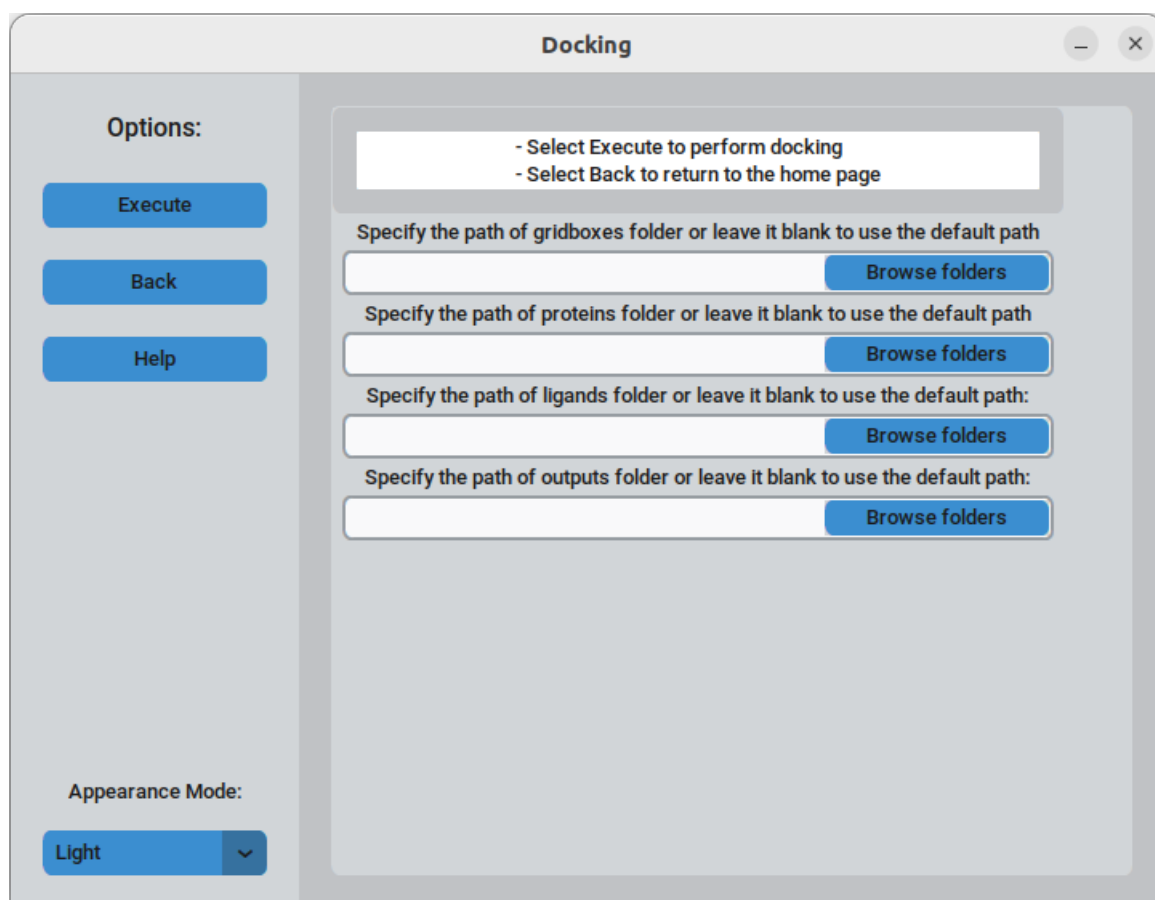


Figura (3.18): Sezione **Receptors** della GUI

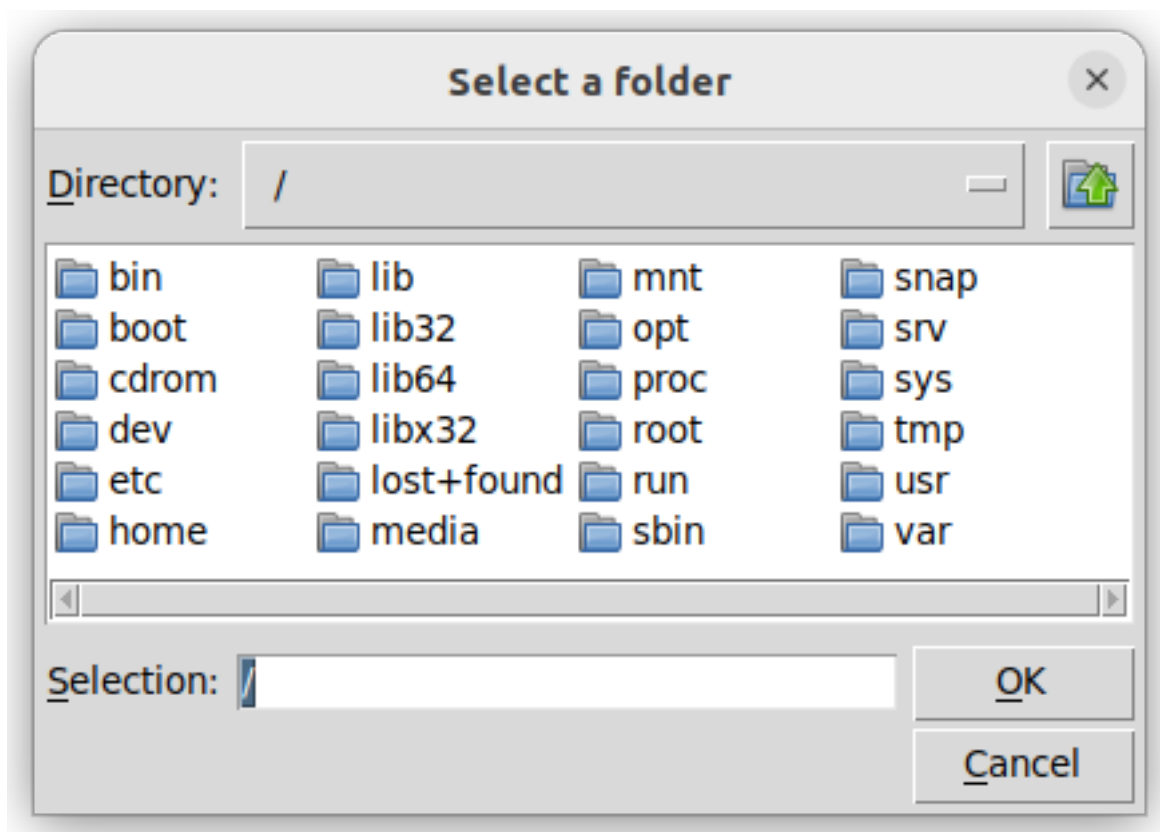
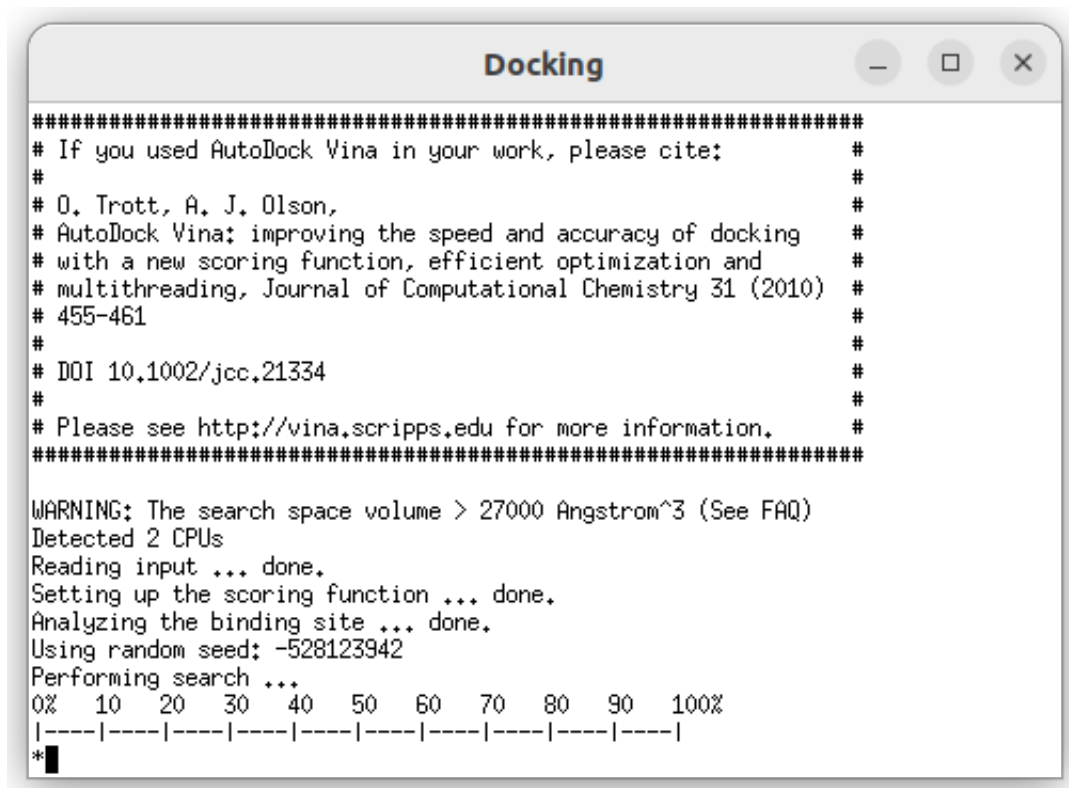


Figura (3.19): **Browse files**

La GUI per il **docking** richiama lo script **performDocking.sh** discusso nella precedente sezioni, 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** (3.20) 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: 3.21) ed in caso di input non valido sarà restituito un messaggio di

errore (figura: 3.22).



```
#####
# 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 2 CPUs
Reading input ... done.
Setting up the scoring function ... done.
Analyzing the binding site ... done.
Using random seed: -528123942
Performing search ...
0%  10  20  30  40  50  60  70  80  90 100%
|----|----|----|----|----|----|----|----|----|----|
*█
```

Figura (3.20): Pannello dell'esecuzione del docking

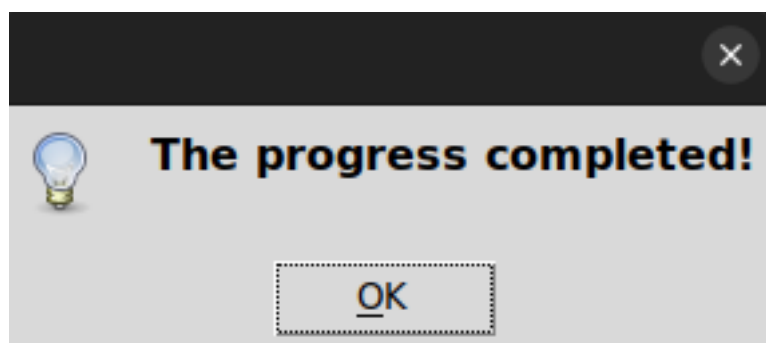


Figura (3.21): Messaggio processo completato con successo



Figura (3.22): Messaggio di errore

Capitolo 4

Risultati sperimentali

Capitolo 5

Conclusioni e sviluppi futuri

Nell'idea iniziale del progetto gli obbiettivi erano quelli di creare un software capace di automatizzare i processi di preparazione degli input per il docking, di esecuzione docking e l'analisi dei suoi risultati, riunendo in una sola applicazione diverse funzioni e strumenti di bioInformatica. Raggiunti tali obbiettivi si è deciso di dotare il software prodotto di un'interfaccia grafica, per permettere anche a chi non abbia particolari conoscenze informatiche e familiarità con l'uso della riga di comando di usufruire dell'applicazione e di completare le funzionalità del software aggiungendo ulteriori analisi ed opzioni.

Il software descritto all'interno di tale documento ha la finalità di dare un apporto importante alla ricerca scientifica, in particolare è progettato con lo scopo di analizzare dell'effetto dei pesticidi sull'apis mellifera. Più nello specifico l'applicazione trattata ha più scopi:

- Riunire in un unico software tutti i tools e strumenti per effettuare il 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.

5.1 Conclusioni

Computational Docking automatizza l'intero processo di screening biochimico partendo dalla preparazione dei **ligandi** e dei **recettori**, passando per il processo di **docking** e concludendo con l'analisi dell'output derivato dalla fase precedente. Viene concessa ampia libertà all'utente per quanto riguarda la scelta degli input e le impostazioni d'uso del software. Il software realizzato, come si evince dai risultati sperimentali, è stato in grado di fornire risultati soddisfacenti dal punto di vista qualitativo, poichè sia i risultati del docking che l'analisi hanno soddisfatto i requisiti e le richieste del nostro esperto del dominio, il dottor Febbraio. 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 **Computational Docking** è dotato di una **GUI** semplice da usare e che si avvicina non solo ai modelli di applicazione maggiormente

diffusi sul mercato, facilitandone la comprensione da parte dell'utente, ma che gode di una buona responsività non solo nei confronti di chi la usa, il quale è sempre tenuto al corrente di ciò che il software stia facendo, ma anche per quanto riguarda i tempi di esecuzione, infatti tramite l'uso del parallelismo e della concorrenza sviluppati attraverso i thread, è possibile effettuare diversi compiti espletati dal software in parallelo e con tempi di risposta contenuti. E' bene ricordare che per andare incontro a macchine le quali non dispongono di molte risorse è possibile utilizzare la versione da riga di comando del software, veloce e leggera. Inoltre l'applicazione è scalabile considerata la facilità di evoluzione per sviluppi futuri. In conclusione **Computational Docking** è in grado di eseguire il docking, preparare i suoi input ed analizzarne gli output in maniera efficiente ed efficace.

5.2 Sviluppi futuri

Il motore di **Computational Docking** è analizzare gli effetti dei composti chimici contenuti all'interno dei pesticidi più diffusi, sulle api del genere *Apis Mellifera*. L'obiettivo a lungo termine dell'applicazione è in realtà allargare il dominio della sua analisi ad altre specie e casi, per esempio l'uomo, questo è sicuramente lo sviluppo futuro principale a cui mira il software.

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 il database di **Pubchem**, ma usufruire anche degli altri archivi di composti chimici ed organici 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 **Computational Docking**.

Detto ciò è chiaro come questo non sia un punto di arrivo ma un solo il punto da cui partire per il progetto **Computational Docking**.

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
Diffufenican	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

5YYL	3BFA	3D78	6LQK	2MLT
7ASD	3BFB	3FE6	6O4M	2MW6
1BH1	3BFH	3FE8	7OXF	2N8V
1CCV	3BJH	3FE9	7ZS6	3D75
1FCQ	3CAB	3R72	2J88	3D76
1FCU	3CDN	3RZS	3QRX	3D77
1FCV	3CYZ	3S0A	6GXN	5OHX
1POC	3CZ0	3S0B	6GXP	5XZ3
1TER	3CZ1	3S0D	4E81	6DST
1TUJ	3CZ2	3S0E	6GXO	6YSS
2H8V	3D73	3S0F	6GWT	6YSU
2LIC	3D74	3S0G	6YST	5O2R

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] Xuan-Yu Meng et al. «Molecular docking: a powerful approach for structure-based drug discovery». In: *Current computer-aided drug design* 7.2 (2011), pp. 146–157.
- [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] Matt Swain. *PubChemPy documentation*. <https://pubchempy.readthedocs.io/en/latest/guide/introduction.html>. 2014.
- [8] 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](https://doi.org/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>.
- [9] The pandas development team. *pandas-dev/pandas: Pandas*. Ver. latest. Feb. 2020. DOI: [10.5281/zenodo.3509134](https://doi.org/10.5281/zenodo.3509134). URL: <https://doi.org/10.5281/zenodo.3509134>.
- [10] John E Grayson. *Python and Tkinter programming*. Manning Publications Co. Greenwich, 2000.
- [11] Paul Barrett et al. «matplotlib—A Portable Python Plotting Package». In: *Astronomical data analysis software and systems XIV*. Vol. 347. 2005, p. 91.

- [12] 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.
- [13] Noel M O’Boyle et al. «Open Babel: An open chemical toolbox». In: *Journal of cheminformatics* 3.1 (2011), pp. 1–14.
- [14] 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.
- [15] 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.