ISTANBUL MEDENIYET UNIVERSITY
INSTITUTE OF GRADUATE STUDIES
DEPARTMENT OF BIOENGINEERING

Developing a Detailed Framework for Covalent Docking, Implementation and Comparative Assessment of Different Tools on a Benchmark Set of Protein-Ligand Complexes

Master's Thesis

Ahmet Can Tekeli

April 2024

ISTANBUL MEDENIYET UNIVERSITY
INSTITUTE OF GRADUATE STUDIES
DEPARTMENT OF BIOENGINEERING

Developing a Detailed Framework for Covalent Docking, Implementation and Comparative Assessment of Different Tools on a Benchmark Set of Protein-Ligand Complexes

Master's Thesis

Ahmet Can Tekeli

Supervisor

Asst. Prof. Saliha Ece ACUNER ZORLUUYSAL

April 2024

THESIS JURY APPROVAL

This Master's thesis titled "Developing a Detailed Framework for Covalent Docking, Implementation and Comparative Assessment of Different Tools on a Benchmark Set of Protein-Ligand Complexes" written by Ahmet Can Tekeli at the Department of Bioengineering was accepted by our jury.

JURY MEMBERS **SIGNATURE**

Supervisor:

Asst. Prof. Saliha Ece ACUNER ZORLUUYSAL

Istanbul Medeniyet University

Other Jury Members:

Asst. Prof. Elif KUBAT ÖKTEM

Istanbul Medeniyet University

Prof. Pemra ÖZBEK SARICA

Marmara University

Date of Defense: 26 / 04 / 2024

STATEMENTS

Style and Reference Manual Statement

Having reviewed this thesis written under my supervision, I confirm that it has been written in accordance with Manual of Style and used its Chicago Manual of Style 17th edition reference format consistently throughout the entire text.

Signature

Asst. Prof. Saliha Ece ACUNER ZORLUUYSAL

Declaration of Originality

I hereby declare that all information in this dissertation has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conducts, I have fully cited and referenced all material and results that are not original to this work.

Signature

Ahmet Can TEKELİ

GENİŞ ÖZET

Kovalent Kenetlenme Metodunun Ayrıntılı Prosedürünün Çıkarılması, Kontrol ve Özgün Olarak Belirlenecek Protein-Küçük Molekül Çiftleri Üzerinde Uygulanması ve Sonuçların Karşılaştırmalı Analizi

Tekeli, Ahmet Can

Yüksek Lisans Tezi, Biyomühendislik Anabilim Dalı

Danışman: Dr. Öğr. Üyesi Saliha Ece ACUNER ZORLUUYSAL

Nisan 2024

Proteinler, biyolojik sistemlerde çok çeşitli işlevlere sahip olan çok önemli makromoleküllerdir. Etkilerini nükleik asitler, hücresel organeller, oksijen ve ligandlar gibi küçük bileşikler de dâhil olmak üzere çeşitli moleküller ve yapılarla etkileşim kurarak ortaya koyarlar. Protein-ligand etkileşimlerinin ve bağlanma mekanizmalarının anlaşılması, rasyonel ilaç tasarım sürecinin temelidir. Bu amaca hizmet etmek için geliştirilmiş birçok deneysel teknik vardır. Kristalografi, statik bir yapı içindeki moleküllerin termodinamiğini analiz ederek protein-ligand etkileşimleri hakkında değerli bilgiler sağlar. NMR spektroskopisi, özellikle protein-ligand komplekslerinin dinamik davranışını incelemek için bir başka güçlü tekniktir. Diferansiyel tarama kalorimetrisi (DSC) ve izotermal titrasyon kalorimetrisi (ITC) gibi kalorimetrik yöntemler, bu komplekslerin termodinamiğini araştırmak için ek bir yaklaşım sunar (Bronowska 2011).

Protein-ligand etkileşimlerini incelemeye yönelik deneysel teknikler karmaşık, pahalı ve zaman alıcı olabilir. Rasyonel ilaç tasarımı genellikle yüzlerce küçük molekülün taranmasını içerdiğinden, bu deneysel yöntemler pratik olmaktan çıkmaktadır. Bu durum, çeşitli koşulları simüle edebilen, entalpi, entropi ve serbest enerji değişikliklerini çok daha hızlı hesaplayabilen hesaplama tekniklerinin değerini ortaya koymakta ve bu teknikleri büyük ölçekli çalışmalar için ideal hale getirmektedir (X. Du et al. 2016). Protein-ligand kenetlenmesi ve ardından moleküler dinamik (MD) simülasyonları, tüm bağlanma süreci ve kompleksin zaman içinde nasıl değiştiği hakkında güvenilir bilgiler sunar.

Moleküler kenetlenmenin özel bir durumu olan kovalent kenetlenmede, hedefledikleri proteinle aralarında kovalent bağ oluşturma potansiyeline sahip ligandların etkileşimleri hesaplamalı olarak modellenmektedir. Bu teknik ilaç tasarımı ve keşfi gibi alanlarda kullanılan önemli bir yöntemdir. Hedefine

kovalent bağlanan potansiyel ilaç adaylarının tasarlanması, tespit edilmesi, mevcut ilaçların optimizasyonu ve yeniden konumlandırılması için kullanılır. Ligandlar, kendilerinde bulunana "warhead" diye adlandırılan başlıklar aracılığı ile nükleofilik amino asit gruplarına bağlanırlar. Çok çeşitli başlıklar ve reaksiyonlar mevcuttur. Kovalent bağ yapma potansiyeli olan ligandlar, hedef reseptöre tersinir olmayan şekilde bağlanmaları nedeniyle kalıcı ve yüksek etkinlik gösteren ilaç adayları olabilirler. Bu yöntem ilaç keşfi ve tasarımının yanı sıra biyolojik sistemlerin etkileşimlerinin anlaşılması için de kullanılabilir.

Moleküler kenetlenme, bir ligandın bir reseptöre bağlanma şeklini ve yakınlığını tahmin etmek için kullanılan hesaplamalı bir yöntemdir. Potansiyel yeni ilaçların tanımlanmasında ve mevcut ilaçların tasarımının optimize edilmesinde kullanılabilir olduğu için ilaç tasarımında değerli bir araçtır. Protein-ligand kenetlenmesi, X-ışını kristalografisi, NMR veya homoloji modelleme gibi tekniklerden elde edilen bağlanma bölgesinin yapısal bir modeline dayanır. Süreci kolaylaştırmak için kenetlenme, ayrı yöntemler veya hedef proteinin önceden bilinmesiyle belirlenen önceden tanımlanmış bir bağlanma bölgesine yoğunlaşır (Grinter ve Zou 2014).

Kovalent kenetlenme, ilaç tasarımında değerli bir yöntemdir, çünkü kovalent inhibitörler, ligand üzerindeki bir elektrofilik kovalent başlık ile hedef protein üzerindeki bir nükleofilik tortu arasında belirgin bir kimyasal bağ oluşturur (Scarpino, Ferenczy ve Keserü 2018). Kovalent bir bağ oluşumunun dikkate alınmasını gerektirdiğinden, mevcut ilaçlardan daha güçlü ve seçici olan potansiyel yeni ilaçların tanımlanmasında kullanılabilir. Ayrıca, bağlanma afinitesini ve seçiciliğini geliştirmenin yollarını belirleyerek mevcut ilaçların tasarımını optimize etmek için de kullanılabilir. Kovalent kenetlenme benzer prensiplere sahip olmasına rağmen bazı limitasyonlardan dolayı moleküler kenetlenmeye göre daha karmaşık bir işlemdir. Tüm bu karmaşıklık ve limitasyonlar kovalent kenetlenmeyi daha az bilinen ve çalışılan bir alan olmaya iter. Ancak kovalent kenetlenme ile keşfedilecek ve tasarlanacak olan kovalent inhibitörler, bağlanmanın tersinir olmaması ve proteinlerle kovalent olmayan inhibitörlere göre daha yüksek yakınlığa sahip olmasından dolayı, hastalıklarda daha etkili ve kalıcı bir çözüm olma potansiyeline sahiptir. Ayrıca kovalent kenetlenme ile ligand üzerinde bulunan kovalent başlık hedef protein üzerindeki nadir ve korunmayan bölgeleri bulabilir ve çok yüksek seçiciliğe sahip bir inhibitörün geliştirilmesine öncü olabilir. Kovalent inhibitörler kovalent olmayan inhibitörlere göre daha yüksek potansiyele sahip olduğu için bağlanma bölünmeleri zor olan proteinlere etki etme olasılıkları çok daha yüksektir.

Bağlanma afinitelerini tahmin etmeyi amaçlayan puanlama fonksiyonları, kenetlenme yöntemleri içinde önemli bir unsurdur. Bununla birlikte, yapıların daha yüksek karmaşıklığı, hidrojen bağları, elektrostatik etkileşimler, van der Waals etkileşimleri veya hidrofobik etkileşimler gibi kovalent olmayan etkileşimlerin artan olasılığı, çözücü etkileri, entropik katkılar, tahmini daha az doğru hale getirir. Bazı puanlama fonksiyonları, yüksek afiniteli ligandlar ile düşük afiniteli ligandlar arasında ayrım yapmakta zorlanır. Puanlama işlevinin iyileştirilmesi, kenetlenme algoritmalarının ana amacıdır. Genel olarak, kenetlenme algoritmasının hızı ile doğruluğu arasında bir denge söz konusudur. Daha gelişmiş fonksiyonlar ve ayrıntılı örnekleme, doğruluğu ancak hesaplama maliyetini de artırır. Protein esnekliği, doğruluğu dolayısıyla hesaplama maliyetini etkileyen bir diğer husustur (Grinter ve Zou 2014). Protein esnekliği de hesaba katıldığında, verimlilik ve örnekleme bütünlüğünü dengelemek bir zorluktur. Kovalent kenetlenme, standart kenetlenmenin sınırlamalarının ötesinde de sınırlamalarla karşı karşıyadır. Kovalent bağların özgüllüğü nedeniyle uygun ligandların bulunması daha da zorlaşmaktadır. Ligandlar, savaş başlığı reaksiyonları ve hedef kalıntılar hakkında ön bilgilerin toplanması ön işleme aşamasına karmaşıklık katmaktadır. Ek olarak, farklı kenetlenme programlarında dosya biçimleri ve hazırlama gereksinimlerindeki farklılıklar, genel sürecin yönetilmesini zorlaştırabilir.

Bu tez, ücretsiz kovalent kenetlenme programları olan AutoDock4, UCSF DOCK6 ve Cresset Flare hakkında rehberlik sağlayarak kovalent yerleştirme konusunda kapsamlı ve güvenilir kaynak eksikliğini gidermeyi hedeflemektedir. Her program için en uygun parametreler belirlenerek oluşturulmuş detaylı prosedür adım adım talimatlar sağlayarak, alana yeni başlayanların bile başarılı simülasyonlar yürütmesini ve aynı zamanda kullanıcıların kovalent kenetlenmenin prensipleri hakkında kapsamlı bir anlayış kazanmalarını sağlar. Prosedürlerin etkinliğini göstermek için, 40 protein-ligand kompleksinden oluşturulmuş özel bir kıstas veri seti üzerinde kovalent kenetlenme deneyleri gerçekleştirmiştir. Bu deneyler ligandın başlangıçtaki konumunun kenetlenme tahmin doğruluğuna etkisini anlamak için; tüm protein-ligand kompleksleri için başlangıç konformasyonu olarak proteine bağlı haldeki orijinal ligand ve koordinatları rastgele belirlenmiş konformasyondaki ligand kullanarak ikişer kez gerçekleştirilmiştir. Daha sonra üç programın sonuçları belirli kriterlere göre karşılaştırılmıştır. Bu yaklaşım, hem kullanıcıları belirli durumlar için en iyi algoritmayı tanımlamaya yönlendirmeyi hem de yüksek kaliteli simülasyonları kolaylaştırmayı amaçlamaktadır.

Bu amaçla yapılan bu tez çalışmasında yüksek çözünürlüklü olarak protein-ligand etkileşimlerini içeren CovPDB veritabanı üzerinden 20 farklı reaksiyon ve 23 farklı kovalent başlık içeren toplam 40 protein-ligand kompleksi bulunarak özel bir kıstas veri seti oluşturulmuştur. Bu veri setinin oluşturulması ve analizi sürecinde Python yazılımının Biopandas (0.4.1), pandas (2.2.1), requests (2.31.0) beautifulsoup (4.12.3) ve matplotlib (2.0.2) gibi kütüphane ve paketleri kullanılmıştır. Veri setindeki her öğe PDB formatındaki protein ve SDF formatındaki ligand yapılarıyla hazır hale geldikten sonra her program için üretilen prosedürler denenmiştir. Örnek bir protein-ligand kompleksi üzerinden ayrıntılı bir prosedür çıkarılmıştır. Python yazılımıyla oluşturulan bir kod ile kıstas veri setindeki kompleksler birbirinden ayrılarak ve tüm ligandlar için ayrıca rastgele başlangıç koordinasyonları oluşturularak, her protein için hem orijinal hem de modifiye edilmiş konumlardaki liganlarla kovalent kenetlenme işlemi yapılmıştır. Böylece, toplamda her program için (40x2) 80 simülasyon olmak üzere 240 simülasyon gerçekleştirilmiştir. Simülasyonlardan çıkan sonuçlardan (eğer yeterli sayıda varsa) en iyi 10 model ele alınarak bir sonuç listesi oluşturulmuştur. Sonuç listesinde elde edilen tüm modeller Python yazılımının RMSD modülü yardımı ile orijinal kompleks yapılarla konum karşılaştırılan ortalama karekök sapma (RMSD) analizlerine tabi tutulmuştur. RMSD değerleri için 1, 2 ve 3 Å olmak üzere üç adet benzerlik eşiği belirlenerek RMSD değeri bu eşiklerin altında kalan bir model başarılı olarak kabul edilmiştir. Bu yaklaşımla her program ve her kompleks için üç farklı eşik değerlerinde başarı oranları çıkartılarak karşılaştırılmıştır. Bir kompleks için en iyi modele ulaşan programa bakılıp hangi programın kaç kere en iyi sonuç verdiği de incelenmiştir. Kenetlenme skoru olarak en iyi sonuç veren komplekslerin orijinal kompleksle benzerliğini gösteren RMSD değerleri listelendiğinde; bir kompleks en iyi kenetlenme skoruna sahip bile olsa orijinal yapıya benzerliğin düşük yani RMSD'lerin çok farklı olabileceği gözlemlenmiştir. Kıstas veri setinde çok çeşitli kovalent başlık ve reaksiyon tipi bulunduğu için farklı durumlarda farklı programların ön plana çıktığı ve bazı durumlarda hiçbir programın başarılı olmadığı gözlemlenmiştir.

Kovalent kenetlenme prosedürünün çıkarılması ve tahmin doğruluğunun karşılaştırmasının yapılması için üç farklı kovalent kenetlenme yazılımı kullanılmıştır: AutoDock4, UCSF DOCK6 ve Cresset Flare. AutoDock4 programı ile kovalent kenetlenme yapabilmek için AutoDock4'ün kendisi dışında bir de MGLTools adında AutoDock tarafından kullanılan bir görüntüleme aracı ve kovalent kenetlenme yapılmasını sağlayan kodlar bilgisayara yüklenmelidir. UCSF DOCK6 programı ile kovalent kenetlenme yapabilmek için DOCK6'nın

kendisi, programın indirilebilmesi ve çalışabilmesi için belirtilmiş tüm gereksinimler ve yine aynı üreticinin DMS adında bir paketi daha indirilmelidir. Cresset Flare programı ile kovalent kenetlenme yapabilmek için ise herhangi bir ön gereksinim veya ekstra görüntüleme, düzenleme araçlarına ihtiyaç yoktur. Programın kendisi tüm bunları sağlamaktadır. Bu program için lisans satın alınması gereklidir; ancak akademik kullanımlar için kaydolunca ücretsiz bir lisans sunmaktadır. Bu lisans ile programdaki her özellik kullanılamasa da kovalent kenetlenme için gerekli kısımlar kullanılabilir durumdadır. Tez boyunca moleküllerin düzenlenmesi ve görüntülenmesinde Schrodinger'in Maestro ve PyMOL programları kullanılmıştır. Ayrıca, UCSF Chimera/ChimeraX aynı üretici tarafından yapılmış bir görüntüleme ve düzenleme yazılımıdır ve bu yazılım da isteğe bağlı olarak kullanılabilir.

AutoDock4, esnek parametrelere sahip kullanıcı dostu bir deneyim sunar ve DOCK6'ya kıyasla süreci kolaylaştırır, bu da onu kenetlenme kavramları ve parametreleri hakkında bilgi edinmek için mükemmel bir başlangıç noktası yapar. DOCK6 kenetlenme parametrelerini özelleştirmek için en yüksek esnekliği sunar. Bu kapsamlı kontrol, özel durumlarda ince ayar yapılmasını sağlayarak DOCK6'yı güçlü bir araç haline getirir. Bununla birlikte, bu karmaşıklık aynı zamanda onu öğrenmesi en zor program yapmaktadır. Flare, kullanıcı dostu olmasıyla öne çıkmaktadır. Otomatik işlevleri, kenetlenme sürecini kolaylaştırarak sanal tarama gibi araştırmalarda büyük veri kümeleriyle uğraşan araştırmacılar için ideal hale getirir. Ancak bu otomasyon, kullanıcıların parametreler üzerinde sınırlı kontrolüne neden olur.

Hesaplama/skorlama farklılığından dolayı kovalent kenetlenme yöntemi kullanılan program ve uygulamalar birbirlerinden farklı sonuçlar verebilir. Bu sonuçların doğruluğundan emin olunması adına programlar ve parametreler değiştirilerek optimum ve daha anlaşılır bir prosedürün oluşturulması, kovalent kenetlenme alanında bir sonraki çalışmalarda daha iyi sonuçlar alınmasını sağlar. Literatürde kovalent kenetlenme için izlenecek sabit bir prosedür ve farklı programların kenetlenme yöntemindeki başarısı konusunda yeterli bilgi bulunmamaktadır. Bu tez, kovalent kenetlenme yazılım ve sunucuları hakkında toplu ve kapsamlı bir bilgi sunacağı gibi yöntemin daha kullanışlı ve en uygun hale gelmesine katkı sunacak ve bu alanda yapılacak gelecek çalışmalarda yol gösterici bir niteliğe sahip olacaktır.

Anahtar Kelimeler: Kovalent Kenetlenme, AutoDock4, UCSF DOCK6, Cresset Flare, İlaç Tasarımı

ABSTRACT

Developing a Detailed Framework for Covalent Docking, Implementation and Comparative Assessment of Different Tools on a Benchmark Set of Protein-Ligand Complexes

Tekeli, Ahmet Can

Master's Thesis, Department of Bioengineering

Supervisor: Asst. Prof. Saliha Ece ACUNER ZORLUUYSAL

April 2024

Molecular docking simulations are a cornerstone of modern drug discovery and optimization processes. Covalent inhibitors, characterized by their ability to form irreversible bonds with target proteins, offer distinct advantages in terms of higher stability and potential for increased potency. Despite these benefits, designing and simulating inhibitors covalently binding to drug targets remains more complex than traditional docking approaches. Literature information on covalent docking remains limited. Additionally, the number of algorithms capable of performing covalent docking simulations is low, with existing algorithms mostly requiring license fees. This thesis aims to address the lack of reliable and comprehensive sources about covalent docking by providing guidance on free covalent docking programs, namely, AutoDock4, UCSF DOCK6 and Cresset Flare. To address the challenges of covalent docking, detailed frameworks are meticulously developed to guide users through the entire process of performing covalent docking simulations with the optimal parameters for each program. These frameworks provide step-by-step instructions, ensuring that even researchers new to the field can execute successful simulations while simultaneously gaining a comprehensive understanding of the principles of covalent docking. Furthermore, to demonstrate and test the effectiveness of frameworks, re-docking experiments are performed on a custom benchmark set of 40 covalently bound protein-ligand complexes including 20 types of reactions and 23 types of warheads. Covalent re-docking is performed for each protein-ligand complex in the set, starting with the original as well as with a randomized conformation of the ligand. All programs are then compared based on their success of the predicted models using specific criteria. This thesis aims to guide users to define the best covalent docking algorithm for specific cases while guiding and facilitating high-quality simulations.

Keywords: Covalent Docking, AutoDock4, UCSF DOCK6, Cresset Flare, Drug Design

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my thesis supervisor, Saliha Ece Acuner Zorluuysal, for her invaluable guidance and support throughout my master study.

I am also particularly grateful to Asst. Prof. Elif Kubat Öktem and Prof. Pemra Özbek Sarıca for accepting to serve on my thesis committee and for their valuable time and comments.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF COMMAND AND PROGRAMS

INTRODUCTION

Molecular docking simulations are a cornerstone of modern drug discovery and optimization processes. Covalent inhibitors, characterized by their ability to form irreversible bonds with target proteins, offer distinct advantages in terms of higher stability and potential for increased potency. Covalent drugs form irreversible bonds with a specific amino acid residue, frequently cysteine, within the target protein's active site. This approach can overcome limitations of traditional drug design and provide a potent strategy for modulating protein function in disease. To support this type of drug discovery, computational tools like covalent docking simulations have been developed. Despite the promise of covalent drugs and the development of covalent docking tools, challenges remain. The accuracy and reliability of these computational methods, along with the availability of user-friendly frameworks and guidelines are factors limiting their broader implementation.

This thesis aims to compare the performance of three different covalent docking software programs: AutoDock4, UCSF DOCK6, and Cresset Flare. A benchmark dataset consisting of 40 protein-ligand complexes is constructed from a high-resolution covalent protein–ligand interaction database, namely CovPDB database, encompassing 20 distinct reactions and 23 unique covalent warheads. The benchmark dataset was constructed and analyzed using Python libraries and packages, including Biopandas (0.4.1), pandas (2.2.1), requests (2.31.0), beautifulsoup (4.12.3), and matplotlib (2.0.2).

AutoDock4 requires the installation of MGLTools, a visualization tool and covalent docking codes, in addition to AutoDock4 itself. UCSF DOCK6 necessitates the installation of DOCK6, all specified prerequisites for its download and operation, and another package called DMS from the same developer. Cresset Flare, on the other hand, does not require any prerequisites or additional visualization or editing tools. The program itself provides these features. A license is required for this program; however, academic users can

obtain a free license upon registration. While not all features are accessible with this license, the necessary components for covalent docking are available. Throughout the thesis, Schrodinger's Maestro and PyMOL programs were used for molecular preparation and visualization. UCSF Chimera/ChimeraX, a visualization and editing software developed by the same developer as DOCK6, was also used optionally.

A detailed procedure was developed for each previously mentioned software using the same example protein-ligand complex (PDB ID: 2GKE). This case study is the crystal structure of a diaminopimelate epimerase, an antibacterial drug target, in complex with an irreversible inhibitor LL-AziDAP. Python was utilized to separate the complexes in the benchmark dataset and generate random initial coordinates for all ligands with using Open Babel (3.1.1), RDKit (Q3 2023) and, in some cases, with Marvin Sketch. Covalent docking was performed for each protein starting with both original and randomly modified ligand conformations. This resulted in a total of 240 docking simulations (80 per software). The top 10 (or less, if 10 models are not available) models resulting from each of the simulations were combined in a results list. All predicted models were subjected to root mean square deviation (RMSD) analysis using Python's RMSD module to compare their positions with the original experimental complex structures. Three similarity thresholds were defined for RMSD values: 1, 2, and 3 Å, smaller numbers indicating increased similarity of the re-docked complex model to the experimental complex structure. Models with RMSD values below these thresholds were considered successful. This approach was used to compare the success rates of each software and each complex at three different threshold values. The software with the highest accuracy in predicting each of the complexes in the benchmark set was also analyzed to determine which software is more reliable for covalent docking (in general and for certain types of reactions, warheads, etc.). While a complex may have the best docking score, its similarity to the reference experimental structure may be low, resulting in high RMSD values. Therefore, the models were also assessed in terms of the relation between

docking scores and RMSD values. Due to the diversity of covalent warheads and reaction types in the benchmark dataset, different software excelled in different situations, and in some cases, no program was successful.

In summary, this thesis aims to develop a detailed framework for covalent docking, implement this framework with different available industry-standard software, and perform a comparative assessment on a comprehensive benchmark set of protein-ligand complexes. The thesis is structured as follows:

- **Literature Review:** Explores the fundamental principles of protein-ligand interactions, the specifics of covalent docking, and the limitations of existing molecular docking and covalent docking approaches.
- **Materials and Methods:** Describes the construction of a benchmark dataset consisting of protein-ligand complexes with known covalent binding modes. It also outlines the specific tools used in this study and how covalent docking software are compared with each other and assessed for their success rates based on the benchmark dataset.
- **Results and Discussion:** Provides detailed, step-by-step frameworks for utilizing three covalent docking programs AutoDock4, UCSF DOCK6, and Cresset Flare. This includes instructions and guidance on ligand and receptor preparation, docking procedures, and the possibility of automation of the entire process. Furthermore, a critical assessment of these tools is presented, identifying their strengths, weaknesses, and potential troubleshooting strategies.
- **Conclusions and Future Work:** Summarizes the findings of the comparative assessment, providing recommendations on optimal tool selection for various use cases. It outlines future research directions for improving covalent docking algorithms and the broader field of covalent drug design.

1. LITERATURE REVIEW

1.1. Protein Ligand Interactions

Proteins are crucial macromolecules with diverse functions throughout biological systems. They exert their effects by interacting with various molecules and structures, including nucleic acids, cellular organelles, and smaller compounds like oxygen and ligands. Understanding protein-ligand interactions and their binding mechanisms is fundamental to many processes such as rational drug design.

There are many experimental techniques developed to serve this purpose. x-ray crystallography provides valuable insights into protein-ligand interactions by analyzing the thermodynamics of molecules within a static crystal structure. NMR spectroscopy is another powerful technique, particularly for studying the dynamic behavior of protein-ligand complexes. Calorimetric methods, such as differential scanning calorimetry (DSC) and isothermal titration calorimetry (ITC), offer an additional approach to investigate the thermodynamics of these complexes (Bronowska 2011). In addition to those classical methods, recent technical improvements have made single-particle cryo-electron microscopy (cryo-EM) a powerful tool for visualizing the detailed structures of macromolecules (Benjin and Ling 2020).

Experimental techniques for studying protein-ligand interactions can be complex, expensive, and time-consuming. Since rational drug design often involves screening hundreds of small molecules, these experimental methods become impractical. This highlights the necessity of computational techniques, which can simulate various conditions, calculate enthalpy, entropy, and free energy changes much faster, making them ideal for large-scale studies (X. Du et al. 2016). Protein-ligand docking followed by molecular dynamics (MD) simulations offer reliable insights into the entire binding process and how the stability of the complex changes over time.

1.2. Covalent Docking

Molecular docking is a computational method used to predict the binding mode and affinity of a ligand to a receptor. It is a valuable tool in drug design, as it can be used to identify potential new drugs and to optimize the design of existing drugs (Grinter and Zou 2014). Protein-ligand docking relies on individual structures as well as a structural model/approximate location of the binding site, derived from techniques like X-ray crystallography, NMR, or predicted by modeling. To streamline the process, docking concentrates on a pre-identified binding site, determined through different methods or prior knowledge of the target protein's binding sites (Grinter and Zou 2014).

Docking involves two key steps: first exploring different potential positions of the ligand within the protein's active site, and then using a scoring function to evaluate how favorable each position is. Sampling algorithms are used to generate conformations of ligand within specified boundary. Widely used sampling algorithms are Matching Algorithms, Incremental Construction, Monte Carlo, and Genetic Algorithms. The boundary is generally defined as a box and atom maps of each atom type are produced by the docking program. Scoring functions are used to estimate affinity between protein and ligand. Scoring functions are divided to three approaches, force-field-based, empirical and knowledge-based (Meng et al. 2011). In the most common approach, potential fits are evaluated using a precalculated grid of energies, and a steepest-descent minimization algorithm refines the highest-scoring conformations (Freddolino, Stone, and Schulten, n.d.).

Covalent docking is a valuable method in drug design, since covalent inhibitors form a distinct chemical bond between an electrophilic functional group on the ligand, called warhead, and a nucleophilic residue on the target protein (Scarpino, Ferenczy, and Keserü 2018). It can be used to identify potential new drugs that are more potent and selective than existing drugs, as it requires the consideration of the formation of a covalent bond. It can also be used to optimize

the design of existing drugs, by identifying ways to improve their binding affinity and selectivity.

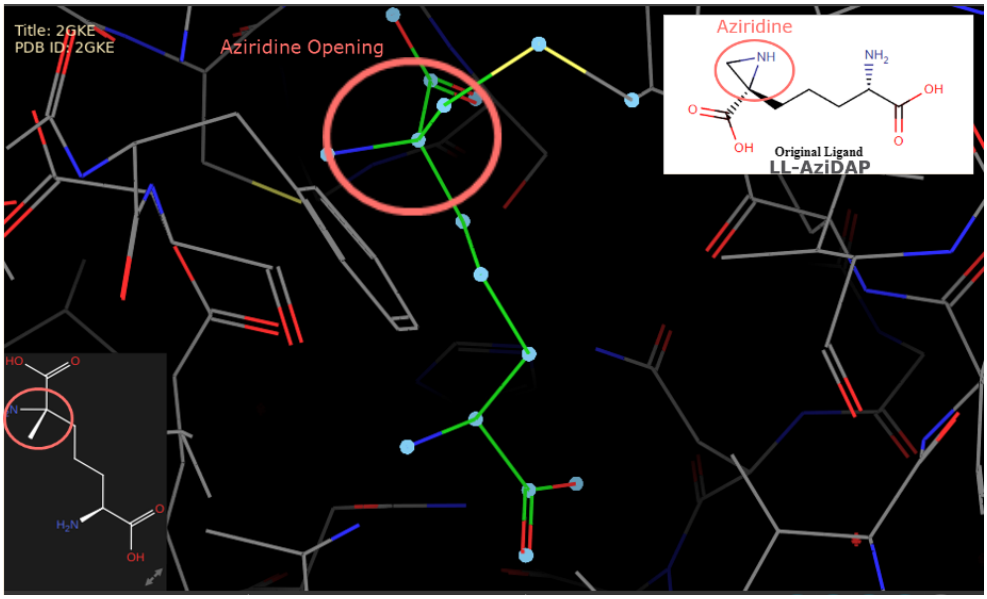

Figure 1.1. Example of Warhead and Warhead Reaction. 2GKE structure is opened in Maestro. Red circles points aziridine warhead of LL-AziDAP, shows on original ligand as well as in 2GKE complex.

1.3. Molecular Docking and Covalent Docking Limitations

Scoring functions used to predict binding affinities are a crucial element within docking methods. However, higher complexity of structures, increasing possibility of non-covalent interactions such as hydrogen bonds, electrostatic interactions, van der Waals interactions, or hydrophobic interactions, solvent effects, entropic contributions make the affinity estimation less accurate. Some scoring functions struggle to differentiate between high- and low-affinity ligands. Improving the scoring function is the main concern of docking algorithms.

In general, there is a trade-off between the speed of the docking algorithm and its accuracy. More advanced functions and exhaustive sampling increase accuracy but also computational cost. Protein flexibility, i.e. proteins' ability to adopt different conformations, is another concern affecting ligand binding and thus, accuracy as well as the computational cost (Grinter and Zou 2014). When

protein flexibility is also considered, balancing efficiency and sampling thoroughness becomes a challenge.

Covalent docking algorithms, on the other hand, face limitations beyond those of standard docking. Due to the specificity of covalent bonds, finding suitable ligands for the target becomes more difficult. Collecting preliminary information about ligands, warhead reactions and target residues adds complexity to the pre-processing stage. Additionally, variations in file formats and preparation requirements across different docking software can make management of the overall process challenging.

1.4. Objective of This Thesis

Researchers without a computational background can often manage molecular docking by following software/server manuals. However, the complexity and relative scarcity of covalent docking software make it a more challenging technique to learn and execute successfully. To address this lack of information, a detailed covalent docking framework and troubleshooting guidance for three docking software that are freely available for academic purposes and suitable for covalent docking, namely, AutoDock4 (Huey et al. 2007, Morris et al. 2009), UCSF DOCK6 (Kuntz et al. 1982, DesJarlais et al. 1988) and Cresset Flare (Cheeseright T. et al. 2006, Bauer and Mackey 2019, Kuhn et al. 2020), are developed. By re-docking experiments on a custom constructed benchmark dataset of forty different protein-ligand complexes, the performance of the framework, accuracy and efficiency of the aforementioned docking algorithms are comparatively tested. The objective and methods of this thesis are summarized as a graphical abstract below (Figure 1.2).

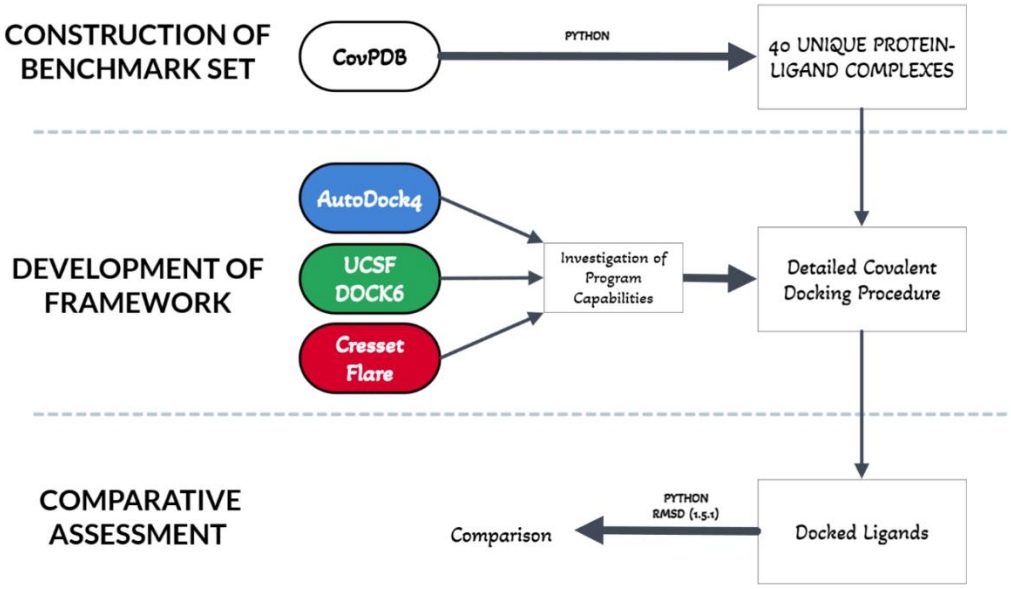

Figure 1.2. Graphical Abstract of the Thesis

2. MATERIALS AND METHODS

2.1. Benchmark Dataset Construction

Covalent docking requires manual implementation of input file creation and binding site identification. For docking experiments with custom ligands or a set of ligands that is not experimentally proven yet, correct covalent bond formation must be decided. A covalent bond can be determined by referencing a ligand like the one being processed or by visually recognizing and matching the warhead. Deciding the binding site presents another challenge that can be addressed through referencing or by trial and error. In contrast, in re-docking experiments, the ligand can be extracted directly from the receptor, eliminating the need to determine the binding residue and the appropriate bond. There are several databases to find covalent ligand-protein complexes, listed in Table 2.1.

In this thesis, 40 distinct receptor-ligand pairs have been constructed from the CovPDB (https://drug-discovery.vm.uni-freiburg.de/covpdb/download) database, as a benchmark dataset, and processed through re-docking to develop an effective and comparative procedure for AutoDock4 (Huey et al. 2007, Morris et al. 2009), UCSF DOCK6 (Kuntz et al. 1982, DesJarlais et al. 1988) and Cresset Flare (Cheeseright T. et al. 2006, Bauer and Mackey 2019, Kuhn et al. 2020). The complex structures in the CovPDB database were downloaded and filtered through a custom Python script, which selects two representative models, with the lowest structural resolution, for each reaction type. All information about the filtered protein-ligand complexes in the CovPDB Database was extracted through web scraping using the Python Requests module. Also, the distributions of the complexes in the benchmark set according to protein class, covalently binding target residue name, covalent reaction name and warhead name are analyzed.

Table 2.1. Databases for Covalently Binding Ligands

Database	Properties	Reference
ZINC20	• It is a large database. • Contains commercially available compounds in 3D format. • Ligand searching by SMILES and SMART, even by drawing. • Useful for identifying ligands with properties similar to those of the one being processed. • **Incapable of specifically identifying covalently binding ligands.**	https://zinc.docking.org/ (Irwin et al. 2020)
Binding Database	• A constructed database specifically designed for drug-like molecules. • Ligand searching by many parameters. Searching by protein target is also possible. • Contains specific datasets like Coronavirus Data. • All data can be downloaded from the database, including detailed information viewable with Excel-like programs. • **Incapable of specifically identifying covalently binding ligands.**	https://www.bindingdb.org/rwd/bind/index.jsp (Liu et al., n.d.)
CovalentInDB	• Limited, only contains ligands with covalent interaction. • Provides detailed information about ligands and their covalent property including reactive warheads of inhibitors and covalent reaction mechanisms. • All data can be downloaded from the database, including detailed information viewable with Excel-like programs.	http://cadd.zju.edu.cn/cidb/ (H. Du et al. 2021)
CovPDB		https://drug-discovery.vm.uni-freiburg.de/covpdb/ (Gao et al. 2021)

2.2. Comparative Assessment of Different Covalent Docking Softwares

A benchmark set of 40 protein-ligand complexes is used for re-docking experiments. Each complex is tested twice using the given procedure: once with the ligand in its default conformation and once with a randomized conformation. A total of 80 docking simulations per program are performed. The top 10 structures from each simulation are selected for further comparison. Each docking result of a ligand has at most 10 structures. A total of 60 docking results are generated at most for each protein-ligand complex. All obtained models are subjected to root mean square deviation (RMSD) analysis. For this analysis, a Python module that calculates RMSD of two molecules (https://github.com/charnley/rmsd) is used. All predicted models were compared to the original experimental complex structures woth RMSD tool. Three similarity thresholds (1, 2, and 3 Å) are used for RMSD values, with lower values indicating closer resemblance to the experimental complex. Models with RMSD values below these thresholds are considered successful. This approach is used to compare the success rates of each software and each complex at three different threshold values. The software with the highest accuracy in predicting each of the complexes in the benchmark set is also analyzed to determine which software is more reliable for covalent docking (in general and for certain types of reactions, warheads, etc.). While a complex may have the best docking score, its similarity to the reference experimental structure may be low, resulting in high RMSD values. Therefore, the models are also assessed in terms of the relation between docking scores and RMSD values.

2.3. List of Tools Employed

The tools and programs listed in Table 2.3 are used for the dataset construction, automation of the procedure and specific error fixing steps of this thesis. Necessary tools for each specific docking software are mentioned in the corresponding detailed frameworks subsection of the "Results and Discussion" section.

Table 2.3. List of Tools Employed in the Thesis

Docking Programs	
AutoDock4	(Morris et al. 2009; 1998)
UCSF Dock6	(Kuntz et al. 1982; DesJarlais et al. 1988)
Cresset Flare	(Cheeseright T. et al. 2006)
Molecule Visualization	
Maestro	("Schrödinger Release 2024-1: Maestro, Schrödinger, LLC, New York, NY," 2024)
PyMOL	(Schrödinger LLC 2015c; 2015a; 2015b)
Molecule Editing	
PyMOL	(Schrödinger LLC 2015c; 2015a; 2015b)
Python 3.12.2	("Python Software Foundation. Python Language Reference, Version 3.12.2. Http://Www.Python.Org," n.d.)
RDKit 2023_09_6 (Q3 2023) Release	("RDKit: Open-Source Cheminformatics. 2023_09_6 (Q3 2023) Release. Https://Www.Rdkit.Org," n.d.)
Open Babel 3.1.1	(O'Boyle et al. 2011)
Marvin Sketch	(ChemAxon 2023)
Cresset Flare interface	(Cheeseright T. et al. 2006)
UCSF Chimera	(Pettersen et al. 2004)
Database	
RCSB PDB	(Berman et al. 2000)
CovPDB	(Gao et al. 2022)
Data Curation and Interpretation	
Biopandas (0.4.1)	(Raschka 2017)
Python Rmsd (1.5.1)	("Calculate Root-Mean-Square Deviation (RMSD) of Two Molecules Using Rotation, GitHub, Http://Github.Com/Charnley/Rmsd," n.d.)
Python pandas (2.2.1)	(Mckinney 2010)
Python requests (2.31.0)	("Python Software Foundation. Python Language Reference, Version 3.12.2. Http://Www.Python.Org," n.d.)
Python beautifulsoup (4 4.12.3)	("Python Software Foundation. Python Language Reference, Version 3.12.2. Http://Www.Python.Org," n.d.)
Python matplotlib (2.0.2)	(Hunter 2007)

3. RESULTS AND DISCUSSION

3.1. Benchmark Dataset Construction and Distribution Analysis

The dataset contains 2 representatives for 20 covalent reaction types, including 23 warheads. The detailed information about the benchmark set is listed in Table 3.1, with each complex's PDB ID, Uniprot ID, reaction type and warhead name for covalent binding, structure resolution and covalent ligand HET codes (heterologous entity in the PDB entry).

The distributions of the complexes in the benchmark set according to protein class, covalently binding target residue name, covalent reaction name and warhead name are analyzed and shown in Figure 2.1. Despite being one of the least abundant amino acids, cysteine is the most common covalent amino acid residue because of its unique properties. Cysteine's thiol group allows it to perform nucleophilic and redox-active functions that other amino acids cannot. This makes cysteine more likely to form covalent bonds with drugs (Huang et al. 2022).

Table 3.1. List of Protein-Ligand Complexes in Benchmark Set

	UNIPROT ID/ACC:	HET (LIGAND) CODE	REACTION NAME	COVALENTLY BINDING RESIDUE	WARHEAD NAME	STRUCTURE RESOLUTION (Å)
2GKE	P44859 (DAPF_HAEIN)	ZDP	Aziridine Opening	CYS 73	Aziridine	1.35
6SXV	Q9XBQ3 (IABF_GEOSE)	LXE	Aziridine Opening	GLU 294	Aziridine	1.4
1PWG	P15555 (DAC_STRSR)	HE0	Beta-Lactam Addition	SER 62	Beta-Lactam	1.07
5A92	E1ANH6 (BLC97_ECOLX)	CEF	Beta-Lactam Addition	SER 70	Beta-Lactam	1.05
4UA9	H6UQI0 (H6UQI0_ECOLX)	CB4	Borylation	SER 70	Boronic Acid	0.84
1M40	P62593 (BLAT_ECOLX)	CB4	Borylation	SER 70	Boronic Acid	0.85
6HMT	P31947 (1433S_HUMAN)	GEH	Disulfide Formation	CYS 42	Disulfide	1.1
4LUC	P01116 (RASK_HUMAN)	20G	Disulfide Formation	CYS 12	Disulfide	1.29
6H5G	P24670 (AROD_SALTI)	FQZ	Epoxide Opening	LYS 170	Epoxide	1.04
2XGB	P16098 (AMYB_HORVU)	EPG	Epoxide Opening	GLU 184	Epoxide	1.2
3HGP	P00772 (CELA1_PIG)	FRW	Hemi(thio)acetalization	SER 195	Ketone	0.94
6BID	Q83883 (POLG_NVN68)	DW4	Hemi(thio)acetalization	CYS 139	Aldehyde	1.15
1JCL	P0A6L0 (DEOC_ECOLI)	HPD	Hemiaminalization	LYS 167	Aldehyde	1.05
4UIO	P24670 (AROD_SALTI)	VAU	Hemiaminalization	LYS 170	Ketone	1.35
6FM7	A0A175VLQ4 (A0A175VLQ4_AEREN)	NXL	Imidazolidinone Opening	SER 62	Imidazolidinone	1.04
6MZ1	Q9L5C7 (Q9L5C7_ECOLX)	NXL	Imidazolidinone Opening	SER 70	Imidazolidinone	1
3CR6	P29373 (RABP2_HUMAN)	LSR	Imine Condensation	LYS 132	Aldehyde	1.22
4A29	Q06121 (TRPC_SACS2)	3NK	Imine Condensation	LYS 210	Ketone	1.1
2CXV	P13901 (POLG_HAVMB)	BBL	Lactone Addition	HIS 102	Beta-Lactone	1.4
2H9H	P08617 (POLG_HAVHM)	BBL	Lactone Addition	HIS 102	Beta-Lactone	1.39
3QZR	E7E815 (E7E815_HE71)	AG7	Michael Addition	CYS 147	Vinyl Carbonyl	1.04
5P9J	Q06187 (BTK_HUMAN)	8E8	Michael Addition	CYS 481	Vinyl Carbonyl	1.08
2AH4	P00760 (TRY1_BOVIN)	GBS	Nucleophilic Acyl Substitution	SER 195	Ester	1.13
6UMZ	P16113 (PYP_HALHA)	QBV	Nucleophilic Acyl Substitution	CYS 69	Carboxylic Acid	0.9
6O8M	D5AT91 (D5AT91_RHOCB)	LSY	Nucleophilic Addition to a Double Bond	CYS 107	Hydrazide	1.46
2O7R	Q0ZPV7 (CXE1_ACTER)	4PA	Nucleophilic Addition to a Double Bond	SER 169	Ester	1.4
3EWW	P11172 (UMPS_HUMAN)	U1P	Nucleophilic Addition to a Triple Bond	LYS 314	Nitrile	1.1
5MAE	P07711 (CATL1_HUMAN)	7KN	Nucleophilic Addition to a Triple Bond	CYS 25	Nitrile	1
2CE2	P01112 (RASH_HUMAN)	XY2	Nucleophilic Aliphatic Substitution	CYS 32	Halomethyl Carbonyl	1
6SFE	P24670 (AROD_SALTI)	L9Z	Nucleophilic Aliphatic Substitution	LYS 170	Aminium Ion	1.08
4ONM	P61088 (UBE2N_HUMAN)	N2F	Nucleophilic Aromatic Substitution	CYS 87	Aryl Sulfone	1.35
5M4Z	Q9NDD3 (Q9NDD3_TRISP)	7K8	Nucleophilic Aromatic Substitution	CYS 189	Aryl Halide	1.18
5U4H	A0A0R0VXX6 (A0A0R0VXX6_ACIBA)	0V5	Phosphorylation	CYS 116	Phosphate	1.05
1YS1	P22088 (LIP_BURCE)	2HR	Phosphorylation	SER 87	Phosphonate	1.1
1YQS	P15555 (DAC_STRSR)	BSA	Ring Opening	SER 62	Beta-Sultam	1.05
1SCW	P15555 (DAC_STRSR)	CP5	Ring Opening	SER 62	1-Hydroxy-2,6-Dioxa-3-Oxophosphorinone	1.13
5AQE	P29600 (SUBS_BACLE)	VOD	Sulfonylation	SER 221	Sulfonyl Halide	1.1
2PFE	Q47SP5 (Q47SP5_THEFY)	AES	Sulfonylation	SER 195	Sulfonyl Halide	1.44
2QXI	P49862 (KLK7_HUMAN)	K7J	Composite Reaction	HIS 57	Halomethyl Carbonyl	1
5EL1	P0A6L0 (DEOC_ECOLI)	1BO	Composite Reaction	LYS 167	Vinyl Carbonyl	1.25

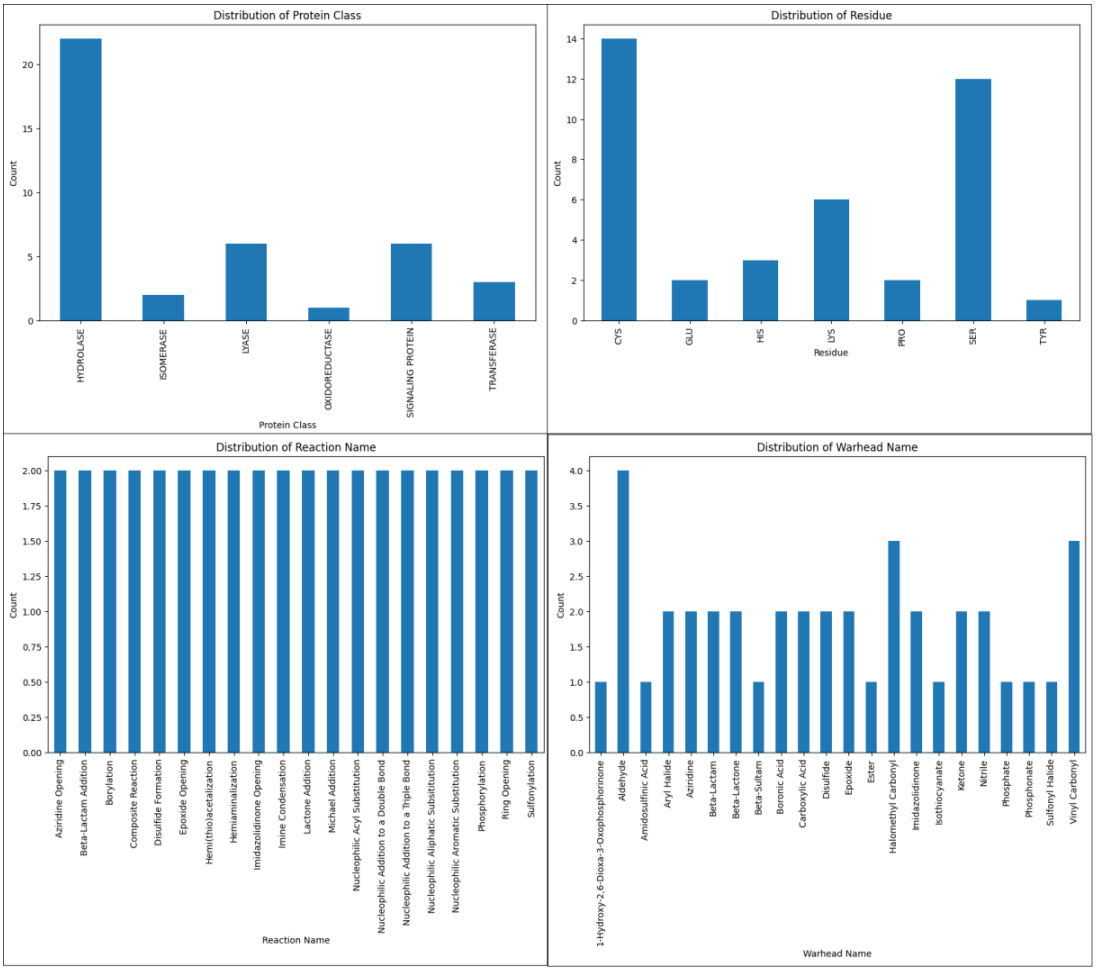

Figure 2.1. Distribution of Protein Class, Residue, Reaction name and Warhead Name of Benchmark Dataset

3.2. Detailed Framework: AutoDock4

Core components of the AutoDock4 covalent docking framework are summarized in Figure 3.1.

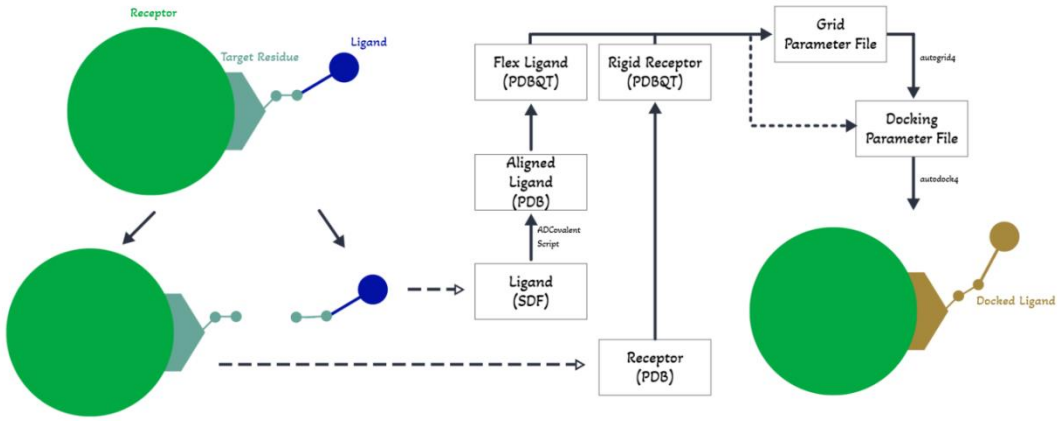

Figure 3.1. Simplified AutoDock4 Covalent Docking Framework

3.2.1. Requirements

Pre-requirements for AutoDock4 covalent docking are:

- AutoDock4 installed (https://autodock.scripps.edu/download-autodock4/).
- The graphical interface MGLTools installed (https://ccsb.scripps.edu/mgltools/downloads/).
- AutoDock4 covalent docking script installed (https://autodock.scripps.edu/resources/covalent-docking/).
    - To use scripts, Python >=2.6 and OpenBabel Python bindings are necessary. However, downloading MGLTools provides both which can be activated with the command *"pythonsh"*.

Non-essential requirements are:

- Schrodinger Maestro for Academics (https://newsite.schrodinger.com/release-download/).
- Schrodinger PyMOL (https://pymol.org/2/)
- Python 3 and RDKit (https://www.rdkit.org/docs/Install.html).

AutoDock4 does not have a built-in covalent docking module yet. When the two methods developed for AutoDock4 are compared, the flexible side chain method was shown to be more efficient and accurate (Scarpino, Ferenczy, and Keserü 2018). The script provided by the developer team is employed to generate the initial alignment between the ligand and the receptor's target residue. During subsequent docking calculations, the ligand side chains will be treated as flexible. MGLTools will be used for the preparation of ligands and receptors, grid generation, and parameter determination, as with any docking simulations performed.

Among the visualization and molecular modeling environments, Maestro has been found to be the best, particularly for the covalent ligand preparation for AutoDock4. Ligand is detected and displayed on an interface automatically in most cases. This ability helps to find ligand and bound receptors faster. Additionally, the built-in optimization tool may assist with covalent docking, which will be addressed in the "Docking Procedure" section. Any other tool, or the use of several tools simultaneously, assessed as suitable for the job is appropriate to use. PyMOL is another visualization tool that is open source and easy to use. PyMOL is good for editing the molecule and visualizing outputs of docking. Python and RDKit are not necessary but used to create a random conformation ligand, creating custom scripts to automate process and converting docking result to *PDB* format to avoid using MGLTools graphical interface.

3.2.2. Terminal Commands over Graphical Interface

MGLTools offers a graphical interface to simplify the docking process, making it more user-friendly and requiring less coding and computer knowledge. After manually processing the ligand with the covalent docking script, it is optional to use MGLTools for continuing procedure. However poor design of graphical interface compels the use of a combination of different visualization tools and terminal commands.

3.2.3. Basic Information About Terminal Commands

The procedure is developed around terminal commands. Some parts of commands should be changed according to file management used. Firstly, "*Right Click* > *Open in Terminal*" in desired folder opens the terminal.

Command 3.1. Built-in Python Interpreter of MGLTools

```
pythonsh
```

The "*autogrid*" and the "*autodock*" modules have their own command. However, to run MGLTools scripts and covalent docking scripts, *pythonsh* will be used.

Command 3.2. Built-in Python Interpreter of MGLTools

```
pythonsh adcovalent/prepareCovalent.py --ligand ligand.mol2
```

This command runs *prepareCovalent.py* in the folder of *adcovalent*. By entering script name with correct location without any addition, details about the script will be written. Using a double dash followed by the appropriate keyword provides detailed information about the process to be executed. In this example, the ligand file is specified, which is in the same directory where the terminal is opened.

Command 3.3. Example Full Command

```
pythonsh adcovalent/prepareCovalent.py --ligand sit_H_optimized.mol2 --ligindices 43,42 --
receptor ../1Receptor/6oim_H.pdb --residue A:CYS12 --outputfile ligcovalent.pdb
```

This is a full command of *prepareCovalent.py*, with a ligand file *sit_H_optimized.mol2*, and receptor file *6oim_H.pdb*. Details will be provided in the appropriate section. The sign "../" represents the previous directory.

3.2.4. File Management

Among the 40 protein-ligand complexes in the benchmark dataset, 2GKE (https://www.rcsb.org/structure/2gke) is used as a case study for explaining

the detailed procedure. 2GKE is the PDB ID of the protein-ligand complex, but it will serve as the name of the receptor during the procedure. The name of the ligand is ZDP, which can be observed in the PDB structure or on the website. AutoDock4 generates numerous files during processing, making file management recommended, though not mandatory.

Suggested file management for AutoDock4 only:

- Main folder: *2GKE*
- Sub-folders:
    - *adCovalent* (The covalent docking script folder)
    - *1Receptor* (Receptor related files will be generated here.)
    - *2Ligand* (Ligand related files will be generated here.)
    - *3Docking* (Docking related files will be generated here.)
        - Runs (Docking run will be executed here.)

Alternatively, whole docking procedure files and command execution can be done in same folder. However, the workspace will not be easier to manage.

3.2.5. Ligand Preparation

All files that are manually processed are saved to the *2Ligand* folder. The original protein-ligand complex downloaded as PDB format, or manually created complex, is placed into main folder.

3.2.5.1. For Re-Docking Experiments

The protein-ligand complex 2GKE is downloaded and opened in Maestro or in the desired modelling tool. For details in this procedure, Maestro is referenced.

ZDP is selected. The two atoms of target residue are added to selection by holding "ctrl" and clicking on the atoms. Right click on the selection and "copy to new object" will create new object with selected atoms, renamed as ZDP. An image of the selection is given in Figure 3.2.

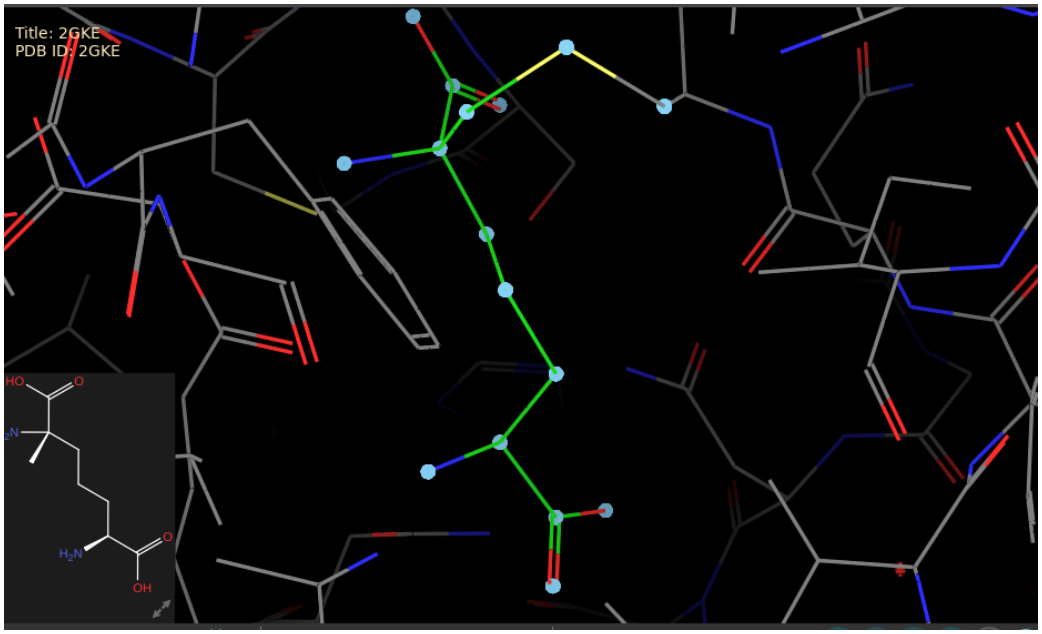

Figure 3.2. ZDP and two connection atoms are selected on Maestro. Green represents carbon on ligand and gray represents carbon on residues. Since cysteine is the target residue, the atoms ZDP, SG, and CB are chosen for consideration.

All atoms of new molecule, as the ligand for docking, are selected and by clicking *"Edit* > *Add Hydrogens"*, all hydrogens are added. *Structure should be minimized to avoid errors.* Export the molecule as *SDF* file.

Adding hydrogen can occasionally disrupt atom charges and coordinates, resulting in error during docking process. The error can be easily avoided by minimizing all atoms by clicking *"Edit* > *Minimize* > *All Atoms"*. Due to this feature being easily accessible without any coding experience, Maestro has been found to be the best for re-docking experiments in AutoDock4.

Consider that the atoms of the extracted ligand will have the exact same coordinates as the ligand in the original PDB file, or slightly same if minimization is done. This might introduce a bias, as the given ligand is already in the perfect position. To avoid bias, a random conformation of the ligand can be generated using RDKit. The following python code (Program 3.1) can be used for creating random conformation of ligand with *SDF* format after typing correct file name

and directory. Save the script as a Python file with the *".py"* extension. Executing the script will create a new file named *"r_ZDP.sdf"* which is ready to use. The script works only *SDF* format, RDKit does not support *MOL2* format. The script does not operate in the presence of atoms B, K, and Na. Alternative approaches should be considered in this case. Example of normal and randomized files in text editor is given in Figure 3.3.

Program 3.1. Python Script to Randomize Coordinates of Ligand with *SDF* Format

```
from rdkit import Chem
from rdkit.Chem import AllChem
import os
#-----------------------------------
# Changeable Parts:
file = '/home/ruzgar/Desktop/2GKE/2Ligand' # File directory
ligand_name = 'ZDP' # File name
#-----------------------------------

# Load SDF file
in_file = ligand_name + '.sdf'
in_path = os.path.join(file , in_file)
supplier = Chem.SDMolSupplier(in_path)

# Prepare an SDF writer
out_file = 'r_' + ligand_name + '.sdf'
out_path = os.path.join(file , out_file)
writer = Chem.SDWriter(out_path)

for mol in supplier:
    if mol is not None:
        # Convert to SMILES and back to a molecule, then add hydrogens
        smiles = Chem.MolToSmiles(mol)
        new_mol = Chem.AddHs(Chem.MolFromSmiles(smiles))

        # Generate a random conformation
        AllChem.EmbedMolecule(new_mol, useRandomCoords=True)

        # Perform a quick minimization
        AllChem.UFFOptimizeMolecule(new_mol)

        # Write to output file
        writer.write(new_mol)
writer.close()
```

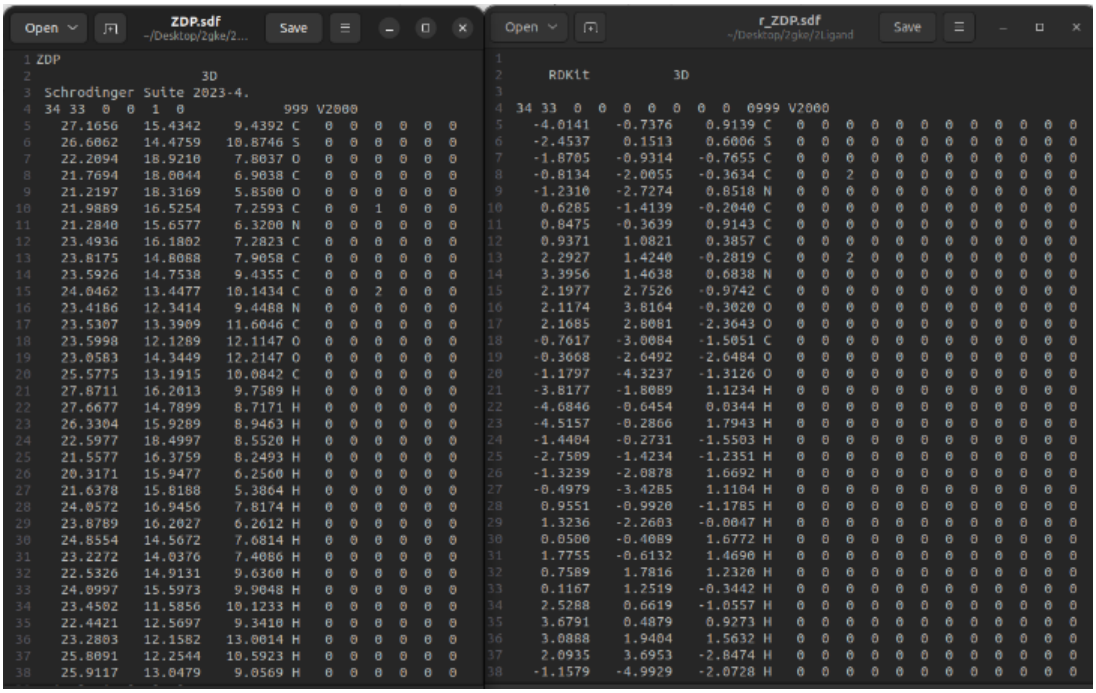

Figure 3.3. ZDP and r_ZDP opened with text editor. Program 1 is executed, and it creates new molecule called *r_ZDP.sdf* with randomized coordinates

3.2.5.2. Creating Manual Covalent Bonds for Docking Experiments with Different Ligands

The protein to be docked is downloaded and directly proceeds to the receptor preparation section. If the protein is also a protein-ligand complex like 2GKE, ligands should be deleted in the desired modelling tool. The main difference between docking and re-docking for covalent docking lies in the preparation of the covalently bonded receptor and ligand. For covalent docking, the receptor and desired ligand with a covalent bond must be manually created. In re-docking, however, the receptor is already part of a protein-ligand complex with a covalent bond. To create covalent bond manually, PyMOL has been found to be the best modelling tool.

First, the model is opened, ligand and waters are deleted, if exist. The molecule to be bonded is also opened in the same window and hydrogens removed. PyMOL is turned to "3-Button Editing" mode by clicking "mouse mode" on the right corner of default window layout. The molecule is moved close to target

residue by clicking one of the atoms and using “shift + mouse wheel” and “shift + right click” combinations. Displaying target residue as sticks will be helpful.

Making mouse mode to “viewing” and changing “Selecting” to “Object” will enable to select whole receptor and the ligand molecule. The selection shows as “(sele)” is copied as new object. After this, both molecules are displayed as one molecule. In editing mode, atoms that covalent bond will be created are selected. On PyMOL terminal, writing “bond” creates a bond between molecules. Example in Figure 3.4 is the latest view before creating bond. After minimization, the protein-ligand complex can be processed as mentioned in section 3.1.5.1.

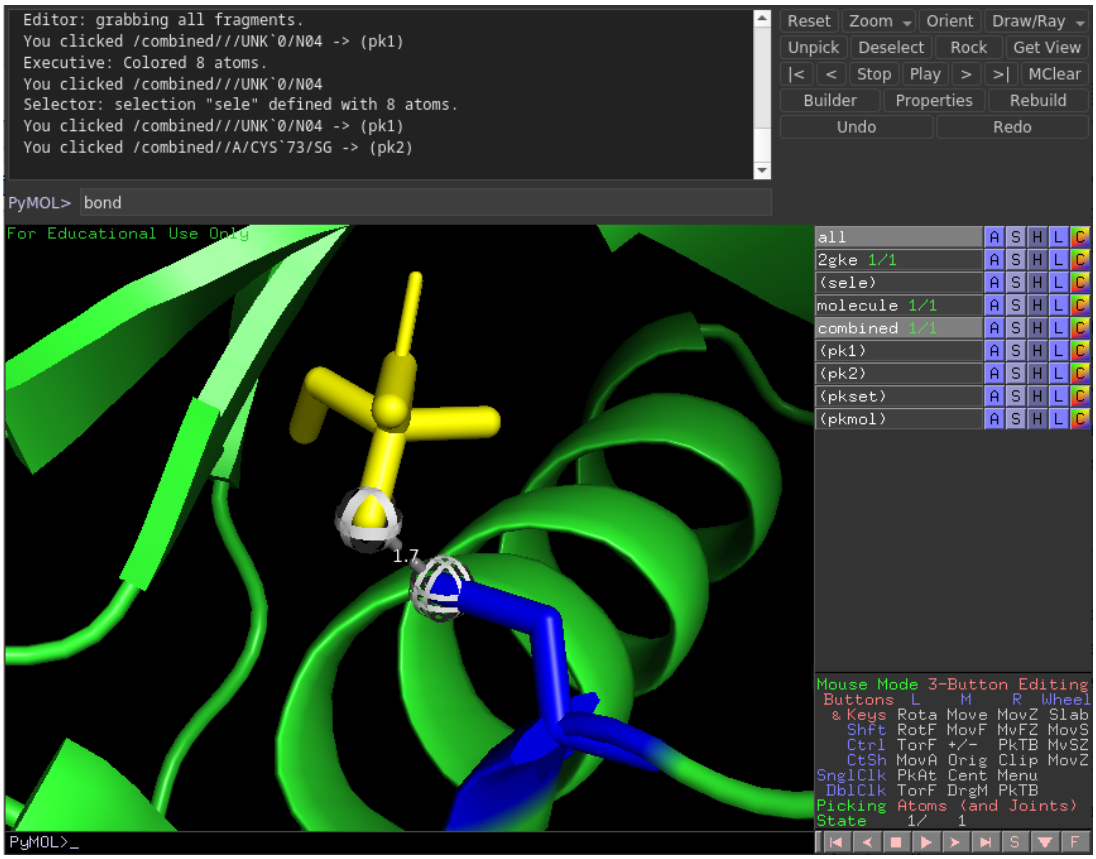

Figure 3.4. Example Bond Formation in PyMOL. The manual bonding procedure is followed. The bonding atom on target residue colored as blue and the bonding atom on a molecule to be docked, colored as yellow, are selected in editing mode. Executing “bond” function will create the bond.

3.2.6. Preparing Ligand for Covalent Docking

After the previous part of the procedure is done, two files will be created, *ZDP.sdf* and *r_ZDP.sdf*. Opening the terminal in this directory and executing the following commands will generate ligand files in two forms, rigid and flexible, suitable for AutoDock4 calculations. These forms will have their connection atoms aligned with the target residue and be saved in the PDBQT format, which is a file type exclusive to AutoDock4 and MGLTools. Repeat the commands separately with r_ZDP by changing ZDP to r_ZDP. All commands of this section should be executed in *2Ligand* file if the suggested file management is used.

Command 3.4. Aligning Connection Atoms to Target Residue

```
pythonsh ../adCovalent/prepareCovalent.py --ligand ZDP.sdf --ligindices 1,2 --receptor
../2gke.pdb --residue A:CYS73 --outputfile ZDPcov.pdb
```

The covalent docking script prepareCovalent.py will align two connection atoms according to the given *ligindices* flag. Numbers are the atom identifier IDs. Correct numbers are, in most cases, 1 and 2. This may differ for each file, so IDs should be verified individually. To check IDs correctly, open ligand file with PyMOL, "*Label* > *Atom Identifiers*" or with Maestro, "*Label* > *Atom Numbers*" (Figure 3.5). This process will label every atom, with the numbers on the connecting atoms referred to as ligindices. The correct sequence for the numbers is "*the second connection atom, which is the farther atom from the ligand, followed by a comma, then the first atom connected to that atom*" as demonstrated in this example, the correct format is "*--ligindices 1,2*". IDs can also be verified by opening ligands as text files.

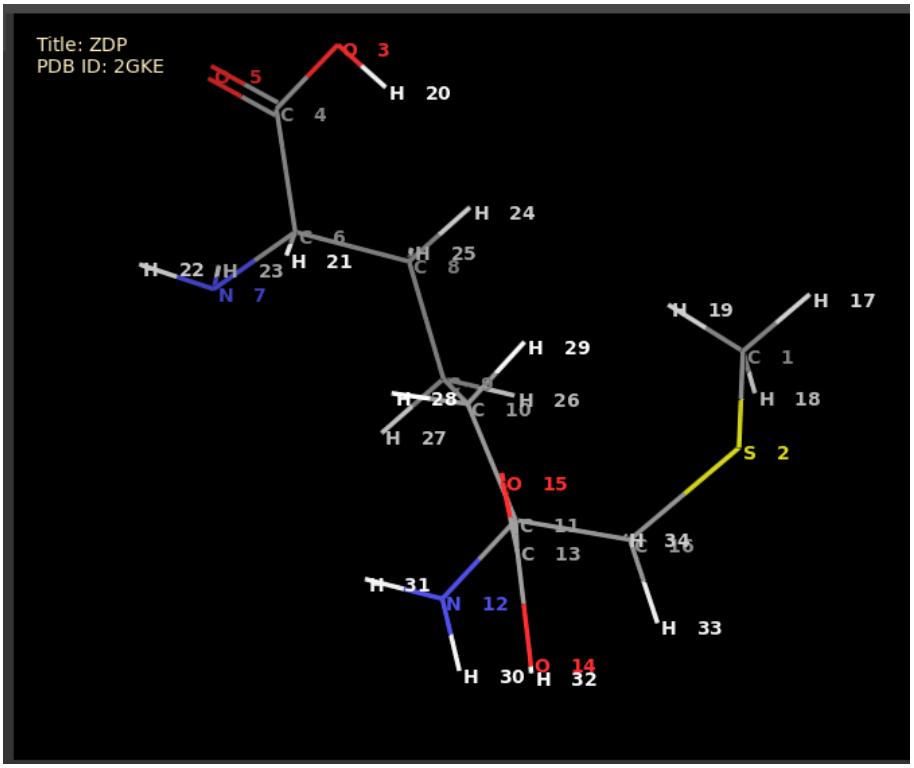

Figure 3.5. The ligand molecule opened with Maestro. The connection atoms are the ones for ligindices parameter. As in this example, connection atoms shown with the number of 1 and 2

The *receptor* flag should indicate the file of the original protein-ligand complex, or a protein-ligand complex created manually and saved as PDB format, as referred to in the previous Section 3.2.4.2. Meaning that the receptor referred to here should be the one that covalent bond exists. In this example, it is in the main folder, which is the previous directory from the one currently being worked on. Correct format is *"--receptor ../2gke.pdb"*.

The *residue* flag is the target residue to create covalent bonding. In this example, it is cysteine 73 of chain A. Correct format is *"-residue A:CYS73"*.

The *outputfile* flag is the name and desired format of the output file. PDB format is ideal. Add –cov to the name to distinguish. Correct format is *"--outputfile ZDPcov.pdb"*.

There is another parameter here hidden for this example. The covalent docking script automatically aligns ligand molecules to target residue. However, some residue types do not support automatic alignment. If the message such as *"[ERROR] The specified residue types are not supported automatically: HIS. Use either --recsmarts or --recindices."* is occurred, atoms that will be aligned on should be indicated with new parameter *"recindices". Atom identifier numbers of receptor atoms should be written* as the correct sequence for the numbers is "*the atom that ligand will be bonded, followed by a comma, then the first atom connected to that atom*". The correct format as an example is *"--recindices 818,816"*. Atom numbers should be checked from the ligand-receptor file located in main folder, not the one prepared for docking procedure which will be covered in next section. Glutamic acid (GLU), and histidine (HIS) are encountered as non-supported resides during docking of constructed dataset.

Command 3.5. Converting Ligand to PDBQT Format

```
pythonsh {MGLToolsPckgs Location}/AutoDockTools/Utilities24/prepare_receptor4.py -r ZDPcov.pdb
```

Replace "*{MGLToolsPckgs Location}*" with the precise directory where the MGLToolsPckgs folder is located, within the installation directory of MGLTools. The *r* flag should be the aligned ligand. Correct format is *"-r ZDPcov.pdb"*.

Command 3.6. Making Ligand Rigid and Flex

```
pythonsh {MGLToolsPckgs Location}/AutoDockTools/Utilities24/prepare_flexreceptor4.py -r ZDPcov.pdbqt -s ZDPcov:A:CYS73
```

The *r* flag is the aligned ligand converted to PDBQT format on previous step. Correct format is *"-r ZDPcov.pdbqt". The s* flag is the specification for flex residues. The specified residues, separated by comma, are given flexibility. The ligand and target residue must be specified. The format of the s flag is *"the name of aligned ligand file, followed by colon, the full name of residue"*. For other residues, residue

names are separated by underscore. Correct format is *"-s ZDPcov:A:CYS73"*. Final files are listed in Table 3.1.

Table 3.1. Files Inside 2Ligand Folder After Ligand Preparation

File Name		Description
ZDP.sdf	r_ZDP.sdf	The manually prepared receptor
ZDPcov.pdb	r_ZDPcov.pdb	Ligand aligned with target residue
ZDPcov.pdbqt	r_ZDPcov.pdbqt	The format supported by AutoDock4 and MGLTools
ZDPcov_rigid.pdbqt	r_ZDPcov_rigid.pdbq	The ligand in rigid form
ZDPcov_flex.pdbqt	r_ZDPcov_flex.pdbqt	The ligand in flex form
Residues.log		Show the target residue as written in s parameter of Command 6
Randomize_sdf.py		The python file for Program 1, naming unnecessary

3.2.7. Receptor Preparation

In this section, the receptor to which the prepared ligand will be docked will be prepared. Files that are generated in this part of the procedure should be saved in *1Receptor* folder. All commands of this section should be executed in *1Receptor* file if the file management is used.

For any docking experiment, the receptor should be the one with no ligand. Downloaded protein-ligand complex is opened in desired modelling tool once again. All ligands and other molecules including water molecules are deleted. After adding hydrogens, the receptor is saved in PDB format. In this example, file saved as *"2gke_clean_H.pdb"*.

If the binding site, within a 6 Ångström radius from target residue, contains another molecule or metal atom that could influence the orientation of the docked molecule, they should not be deleted. If the receptor has more than one chain and

the other chains do not intersect with the binding site, these other chains can be deleted to reduce computation.

Opening the terminal in *1Receptor* folder which the prepared receptor present and executing following commands will generate rigid and flex receptor files suitable for AutoDock4 calculations.

Command 3.7. Converting Receptor to PDBQT Format

```
pythonsh {MGLToolsPckgs Location}/AutoDockTools/Utilities24/prepare_receptor4.py -r 2gke_clean_H.pdb
```

Command 3.8. Making Receptor Rigid and Flex

```
pythonsh {MGLToolsPckgs Location}/AutoDockTools/Utilities24/prepare_flexreceptor4.py -r 2gke_clean_H.pdbqt -s 2gke_clean_H:A:CYS73
```

These commands function in the same way as those used for the ligand file. Final files are listed in Table 3.2.

Table 3.2. Files Inside *1Receptor* Folder After Receptor Preparation

File Name	Description
2gke_clean_H.pdb	The manually prepared receptor
2gke_clean_H.pdbqt	The format supported by AutoDock4 and MGLTools
2gke_clean_H_rigid.pdbqt	The receptor in rigid form
2gke_clean_H_flex.pdbqt	The receptor in flex form

3.2.8. Docking

In the final part of the procedure, parameters for generating the grid box and parameters for docking will be defined, and AutoDock4 will be executed. Opening the terminal in *3Docking* folder and executing following command will generate *GPF* file which the docking parameter is defined in it. All commands of this section should be executed in 3Docking file if the file management in section 3.3.3 is used.

Command 3.9. Generating Grid Parameter File (GPF)

```
pythonsh {MGLToolsPckgs Location}/AutoDockTools/Utilities24/prepare_gpf4.py -r
2gke_clean_H_rigid.pdbqt -l ZDPcov_flex.pdbqt -x ZDPcov_flex.pdbqt -y -I 20 -o
ZDP_[rigid_rec+flex_lig].gpf
```

When the receptor and ligand files are in different folders, the command gives error. That is why the receptor, for this example rigid one, and the ligand to be docked, for this example flex one, is copied to *3Docking* folder.

The *r* flag is the processed receptor, in either rigid or flex forms. Correct format is *"-r 2gke_clean_H_rigid.pdbqt"*.

The *l* flag is a processed ligand, in either rigid or flex forms. Correct format is *"-l ZDPcov_flex.pdbqt"*. The *x* flag is flex residue file name same as ligand file in this procedure.

The *y* flag sets the centers grid as the center of the ligand given with *l* flag. The *I* flag increments the given number by 3 for all dimensions, thereby defining the size of the grid box. In this example, the size of the grid box is 60x60x60 number of grid points (npts).

The *I* flag defines the name and directory of output file. In this example, all docking related files will be generated in 3Docking file, therefore correct format is *"ZDP_[rigid_rec+flex_lig].gpf"*. In this example, flexible ligands will be docked to rigid receptor. Other combinations can also be performed, so creating distinctive file names would facilitate the process.

In the GPF file created, a new parameter named *"parameter_file"* should be added and the file comes with MGLTools installation, *"AD4_parameters.dat"* should be indicated. This file is in *"AutoDockTools"* folder in MGLTools installation path. The new line should be added at the end of GPF file is

"parameter_file {MGLToolsPckgs Location}/AutoDockTools/AD4_parameters.dat"

Command 3.10. Generating Docking Parameter File (DPF)

```
pythonsh {MGLToolsPckgs Location}/AutoDockTools/Utilities24/prepare_dpf4.py -r
2gke_clean_H_rigid.pdbqt -l ZDPcov_flex.pdbqt -x ZDPcov_flex.pdbqt -o
ZDP_[rigid_rec+flex_lig].dpf -p move='docking_param'
```

Executing Command 10 will generate a *DPF* file. The *r, l, x* and *o* flags function in the same manner as those in command 9. The *p* flag is an indication to change any parameter. If the flag is given with *"move"* as in command 10 and a file, the parameters written inside the file will be applied. This can be used to change desired parameters automatically once the parameter file is created. For example, default run amount, 10, can be changed with *"ga_run 100"* by adding to the *"docking_param"* file or can be changed from command, *"- p ga_run=100"*.

All the parameters written inside *GPF* and *DPF* files are default values are default values, unless altered by different flags, and can be manually modified using a text editor. In any situation, for covalent docking, the value "unbound_model extended" inside *DPF* file must be replaced with "unbound_energy 0.0" (Figure 3.6). This change tells AutoDock4 to ignore the default unbound model calculations and use a fixed unbound energy value instead, which can lead to more accurate docking results for covalent interactions by avoiding incorrect assumptions about the ligand's energetic when it's not bound to the receptor.

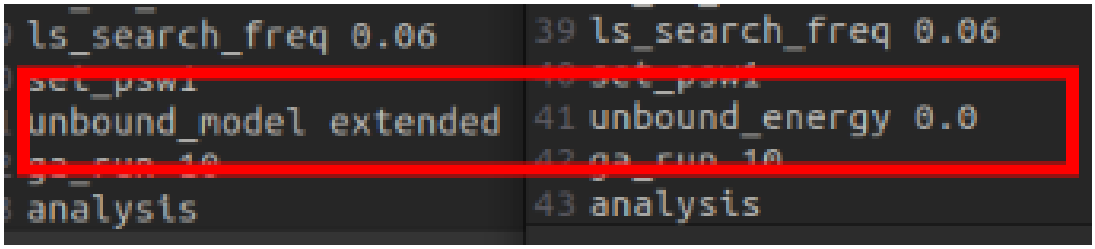

Figure 3.6. Changing parameters manually. DPF file is opened with text editor. The file on the left is the original file created after Command 10. The parameter is replaced with a new one as seen on the right.

After creating parameter files, docking can be executed. First, the grid box is generated by *autogrid4*, then docking is executed by *autodock4* commands.

Creating a "run" folder will consolidate the output from both executions, keeping the workspace organized and easier to manage.

Command 3.11. Grid Box and Map Generation with "*autogrid4*"

```
autogrid4 -p ZDP_[rigid_rec+flex_lig].gpf -l run/ZDP_autogrid_rig+flex_log.glg
```

Command 3.12. Docking with "*autodock4*"

```
autodock4 -p ZDP_[rigid_rec+flex_lig].dpf -l run/ZDP_docking_rig+flex_log.dlg
```

For both commands, the *p* flag should be the parameter file created before and the *l* flag is the log file of the process, *GLG* format for *autogrid4* and *DLG* format for *autodock4*.

During autogrid4 execution, some warnings like *"autogrid4: WARNING: Found an H-bonding atom with three bonded atoms, atom serial number 927"* are likely to be seen and can be ignored. After execution of Command 11, autogrid4, will generate bunch of map files in *3Docking* folder and GLG file in *run* folder.

DLG file contains all the details about the docking performed and contains also the docking results. Thus, the file is saved to the main folder, the previous location, to make it easier to find. This is because the *3Docking* folder will contain many files after *autogrid4* runs.

The DLG file can only be visualized by MGLTools by clicking *"Analyze* > *Docking* > *Open"* and selecting the *DLG* file. MGLTools is considered a less effective visualization tool because it offers less control over molecules compared to tools like PyMOL. The custom script in Program 2 is designed to extract the sections that display the coordinates of docked models and save them in *PDB* format. This allows the files to be opened with any visualization tool. Save the script as a Python file with the *".py"* extension. Opening a terminal in the same directory as *DLG* file, in this case main folder, and executing Program 2 generates a PDB file

with the name defined as variable *"output"* in Program 2. Final files are listed in Table 3.3.

Program 3.2. Python Script to Extract Docked Models and Save as *PDB* File

```
import os

#-----------------------------------
# Changeable Parts:
dlg_file = 'r_ZDP_docking_rig+flex_log.dlg' # DLG file name
output = 'r_ZDP_rig+flex_docked.pdb' # Output file name
#-----------------------------------

# Function to extract poses
def extract_poses(dlg_file):
    poses = []
    current_pose = []

    with open(dlg_file, 'r') as file:
        for line in file:
            # To find out start and end point of model
            if line.startswith("DOCKED: MODEL"): # Proceed with current pose
                current_pose = []
            elif line.startswith("DOCKED: ENDMDL"):  # End process with current pose
                poses.append(current_pose)
                current_pose = []
            elif current_pose is not None and line.startswith("DOCKED:"):
                current_pose.append(line[len("DOCKED: "):])
    return poses

# Extract poses and saving poses to a PDB file
extracted_poses = extract_poses(dlg_file)
with open(output, 'w') as file:
    for i, pose in enumerate(extracted_poses):
        file.write(f"MODEL       {i+1}\n")
        file.writelines(pose)
        file.write("ENDMDL\n")
```

Table 3.3. Files Inside *run* Folder After Docking

File Name		Description
ZDP_autogrid_rig+flex_log.glg	r_ZDP_autogrid_rig+flex_log.glg	GLG files contains details about autogrid4 execution
ZDP_docking_rig+flex_log.dlg	r_ZDP_docking_rig+flex_log.dlg	DLG files is the result
ZDP_rig+flex_docked.pdb	r_ZDP_rig+flex_docked.pdb	Docking results as PDB file
convert_to_pdb.py		The python file for Program 2, naming unnecessary

3.2.9. Automation of Docking Procedure

Apart from setting up the initial molecules (like a ligand with two bond sites and a receptor without its ligand) which are created with desired visualization tool, the rest of the process involves python codes and terminal commands. All commands can be arranged in correct order to create a shell script. Manual editing of the parameter files is also possible with shell scripting language. If it is combined with a visualization tool that also uses a scripting language, docking procedure can be automized. This automation would be great for repeating tasks, but its limited flexibility might prevent it from handling every scenario. Additionally, it requires intermediate coding and computer knowledge and Unix based shell terminal.

3.3. Detailed Framework: UCSF DOCK6

Core components of the AutoDock4 covalent docking framework are summarized in Figure 3.7.

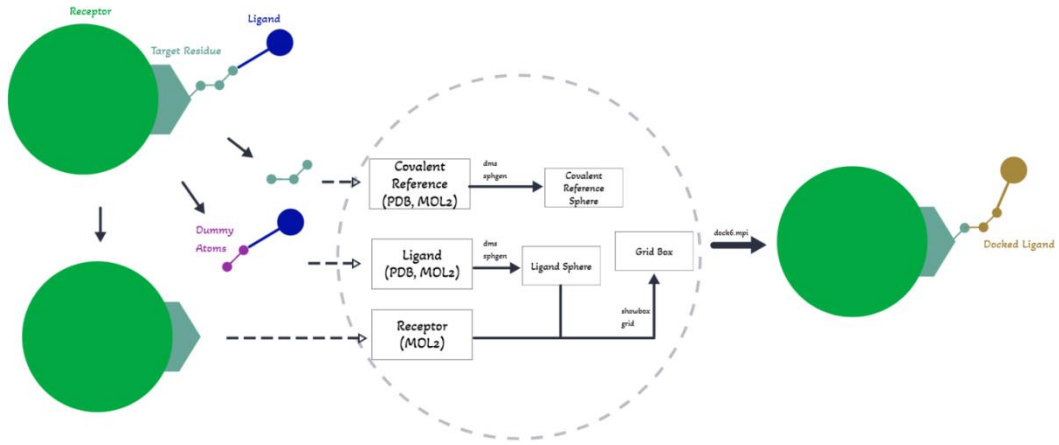

Figure 3.7. Simplified UCSF DOCK6 Covalent Docking Framework

3.3.1. Requirements

Pre-requirements for UCSF DOCK6 covalent docking:

- UCSF DOCK6 installed (https://dock.compbio.ucsf.edu/DOCK_6/index.htm).
- Base requirements for DOCK6 and *make* package.
- DMS (http://www.cgl.ucsf.edu/Overview/software.html#dms).

Non-essential requirements:

- UCSF Chimera or ChimeraX (https://www.cgl.ucsf.edu/chimera/).
- Schrodinger PyMOL (https://pymol.org/2/).
- Python 3 and RDKit (https://www.rdkit.org/docs/Install.html).

UCSF DOCK6 includes a covalent docking feature that can be used when connection atoms are assigned as "*dummy atoms*". The method works like the flexible side chain method in AutoDock4, ligand with two dummy atoms is aligned and replaced with ones on target residue. DOCK6 can be installed with several configurations during installation including the one that enables parallel

processing capability to reduce docking time significantly. Details can be checked from UCSF DOCK6 manual.

Lacking base requirements will be encountered during installation. Since it is installed via building "*makefile*", some modules that are needed by *make* package should also be downloaded. The terminal used for installation will provide the information.

DMS is a module to create surface required during procedure. This is terminal based program; however same function can be used on user interface of Chimera (not available in ChimeraX).

UCSF Chimera's extensive functionality makes it a good choice for molecular editing tasks. Tool is preferred alongside UCSF DOCK6 since they share the same provider. Any modelling and visualization tool can be used. PyMOL is a safe zone that can be used for any task and especially validating the edited molecules, also good at virtualizing docking results, as mentioned in section 3.1.1. Python and RDKit are also used for the same purpose as mentioned earlier. Section 3.1.3 can be viewed for further information about terminal commands.

3.3.2. File Management

Among the 40 protein-ligand complexes in the benchmark dataset, 2GKE (https://www.rcsb.org/structure/2gke) is used as a case study for explaining the detailed procedure. The protein-ligand complex 2GKE will serve as the name of the receptor and ZDP is the ligand bound. DOCK6 generates numerous files during procedure. Making file management is optional, however organization can be enhanced by sorting working folder with "*Date Modified*" in descending order.

Suggested file management for DOCK6 only:

- Main folder: *2GKE*
- Sub-folders:
    - *1Receptor* (Receptor related files will be generated here.)

- *2Ligand* (Ligand related files will be generated here.)
- *3SphereAndGrid* (Grid related files will be generated here.)
- *4Docking* (Docking related files will be generated here.)

3.3.3. Ligand Preparation

All files that are manually processed in this section are saved to the *2Ligand* folder. The original protein-ligand complex downloaded as PDB format, or manually created complex, is placed into main folder. 2GKE is already a protein-ligand complex which will be used for re-docking experiments. For docking experiments with different ligands, refer to section 3.2.4.1, *"AutoDock4 / For Re-Docking Experiments"*.

Complex downloaded is opened with Chimera. Ligand and the target residue are selected by clicking or typing *"select :ZDP:73"* to Chimera terminal as ZDP is the name of ligand and 73 is the target residue. For receptors with multiple chains, *"select :ZDP.a:73.a"* to select molecules on chain a. Selection is inverted from top dropdown menu, *"Select > Invert (all models)"* and delete selected atoms or by typing *"sel invert true"* and *"delete sel"*. Hydrogens and charge are added from *"Tools > Structure Editing > AddCharge"*, or *"addh"* and *"addcharge"*. It is recommended to add charges to the molecule as considering the ligand as non-standard residue. To achieve this, type *"addcharge nonstd :ZDP 1"* which the number is total charge or can be choose from Add Charges UI. For this thesis, ligands are assigned a charge of 1 unless doing so causes errors. In error cases, they receive a charge of 0 or adding charge without defining non-standard residue in order. The molecule is saved as *MOL2*, in this example, *"ligprep.mol2"*.

The ligprep is the molecule both target residue and ligand exist. After saving ligprep, the Chimera window is cleaned by closing the section and ligprep is opened. ZDP and two connection atoms are selected, in this case target residue is cysteine, atoms are SG and CB. Typing *"select :ZDP:@sg,cb"* does the same job. The symbol "@" is used for selection of single atoms with specified names afterwards. Selection is inverted and deleted again. Chimera window sometimes

hide hydrogens even if they exist. All atoms are enabled by menu, "Actions > Atom/Bonds > show". If any hydrogen on connection atoms remains, it is crucial that they should be deleted. The resulting molecule should be saved as both *MOL2* and PDB formats, in this example *"ZDP.mol2"* and *"ZDP.pdb"*. The molecule with MOL2 format is the ligand molecule to be docked and *PDB* format is used for creating random conformation of is since RDKit and DMS do not support *MOL2.*

Command 3.13. Chimera Terminal Commands for Ligand Preparation in Order

```
open pdb:{NAME_OF_COMPLEX}
select :{LIGAND_NAME}:{RESIDUE_NUMBER}
sel invert true
delete sel
addh
addcharge nonstd :{LIGAND_NAME}{CHARGE}
write format mol2 0 ligprep.mol2
close all

(Ligprep file is opened)
select :{LIGAND_NAME}:@{ATOM_1},{ATOM_2}
sel invert true
delete sel
write format mol2 0 {LIGAND_NAME}.mol2
write format pdb 0 {LIGAND_NAME}.pdb
```

During the process, some errors were encountered with the randomized ligand. Program 1 selectively randomizes atoms within the ligand file, excluding the two connection atoms and save as *"r_ZDP.mol2"* and *"r_ZDP.pdb"* in this example. Maintaining these connection atoms helps avoid future errors caused by ligand randomization. Running this script requires Open Babel, or manually converting output in *PDB* format to *MOL2*.

Program 3.3. Python Script to Randomize Coordinates of Ligand with *MOL2* Format

```
from rdkit import Chem
from rdkit.Chem import AllChem
import os
import subprocess

#---------------------------------
# Changeable Parts:

file = '/home/ruzgar/Desktop/2GKE/2Ligand' # File directory
ligand_name = 'ZDP' # File name
#---------------------------------

def convert_pdb_to_mol2(pdb_path, mol2_path):
    subprocess.run(["obabel", pdb_path, "-O", mol2_path])

def generate_conformation(mol):
    # Generate a random conformation
    AllChem.EmbedMolecule(mol, useRandomCoords=True, coordMap={0:
mol.GetConformer().GetAtomPosition(0), 1: mol.GetConformer().GetAtomPosition(1)})
    # Clear the constraint
    mol.ClearComputedProps()

def conversion(ligand):
    # Load PDB file
    in_file = ligand + '.pdb'
    mol = Chem.MolFromPDBFile(in_file, sanitize=True, removeHs=False)

    if mol:
        # Generate a random conformation with two atoms fixed
        generate_conformation(mol)

        # Save the molecule to a new PDB file
        out_file_pdb = 'r_' + ligand_name + '.pdb'
        Chem.MolToPDBFile(mol, out_file_pdb)

        # Convert PDB to MOL2 using Open Babel via subprocess
        out_file_mol2 = 'r_' + ligand + '.mol2'
        convert_pdb_to_mol2(out_file_pdb, out_file_mol2)

conversion(ligand_name)
```

After this part of ligand preparation procedure, two ligands, one with MOL2 and another with PDB format, are created. PDB files are used for DMS which will be covered in section 3.2.5 MOL2 files are the ones to be docked. As mentioned before, the two connection atoms should be assigned as dummy atoms. The

ligand molecule is structured with the two connection atoms followed by the main body of the ligand after editing as in Figure 3.8. The outermost atom is modified as "D2" and the other atom (the one closest to ligand and involving in the formation of the covalent bond) as "D1". Also, atom types are changed as "Du" in MOL2 files and "LP" in PDB files. Final files are listed in Table 3.4.

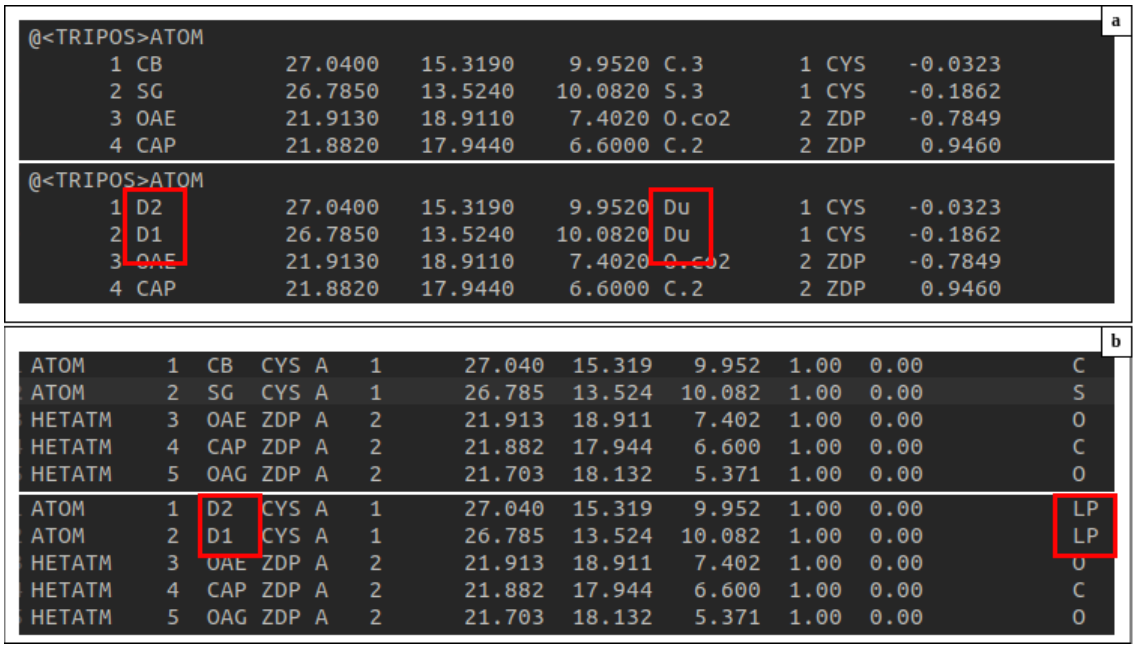

Figure 3.8. Modifying the Connection Atoms as Dummy Atoms. The file on the top is the original one and the file on the bottom is modified one. (a) *ZDP.mol2* file created after ligand procedure. Connection atoms are changed as D1, D2 and Du as shown in red boxes. (b) *ZDP.pdb* file created after ligand procedure. Connection atoms are changed as D1, D2 and LP as shown in red boxes.

Table 3.4. Files Inside 2*Ligand* Folder After Ligand Preparation

File Name		Description
Ligprep.mol2		Molecule formed by the ligand and target residue
ZDP.mol2	r_ZDP.mol2	The ligand atom that connection atoms are modified as dummy atoms, those are the ligands to be used in docking
ZDP.pdb	r_ZDP.pdb	Ligand with randomized coordinates and modified. Those are the ligands to be used for DMS.
randomize_mol2.py		The script contains Program 1

3.3.4. Preparation of Receptor and Covalent Bond Reference

All files that are manually processed in this section are saved to the *1Receptor* folder. The ligprep file is opened with Chimera once again. Only three atoms will be selected to be covalent reference file, all of them are the atoms of target residue. Two connection atoms and additionally one nearest atom on receptor is selected. In this case, since the target residue is cysteine, SG, CB and CA atoms are selected. The third atom, CA, is used to attach docking to receptor while other two for aligning the ligand. This can also be achieved by typing *"select :cys@sg,cb,ca"* Selection is inverted and deleted (Figure 3.9). A molecule of three atoms is saved as both *PDB* and *MOL2* format, since DMS requires PDB format, in this example *"2gke_ref.pdb"* and *"2gke_ref.mol2"*.

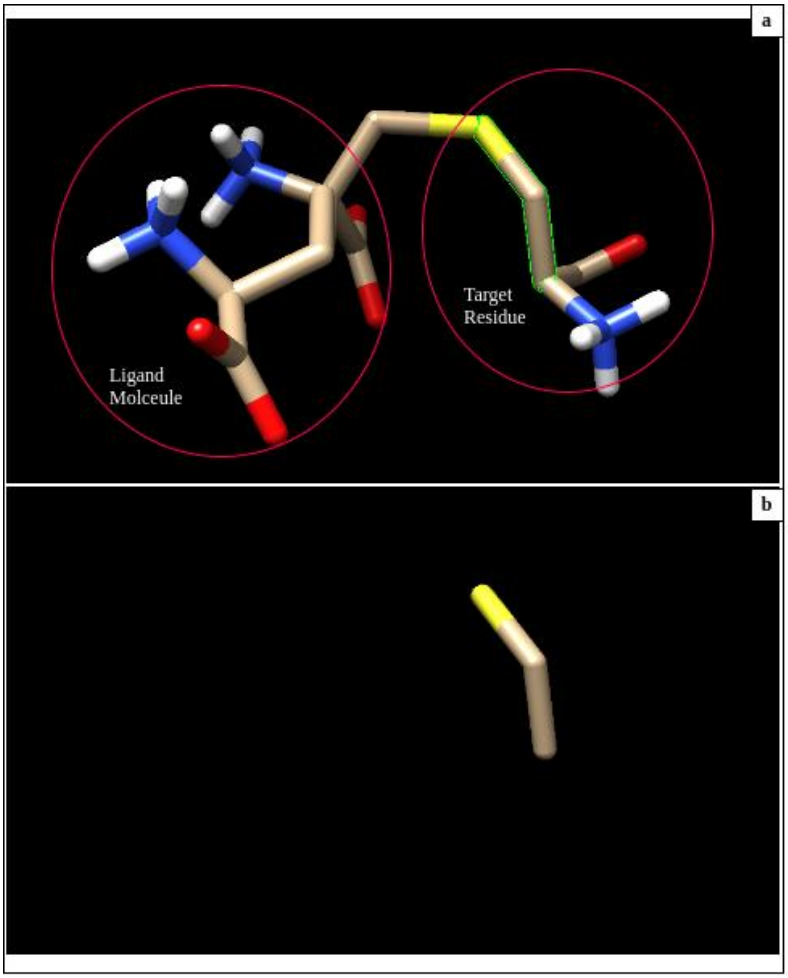

Figure 3.9. Covalent Reference in Chimera. (a) The green selection takes two connection atoms SG and CB, and the next atom on target residue. (b) Covalent reference

The Chimera window is cleaned and the original complex, 2GKE, is opened. Water molecules are removed by typing *"show solvent"* and *"delete solvent"*. All ligands and other molecules are deleted. If the binding site, within a 6 Ångström radius from target residue, contains another molecule or metal atom that could influence the orientation of the docked molecule, they should not be deleted. If the receptor has more than one chain and the other chains do not intersect with the binding site, these other chains can be deleted to reduce computation. After cleaning the structure, hydrogen and charge are added. Later, the two connection atoms of target residue are selected and deleted. This can also be achieved by

typing *"select :73@sg,cb"* which is the command deleting SG and CB atoms of residue 73. Deleting hydrogen atoms of target residue should not be skipped after deleting connection atoms. Atoms can also be selected by mouse click while holding ctrl and shift (Figure 3.10). The structure is saved as MOL2, in this example *"2gke_clean_H_notarget.mol2"*. Final files are listed in Table 3.5.

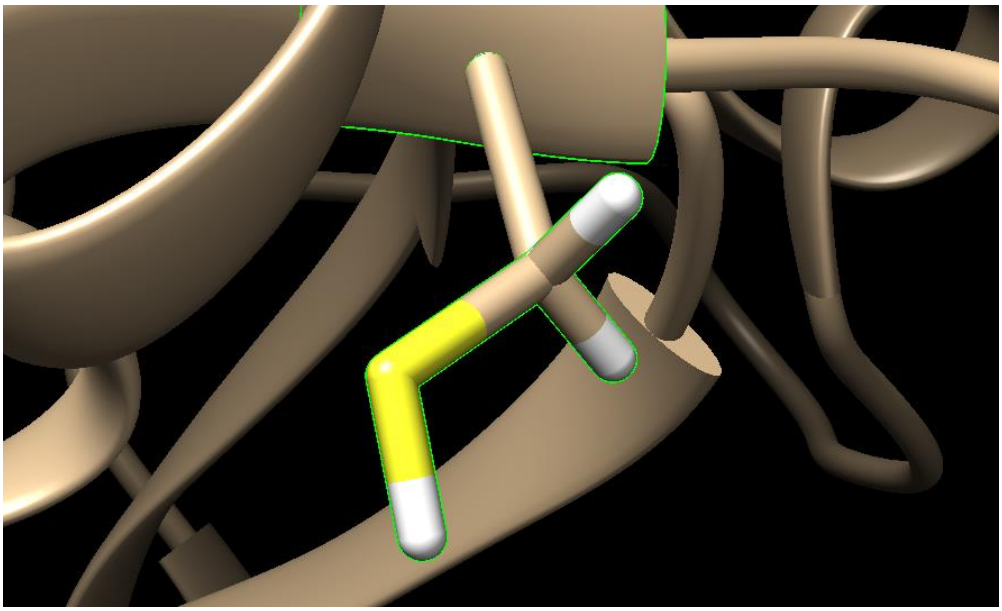

Figure 3.10. 2GKE in Chimera. Two connection atoms of CYS 73 and the hydrogens bound to them are selected.

Table 3.5. Files Inside *1Receptor* Folder After Receptor Preparation

File Name	Description
2gke_ref.mol2	Covalent reference file to be used in docking
2gke_ref.pdb	Covalent reference file to be used for DMS
2gke_clean_H_notarget.mol2	The receptor to be used in docking

Command 3.14. Chimera Terminal Commands for Receptor Preparation in Order

```
(Ligprep file is opened)
select :{TARGET_RES_TYPE}:@{ATOM_1},{ATOM_2},{ATOM_3}
sel invert true
delete sel
write format mol2 0 {RECEPTOR_NAME}_ref.mol2
write format pdb 0 {RECEPTOR _NAME}_ref.pdb
close all

(Complex is opened)
show solvent
delete solvent
show ligand
delete ligand
addh
addcharge
select :{TARGET_RESIDUE_NUMBER}:{ATOM_1},{ATOM_2}
delete sel
select :{TARGET_RESIDUE_NUMBER}:{HYDROGEN_ATOM_IDS}
delete sel
write format mol2 0 {RECEPTOR_NAME}_clean_H_notarget.mol2
close all
```

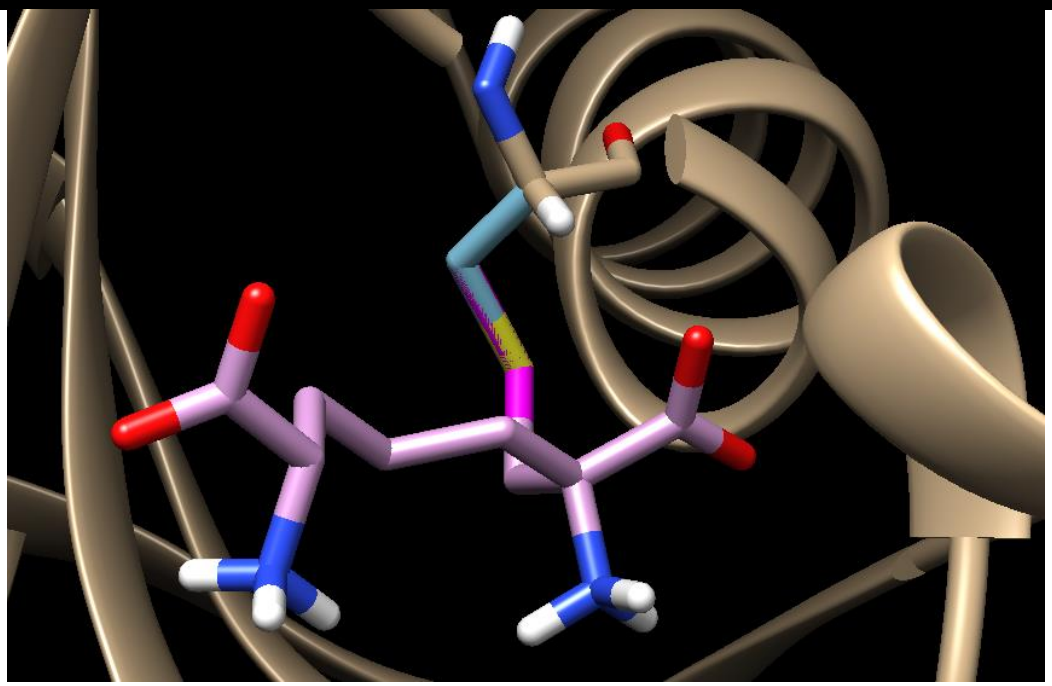

Figure 3.11. Prepared Receptor, Ligand and Covalent Reference

Figure 3.11 shows all prepared molecules at once. Magenta colored atoms are connection atoms on target residue that are modified as dummy atoms. Light

blue colored atoms are covalent reference. Brown is the receptor. As the connection atoms of receptor are deleted, they are not seen. Covalent reference molecule acts as bridge. One of the atoms intersects with target residue and completes deleted atom. The other two atoms intersect with magenta atoms. Magento atoms are used to align on those two atoms of covalent reference and replaced during docking.

3.3.5. Generation of Spheres

From this section onward, the manually created molecules will undergo processing steps for docking calculation. For molecular docking in DOCK6, surface creation of the target receptor is a critical step to create spheres which selected spheres will be used to predict interactions and to generate proper grid box. It is representations of potential binding pockets within a receptor molecule and act as target placing the ligand during process. In docking, generated spheres are selected to create grid box to be docked. However, in covalent docking since the binding site and covalent interaction are already known, the covalent reference molecule is used to align dummy atoms and the ligand itself will be used for grid box. Note that ligand is already in the desired place in re-docking experiments. In docking of a random ligand to molecule, as it is mentioned in section 3.2.4.1, *"AutoDock4 / For Re-Docking Experiments"*, the ligand is already moved manually to the place near target residue to create manual covalent bond. So, the moved ligand is used to create grid box.

There are two ways to create surface file to process into sphere. The covalent reference file and ligand file is separately opened in Chimera and surface of molecule is shown by *"Action* > *Surface* > *Show"*, then Dot Molecular Surface (*DMS*) is exported by *"Tools* > *Structure Editing* > *Write DMS"* (ChimeraX has not this function). This will create a file with *DMS* extension. File is placed to *3SphereAndGrid* folder.

The second way is creating the same file easier with "*DMS module*" but with use of terminal. Saving the molecules in both *MOL2* and *PDB* format is because *DMS*

module requires the molecule in *PDB* format to process. A terminal is opened in *3SphereAndGrid* folder and command 3 is executed to create *DMS* file. In this example, files saved as "*ref_surface.dms*" and "*ZDP_surface.dms*".

Command 3.15. *DMS* creation with "*DMS module*"

```
dms ../1Receptor/2gke_ref.pdb -a -n -o ref_surface.dms
dms ../2Ligand/ZDP.pdb -a -n -o ZDP_surface.dms
```

The *a* flag is used to select all atoms and the *n* flag is used to calculate surface normals, both are required for covalent docking. The *o* flag is used to specify output name.

In the next step, by referencing the *DMS* files created, spheres will be generated. The *sphgen* module, which comes with UCSF DOCK6 installation, is used for this job. The sphgen module requires a file named INSPH that includes parameters to work with. Detailed information about parameters is written in UCSF DOCK6 manual. The best parameters for covalent docking are given in Parameters 1 and Parameters 2 which can be copied into the file. A file named INSPH is created in *3SphereAndGrid* folder, and parameters are added.

Parameters 3.1. Contents of INSPH File for Covalent Reference

```
ref_surface.dms
L
X
0.9
4.0
1.4
ref_sphere.sph
```

Parameters 3.2. Contents of INSPH File for Ligand

```
ZDP_surface.dms
L
X
0
4.0
1.4
ZDP_sphere.sph
```

Executing Command 4 using INSPH file created will generate desired sphere file as the name of the last line in file. This step is performed separately because both parameters require to be written in a file named INSPH and gives output file OUTSPH. Also, OUTSPH should be renamed or deleted before running again. In this example, "*ZDP_sphere.sph*" and "*ref_sphere.sph*" are generated.

Command 3.16. Command to Execute *Sphgen* Module

```
sphgen -i INSPH -o OUTSPH
```

The sphere file of covalent reference is created. However, the molecule order is required to be reversed compared to the original sequence for proper alignment of dummy atoms. The reference file can be opened as text and the order can be checked. In most cases, the order will be shown from the nearest atom of receptor to the further. In this example, the order is CA-CB-SG, but desired order of surface file is SG-CB-CA, the connection atom as first. Since the surface file has the same order, the third line of cluster 1 in *ref_sphere.sph* file is changing place with the first line as shown in Figure 3.12. In this example, the file is saved as "*ref_sphere_modified.sph*".

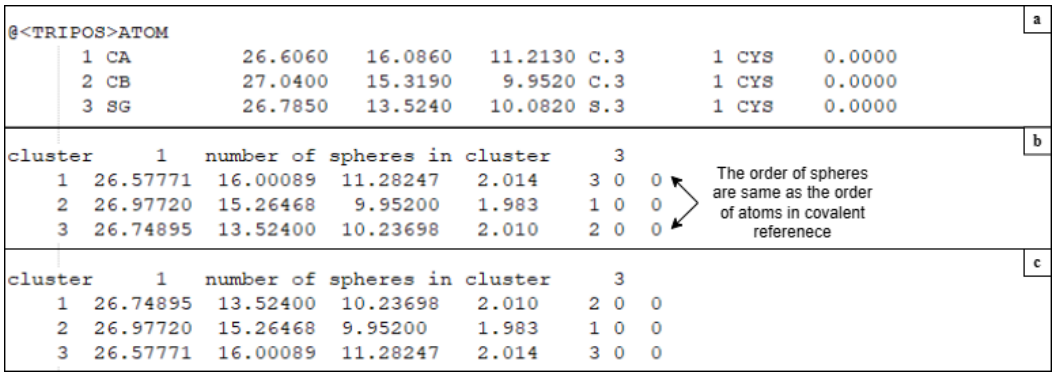

Figure 3.12. Swapping Positions of Spheres. (a) Atoms in *2gke_ref.mol2, (b)* Spheres in *ref_sphere.sph* after sphgen command, (c) *ref_sphere.sph* as the first and third spheres are swapped their position.

3.3.6. Grid Box Generation

The grid box is generated in a radius of ligand spheres. The *showbox* module, which comes with UCSF DOCK6 installation, is used to visually display a box around a selected set of spheres. Visualizing the spheres with the showbox allows to access whether the sphere placement is meaningful and adjust the box to fine-tune. In this section, there is no need to create separate boxes for ligand and randomized one.

Showbox module requires a file named *"box.in"* that includes the parameters for box generation. The best parameters for covalent docking are given in Parameters 3. Executing showbox in *3SphereAndGrid* folder creates a box file needed for grid generation. In this example, *"ZDP.box.pdb"* is created.

Parameters 3.3. Contents of "box.in" File for Showbox

```
Y
8.0
ZDP_sphere.sph
1
ZDP.box.pdb
```

Command 3.17. Command to Execute *Sphgen* Module

```
showbox < box.in
```

Grid generation is performed with a module *grid*, which comes with UCSF DOCK6 installation, and requires a file named "grid.in" with proper parameters. Optimum parameters are given in Parameters 4. The receptor_file parameter is the receptor, connection atoms of target residues deleted and saved as *MOL2* format. Van der Waals and chemical definition files come with UCSF DOCK6 installation and located in *"parameters"* folder in installation path. Executing grid in *3SphereAndGrid* folder creates three files named "gridinfo.out", "grid.bmp" and "grid.nrg" which will be used in docking process. Once the proper grid box is created, there is no need to create a new box for other docking processes. In this example, the grid box created is used for both ZDP and r_ZDP file. All

created molecules and files are opened in Chimera to be sure (Figure 3.13). Final files are listed in Table 3.6.

Parameters 3.4. Contents of "grid.in" File for Grid

```
compute_grids              yes
grid_spacing               0.4
output_molecule            no
contact_score              no
energy_score               yes
energy_cutoff_distance     9999
atom_model                 a
attractive_exponent        6
repulsive_exponent         9
distance_dielectric        yes
dielectric_factor          4
allow_non_integral_charges      yes
bump_filter                yes
bump_overlap               0.75
receptor_file              ../1Receptor/2gke_clean_H_notarget.mol2
box_file                   ZDP.box.pdb
vdw_definition_file        {DOCK6 Location}/parameters/vdw_AMBER_parm99.defn
chemical_definition_file {DOCK6 Location}/parameters/chem.defn
score_grid_prefix          grid
```

Command 3.18. Command to Execute *Grid* Module

```
grid -i grid.in -o gridinfo.out
```

Table 3.6. Files Inside *3SphereAndGrid* Folder After Grid Generation

<table>
<tr><th colspan="2">File Name</th><th>Description</th></tr>
<tr><td colspan="2">Ref_surface.dms</td><td rowspan="2">DMS files</td></tr>
<tr><td>ZDP_surface.dms</td><td>r_ZDP_surface.dms</td></tr>
<tr><td colspan="2">INSPH</td><td rowspan="2">Input and log output of sphgen</td></tr>
<tr><td>OUTSPH_ZDP</td><td>OUTSPH_rZDP</td></tr>
<tr><td>ZDP_sphere.sph</td><td>Ref_sphere.sph</td><td>Sphere files from sphgen, can be viewed with Chimera</td></tr>
<tr><td colspan="2">Box.in</td><td rowspan="2">Input and output of showbox. Box can be viewed with Chimera</td></tr>
<tr><td colspan="2">ZDP.box.pdb</td></tr>
<tr><td colspan="2">Grid.in</td><td rowspan="2">Input and log output of grid.</td></tr>
<tr><td colspan="2">Gridinfo.out</td></tr>
<tr><td>Grid.bmp</td><td>Grid.nrg</td><td>The grid files to be used in docking, can be viewed with Chimera.</td></tr>
</table>

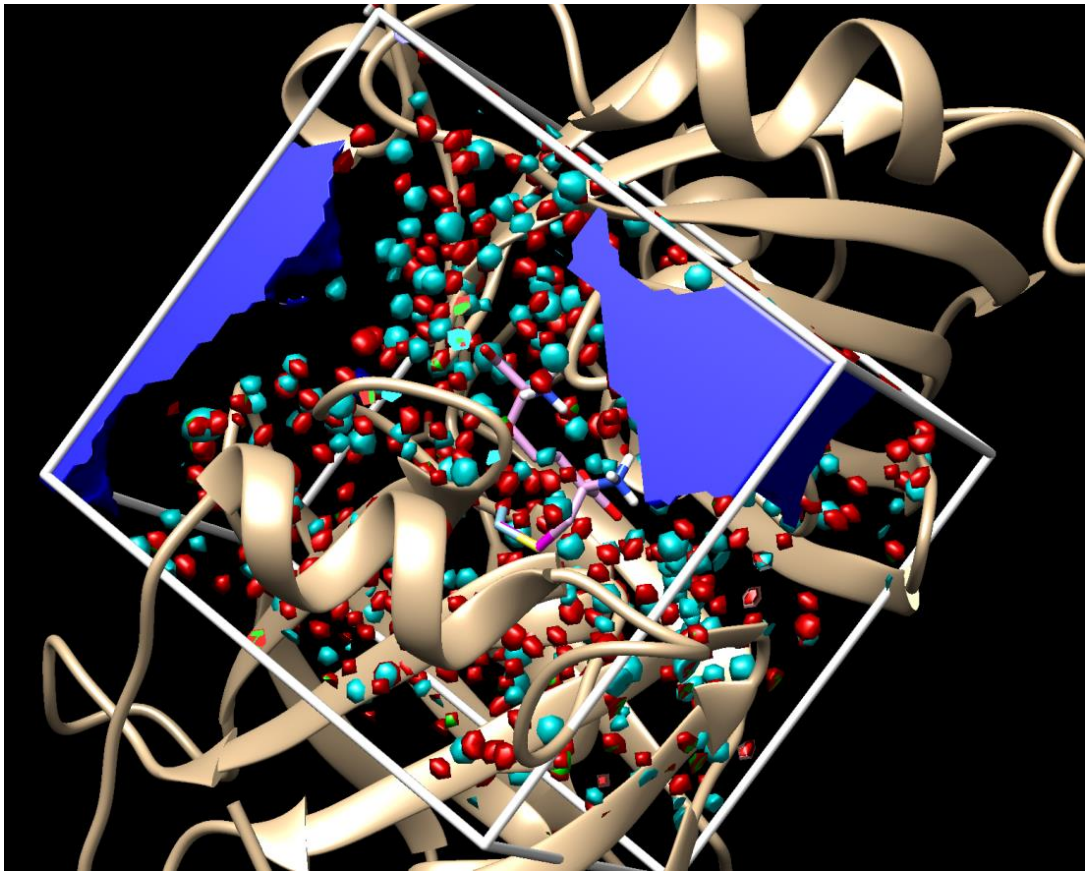

Figure 3.13. All Prepared Atoms and Grid Properties

3.3.7. Docking

For the docking, a module named *dock6* which comes with UCSF DOCK6 installation will be used. For different configuration of DOCK6 as mentioned in section 4.1 *"Requirements"*, the module name is different. The module requires an input file containing specific parameters for it to function. Inside *4Docking* folder, a file named *"covalent.in"* is created.

The best and most necessary docking parameters for covalent docking can be copied from Parameters 5. The conformer_search_type must be assigned as covalent. The ligand_atom_file parameter is the ligand to be docked. The receptor_site_file is the receptor to dock ligand. However, for covalent docking, covalent reference will be assigned as receptor. The grid_score_grid_prefix is two grid files, with BMP and NRG extension, generated on previous section. The location of grid files and prefix names, for this example prefix is grid, should be assigned. The vdw_defn_file, flex_defn_file, and flex_drive_file is assigned as *"parameters"* folder in installation path. The ligand_outfile_prefix is the file name prefix of docked molecules.

Although "use_rmsd_reference_mol" and "use_ligand_spheres" are set to "yes", they don't significantly affect the covalent docking process and could potentially be set to "no" automatically. While "write_conformations" and "num_scored_conformers" aren't strictly required, they are set to "yes" and "100" to ensure DOCK6 generates up to 100 conformations if it is possible. The ligand_outfile_prefix is the name of docked molecules. Two files, one with best score and one with all possible conformations in best to worse order, are created with this name prefix after docking process.

Parameters 3.5. Contents of "covalent.in" File for Covalent Docking

```
conformer_search_type                 covalent
pruning_use_clustering                yes
pruning_max_orients                   100
pruning_clustering_cutoff             100
bondlenth                             1.8
dihideral_step                        10.0
pruning_conformer_score_cutoff        100.0
use_clash_overlap                     no
write_growth_tree                     no
use_internal_energy                   yes
internal_energy_rep_exp               12
internal_energy_cutoff                100.0
ligand_atom_file                      ../2Ligand/ZDP.mol2
limit_max_ligands                     no
skip_molecule                         no
read_mol_solvation                    no
calculate_rmsd                        yes
use_rmsd_reference_mol                yes
rmsd_reference_filename               ../2Ligand/ZDP.mol2
use_database_filter                   no
orient_ligand                         yes
automated_matching                    yes
receptor_site_file                    ../3SphereAndGrid/ref_sphere.sph
max_orientations                      1000
critical_points                       no
chemical_matching                     no
use_ligand_spheres                    yes
ligand_sphere_file                    ../3SphereAndGrid/ZDP_sphere.sph
bump_filter                           no
score_molecules                       yes
contact_score_primary                 no
grid_score_primary                    yes
grid_score_rep_rad_scale              1
grid_score_vdw_scale                  1
grid_score_es_scale                   1
grid_score_grid_prefix                ../3SphereAndGrid/grid
minimize_ligand                       yes
minimize_anchor                       no
minimize_flexible_growth              yes
use_advanced_simplex_parameters       no
minimize_flexible_growth_ramp         no
simplex_max_cycles                    1
simplex_score_converge                0.1
simplex_cycle_converge                1.0
simplex_trans_step                    1.0
simplex_rot_step                      0.1
simplex_tors_step                     10.0
simplex_anchor_max_iterations         0
simplex_grow_max_iterations           0
simplex_grow_tors_premin_iterations   1000
simplex_random_seed                   0
simplex_restraint_min                 yes
simplex_coefficient_restraint         10.0
atom_model                            all
vdw_defn_file                         {DOCK6 Location}/parameters/vdw_AMBER_parm99.defn
flex_defn_file                        {DOCK6 Location}/parameters/flex.defn
flex_drive_file                       {DOCK6 Location}/parameters/flex_drive.tbl
ligand_outfile_prefix                 ZDP_output
write_orientations                    no
num_scored_conformers                 100
write_conformations                   yes
cluster_conformations                 yes
rank_ligands                          no
```

The docking process can be started by executing Command 7. The *i* flag is parameter file, and the *o* flag is the file detailed information about docking process written. The first command can be used for default installation of UCSF DOCK6 while the second command is the one with parallel processing feature which is used in this procedure.

Command 3.19. Command to Execute *Grid* Module

```
dock6 -i covalent.in -o covalent.out
dock6.mpi -i covalent.in -o covalent.out
```

File named *"ZDP_output_conformers.mol2"* contains all docked molecules results. If the file is created empty, there should be a mistake during any part of the process. If changing parameters in covalent.in file according to log file, "*covalent.out*", does not help, whole process should be validated one by one to find problem. The most problematic part is the molecule preparation part. It should be ensured that molecules are prepared correctly. The *"debug.mol2"* file created after docking may help to resolve the problem. It should be ensured that molecule in debug.mol2 correctly aligned to receptor and ligand, can be checked by opening all molecules on PyMOL.

3.3.8. Automation of Docking Procedure

UCSF DOCK6 is a terminal based docking program. All files including the parameter files like INSPH etc. and their contents can be created and modified by terminal commands. UCSF Chimera also has its own scripting language run with Python. The target residue connection atom removal procedure will be the same for every sample if the target residues are the same. Atom IDs will also be the same. Changing parameters are ligand, residue number etc. In theory, an automatic receptor, ligand and covalent reference file creation script is possible. Thus, it's feasible to create a shell script that automates the entire process, but it would likely be quite complex.

3.4. Detailed Framework: Cresset Flare

Core components of the AutoDock4 covalent docking framework are summarized in Figure 3.14.

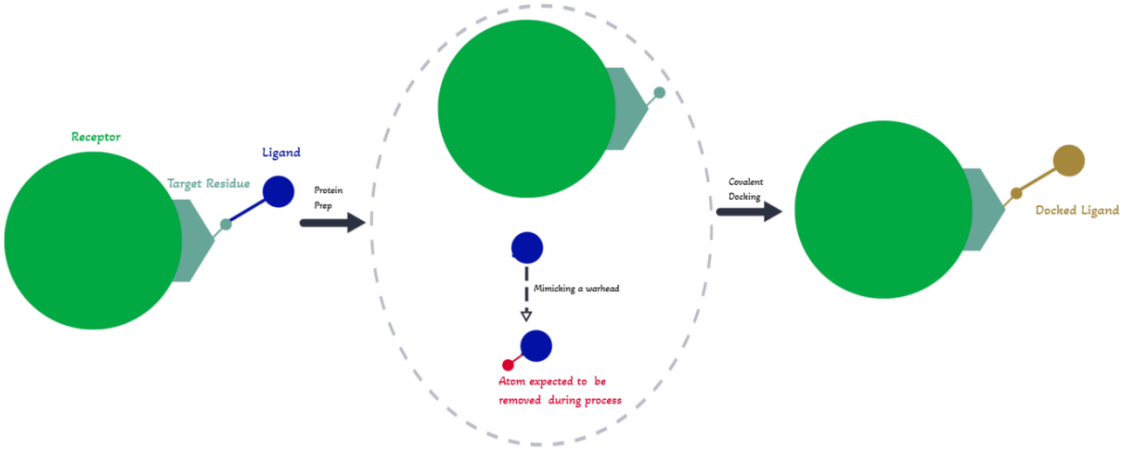

Figure 3.14. Simplified Cresset Flare Covalent Docking Framework

3.4.1. Requirements

Pre-requirements for Cresset Flare covalent docking:

- Licensed Cresset Flare (https://www.cresset-group.com/software/flare/).

Non-essential requirements:

- Schrodinger PyMOL (https://pymol.org/2/).
- Python 3 and RDKit (https://www.rdkit.org/docs/Install.html).

Flare is a software platform designed specifically for structure-based drug discovery. It offers a comprehensive suite of features, ranging from molecule creation to running molecular dynamics simulations. The covalent docking feature is available in Flare version 7 or later versions. Although Flare is typically a commercial software, it provides a free education license option for academics that includes the necessary functionalities for performing covalent docking.

Flare has all the necessary features already; a covalent docking procedure might not need any additional or supportive tools. However, RDKit is used to create random conformation of ligand and PyMOL is used to virtualize all docking

results, in same way done in AutoDock4 and DOCK6 procedures. Python is needed for RDKit and pyflare, which allow Flare functionality in Python. Flare does not need a file management system as it does in other programs. However, a separate folder can be created to store Flare saves and ligands, receptors etc.

Among the 40 protein-ligand complexes in the benchmark dataset, 2GKE (https://www.rcsb.org/structure/2gke) is used as a case study for explaining the detailed procedure.

3.4.2. Ligand and Receptor Preparation

For re-docking experiments, the receptor is downloaded from RCSB in PDB format. Although 40 different protein-ligand complexes were used, 2GKE (https://www.rcsb.org/structure/2gke) will be used as an example for this procedure. From the small molecules window, the ligand molecule should also be downloaded as SDF format, the details will be covered. Installed ligand can be randomized with the program 3.1. For docking experiments with different ligands, Flare does not need a separate preparation of ligand molecules. The receptor and ligand files can be directly used without any modification. All files are saved in the folder.

Flare is opened and all files directly imported to flare by clicking *"File > Open File"* or directly drag-and-dropping. Receptors should be assigned as protein and ligands as ligand in the pop-up window after importing molecules. Automatic functions easily differentiate in most cases. The "Proteins" window on the bottom part of default Flare layout shows chains, ligands, water and other molecules.

On the protein tab of top menu, clicking *"Protein Prep"* automatically separates receptors and ligands and makes receptor ready for docking if *"Auto-extract ligand"* is selected. In most cases, the ligand to be used in re-docking experiments is detected and extracted automatically. However sometimes the ligand can be assigned as other molecules, so they should be deleted manually. This creates

new protein in protein window, tagged with "P". Other ligands and atoms can be manually deleted if needed. In this example, "2GKE_P" is created. Water and other molecules are deleted.

While preparing the receptor is generally straightforward, managing the complexities of ligands can sometimes prove to be quite challenging. Auto-extract function extracts the ligand if the ligand is detected by program during import and the extracted ligand is the same as bound one. However, in covalent docking, some ligands are bound with a specific reaction that a part of ligand might break off or atoms and bonds might be replaced with another atom and bonds. Since auto-extract only extracts the molecule as it is bonded form, the extracted ligand will not be the desired ligand in those cases. That is why working with the exact version of ligand downloaded from PDB database or a ligand database is the best approach for re-docking experiments, for other experiments it does not a concern. In this example, "ZDP.sdf" and "r_ZDP.sdf" that is created with the script program in 3.2.4.1, added to Flare.

3.4.3. Docking

On the protein tab of top menu, clicking "*3D Pose* > *Dock* > *Covalent*" opens a pop-up window that docking parameters are set. To check covalent binding capability of ligand to receptor and to check if covalent bond is right, calculation method is selected as "Quick". Ligands are selected by clicking with mouse while holding ctrl key and "Selected Ligands" box is checked. Protein tagged with "P" is selected as protein. Chains and molecules are excluded if they do not have any impact.

For grid box, "*Atom Pick*" is selected. By holding the ctrl key, one of the atoms on target residue (CYS 73) is selected. On the home tab of top menu, clicking "*Grow*" automatically selects all atoms in radius given, which is 6 Ångström for this case. Borders of the box are displayed blue. In some cases, Grow will select a proper box. Instead of automatic selection with grow, after selection of atom on target, selecting atoms on border of desired box are manually by clicking atoms

while holding shift key and atoms are kept selected until desired box is created (Figure 3.15). Once docking calculation is started and finished, the same grid box can be used unless any changes are made. The "Existing Grid" check box will automatically be confirmed after protein is selected.

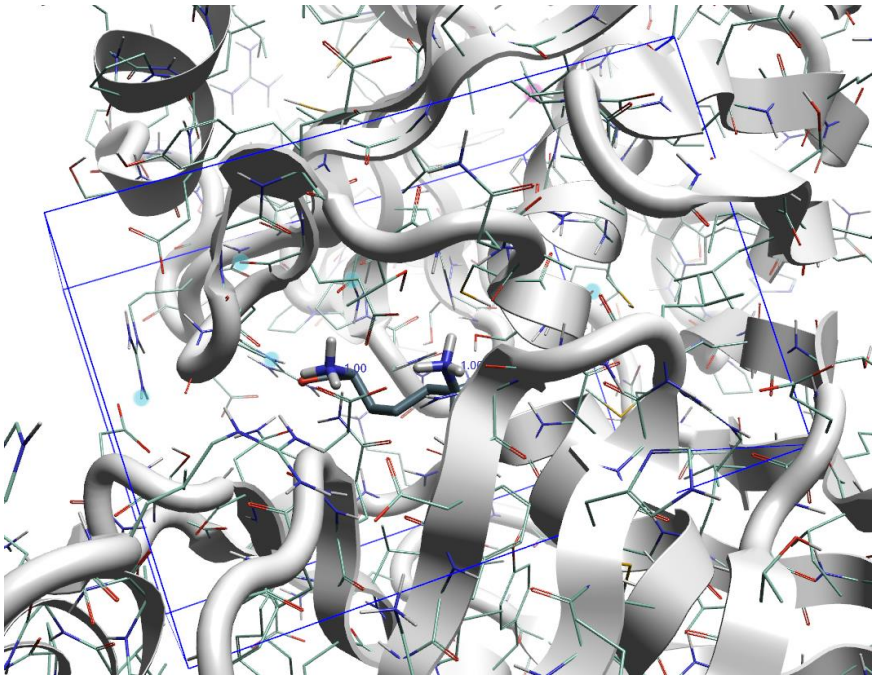

Figure 3.15. Atoms Selected to Create Box. Atom highlighted with blue light are manually selected atoms. By selection or deselecting atoms one by one, desired box is created.

The target residue is selected after defining the grid. In this example, CYS is selected in the dropdown menu and A CYS 73 is selected as residue. If the target residue does not exist in defined grid, an error will be prompted. In Flare covalent docking, there are five amino acid types (Cysteine, Serine, Lysine, Threonine, and Tyrosine) that can be selected as target residue and custom amino acid specifications cannot be added.

The "Show Options" button can be used to experiment on changeable parameters. There is an option to restrict covalent binding to specify warhead using its SMARTS pattern, if the ligand has more than one reactive site. For virtual screening, max poses are reduced to 1 and calculation is started. After

being sure of ligand and receptor binding and covalent bond type, “Normal” as calculation method and 100 as max poses are selected, and ligand docked. Results will be displayed as defined in “Copy Ligands” part on the window before docking. 10 best results of docking results are saved as pdb format by clicking “Export” on ligand right click menu. In this specific example, Flare won't be able to perform docking because it cannot recognize a suitable covalent warhead reaction between ZDP and cysteine.

Flare has a feature to detect covalent warhead of ligand and use the most suitable reaction. The warhead types, warhead detection specifications, and reactions are pre-defined by developers, detailed in their website (https://www.cresset-group.com/about/news/covalent-docking-flare-v7-new-covalent-warheads/). Flare use their own pattern matching syntax called *ATPAT* to write covalent warhead reactions stored in a file with *PAT* format. If a custom warhead and reaction is needed, customer service would help to write *PAT* file. Otherwise, the docking is limited to the pre-definitions which will be enough for many cases.

3.4.4. Forcing Ligand to Bind Target Residue

In some cases, covalent warhead could not be detected by the program. By clicking “Edit a Copy” on ligand right click menu, molecule editing screen is enabled. Based on the ligand and the residue, ligand can be modified to match the warhead reaction of the program. For example, the ligand that proper warhead could not be found can be docked by changing the bond type to double bond on binding site.

In another example, if a reaction that one atom on ligand is eliminated and bond is created with target atom, covalent bond can be forced by changing the eliminated atom element type. Since the atom will be eliminated, any modifications are allowed. However, for all those try-and-error type tests, the actual structure of the ligand should not be changed for proper comparison. As Figure 3.16 shows, even if the structure of molecule is changed, basic structure

remains, and atom element types also remains. After docking, the molecule will be docked exactly as desired.

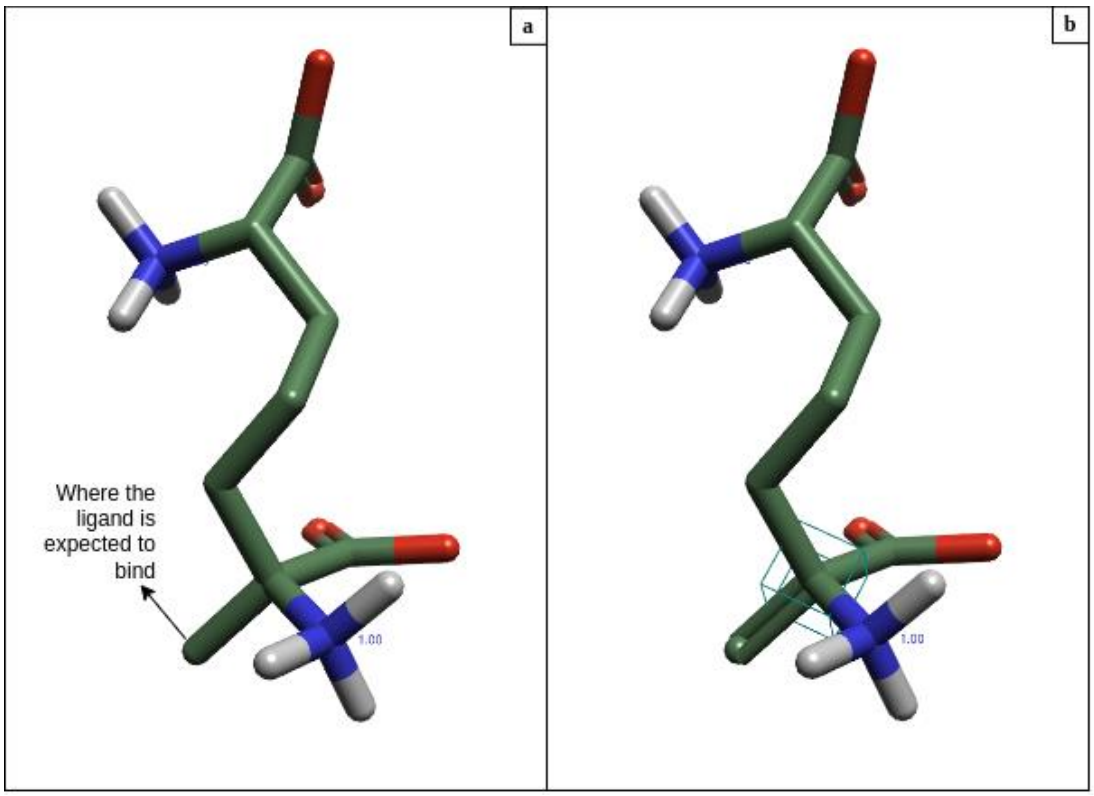

Figure 3.16. Editing ZDP Molecule. (a) Warhead of the original ZDP molecule is not detected by Flare. The place where the binding occurs is pointed. (b) Single bond is changed with double bond. The carbon in the middle of box has a total of 5 bonds. This will not affect calculation since double bond will be broken and a new bond will be created between C on ligand and S on Cysteine.

3.4.5. Automation of Docking Procedure

Besides the Flare docking is mostly automatic, all features of the Flare can be accessed by pyflare (https://gitlab.com/cresset/flare-python-pyflare). Creating a script to automate would rather be easy and can theoretically be used for a mass process of an entire dataset. However, creatin custom warhead reaction by contacting customer service and forcing the existing ligand covalently bond by modifying cannot be automated. After running the automated process, any failed ligands or ligand-receptor combinations can be manually investigated.

3.5. Troubleshoot of Available Covalent Docking Tools

Before the program specific advantages, disadvantages and troubleshooting, all programs are commented in certain categories in Table 3.7. Flare is the easiest but has a lot of limitations. AutoDock4 has a few difficulties in its framework, but it is best for starters. DOCK6 is the most advanced one but also the hardest one.

Table 3.7. User Experience Comparison

	AutoDock4	**DOCK6**	**Flare**
Installation	Easy	Difficult	Easy but license is needed
User interface and terminal commands	Easy to use in terminal, Available as Add-on in many programs, Poor interface in MGLTools	Hard to use, Poor interface in its own visualization tool	Interface based program; user friendly
Ligand - Receptor Preparation	Manual	Manual and carefully prepared	Automatic
File preparation and formats	Easy to handle	Very sensitive to file formats	Automatic and easy
Additional modifications on files	Not required	Required	Not required
Supported target residue	Cys, Ser, Thr, Lys	All residues (except Pro) are suitable as long as they are specified manually	Cys, Ser, Thr, Lys, Tyr
Unsupported target residue	Atom IDs must be specified		No built-in function to use them. Customer service may help
Changeable docking parameters	Changeable	A wide variety of changeable parameters exist	A few parameters are changeable
Automation of covalent docking	Difficult but possible	Difficult but possible	Already automatic
If ligand is not docked ...	Parameters and input files can be tweaked	Parameters and input files can be tweaked	Ligand file can be tweaked to force covalent docking
Ease of error handling	Medium	Difficult to find source of error since it has too many steps	Easy but not possible in some cases since program is not open source
Framework difficulty	Medium	Difficult	Easy

3.5.1. AutoDock4 Troubleshoot

AutoDock4 offers a user-friendly experience with flexible parameters and simplifies the process compared to DOCK6, making it an excellent starting point for learning about docking concepts and parameters. In addition, AutoDock4 maintains relatively high accuracy in covalent docking.

Although AutoDock4 does not need specific modifications on ligand and receptor files, sometimes proteins-ligand complexes with multiple alternative positions may lead to grid center error which caused by different grid centers on different atom maps after autogrid4 command. Files with wrong center can be manually modified in text editor. Before file receptor and ligand preparation, alternative positions of the complex can be removed in PyMOL with *"remove alt b, remove alt c …"* command or alternative position of target residue can be removed specifically, as example *"cmd.remove("resi 170 and resn LYS and alt B")"*.

In some parts of framework, AD4_parameters.dat file should be specified. In some cases, AD4_parameters.dat may not include proper information about ligand atom. For example, boron atom in borylation reaction is not included in parameter file. In this case, required atom features should be searched and added in correct format in parameter file. In this benchmark, B, K and Na atoms needed to be added.

Some residues are automatically aligned with covalent docking script. If the target residue is not supported, an error will be displayed like *"[ERROR] The specified residue types are not supported automatically: HIS. Use either --recsmarts or --recindices."*. Atom IDs of target residue connection atoms should be checked from the original protein-ligand complex that covalent bound ligand is not removed yet. Atom IDs might be changed after modification, and it leads to an error that will be encountered later in docking section.

In this benchmark, there are some complexes that target residue is histidine. Histidine has a ring that involves covalent bonding. Ligand is bonded to N atom

on histidine. There are 2 N atoms on histidine and the one which participates in covalent bonding and the C atom which is not in the middle of 2 N atom is selected as recidices parameter.

3.5.2. UCSF DOCK6 Troubleshoot

DOCK6 offers the greatest flexibility for customizing docking parameters. This extensive control allows for fine-tuning in specific cases, making it a powerful tool. However, this complexity also makes it the most challenging program to master.

DOCK6 is overly sensitive to input file errors. Unfortunately, these errors may not be immediately apparent and could only become obvious after reviewing the docking results. Source of the error can be investigated by controlling all files from the latest to oldest one. After docking runs without an error, a debug file will be created. Debug file and ligand file is opened in PyMOL to check debug is correctly aligned with ligand.

In most cases, debug will not be correct place and output ligand file will be empty. In those cases, the source of the error must be any of input files. The Framework of DOCK6 is the most complicated one and it is likely to make mistakes. After being sure about dummy atoms, those files should be used in dms and sphere generation. If a file without dummy atoms or a file with wrong dummy atoms is used, docking will not be successful.

Randomized ligands can be problematic in DOCK6. Because it is highly sensitive to input ligands, a randomized ligand cannot be docked in some cases. The reason for this is unknown but it can be said that using DOCK6 might lead to hard times for a custom ligand. However, if the connection atom of randomized or custom ligand is aligned with those in target residue, covalent docking will be successful. The randomized ligand script provided in DOCK6 framework randomized ligand atom without changing coordination of two connection atoms.

3.5.3. Cresset Flare Troubleshoot

Flare stands out for its user-friendliness. Its automated functionalities facilitate the docking process, making it ideal for virtual screening and for those dealing with large datasets for virtual screening. However, this automation comes at the cost of limited user control over parameters.

Flare failed to dock in some cases because covalent docking is limited to five target residue types as mentioned in Table 3.8. Also in more complex structures, Flare could detect proper warhead. Due to the strict limitations, the benchmark set was carefully constructed to include only complexes with supported target residues. However, some reaction types have not enough examples in CovPDB database. If the docking fails, the ligand file was modified in order to force covalent docking. As mentioned before in framework section, ligand atoms and/or bonds can be changed, and the proper covalent warhead can be mimicked. However, in this case, the chemical form of docked ligand should be same with original one.

Table 3.8. Failed Complexes and Successful Complexes with Ligand Modification in Flare Covalent Docking

Failed	Target Residue	Reason	Failed	Target Residue	Reason
6SVX	GLU 294	GLU is not supported	**6O8M**	CYS 107	Warhead could not be found
2XGB	GLU 184	GLU is not supported	**2O7R**	SER 169	Warhead could not be found
3HGP	SER 195	Warhead could not be found	**5M4Z**	CYS 189	Warhead could not be found
1JCL	LYS 167	Warhead could not be found	**1YS1**	SER 87	Warhead could not be found
2CXV	HIS 102	HIS is not supported	**1SCW**	SER 62	Warhead could not be found
2H9H	HIS 102	HIS is not supported			

Ligand Forced to Dock	Target Residue	Ligand Forced to Dock	Target Residue	Ligand Forced to Dock	Target Residue
2GKE	CYS 73	**4UIO**	LYS 170	**2CE2**	CYS 32
6H5G	LYS 170	**4A29**	LYS 210	**5U4H**	CYS 116
6BID	CYS 139	**2AH4**	SER 195	**1YQS**	SER 62

Flare warhead detection has a priority order. If the ligand has more than one warhead and covalent docking is performed on a different warhead than expected one, ligand should be tweaked to not recognize that warhead.

3.6. General Comparison and Assessment of Software Success Rates

Figure 3.17 displays the distribution of top-scoring models across docked ligands. It indicates the program that achieved the lowest RMSD value. The pie chart suggests that each program excels in specific scenarios.

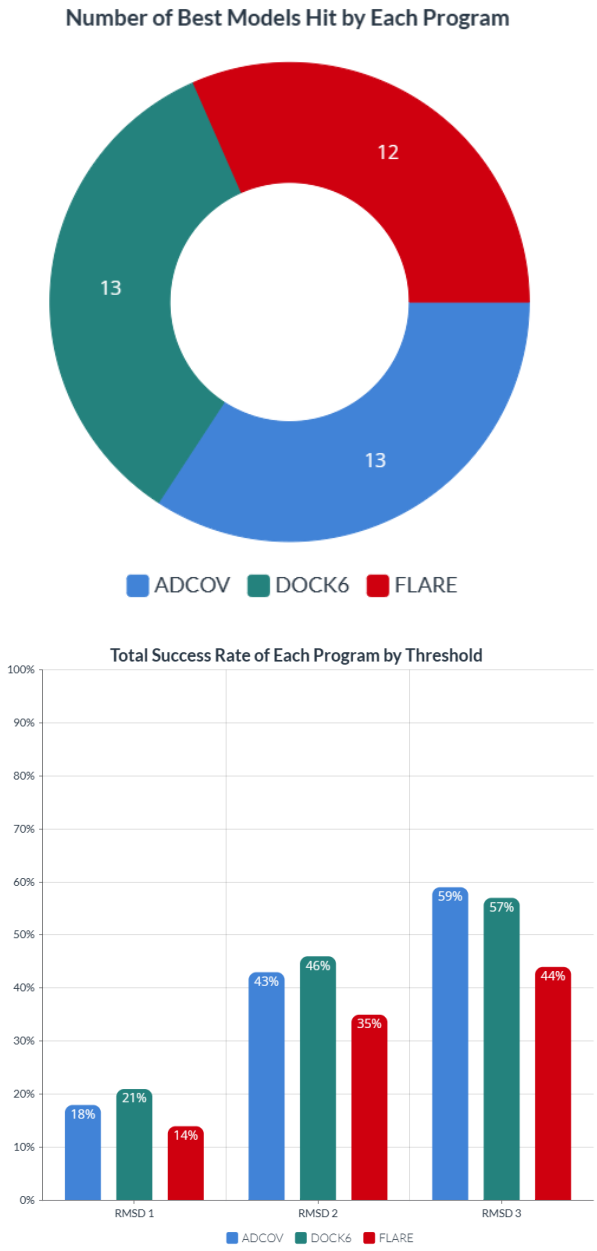

Figure 3.17. Summary of All Results. The chart on the left shows the number of best hits, meaning that the program giving lowest RMSD value among docking results. The chart on the right shows the total success rate of entire docking results of a program.

The chart on the left in Figure 3.17 compares the overall success rates across all docked ligands of different docking programs. While a higher RMSD threshold improves the success rate, there are far fewer docking results with an RMSD below 1 Å. All complexes are listed in Table 3.9 with the information of which program docked it best and what the RMSD value of the best structure is.

Table 3.9. Best Hit of All Complexes in Benchmark Set

PDB ID	BEST MODEL	RMSD	PDB ID	BEST MODEL	RMSD
2GKE	ADCOV	0.264094614	**2H9H**	ADCOV	2.3293017
6SXV	DOCK6	0.512848459	**3QZR**	DOCK6	1.15963001
1PWG	FLARE	0.584695344	**5P9J**	FLARE	0.422754997
5A92	DOCK6	0.546308106	**2AH4**	DOCK6	0.387392853
4UA9	DOCK6	1.930101941	**6UMZ**	DOCK6	0.149561868
1M40	ADCOV	4.010625203	**6O8M**	ADCOV	0.913354988
6HMT	FLARE	1.613595955	**2O7R**	ADCOV	0.30776178
4LUC	FLARE	0.822515342	**3EWW**	ADCOV	0.268332383
6H5G	ADCOV	0.155290142	**5MAE**	FLARE	0.769043925
2XGB	DOCK6	0.443613101	**2CE2**	FLARE	1.091833804
3HGP	DOCK6	0.759673891	**6SFE**	FLARE	0.127097462
6BID	FLARE	0.635632617	**4ONM**	ADCOV	0.098672717
1JCL	DOCK6	0.617234092	**5M4Z**	ADCOV	0.330672211
4UIO	DOCK6	0.835125908	**5U4H**	FLARE	0.167683429
6FM7	FLARE	0.640210715	**1YS1**	DOCK6	0.495218064
6MZ1	ADCOV	0.418265681	**1YQS**	ADCOV	0.841762419
3CR6	FLARE	0.467966304	**1SCW**	ADCOV	0.25510626
4A29	FLARE	0.381086042	**5AQE**	DOCK6	0.452453342
2CXV	ADCOV	2.357357036	**2PFE**	DOCK6	0.491899952

Success rates of AutoDock4 and DOCK6 are close while Flare shows lowest success rate even the figure 3.15 shows the amount of best hits are remarkably close. The main reason for this might be the number of total docking result of Flare. Flare could not dock some samples in benchmark set due to its limitations, resulting with least amount of docking result which might lower the possibility of success in difficult conditions. Table 3.10 shows successful and unsuccessful docking experiments of the entire benchmark with all programs.

Table 3.10. Success of Programs on Protein-Ligand Complexes.

	AutoDock4	DOCK6	Flare		AutoDock4	DOCK6	Flare
2gke				3qzr			
6sxv				5p9j			
1pwg				2ah4			
5a92				6umz			
4ua9				6o8m			
1m40				2o7r			
6hmt				3eww			
4luc				5mae			
6h5g				2ce2			
2xgb				6sfe			
3hgp				4onm			
6bid				5m4z			
1jcl				5u4h			
4uio				1ys1			
6fm7				1yqs			
6mz1				1scw			
3cr6				5aqe			
4a29				2pfe			
2cxv				2qxi			
2h9h				5el1			

* Green indicates that the program successfully docked the ligand and generated at least 10 docked structures. Yellow indicates that the program successfully docked the ligand but generated fewer than 10 unique structures. Red indicates that the program failed to dock.

Radar charts in Figure 3.18 reveal docking program success rates across three RMSD thresholds. Most results fall between RMSD 1 and 2. AutoDock4 excels in Nucleophilic Addition to a Double Bond, Nucleophilic Aliphatic Substitution,

and Nucleophilic Aromatic Substitution. DOCK6 excels in Aziridine Opening, Epoxide Opening, Hemi(thio)acetalization, Hemiaminalization, and Nucleophilic Acyl Substitution. Flare excels in Imine Condensation, Imidazolidinone Opening, and Disulfide Formation. While Borylation reaction samples were docked successfully, none achieved an RMSD threshold below 3. Michael addition and lactone addition show high RMSD in general.

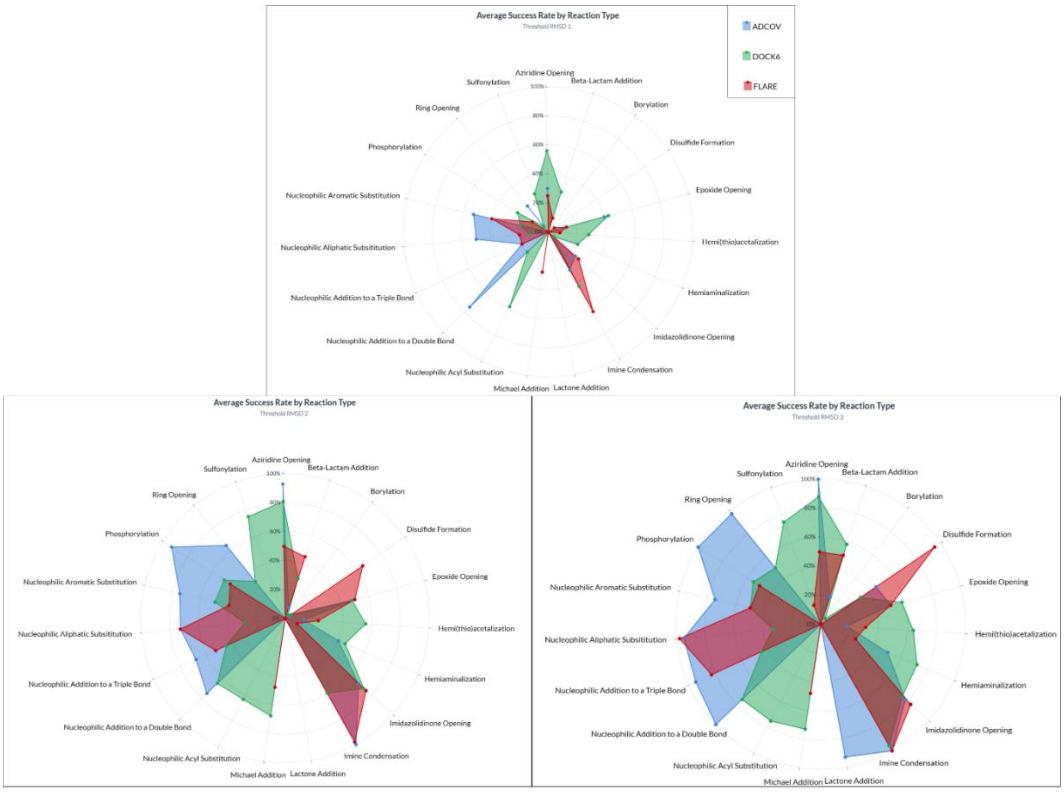

Figure 3.18. Average Success Rate by Reaction Type

Similar results can be obtained when warhead type is considered as in Figure 3.19. However, strengths of each program can be revealed easier with those charts as AutoDock4 and DOCK6 cover separate areas in charts. Flare does not show strength on any warhead type except disulfide warhead. Ligands with disulfide warhead were only docked with Flare.

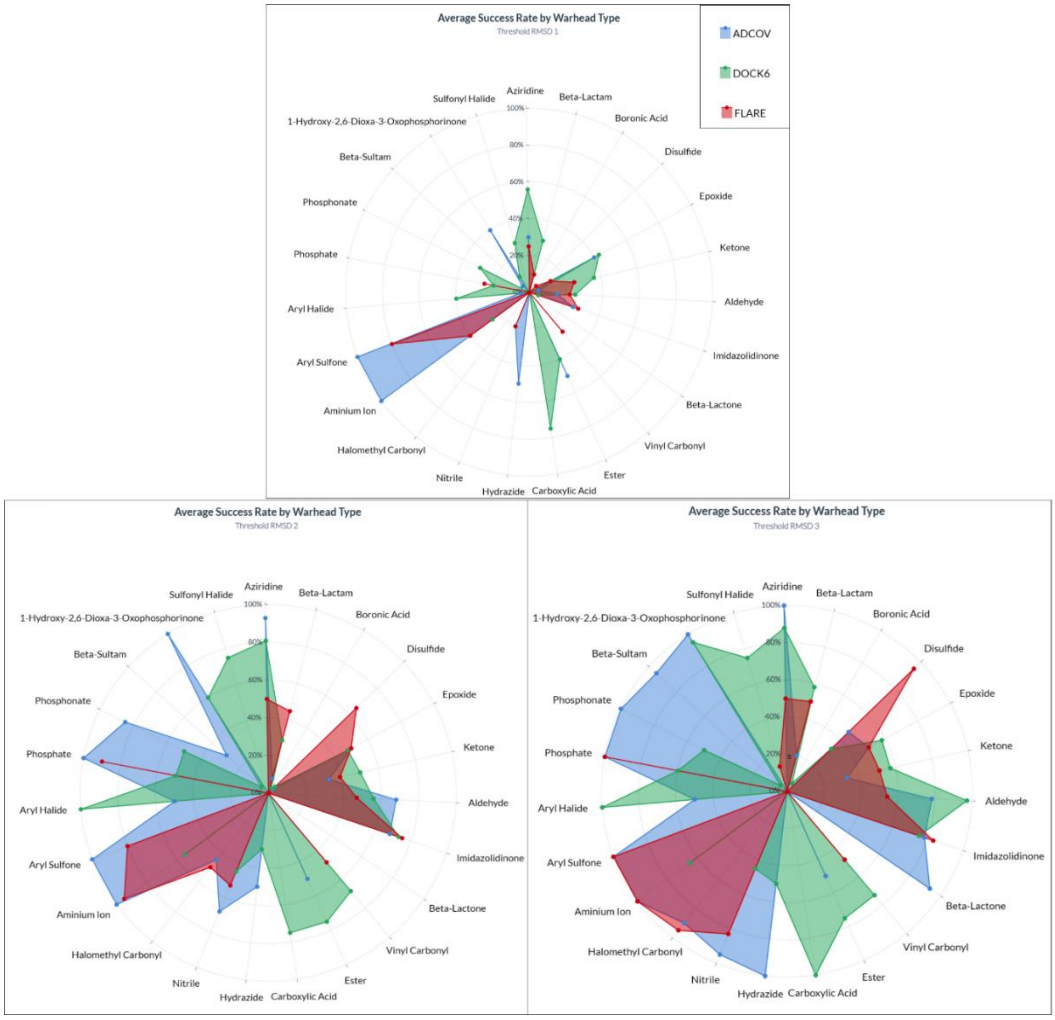

Figure 3.19. Average Success Rate by Warhead Type

The provided graphs in Figure 3.20 demonstrate the correlation between the average success rate over hydrogen bond donor - acceptor of ligand and number of rotatable bonds of ligands. All three programs exhibit success rate of around 50 percent when docking molecules with zero rotatable bonds. However, as the number of rotatable bonds increases, the success rate for each program declines. Same correlation occurs on success and hydrogen bond properties in every program except DOCK6. The reason could be related to DOCK6's calculation methods, which may place less emphasis on the influence of hydrogen bonds in its scoring.

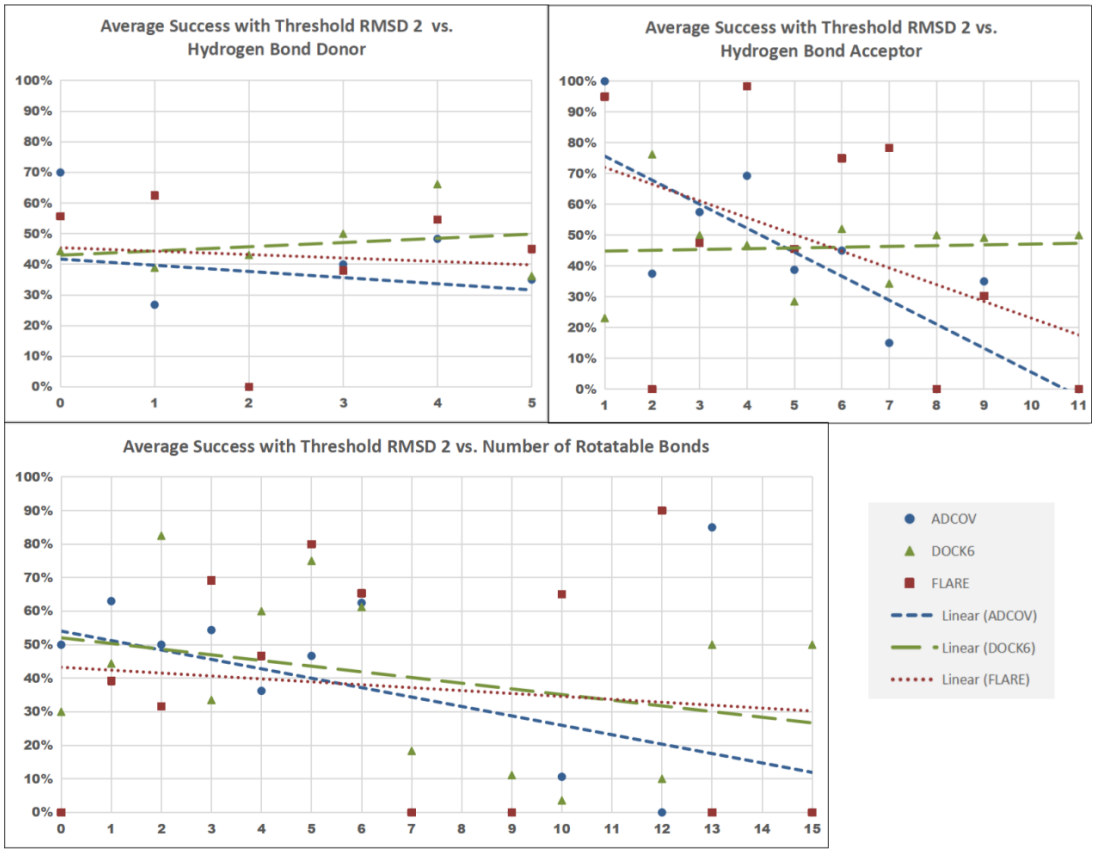

Figure 3.20. Average Success in Different Ligand Properties in RMSD Threshold of 2

The best RMSD value of a category is not always the first structure. First structure is scored as lowest energy in any program. When RMSD values are considered, the first structure is not the best structure in most cases. Table 3.11 shows the count of structure among docked ligands in certain categories.

Table 3.11. Number of Complexes in Categories. Count indicates that the best RMSD of that category is either the first docked ligand or the docked ligand in top 10.

	AutoDock4 Normal	AutoDock4 Randomized	DOCK6 Normal	DOCK6 Randomized	Flare Normal	Flare Randomized
In top 10	35	32	29	32	21	21
First Structure	3	6	9	6	6	6

5A92 is the one that DOCK6 shows its strength (Figure 3.21). One of two rings of ligand molecule (cefotaxime) has β-lactam ring that involves in covalent bonding, β -Lactam addition reaction. In general, each program having a hard time with ring structures and acquiring low RMSD is less likely. As it is shown in Figure 3.21.a, normal ligand docking in DOCK6 generated all top 10 model as RMSD lower than 1 Å. Even though DOCK6 is the most successful program for this case, randomized results give significantly more RMSD value that is still more successful than any other model of other programs. However, this indicates that docking with original ligand, that supposed to be best model already, can create a bias and can impact the docking result.

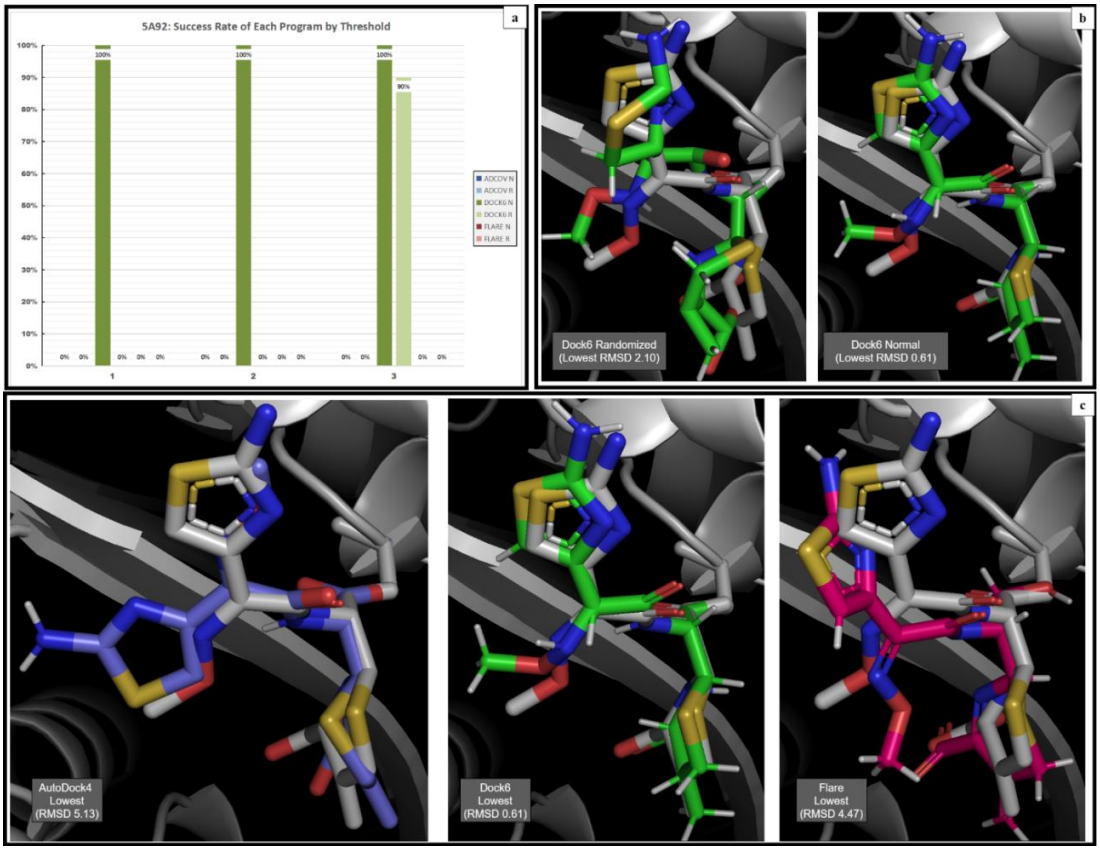

Figure 3.21. Specific Case: 5A92. 5A92 is a hydrolase protein named Beta-lactamase CTX-M-97 with UniprotID E1ANH6 (BLC97_ECOLX). (a) Success rate in top 10 model for each model. (b) Best hit model is generated by DOCK6. Images show models with lowest RMSD for both randomized and normal ligand experiments of DOCK6. (c) Images show models with lowest RMSD for each program.

2CE2 is the one that Flare shows its strength (Figure 3.22). Ligand name is N,N'-dimethyl-N-(acetyl)-N'-(7-nitrobenz-2-oxa-1,3-diazol-4-yl)ethylenediamine with HET code of XY2. It has nitrobenzene that increases complexity of docking. The binding site of 2CE2 is located on the outside of the structure. The structures with exposed binding sites can lead to a docking result with a split model which means most parts on ligand tend to move away from structure, as it can be seen in Figure 3.22.c DOCK6 result. As in this specific case, Flare generates good

models on exposed binding sites while DOCK6 needs specific parameter changes or stricter and smaller grid box.

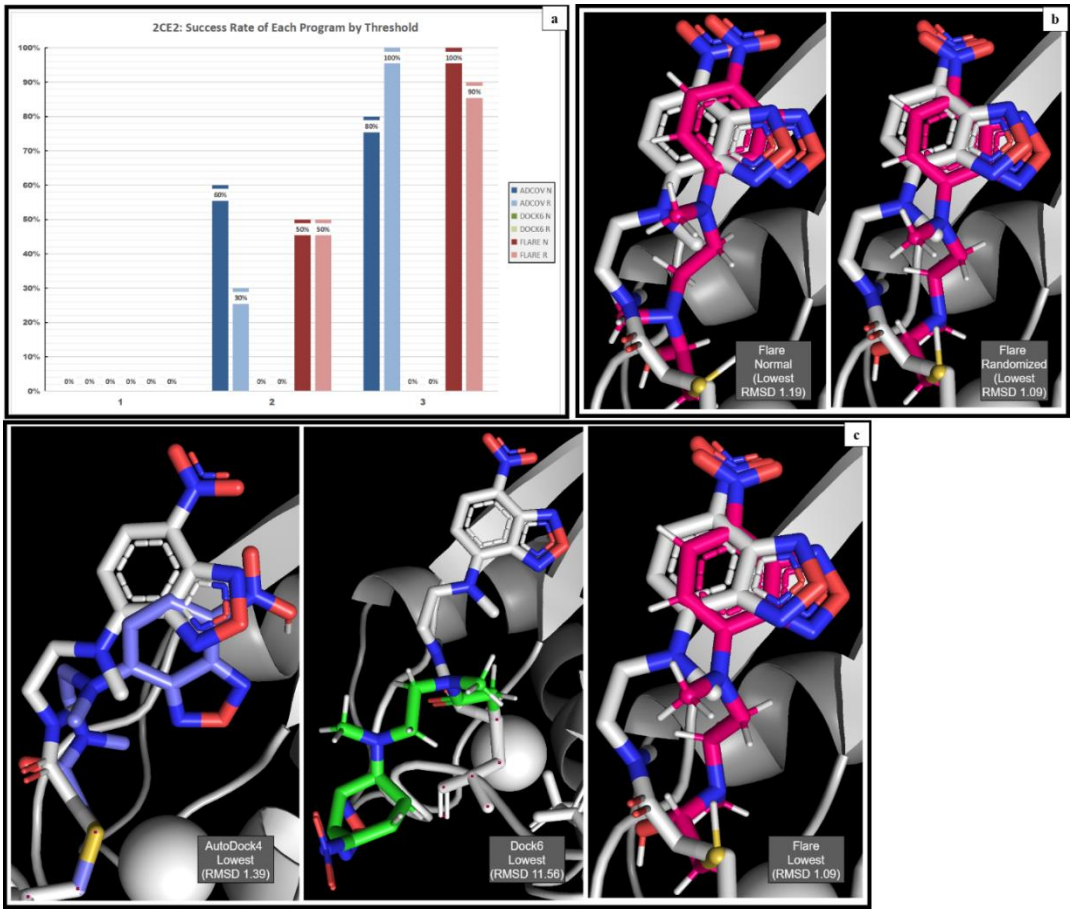

Figure 3.22. Specific Case: 2CE2. 2CE2 is a signaling protein named GTPase HRas with UniprotID P01112 (RASH_HUMAN). (a) Success rate in top 10 model for each model. (b) Best hit model is generated by Flare. Images show models with lowest RMSD for both randomized and normal ligand experiments of Flare. (c) Images show models with lowest RMSD for each program.

1M40 is the one that Flare shows its strength (Figure 3.22). Ligand name is LP06 with HET code of CB4. CB4 has a unique part among every ligand in benchmark set, it has boron. Boron atom made all frameworks difficult in different parts for each program. Python ligand randomization codes do not work for this ligand but the minimalization tool of Marvin Sketch can successfully randomize any ligand. In this specific case, along with exposed binding site, most of the residues

around binding site stand on loops of protein. This might be the reason for low accuracy in terms of RMSD.

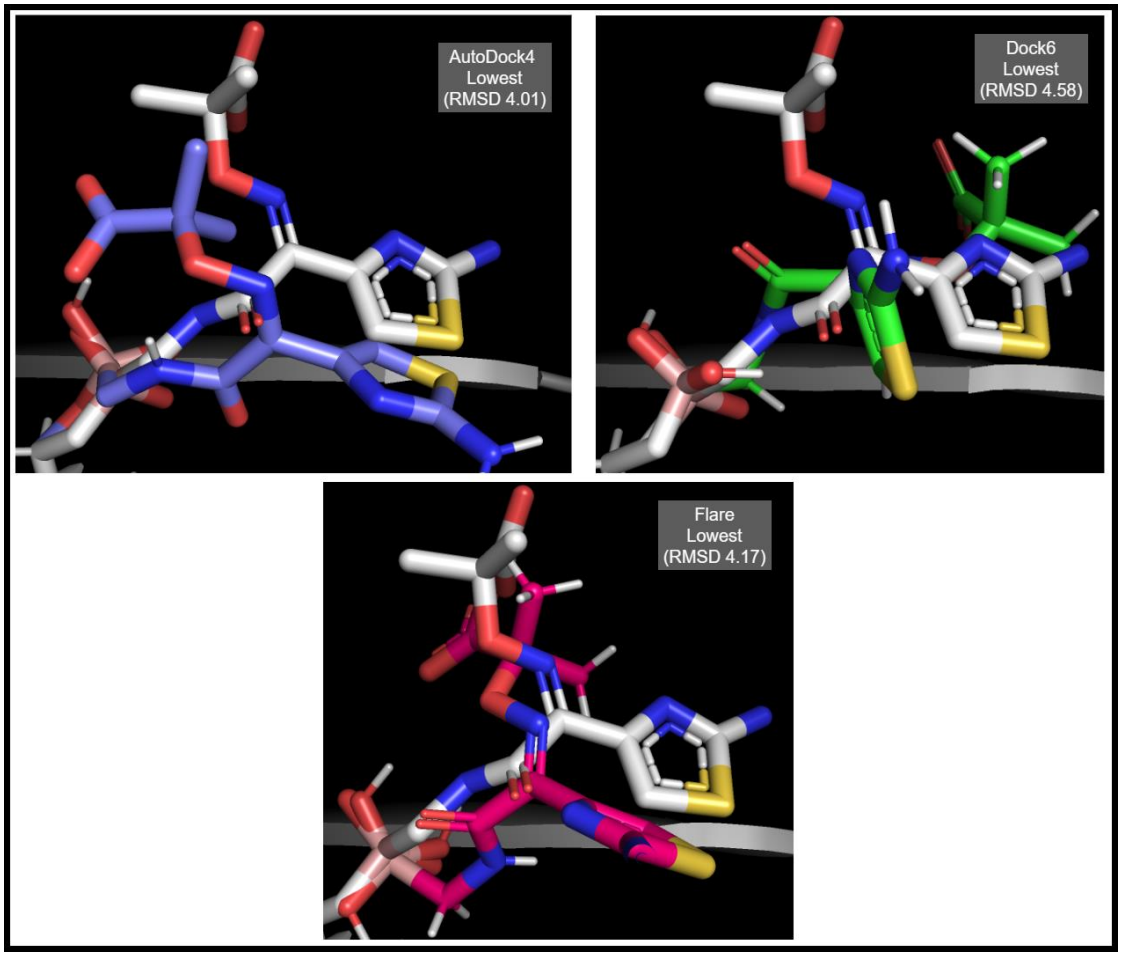

Figure 3.23. Specific Case: 1M40. 1M40 is a hydrolase protein named Beta-lactamase TEM with UniprotID P62593 (BLAT_ECOLX). Images show models with the lowest RMSD for each program.

4. CONCLUSIONS

Molecular docking is a widely used method in drug design. Covalent docking, while based on similar principles, is a more complex process due to certain limitations compared to molecular docking. All these complexities and limitations make covalent docking a less studied and known field. However, covalently binding inhibitors discovered and designed through covalent docking have the potential to be more effective and permanent solutions as drugs due to the irreversibility of binding and higher affinity compared to non-covalent protein inhibitors. Additionally, with covalent docking, the covalent warhead on the ligand can target rare and unprotected regions on the target protein and lead to the development of a highly selective inhibitor. Covalent inhibitors have higher potential compared to non-covalent inhibitors and are therefore much more likely to act on proteins with difficult binding cleavages.

Different software and applications used in covalent docking methods can give different results. To ensure the accuracy of these results, the diversity of programs should be changed, and an optimal and more understandable procedure should be developed to ensure better results in future studies in the field of covalent docking. There is no standardized procedure to follow for covalent docking in the literature, and there is not enough information on the success of different programs in the docking method. This thesis provides comprehensive and comparative information on covalent docking programs, contributes to making the method more useful and optimal, and is a potential guide for future studies in this field.

REFERENCES

1. Bauer, Matthias R., and Mark D. Mackey. 2019. "Electrostatic Complementarity as a Fast and Effective Tool to Optimize Binding and Selectivity of Protein-Ligand Complexes." Journal of Medicinal Chemistry 62 (6): 3036–50. https://doi.org/10.1021/ACS.JMEDCHEM.8B01925/SUPPL_FILE/JM8B01925_SI_002.ZIP.
2. Benfield, Cory, Ian Stapleton Cordasco, and Nate Prewitt. 2011. "Requests 2.31.0 / 22 May 2023 Release."
3. Benjin, Xu, and Liu Ling. 2020. "Developments, Applications, and Prospects of Cryo- electron Microscopy." Protein Science : A Publication of the Protein Society 29 (4): 872. https://doi.org/10.1002/PRO.3805.
4. Berman, Helen M., John Westbrook, Zukang Feng, Gary Gilliland, T. N. Bhat, Helge Weissig, Ilya N. Shindyalov, and Philip E. Bourne. 2000. "The Protein Data Bank." Nucleic Acids Research 28 (1): 235–42. https://doi.org/10.1093/NAR/28.1.235.
5. Bianco, Giulia, Stefano Forli, David S. Goodsell, and Arthur J. Olson. 2016. "Covalent Docking Using Autodock: Two-Point Attractor and Flexible Side Chain Methods." Protein Science 25 (1): 295–301. https://doi.org/10.1002/PRO.2733.
6. Bronowska, Agnieszka K. 2011. "Thermodynamics of Ligand-Protein Interactions: Implications for Molecular Design." In Thermodynamics, edited by Juan Carlos Moreno-Pirajan. Rijeka: IntechOpen. https://doi.org/10.5772/19447.
7. Cheeseright T., Mackey M., Rose S., and A. Vinter. 2006. "FlareTM, 7, Cresset®, Litlington, Cambridgeshire, UK." Molecular Field Extrema as Descriptors of Biological Activity: Definition and Validation J. Chem. Inf. Model.
8. Cheeseright, Tim, Mark Mackey, Sally Rose, and Andy Vinter. 2006. "Molecular Field Extreme as Descriptors of Biological Activity:

Definition and Validation." Journal of Chemical Information and Modeling 46 (2): 665–76. https://doi.org/10.1021/CI050357S/SUPPL_FILE/CI050357SSI20051111_062241.PDF.

9. ChemAxon. 2023. "Marvin Was Used for Drawing, Displaying and Characterizing Chemical Structures, Substructures and Reactions, Marvin 24.1.2. Http://Www.Chemaxon.Com."
10. DesJarlais, Renee L., George L. Seibel, Irwin D. Kuntz, Robert P. Sheridan, R. Venkataraghavan, and J. Scott Dixon. 1988. "Using Shape Complementarity as an Initial Screen in Designing Ligands for a Receptor Binding Site of Known Three-Dimensional Structure." Journal of Medicinal Chemistry 31 (4): 722–29. https://doi.org/10.1021/JM00399A006/ASSET/JM00399A006.FP.PNG_V03.
11. Du, H., Gao, J., Weng, G., Ding, J., Chai, X., Pang, J., Kang, Y., Li, D., Cao, D., & Hou, T. (2021). CovalentInDB: a comprehensive database facilitating the discovery of covalent inhibitors. Nucleic Acids Research, 49(D1), D1122. https://doi.org/10.1093/NAR/GKAA876
12. Du, Xing, Yi Li, Yuan Ling Xia, Shi Meng Ai, Jing Liang, Peng Sang, Xing Lai Ji, and Shu Qun Liu. 2016. "Insights into Protein–Ligand Interactions: Mechanisms, Models, and Methods." International Journal of Molecular Sciences 17 (2). https://doi.org/10.3390/IJMS17020144.
13. Freddolino, Peter, John Stone, and Klaus Schulten. n.d. "Grid Generation and Matching for Small Molecule Docking." Accessed April 19, 2024. http://www.ks.uiuc.edu/Research/vmd/projects/ece498/.
14. Gao, M., Moumbock, A. F. A., Qaseem, A., Xu, Q., & Günther, S. (2022). CovPDB: a high-resolution coverage of the covalent protein–ligand interactome. Nucleic Acids Research, 50(D1), D445. https://doi.org/10.1093/NAR/GKAB868

15. Grinter, Sam Z, and Xiaoqin Zou. 2014. “Challenges, Applications, and Recent Advances of Protein-Ligand Docking in Structure-Based Drug Design.” Molecules 19: 10150–76. https://doi.org/10.3390/molecules190710150.
16. Huang, F., Han, X., Xiao, X., & Zhou, J. (2022). Covalent Warheads Targeting Cysteine Residue: The Promising Approach in Drug Development. https://doi.org/10.3390/molecules27227728
17. Huey, Ruth, Garrett M. Morris, Arthur J. Olson, and David S. Goodsell. 2007. “A Semiempirical Free Energy Force Field with Charge-Based Desolvation.” Journal of Computational Chemistry 28 (6): 1145–52. https://doi.org/10.1002/JCC.20634.
18. Hunter, J D. 2007. “Matplotlib: A 2D Graphics Environment.” Computing in Science & Engineering 9 (3): 90–95. https://doi.org/10.1109/MCSE.2007.55.
19. Irwin, John J., Khanh G. Tang, Jennifer Young, Chinzorig Dandarchuluun, Benjamin R. Wong, Munkhzul Khurelbaatar, Yurii S. Moroz, John Mayfield, and Roger A. Sayle. 2020b. “ZINC20 - A Free Ultralarge-Scale Chemical Database for Ligand Discovery.” Journal of Chemical Information and Modeling 60 (12): 6065–73. https://doi.org/10.1021/ACS.JCIM.0C00675/ASSET/IMAGES/LARGE/CI0C00675_0007.JPEG.
20. Kuhn, Maximilian, Stuart Firth-Clark, Paolo Tosco, Antonia S.J.S. Mey, Mark MacKey, and Julien Michel. 2020. “Assessment of Binding Affinity via Alchemical Free-Energy Calculations.” Journal of Chemical Information and Modeling 60 (6): 3120–30. https://doi.org/10.1021/ACS.JCIM.0C00165/SUPPL_FILE/CI0C00165_SI_002.ZIP.
21. Kuntz, Irwin D., Jeffrey M. Blaney, Stuart J. Oatley, Robert Langridge, and Thomas E. Ferrin. 1982. “A Geometric Approach to Macromolecule-

Ligand Interactions." Journal of Molecular Biology 161 (2): 269–88. https://doi.org/10.1016/0022-2836(82)90153-X.

22. Liu, Tiqing, Yuhmei Lin, Xin Wen, Robert N. Jorissen, and Michael K. Gilson. 2007. "BindingDB: A Web-Accessible Database of Experimentally Determined Protein–Ligand Binding Affinities." Nucleic Acids Research 35 (Database issue): D198. https://doi.org/10.1093/NAR/GKL999.
23. Mckinney, Wes. 2010. "Data Structures for Statistical Computing in Python." PROC. OF THE 9th PYTHON IN SCIENCE CONF. https://www.researchgate.net/publication/265001241.
24. Meng, Xuan-Yu, Hong-Xing Zhang, Mihaly Mezei, and Meng Cui. 2011. "Molecular Docking: A Powerful Approach for Structure-Based Drug Discovery." Curr Comput Aided Drug Des. 2011 Jun 1; 7(2): 146–157. https://doi.org/10.2174/157340911795677602.
25. Meng, Elaine C., Brian K. Shoichet, and Irwin D. Kuntz. 1992. "Automated Docking with Grid-Based Energy Evaluation." Journal of Computational Chemistry 13 (4): 505–24. https://doi.org/10.1002/JCC.540130412.
26. Morris, G. M., Ruth, H., Lindstrom, W., Sanner, M. F., Belew, R. K., Goodsell, D. S., & Olson, A. J. (2009). AutoDock4 and AutoDockTools4: Automated docking with selective receptor flexibility. Journal of Computational Chemistry, 30(16), 2785–2791. https://doi.org/10.1002/JCC.21256
27. Morris, Garrett M, David S Goodsell, Robert S Halliday, Ruth Huey, William E Hart, Richard K Belew, and Arthur J Olson. 1998. "Automated Docking Using a Lamarckian Genetic Algorithm and an Empirical Binding Free Energy Function." Journal of Computational Chemistry 19 (14): 16391662. https://doi.org/10.1002/(SICI)1096-987X(19981115)19:14<1639::AID-JCC10>3.0.CO;2-B.
28. Morris, Garrett M., Huey Ruth, William Lindstrom, Michel F. Sanner, Richard K. Belew, David S. Goodsell, and Arthur J. Olson. 2009a.

"Software News and Updates AutoDock4 and AutoDockTools4: Automated Docking with Selective Receptor Flexibility." Journal of Computational Chemistry 30 (16): 2785–91. https://doi.org/10.1002/JCC.21256.

29. O'Boyle, Noel M, Michael Banck, Craig A James, Chris Morley, Tim Vandermeersch, and Geoffrey R Hutchison. 2011. "Open Babel: An Open Chemical Toolbox." Journal of Cheminformatics 3 (1): 33. https://doi.org/10.1186/1758-2946-3-33.
30. Perozzo, Remo, Gerd Folkers, and Leonardo Scapozza. 2004. "Thermodynamics of Protein–Ligand Interactions: History, Presence, and Future Aspects." Journal of Receptors and Signal Transduction 24 (1–2): 1–52. https://doi.org/10.1081/RRS-120037896.
31. Pettersen, Eric F., Thomas D. Goddard, Conrad C. Huang, Gregory S. Couch, Daniel M. Greenblatt, Elaine C. Meng, and Thomas E. Ferrin. 2004. "UCSF Chimera—A Visualization System for Exploratory Research and Analysis. Http://Www.Cgl.Ucsf.Edu/Chimera." Journal of Computational Chemistry 25 (13): 1605–12. https://doi.org/10.1002/jcc.20084.
32. Raschka, Sebastian. 2017. "BioPandas: Working with Molecular Structures in Pandas DataFrames." The Journal of Open Source Software 2 (14): 279. https://doi.org/10.21105/joss.00279.
33. Richardson, Leonard. 2004. "Beautiful Soup 4, Code.Launchpad.Net/Beautifulsoup/."
34. Scarpino, Andrea, György G. Ferenczy, and György M. Keserü. 2018. "Comparative Evaluation of Covalent Docking Tools." Journal of Chemical Information and Modeling 58 (7): 1441–58. https://doi.org/10.1021/ACS.JCIM.8B00228/SUPPL_FILE/CI8B00228_SI_001.PDF.
35. Schrödinger LLC. 2015a. "The AxPyMOL Molecular Graphics Plugin for Microsoft PowerPoint, Version~1.8."

36. Schrödinger LLC. 2015b. “The JyMOL Molecular Graphics Development Component, Version~1.8.”
37. Schrödinger LLC. 2015c. “The PyMOL Molecular Graphics System, Version~1.8.”
38. Schrödinger LLC. 2024. "Maestro, Schrödinger"
39. “Calculate Root-Mean-Square Deviation (RMSD) of Two Molecules Using Rotation, GitHub, Http://Github.Com/Charnley/Rmsd.”
40. “Python Software Foundation. Python Language Reference, Version 3.12.2. Http://Www.Python.Org.”
41. “RDKit: Open-Source Cheminformatics. 2023_09_6 (Q3 2023) Release. Https://Www.Rdkit.Org.”