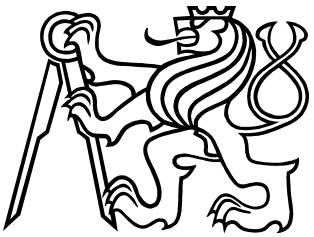


Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Cybernetics

Computational Detection of Pathways for Flexible Ligands in Protein Structures

Diploma Thesis

Bc. Tom Jankovec

Thesis supervisor

Ing. Vojtěch Vonásek Ph.D.

Prague, May 2018

MASTER'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Jankovec Tom** Personal ID number: **420257**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Cybernetics and Robotics**

Branch of study: **Robotics**

II. Master's thesis details

Master's thesis title in English:

Computational Detection of Pathways for Flexible Ligands in Protein Structures

Master's thesis title in Czech:

Výpočet cest flexibilních ligandů v proteinových strukturách

Guidelines:

1. Get familiar with ligand pathway computation [1] and tunnel detection in protein structures [2].
2. Get familiar with sampling-based motion planning [3], namely RRT [4] and PRM [5] and their extensions (e.g. ML-RRT [8]).
3. Design a model for flexible ligands [6] either as a multi-link kinematic chain or utilize an existing library (e.g., Vina Autodock) to realize the model. Inputs to the model are the torsion angles of the ligand, output is the 3D position of all atoms.
4. Utilize Vina Autodock to calculate potential energy of the protein/ligand system.
5. Design and implement a sampling-based motion planning method to find pathways for the flexible ligand between an active site in the protein and the outer space. Detect the pathways in static (single) frames.
6. Compare the approach from 5) with MoMa-LigPath [7] tool on a a set of proteins/ligands.

All necessary data (protein/ligand structures, location of active sites) will be provided by the supervisor in the form of PDB or PDBQT files. Access to MoMa-LigPath will be provided by the supervisor.

Bibliography / sources:

[1] Cortés, Juan, et al. "A path planning approach for computing large-amplitude motions of flexible molecules." Bioinformatics 21.suppl 1 (2005): i116-i125.
[2] Chovancova, Eva, et al. "CAVER 3.0: a tool for the analysis of transport pathways in dynamic protein structures." (2012): e1002708.
[3] LaValle, Steven M - Planning algorithms - Cambridge university press, 2006.
[4] LaValle, Steven M., and James J. Kuffner Jr. - Rapidly-exploring random trees: Progress and prospects - (2000).
[5] Kavraki, Lydia E., et al. - Probabilistic roadmaps for path planning in high-dimensional configuration spaces - Robotics and Automation, IEEE Transactions on 12.4 (1996): 566-580.
[6] Introduction to Protein Structure, Carl Branden, John Tooze, Garland Science; 2 edition (January 3, 1999)
[7] Cortés, J., Siméon, T., Ruiz de Angulo, V., Guieysse, D., Remaud-Siméon, M. and Tran, V., 2005. A path planning approach for computing large-amplitude motions of flexible molecules. Bioinformatics, 21(suppl_1)
[8] Cortés, Juan, Léonard Jaillet, and Thierry Siméon. "Disassembly path planning for complex articulated objects." IEEE Transactions on Robotics 24, no. 2 (2008): 475-481.

Name and workplace of master's thesis supervisor:

Ing. Vojtěch Vonásek, Ph.D., Multi-robot Systems, FEE

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **09.01.2018** Deadline for master's thesis submission: **25.05.2018**

Assignment valid until: **30.09.2019**

Ing. Vojtěch Vonásek, Ph.D. Supervisor's signature	doc. Ing. Tomáš Svoboda, Ph.D. Head of department's signature	prof. Ing. Pavel Ripka, CSc. Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature

Author statement for undergraduate thesis
I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, Date ....
Signature

Acknowledments
I would like to express my immense gratitude to my thesis advisor, Ing. Vojtěch Vonásek Ph.D., for his input, endless enthusiasm and for helping me throughout my entire academic career. I would also like to thank my family for their support throughout my studies, and for never giving up on me.

We would like to thank Mgr. Ondřej Vávra from Loschmidt Laboratories, Faculty of Science, Masaryk University, for preparing the dataset.

Access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum provided under the programme "Projects of Large Research, Development, and Innovations Infrastructures" (CESNET LM2015042), is greatly appreciated.

Abstract

Characteristics of protein molecules and other biochemical compounds are investigated via protein tunnels or ligand pathways. To ensure trustworthiness of the results, it is imperative that both the shape and the flexibility of ligand molecules are taken into account. A majority of the currently utilized software tools do not allow this. Motion planning for mobile robots is one of the approaches enabling the use of these characteristics. This thesis presents a novel algorithm for ligand pathways detection, which takes into account the shape, flexibility and the intramolecular energy of ligand molecules. To speed up the algorithm, a precomputed knowledge about free spaces in the protein molecules is used. The algorithm also enables the computation of potential energy of the intermolecular receptor–ligand system for the generated pathways. The presented algorithm was tested on a real dataset of several hundred conformations and tunnels of three protein receptors and seven ligand molecules. The algorithm's performance was compared to the state–of–the–art tool MoMA–LigPath.

Abstrakt

Charakteristiky proteinových molekul a jiných biochemických sloučenin jsou zkoumány skrze proteinové tunely, případně ligandové cesty. Pro zajištění věrohodnosti výsledků je nutné při hledáni uvažovat tvar i flexibilitu ligandových molekul, což v současnosti nejčastěji používané softwarové nástroje neumožňují. Jedním z přístupů, schopných tyto podmínky zajistit, je plánování pohybů pro mobilní roboty. V této práci je představen nový algoritmus pro výpočet ligandových cest, který uvažuje tvar, flexibilitu a potenciální energii ligandu. Pro zrychlení algoritmu je využívána předem dostupná znalost volného místa v receptorových molekulách. Algoritmus umožňuje výpočet intermolekulární potenciální energie systému receptor–ligand pro vygenerované ligandové cesty. Prezentovaný algoritmus byl otestován na reálném datasetu několika set konformací a tunelů třech proteinových receptorů a sedmi ligandových molekul a porovnán se state–of–the–art nástrojem MoMA–LigPath.

Keywords
Motion planning, protein molecules, ligand pathways, protein tunnels.

Klíčová slova
Plánování pohybů, proteinové molekuly, ligandové cesty, proteinové tunely.

Contents

List of Algorithms

List of Figures

1 Introduction

This chapter introduces the field of protein molecules research to the reader, describes the task of protein tunnel detection and path planning and elucidates the problem at hand.

1.1 Motivation and Goals of this Thesis

Proteins represent a very large and diverse group of biomolecules. Some proteins possess the capability to interact with the environment around them to the extent of, e.g., catalyzing chemical reactions or self-replicating. Other proteins can serve as structural elements, signal receptors inside organisms or even as transport molecules. These interactions can be carried out between the receptor protein molecules, and smaller ligand molecules. Protein tunnel traversability problem consists of determining whether a ligand molecule can reach a receptor's *active site* (explained later). The influencing factors include not only the molecule's chemical composition, but also physical structures and the locations said active sites. Applications span such areas as drug design or biochemical processes research.

Molecular dynamics simulations[1] can be employed to solve this task. They are, however, very computationally demanding and require extensive resources.

Geometric molecular analysis has been successfully employed in the past to solve this problem. It focuses on detecting protein tunnels (explained later) and evaluating their characteristics, to determine whether they can be traversed by a supplied ligand molecule. One of its main disadvantages is the fact that the a majority of the currently available tools analyze the receptor molecules using a spherical probe, thus disregarding the shape and flexibility of the ligand molecule. The resulting solutions are therefore approximate.

This thesis focuses on tunnel pathways detection. We aim to develop a software tool which, when supplied with the structures of the receptor molecule, ligand molecule and the respective protein tunnel containing the active site, predicts whether ligand tunnel traversability is possible from the geometrical point of view.

[1] Computer simulations of molecules through a selected time period. Analytical solutions are often impossible, leading to numerical solutions which diverge from the real scenarios as time progresses.

We approach the problem using state–of–the–art motion planning algorithms and aim to enable pathways predictions for flexible ligand molecules while taking into consideration ligand potential energy functions. We expand on our previous work, which explored the possibilities of protein tunnel[2] detection using a specifically tailored modification of the RRT algorithm.

Chapter 1 gives the reader an insight into the field of protein biochemistry and presents the problem from the biochemical standpoint. It establishes the terminology necessary to describe the task at hand.

Chapter 2 presents the problem from the robotic paradigm and discusses possible approaches of tackling this issue from a theoretical point of view. It presents algorithms from the field of robotics suitable for tackling this issue and discusses their applicability.

Chapter 3 presents the chosen algorithm in great detail, discusses implementation details and presents the final solution.

Chapter 4 describes the testing methodology and conducted experiments.

Chapter 5 discusses the results of the presented experiments and their implications.

Finally, **Chapter 6** contains the conclusion and **Addenda** describing some minute details and technicalities of the final implementation.

1.2 Protein Molecules

Protein molecules are large biomolecules, specifically long polymers of amino acids, which distinguish themselves from other molecules by a high degree of complexity and organization. See *Figure 1.1* for an example of a protein molecule.

The structure of protein molecules is made up of amino acids, which are organic compounds consisting mainly of carbon and hydrogen atoms. They represent the smallest protein building blocks. Each protein has its characteristic sequence of aminoacids, and also a highly unique and complex structure, which arises from this sequence [30]. The size of protein molecules can vary immensely, from 20 amino acids in the smallest protein[3] in the world (Trp-Cage Miniprotein [33]), to 27000 amino acids in the largest protein

Figure 1.1: An example of a protein molecule, the 2CI2 molecule, located in barley seeds. Protein subunits (tertiary structures, as explained later) are denoted in different colors. PDB file courtesy of [3].

[2]Protein tunnels are voids inside the receptor structure, which originate at a specific location in the molecule and reach the molecular surface, as explained later.

[3]Proteins with less than 30 or 40 amino acids are considered to be peptides, rather than proteins, according to some literature sources

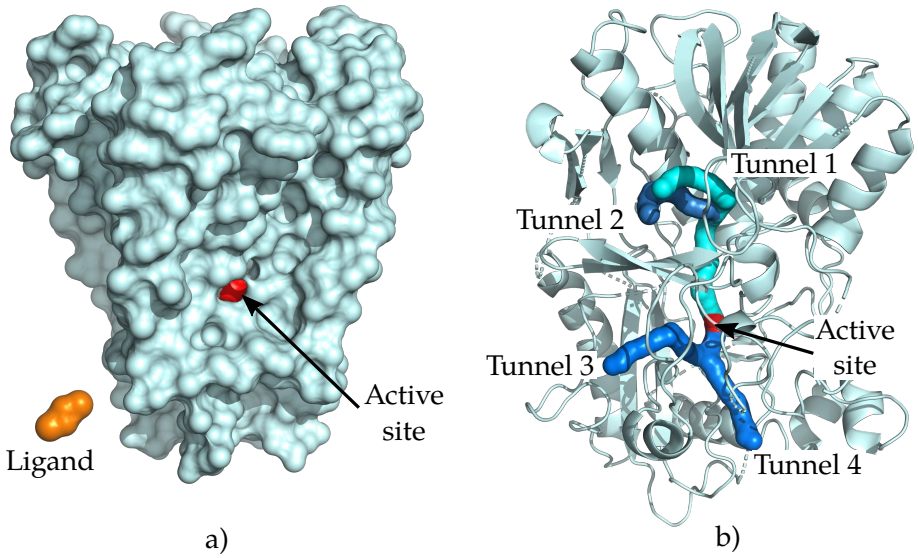


Figure 1.2: a) *Protein active site illustration inside the 1BL8 protein molecule. The active site, located inside a molecular cavity, is denoted in red, whereas the computed protein surface is denoted in gray. A ligand molecule example is denoted in orange.* b) *An example of protein tunnels, originating at an active site. Individual tunnels are colored in different shades of blue. PDB files courtesy of [3].*

in the world, Titin [39]. The chemical properties of protein molecules are determined by the constituent atoms of their amino acids and their relative spatial positions [30].

1.2.1 Protein Active Sites

A very important characteristic of a protein molecule is the presence of its so–called *active sites*. These are locations inside the protein structure containing particular arrangements of atoms, which can repeatedly catalyze chemical reactions with other molecules. The presence and locations of these active sites can greatly affect the characteristics of the respective molecule, and are therefore an intensively studied topic in the field of biochemistry. An example of an active site is given in *Figure 1.2*.

Active sites are usually located deep inside the protein molecules, near void spaces. Void spaces connected to the surface of the molecule are called *protein tunnels*. An example of protein tunnels is presented in *Figure 1.2.b*. The characteristics of these tunnels, mainly the width of the tunnel's bottleneck (the narrowest spot), the length and curvature of the tunnel and the surrounding amino acid residues helps to determine the properties of the protein molecule from the ligand traversability standpoint. See *Figure 1.3* for illustrations.

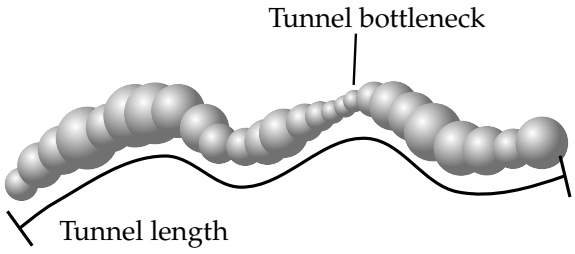


Figure 1.3: Protein tunnel diagram, illustrating tunnel bottleneck (narrowest part) and tunnel length.

Tunnel analysis helps to determine whether the two molecules can react and is therefore a very actively studied topic, with numerous software tools being proposed lately (see *Section 3.2*). This analysis has applications in protein engineering, consisting of changing selected properties of a protein molecule. This can be achieved by altering the amino acids forming its protein tunnels, significantly changing ligand traversability for the respective tunnels. Other examples are available in a recent survey [24].

1.2.2 Spatial Protein Arrangement

As stated previously, the structure of a protein molecule, as well as its function, is determined by the sequence of its amino acids. These amino acids form monomer units in long polymer chains which can be thousands of units long. Although these chains are in essence monodimensional, they can give rise to intricate and complex spatial molecular structures in what is called the protein folding process [6].

There are four levels of protein structures, as visualized in *Figure 1.4*:

- **The primary structure** describes the sequence of amino acids in the polymer chain. It originates from inter–atomic bonds which hold together the molecular chain (*Figure 1.4.a*).
- **The secondary structure** refers to regularly occurring structural patterns in the polymer chains, specifically to α–helices and to β–sheets, which are two of the main structural types (*Figure 1.4.b*).
- **The tertiary structure** describes the spatial formations created by α–helices and to β–sheets, which are tightly packed to form a compact structure (*Figure 1.4.c*).
- **The quaternary structure** refers to the structure created by combining several tertiary structures. It describes the resulting protein molecule created by combining the tertiary subunits (*Figure 1.4.d*).

These structures represent information about the protein molecules which are applicable for the field of biochemistry, but not very useful for our problem at hand. A different approach, described in the next section, is therefore used.

1.2.3 Protein Structure Representation

Given the complexity of protein spatial structures, different representations exist to fulfill individual purposes.

- **Space-filling model** (*Figure 1.5.a*) is a three-dimensional molecular representation where the atoms are portrayed by spheres whose radii are proportional to their Van-der-Walls radii, and whose center-to-center distances are proportional to the distances between the atomic nuclei, all in the same scale. Atoms of different chemical elements are commonly visualized by differently colored spheres [4]. This model is applicable for protein cavity and tunnel detection, since it gives the most precise information about the space occupied by the molecule's atoms. Given the simplicity of this approach's implementation and its credibility, this model has been implemented to represent the molecules in our algorithms.

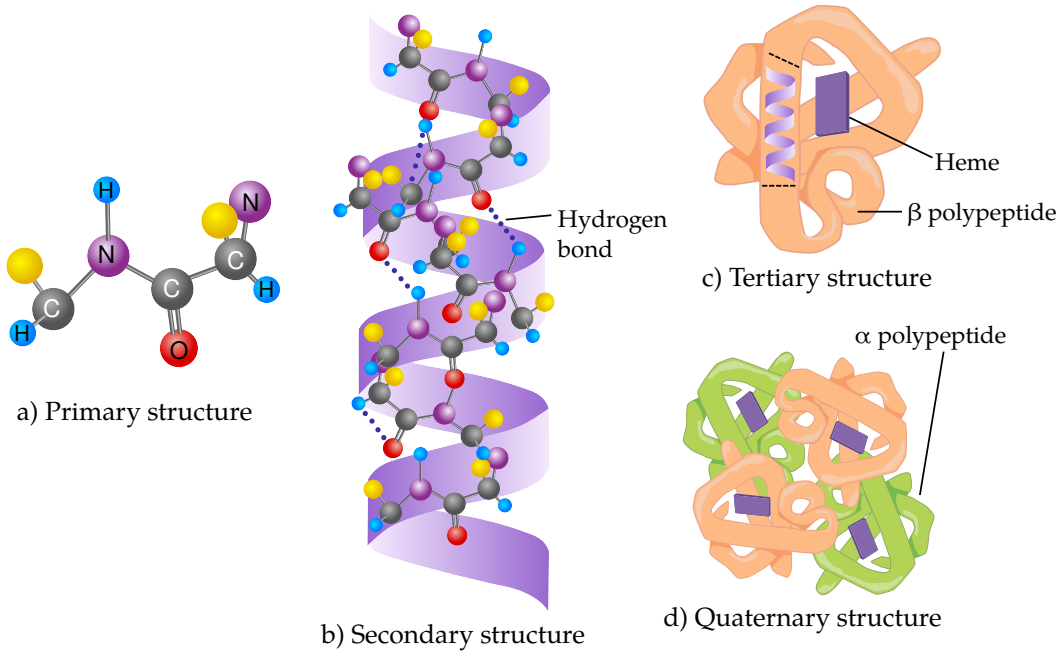


Figure 1.4: Different levels of protein structures. Original image courtesy of [36].

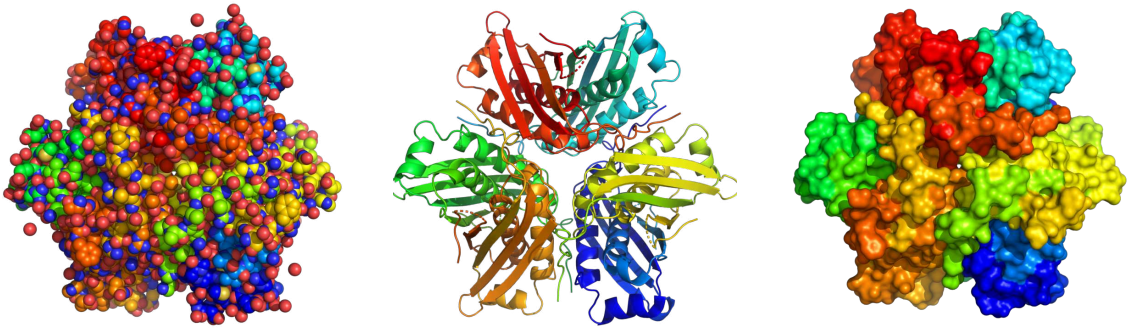


Figure 1.5: Different types of spatial representations of the MOAC protein from E. Coli. Subfigure a) *shows the space-filling model, subfigure* b) *the ribbon model and subfigure* c) *the molecular surface model. PDB file courtesy of [3].*

- **Ribbon diagram** (*Figure 1.5.b*) is a 3D schematic representation of protein structure. It is one of today's most commonly used methods of protein visualization. The ribbons show the overall spatial position and organization of the polymer chain and serve as a visual framework on which we can add details of the full spatial atomic structure. α-helices are shown as coiled ribbons or thick tubes, β-strands as arrows and lines or thin tubes for non-repetitive coils or loops. The direction of the polypeptide chain is shown locally by the arrows, and may be indicated overall by a colour ramp along the length of the ribbon.

  Ribbon diagrams are a powerful tool capable of expressing the overall functional structure of a protein molecule. They are, however, misleading for our purpose, in their way of portraying large parts of the spatial protein structure as an open space. While inapplicable for path planning, this model is excellent in providing insight regarding the protein's general structure and the overall location of the found path in the protein molecule [35].

- **Molecular surface models** (*Figure 1.5.c*) visualize the outer molecular layer, which is generated using state-of-the-art computational techniques. They are very useful because they can facilitate excellent visualizations, as well as numerical simulations of protein molecules. One example of such surface model is the recent *Ligand Excluded Surface* [27], which uses a grid–based algorithm to approximate not only the molecular surface but also molecular cavities capable of containing the ligand molecule. The model discretizes a configuration space of a ligand inside and near a protein molecule and then exhaustively searches for all colliding molecular locations. After the search completes, a 3D surface model is then created from the found information. Since this model uses the supplied ligand molecule to find the molecular surface, it provides the most precise molecular surface approximation for the given ligand molecule. It is, however, very computationally demanding and therefore unsuitable for our purposes, other than rendering visualizations.

1.2.4 Protein Conformations

The spatial arrangement of atoms in a molecule is termed *molecular conformation* [30]. This term is somewhat analogous to the, perhaps more robotics–oriented term, configuration, which will be explained in greater detail in the following *Chapter 2*. The structure of a typical protein contains hundreds of individual chemical bonds. Because free rotation is possible around many of these bonds, the protein can assume an unlimited number of structural configurations (conformations). Whether these bonds rotate and the axis of their rotation is determined by the atoms which share the bond, the types of bonds and the neighboring atoms (see *Figure 1.6* for illustration). The determination of the number of degrees of freedom in a protein molecule and their properties is a widely studied topic, from both the biochemical and the algorithmic perspective (see e.g. [48]). It is, however, beyond the scope of this thesis, which focuses on this problem from the more robotics–oriented, path–planning perspective. A number of tools with this functionality is available, such as the AutoDock Vina [19], which has been utilized in this thesis. Other sources, discussing this topic include, e.g., [40].

1.2.5 Intramolecular Potential Energy

Two atoms positioned in space can interact with each other in various ways, with both attractive and repulsive forces. The intensity of these forces depends on the distance of the atoms. When a chemical bond is present, the Van der Waals forces keep the bonded atoms at a fixed distance – the equilibrium point. Any variation from this fixed distance results in a change of the intramolecular potential energy. As we have established previously, a ligand molecule can assume different conformations, thanks to rotatable bonds. When three atoms are connected via chemical bonds, these bonds assume a certain conformation, establishing a fixed angle between the atoms. Again, any variation of this angle changes the intramolecular potential energy [9]. An example of a potential energy function is presented in *Figure 1.7*.

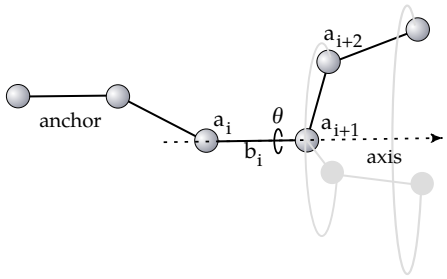


Figure 1.6: Protein conformation with a rotating bond. The atoms indexed after the atom a_i rotate by angle θ around axis $\vec{b}_i$. Original image courtesy of [32].

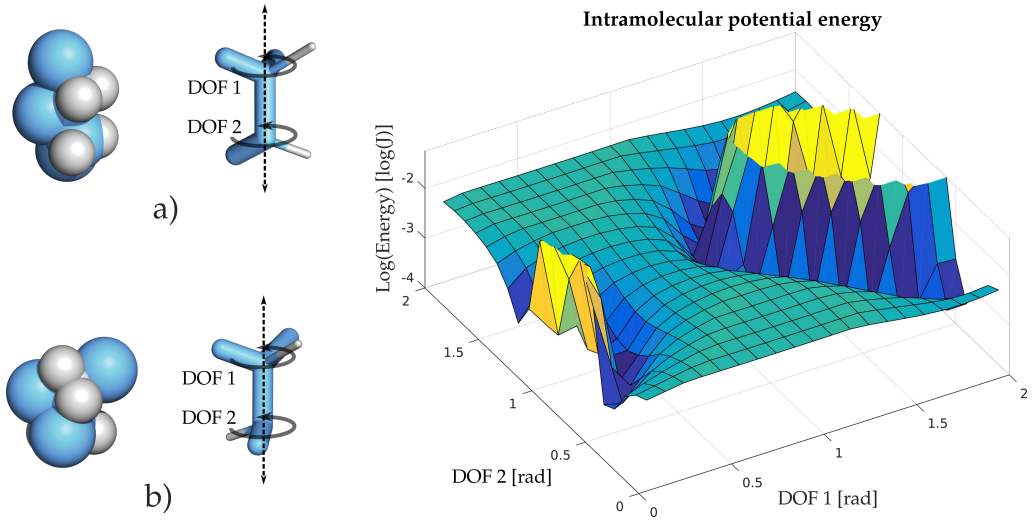


Figure 1.7: Example of an intramolecular potential energy function for a 2 degrees of freedom molecule 1,2,3-Trichloropropane, depicted in Subfigure a). *Molecular potential energy is a function of internal molecular angles. The probable, low–energy conformations are depicted in dark blue, and improbable, high–energy conformations are depicted in yellow. The energy value was log–scaled to better illustrate minute differences between low–energy conformations. The globally lowest energy conformation is depicted in* Subfigure b).

As is usually the case in physics, the most probable molecular state is the conformation with the lowest intramolecular potential energy. When planning a pathway for a flexible ligand molecule purely from the geometric point of view, we assume that the molecule can take on any conformation. This, however, is not the case in nature, since high–energy conformations are much less probable. Thus, what may seem like a straightforward motion from the path planning point of view can actually be exceedingly

improbable in the real molecules.

Taking potential energy of the ligand molecule into account enables us to further judge the feasibility of the found pathway. If the molecule has to assume high–energy conformations for the majority of the motion to facilitate its passage through the protein tunnel, we can safely assume that such a motion is improbable and the pathway is not very feasible. To enable this approach, the Transition–RRT (T–RRT) planner [20] was implemented. Our solution facilitates planning in continuous cost–spaces and thus allows us to take ligand intramolecular potential energy into account. See *Section 3.4.7* for details.

2 Configuration–Space Search

This chapter introduces the area of configuration–space searching, defines the necessary terminology and presents current state of the art methods and algorithms for path planning in the environment of protein molecules [15].

2.1 Configuration–Space

The problem of finding a path through various environments and for multiple types of robots has been an intensively studied topic for several past decades. The complexity and diversity of the some of the possible environments, combined with infinitely many continuous robot configurations, makes the task of finding an optimal path between two locations, which the robot is able to traverse without colliding with the environment, a particularly difficult challenge. There exists a plethora of algorithms, with each usually suited best for a particular problem. Some of these algorithms, suitable for the topic of this thesis, will be discussed later. Generally the motion planning problem can be formulated using the concept of configuration space [28], which is explained in the following paragraphs.

A **workspace** of a robot describes the area in which the robot operates, using the common Euclidean coordinates. It includes all spatial positions reachable by the robot's end effector. Workspaces are described in terms of spatial coordinates. This makes them easy to understand for humans, but, depending on the construction of the robot, it can also make them difficult to utilize for path planning. *Figure 2.1* shows an example of a robotic workspace.

A **configuration** c of a robot is a vector of parameters with length equal to the robot's number of degrees of freedom (abbreviated as DOF for the rest of this thesis). A configuration fully describes the current state of the robot. Each of the parameters represents a particular DOF.

A **configuration space** $\mathcal{C}$ of a robot is a set of all possible configurations. Configuration spaces can include invalid states, in which the robot is colliding with the environment, or with itself. Depending on a particular robot's construction, the configuration space may form such complex objects as manifolds, which greatly increases the difficulty of planning a collision-free path for such a robot. *Figure 2.1* shows an example of a configuration space and its mapping onto a Euclidean workspace.

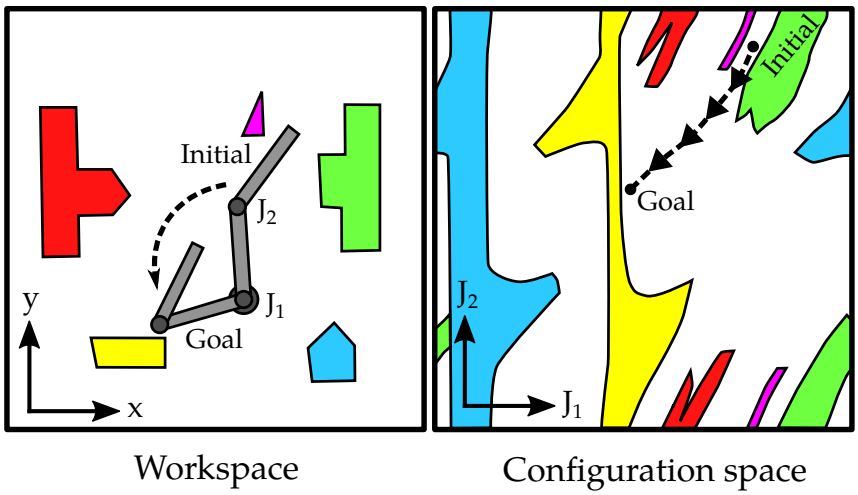


Figure 2.1: Example of a robot workspace and configuration space. The left subfigure shows the original workspace, with initial and goal robot configurations and obstacles. The right subfigure shows how the workspace and the obstacles map onto the robot's configuration space (joint space). A collision–free path between the two configurations is shown as a striped line. Original image courtesy of [34].

A **path** in a configuration space $\mathcal{C}$ is a sequence of configurations, which satisfies some imposed constraints. Let c_{init} be the robot's initial configuration and c_{goal} the goal configuration. Let $colliding(c)$ be a predicate, which holds true iff the robot is colliding with itself or with the environment in the supplied configuration c. Let $interpolate(c_1, c_2)$ be a function which interpolates the supplied configurations c_1 and c_2, and creates a set of infinitely many configurations between the two. Let there be a trajectory $\mathcal{T}$ of n configurations $\mathcal{T} = \{c_1, c_2, ..., c_n\}$. Let an interpolated trajectory $\mathcal{T}_{inter}$ be a set of all configurations created by interpolating the trajectory $\mathcal{T}$ in the following manner:

$$\mathcal{T}_{inter} = \mathcal{T} \cup \{\forall i : 0 \leq i < n, interpolate(c_i, c_{i+1}), c_i \in \mathcal{T}\}.$$

We say a path is collision–free iff the following condition holds true

$$\forall c(c \in \mathcal{T}_{inter}), \neg colliding(c).$$

A sharp–eyed reader might object, since checking collisions for an infinite amount of configurations is quite difficult. Analytical solutions for collision detection checking exist only for a set of geometric primitives and under certain conditions [11]. In reality, the conditions for collision detection are often relaxed. Collision checking is usually performed for a set of configurations which have been interpolated with a certain resolution. This resolution is set in a way which allows us to trust that the found path is collision–free and still perform the calculations in a finite amount of time.

2.2 Path Planning Algorithms

We have now established, what the act of "being able to get from configuration a to b" means. However, as is usually the case in the field of robotics, there is another problem – knowing how to get from a to b. As stated previously, a particular configuration space can form such simple objects as a line, or such complex objects as a multidimensional manifold. Navigating such spaces quickly becomes intractable for human operators, as the number of DOF of the robot increases. Since even a simple non-constrained 3D object in a 3D Euclidean space has 6 DOF and its configuration space forms a manifold, due to the SO(3) rotation group, path planning presents a non–trivial challenge, even for such simple cases.

A path–planning algorithm searches for a path satisfying specified constraints in the supplied configuration space. Existing path–planning algorithms can be divided into several groups:

Deterministic algorithms, which search on a grid or a graph–based configuration space in a predictable manner. They are usually employed when there exists a finite number of configurations. They are guaranteed to find the path in a feasible amount of time. Examples include the Breadth–First–Search algorithm (BFS), Depth-First–Search algorithm (DFS), A* algorithm and a variety of others. Their counterpart are **stochastic** algorithms, which incorporate an element of randomness when choosing the next configuration to explore. They are mainly used when we are planning in an infinite configuration space, such as the Euclidean space. Examples include, e.g., the Probabilistic RoadMap algorithm (PRM) [14], the Rapidly–Exploring Random Tree Search algorithm (RRT) [29], the Rapidly–Exploring Dense Trees algorithm (RDT) and many others.

Graph–based algorithms which search for the path on a predetermined graph with known connections and nodes (BFS, DFS, A*, ...) and **sampling** algorithms which randomly create nodes and connections between them in the provided planning domain (PRM, RRT, RDT, ...). The randomly created graph's nodes can be interconnected (PRM) or can be organized into a tree (RRT)[1]. The benefit of utilizing trees is the fact that when the goal region is reached, the algorithm does not need to search the built graph, but can simply traverse the tree to its root, in order to extract the found path.

Offline and **online** algorithms. When planning a path for a robot in a dynamic environment, being able to replan the path according to the observed changes and therefore react quickly is more important that knowing the path in its entirety. Offline path planners compute and return the entire found path, whereas online path planners only focus on the immediate portion. This makes them able to quickly react to changes in the environment, at the expense of possible non–optimal paths.

The topic of this thesis concerns searching a continuous high–dimensional configuration space with infinitely many configurations. We will therefore focus on offline, sampling–based path–planning algorithms and describe them in more detail in the following sections.

[1] A tree is a connected graph with no cycles

2.3 Sampling–Based Algorithms

Sampling–based path–planning algorithms work in a continuous configuration space with infinitely many configurations. Instead of systematically visiting every possible configuration, they randomly sample configurations in the space and try to find possible connections between them. If a connection is found, the new node is added to a graph of previously found connections. Once this graph reaches the goal location, a path is reconstructed from the graph. There exist many modifications, each of which is suitable in a different environment and fulfills a different purpose. The most prominent examples of such algorithms are presented below.

2.3.1 Probabilistic Roadmap Algorithm

The PRM algorithm [14] consists of two phases. First, the algorithm constructs a graph by randomly sampling the configuration space and interconnecting all available nodes (the learning phase). Then the algorithm searches for a path between two supplied locations (the query phase). See *Figure 2.2* for an illustration. While the algorithm does require some parameter tuning to adjust to a general scenario case, it is a well tested, robust algorithm which is able to find a solution in general environments. Furthermore, unlike the following algorithms, if the environment is static, there is no need to repeat the learning phase when planning between different start and goal configurations. Thus the following planning queries can be finished in fractions of the time required for the learning phase to finish, enabling multiple–query planning[2]. It does, however, struggle to find a path in a narrow corridor or in a bug trap type situation [14].

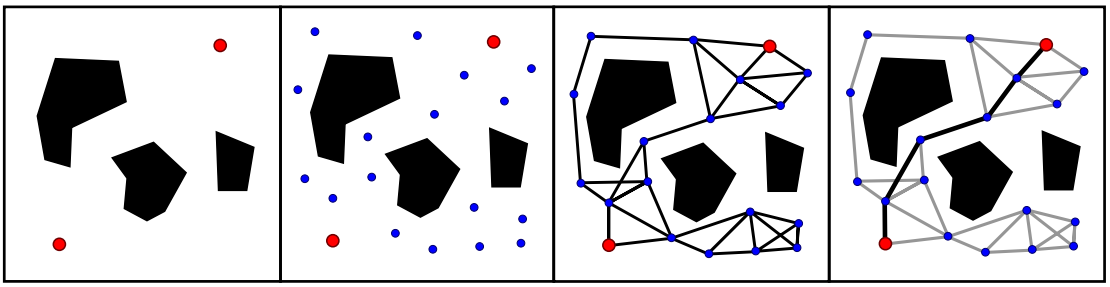


Figure 2.2: Functionality of the Probabilistic Roadmap algorithm. Subfigure a) *displays the planning environment (workspace), with the initial and goal locations denoted in red. Subfigure* b) *shows the first stage of the learning phase, which consists of sampling collision–free states in the configuration space. Subfigure* c) *shows the second phase of the learning phase, which tries to connect the collision–free nodes, where possible. A generated graph of nodes connected by collision–free trajectories is shown. Subfigure* d) *displays the query phase and the found shortest path.*

[2]Multiple–query planners are particularly useful for multirobot systems, when we're dealing with the task of planning paths for a large amount of robots in a static or dynamic environment.

2.3.2 Rapidly–Exploring Random Trees Algorithm

The RRT algorithm [29] is a single–query incremental algorithm designed specifically to quickly explore high–dimensional configuration spaces of any shape and size. It works by building an oriented tree from the initial supplied location. The planner samples the configuration space and tries to connect the new samples to the previously built tree. If the samples can be connected by a collision–free trajectory, the new sample is added to the tree and the process repeats. After every iteration the algorithm checks whether the newly added sample lies within a preset region of the supplied goal location. If it does, the algorithm terminates and a path is extracted by following the tree edges to the root of the tree, i.e., the initial location. The functionality is illustrated in *Figure 2.3*. The algorithm is shown to be able to handle high DOF applications with both holonomic and non–holonomic constraints [29] and is one of the most popular currently used path–finding algorithms. It spawned a variety of modifications, some of which are described in the following subsections. Like the PRM algorithm, it can struggle to find a path through a narrow corridor or a bug trap type situation. Unlike the PRM algorithm, RRT is a single–query planner, requiring a fresh start for every task.

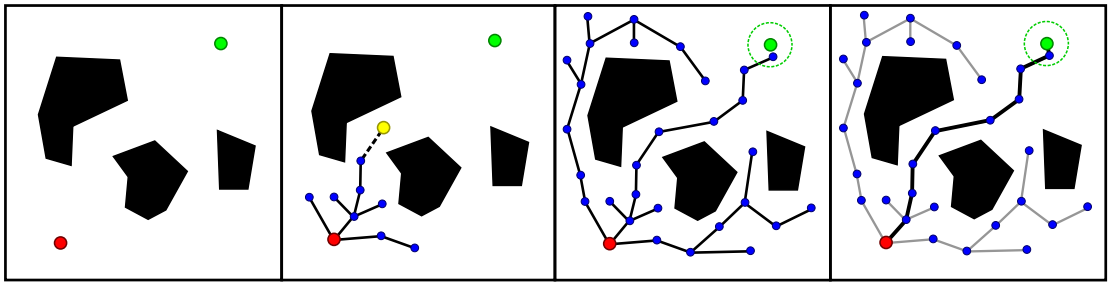


Figure 2.3: Functionality of the Rapidly–exploring Random Trees algorithm. a) *The environment, with the initial location in red and goal location in green.* b) *Tree growth, with a new node being added in yellow.* c) *A fully grown tree, with a node, which lies in the goal region, shown as dashed green line.* d) *The extracted path.*

2.3.3 Rapidly–Exploring Dense Trees Algorithm

Rapidly–exploring Dense Trees algorithm [25] is a modification of the RRT algorithm, which yields good performance out of the box, without a lot of parameter tuning [25]. It functions similarly to the RRT algorithm, however, when a nearest neighbor of the newly sampled node is being searched, the points can be connected to the nearest position on the edge, alongside the previously sampled tree nodes. This results in a dense covering of the searched space. See *Figure 2.4* for an illustration of the functionality.

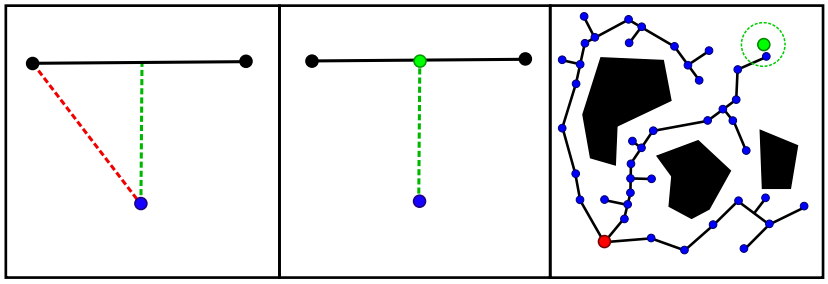


Figure 2.4: Functionality of the Rapidly–exploring Dense Trees algorithm. a) *Trajectory segment with a newly sampled node depicted in blue. A conventional trajectory which would be used by the RRT algorithm is shown in red, whereas the trajectory found by the RDT algorithm is shown in green.* b) *When a node is connected to an existing trajectory segment, the segment is split into two and a new node is created, to which the sampled node is connected.* c) *This behavior results in the characteristic RDT graph, which contains a high number of right angles, when compared to the graph resulting from the RRT algorithm.*

2.3.4 Optimal RRT Algorithm (RRT*)

Both the PRM and the RRT (RDT) algorithms are non–optimal methods. When a path is found, it is nearly always longer than the shortest possible path and requires further refining. The RRT* algorithm [22], however, is asymptotically–optimal[3]. It also builds a tree graph, like the RRT algorithm. Its key defining characteristic is the fact that it can re–route non–optimal parts of the search tree. When adding a new node to the tree, all nearby nodes in a certain region are tested for shorter paths. If the cost of reaching the root of the tree from the nearby nodes is higher through the tree than through the new added node, the respective edges are rerouted through the new node. See *Figure 2.5* for more details.

As the name suggests, the algorithm is guaranteed to eventually find the optimal (shortest possible) path. However, given the scenario we're concerned with, path optimality isn't as important as finding any path through the complex environment. The additional steps during the rewiring phases of this algorithm cause a slowdown, which is unnecessary for our purpose and for this reason we haven't utilized it in our approach.

2.3.5 RRT-Path Algorithm

The RRT-Path algorithm [43] combines the best of the PRM and the RRT algorithm. It employs the concept of a so–called auxiliary path, which represents additional knowledge about the configuration space. The path is precomputed and represents void regions of the configuration space, which have higher probability of collision–free configurations. The algorithm works by growing a tree graph from the initial location, while employing the auxiliary path as a guide for biased sampling. When creating a new sample, a

[3]Asymptotic optimality means that the limit of the probability of finding an optimal path converges to 1 as runtime approaches infinity. Given enough time, the algorithm will always find the shortest path.

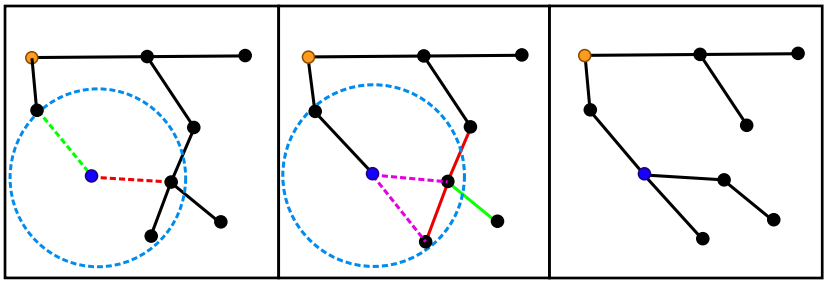


Figure 2.5: Functionality of the RRT algorithm.* a) *When a new node (blue) is added to the search tree, instead of simply choosing its nearest neighbor as its parent (red dashed line) like the RRT algorithm would, RRT* considers routes to all its neighbors in the region* X_{near} *(blue dashed circle). A trajectory (green dashed line) is clearly more optimal.* b) *The new node is added and all existing search tree nodes in the region* X_{near} *are considered for rerouting. Purple dashed lines represent possible alternatives for the current non-optimal (red) lines. The alternatives are clearly more optimal. Note how the green line is not rerouted, since it doesn't lie in the region* X_{near}. c) *A more optimal tree is constructed with the new node being a parent of some already existing nodes.*

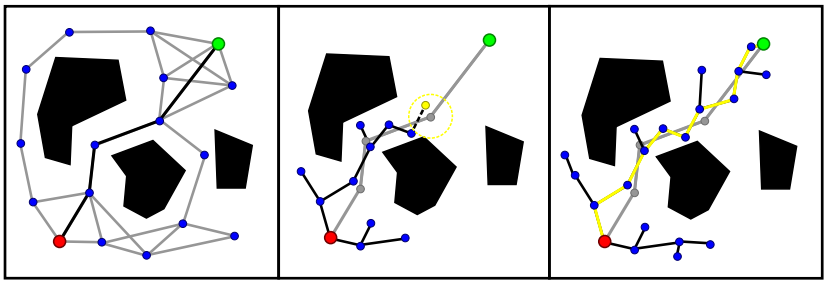


Figure 2.6: Functionality of the extended RRT-Path algorithm. a) *In the first phase, a roadmap is created (gray), and an auxiliary path for a scaled–down robot is found (black).* b) *A tree graph (black) is then grown, using the auxiliary path (gray) as a guide. A newly sampled node near in the auxiliary path region is depicted in yellow.* c) *A fully grown tree, with the final found path in yellow. Note how the final tree graph exhibits much smaller branching, and is able to reach the goal location with fewer nodes, than the original RRT implementation.*

configuration in the auxiliary path region is picked with a non–zero probability. This region moves as parts of the auxiliary path are being reached by the tree graph. The functionality is illustrated in *Figure 2.6*. This ensures rapid progress through empty environments, while providing flexibility when navigating around and through more complex obstacles. This algorithm has been shown to exhibit good performance when planning through narrow passages, to the point of surpassing other planners [42], [43].

An extension of the algorithm is presented [44]. The modified algorithm first finds a path for a scaled–down version of the mobile robot. The found path is then used by the RRT-Path algorithm, when finding a path for the robot with its original size. The features and characteristics of this algorithm (described in greater detail in the next chapter) make it a very suitable choice for the problem at hand.

2.4 Conclusion

The concept of configuration spaces can greatly aid us in pathways detection problem solving. We have presented some of the most prominent modern sampling–based path–planning algorithms. There exists a large amount of other path-planning algorithms, however, as the results of [41] have shown, tree-based planners such as the RRT or RRT-Path algorithms offer good results when planning in complex and narrow environments, while being relatively easy to implement.

This thesis does not focus on the quality of the found path, but rather on its existence. It is therefore pointless to require an optimal path, or to refine the path once it has been found. Furthermore, the nature of the task offers an excellent opportunity of utilizing the abilities of the RRT–Path algorithm, since there exist tools designed for protein cavity analysis. These tools can be used to precompute guiding paths, which are then utilized by the RRT–Path algorithm to speed up the search.

We have therefore decided to employ the RRT–Path algorithm as a basis for our implementation in this work, and extend it using ligand flexibility, atom scaling, potential energy planning and a number of other modifications, for the purpose of solving the ligand pathways detection problem.

3 Pathways Computation Algorithm

This chapter presents our solution to the problem introduced in *Chapter 1*, using the terminology presented in *Chapter 2*. We describe the selected state–of–the–art algorithm for configuration space searching, the modifications required to enable its use for molecular pathways computation and the motivation behind them, and the resulting implementation.

3.1 RRT-Path

As stated previously in *Section 2.3.5*, the RRT-Path algorithm is a modification of the single–query incremental sampling–based RRT algorithm. Like the RRT algorithm, it is able to quickly traverse high–dimensional configuration spaces and find a solution to a path–planning problem. Unlike the RRT algorithm, thanks to the introduction of auxiliary guiding paths, it is able to quickly traverse even narrow passages, assuming a good guiding path is supplied [43].

Auxiliary paths represent additional knowledge about the supplied configuration space, being a rough estimation of the possible final path. When creating these guiding paths, a simplified robot[1] is used. As a result, the found paths can be infeasible[2], due to constraints imposed by the robot, thus being invalid as the final solution. A variety of methods for computing these auxiliary paths is available, such as visibility graphs [21], Voronoi diagrams [8] (employed by CAVER) or even the PRM algorithm with a simplified mobile robot model.

Finally, during the searching phase, the algorithm exhibits the so–called *temporary goal bias* for selecting nodes from the auxiliary path when sampling new states. The auxiliary path is parameterized and a beginning node is selected as a temporary goal.

[1] The term simplified robot denotes using simplified geometry, such as a primitive shape, to represent the robot. CAVER 3.0 uses a spherical probe with a set radius to represent the ligand molecule.

[2] Examples of infeasibility include such cases as collisions between the robot and the environment, or not satisfying differential constraints. Differential constraints can arise when we planning a motion for nearly any mechanical system. They encompass such cases of problems as non–holonomic planning, kinodynamic planning, and trajectory planning, all of which are intensively studied topics in the field of robotics [5]. Differential constraints restrict the DOF of the robot, based on its state. An often cited example on non–holonomic planning is that of a car, which can move forward or backwards, but has to change the angle of its front wheels in order to move left or right.

Once this, or any other node positioned further down the path is reached, the temporary goal node is moved to accommodate this, thus ensuring the tree graph grows along all portions of the supplied auxiliary path. Pseudocode of the algorithm is listed in *Algorithm 1*.

```
1  Function BuildRRTPath
       input  : Initial configuration q_init, configuration space C, goal region G, auxiliary
                path P
       output: RRT graph T
2      T.addNode(q_init);
3      tempGoal ← initializeTemporaryGoal(P);
4      while ¬goalReached() do
5          q_rand ← createRandomSample(tempGoal);
6          q_near ← nearestVertex(q_rand);
7          if collisionFree(q_near, q_rand) then
8              T.addNode(q_rand);
9              T.addEdge(q_near, q_rand);
10             moveTemporaryGoalNode(T, tempGoal, P);
11         end
12     end
13     return T;
```

Algorithm 1: RRT-Path algorithm. The member function *addNode* inserts a vertex into the tree, similarly the function *addEdge* inserts an edge. Function *initializeTemporaryGoal* selects the first auxiliary guiding path node to create samples. Function *goalReached* checks whether a vertex of the search tree lies within the goal region G. A goal region is a defined area in the Euclidean or configuration space, usually a sphere, used for stopping the search, after the planner reaches it. Function *createRandomSample* chooses a sample from the temporary goal node or the entire configuration space, depending on the *goal bias* parameter g_b. Function *nearestVertex* returns a nearest neighbor of the supplied vertex, located in the search tree. Function *collisionFree* checks whether a path, formed by interpolating between the two supplied vertices, representing configurations, is collision–free. Finally, function *moveTemporaryGoalNode* checks whether the search tree has reached the current temporary goal node and if it did, selects a different one further down the guiding path (see *Algorithm 2*). Since the search graph is a tree, the final path can be extracted from the resulting graph by iterating over the parents of the respective vertices, against the direction of the directed edges.

3.2 Guiding Paths Computation

The utilized RRT–Path algorithm requires a pre–supplied guiding path in order to properly utilize its full potential. In our case, the guiding paths consist of protein tunnels, represented as spheres with varying radii. These tunnels can either be created manually

```
1 Function moveTemporaryGoalNode
      input : Search tree T, current temporary goal node (tempGoal), initial path
              P = (P_1, ..., P_N)
      output: New temporary goal node
2     for p = P_N : −1 : tempGoal do
3         nearVertex ← nearestNeighbor(V, p);
4         if ||p − nearVertex|| < p.radius then
5             return p
6         end
7     end
8     return tempGoal;
```

Algorithm 2: Temporary goal node moving function. The auxiliary path nodes contain radius of the tunnel in the spot they represent, which can be utilized to aid detection. Function *nearestNeighbor* find a tree vertex nearest to the supplied auxiliary path node. The notation $||\cdot||$ represents the L2 vector norm. The auxiliary path nodes are checked from the end of the path, to prevent unnecessary slowdowns, when a path node following after the current temporary goal node is reached first.

by the user (which would be extremely tedious and time consuming) or automatically generated. There currently exists a variety of specialized tools for protein cavity detection. Examples include FPocket [26], PocketPicker [45], or CAVER 3.0 [12], which was utilized in this thesis for guiding paths computation.

CAVER 3.0 is a software tool which enables protein cavity identification and characterization, for the purpose of docking problem solution estimation and virtual molecular screening [12]. It can automatically analyze supplied protein molecules with a given probe size in static as well as dynamic protein molecule snapshots.

The pathways are identified in the supplied protein structure using user–specified starting points (protein active sites). The protein structure's void space is then divided using Voronoi diagrams. Voronoi edges in the free space are clustered into individual tunnels, which are separated using a dissimilarity function. An example of CAVER output is given in *Figure 3.1*.

The detection of pathways allows us to determine the main transport tunnels for the supplied molecule, which in turn enables us to characterize them and to judge whether intermolecular ligand–receptor interaction could occur in the respective real, physical counterparts.

CAVER is a widely used tool that can identify even complex protein cavities and provides robust results, which can be flexibly tailored to fit the needs of the user using a number of parameters. It was therefore employed to compute the auxiliary guiding paths used by our algorithm. The output of CAVER consists of a multitude of tunnels, represented as spheres with corresponding radii. These tunnels originate inside the protein active site as one tunnel and then split inside the molecule into multiple tunnels, which ends of separate locations on the protein surface.

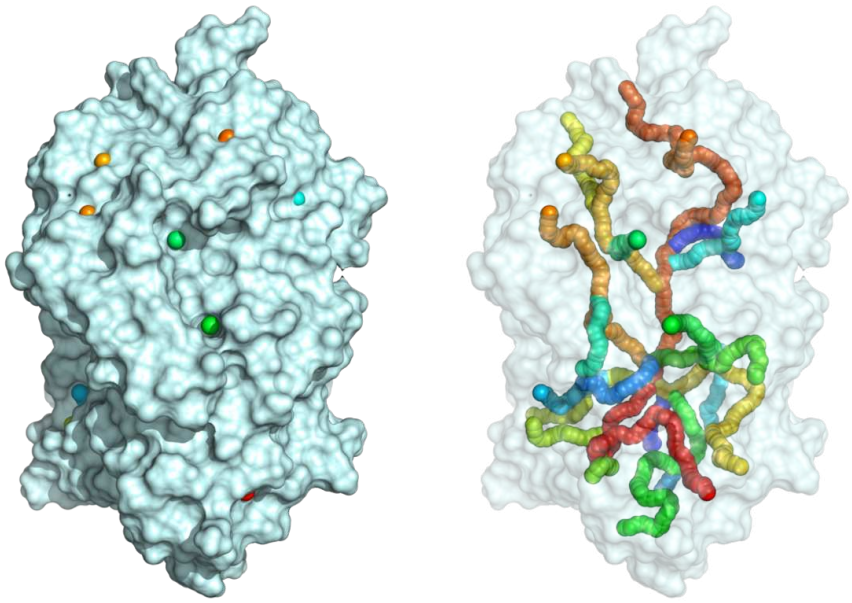


Figure 3.1: An example of tunnel structures found by CAVER 3.0. a) *shows the surface of a protein molecule with tunnel ends visible.* b) *shows the tunnel structures inside the protein molecule. Different tunnels are labeled with different colors. There are 25 found tunnels for this (1MXTa) protein molecule. These tunnels are used by the final version of our algorithm as auxiliary guiding paths.*

3.3 Ligand Conformation Computation

Section 1.2.4 established the terminology regarding protein conformation and explained the core concepts. To reiterate, molecules may exhibit flexibility in certain chemical bonds. Whether this occurs depends on constituent atoms of the chemical bond and the atoms surrounding them. These bonds rotate around an axis, the direction of which also depends on the constituent and surrounding atoms. The conformation (configuration) of a protein molecule may be fully described by a set of parameters denoting the degrees of rotation in the rotating bonds.

The determination of the number of DOFs requires expert knowledge of physical chemistry, and is out of the scope of this thesis. Since a variety of suitable software solutions exists, we have utilized an existing program AutoDock Vina.

AutoDock Vina [19] is a program developed to search for solutions to molecular docking and virtual screening problems (prediction of molecular characteristics). It utilizes custom scoring functions to find rotatable bonds and binding energy of the molecules. It also employs sophisticated optimization methods to speed up the computation process of discretizing the search space to find suitable binding conformations and location of ligand molecules.

Given the fact that the tool is written in the C++ programming language and its source code is available, we have employed it to model ligand molecule conformations. A wrapper library was created which, when supplied with the molecule file and the

configuration of its angles, returns relative locations of individual ligand atoms. These locations are then used by our implementation to construct the spatial ligand structure.

3.4 Novel Modifications of the RRT–Path Algorithm

The original RRT–Path algorithm uses a single guiding path and is suitable for a rigid mobile robot, with the extension [44] applicable for 6 DOF. We need to plan for flexible bodies (and consequently many DOF) and take intramolecular potential into account, for the algorithm to be applicable in our scenario. The following sections describe our modifications.

The resulting implementation is illustrated in *Figure 3.2*. The receptor molecule and the locations of its active sites are inputted into CAVER 3.0, in order to generate receptor tunnels. These tunnels are then, along with the receptor and ligand molecules, used as an input for our algorithm. The final algorithm has been termed *Protein–RRT–Path* (P–RRT–Path in short).

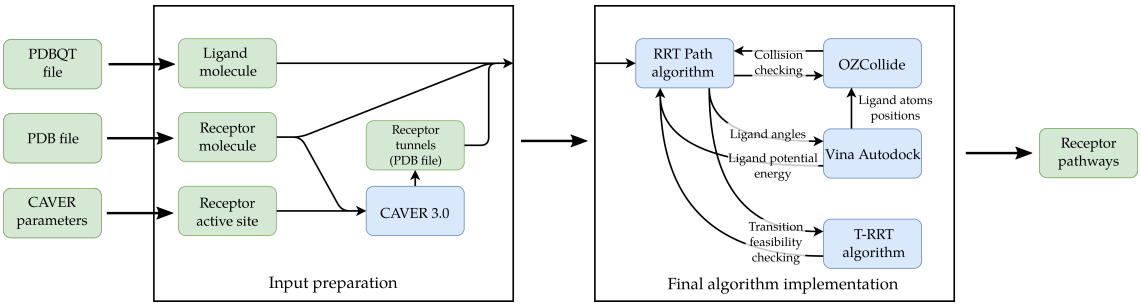


Figure 3.2: Workflow diagram of the final implementation. The left block displays the workflow required to prepare input data for our algorithm. The right block illustrates the synergy of the individual components, required in order to complete the planning task, and the resulting output.

The basis of our implementation is the RRT–Path algorithm, which guides the sampling alongside one the supplied receptor tunnels. It decides on the position and orientation of the ligand molecule, decoupled from the ligand's conformation. The Transition–RRT algorithm selects a feasible conformation of the ligand molecule, in order to maintain the lowest possible intramolecular potential energy. After the ligand molecule configuration[3] is chosen, the algorithm uses the AutoDock Vina wrapper in order to compute ligand atom positions from ligand intramolecular angles. These positions are then passed to OZCollide, along with the chosen configuration, in order to find out whether the configuration is collision–free or not. If it is, these steps repeat in order to check the validity of the possible trajectory, up to a selected resolution threshold. If the resulting trajectory is collision–free, the configuration is added to the search tree. Once a newly added tree node lies within the goal region (the last temporary goal node) the algorithm finishes, otherwise the process repeats itself. After the planner stops, a path can be easily ex-

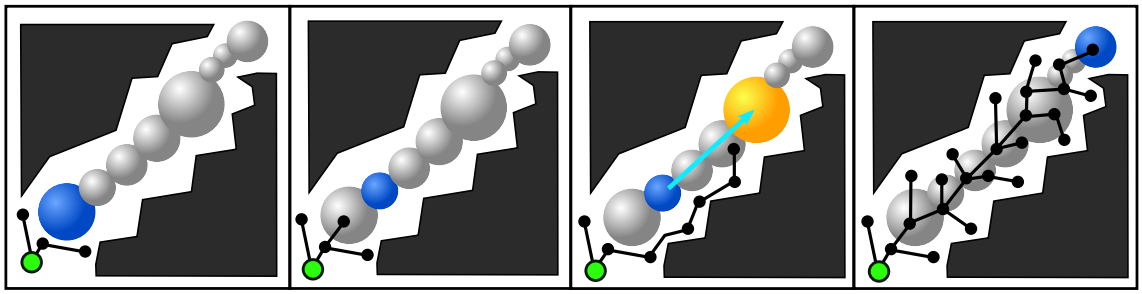
[3]Position, orientation and conformation

tracted by moving up the tree to the root, and keeping track of the visited nodes. See *Algorithm 3* for pseudocode of the final implementation.

The following sections describe in greater detail the novel modifications of the RRT–Path algorithm and the motivation behind them.

3.4.1 Tree Growth Stagnation Detection

As stated previously, the algorithm utilizes the concept of auxiliary guiding paths to speed up the tree growth. While this offers the ability to sample in a free configuration space, the resulting trajectories might still collide with the environment if the guiding path is curved and the new sampled node is sufficiently far. To mitigate this issue, we've devised the concept of temporary goal nodes, which limit the sampling area of the guiding path. When a new sample is created from the auxiliary guiding path, it may only be sampled in a sphere denoted by the temporary goal node. This leads to better tree growth while limiting the amount of samples which would in invalid new trajectories.



a) b) c) d)

Figure 3.3: Illustration of a temporary goal node. a) *A tree is grown from the initial node (green) towards the temporary goal node (blue) located on the auxiliary guiding path (gray). New samples can either be created in this temporary goal node with a non–zero probability, or in the entire configuration space.* b) *The first temporary goal node has been reached so a new temporary goal node is created further down the path.* c) *If any of the nodes on the path after the current temporary goal node is reached. A new temporary goal node (orange) is created to reflect this, keeping the tree growth as fast as possible.* d) *Once the last temporary goal node is reached, the planner stops and the final path is extracted.*

When the planner is started, the guiding path is analyzed and split into separate spherical nodes. The first node of the path is then set as the temporary goal node. If the search tree reaches the temporary goal node or any of the following nodes of the guiding paths, the temporary goal node is moved further down the guiding path to better stimulate the search tree growth (see *Algorithm 2*). Once the last node of the path is reached, the planning is complete and the algorithm may halt.

We have utilized the concept of temporary goal nodes to track the progress of the planner. As stated previously, the planner can occupy two states, a normal state and a stagnation state. The detection of this stagnation state is performed by monitoring the amount of iterations since the temporary goal node was last moved. If the iterations count crosses a certain threshold, denoted by parameter I_s, we assume that the planner

```
1  Function Protein-RRT-Path
       input : Initial configuration q_init, configuration space C, goal region G, auxiliary
               path P
       output: Ligand Pathway L_P
2      T.addNode(q_init);
3      tempGoal ← initializeTemporaryGoal(P);
4      while ¬goalReached() do
5          q_rand ← createRandomSample(tempGoal);
6          q_near ← nearestVertex(q_rand);
7          updateAtomScale();
8          q_rand ← limitNodeDistance(q_rand, q_near);
9          q_rand ← limitLigandAngles(q_rand, q_near);
10         if ¬checkNodeTunnelDistance(q_rand) ∨
             ¬checkTransitionTRRT(q_near, q_rand) then
11             continue ;
12         end
13         if collisionFree(q_near, q_rand) then
14             T.addNode(q_rand);
15             T.addEdge(q_near, q_rand);
16             moveTemporaryGoalNode(T, tempGoal, P);
17         end
18     end
19     return extractPath(T);
```

Algorithm 3: Pseudocode of the P–RRT–Path algorithm. The member function *addNode* inserts a vertex into the tree, similarly the function *addEdge* inserts an edge. Function *initializeTemporaryGoal* selects the proper region of the supplied guiding path as a temporary goal node. Function *goalReached* checks whether the planner has reached the goal region. Function *createRandomSample* returns a new configuration, depending on the current planner state, according to the later described functionality. Function *nearestVertex* picks a nearest neighbor from the tree. Function *updateAtomScale* changes the multiplier of atomic radii (also explained later). Function *limitNodeDistance* checks whether the distance of the supplied two nodes does not cross the set threshold and if it does, moves the newly sampled node. Function *limitLigandAngles* clamps the difference in the intramolecular angles to a preset value. Function *checkNodeTunnelDistance* checks if the distance of the sample from the guiding path does not cross a preset threshold. Function *checkTransitionTRRT* calls the T–RRT algorithm, to determine, whether the transition is admissible, with regards to intramolecular potential energies. Function *collisionFree* checks whether a trajectory formed between the two supplied configurations is collision–free. Function *moveTemporaryGoalNode* updates the current temporary goal node, if is has been reacehd by the tree and finally, function *extractPath* extracts the final path from the tree by iterating through it over parent nodes.

is stuck, the tree growth has stopped progressing and we take corresponding corrective actions to enable further growth, as described in the previous and the following sections.

The planner remains in the stagnation state and the corrective actions remain in place until the current temporary goal node is reached. The planner then switches back to the normal state and the corrective actions are disabled until the planner reaches the stagnation state again.

3.4.2 Atom Scaling

When interacting with each other, both ligands and proteins are not rigid, but adapt to, and affect each other. Molecular dynamics simulations of protein and ligand molecules have revealed the dynamic nature of tunnels, which can move, merge and adapt themselves to the shape of the passing ligand. The ligand can in turn adapt its shape to facilitate its passage through the tunnel.

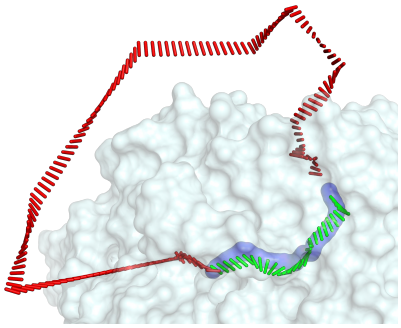


Figure 3.4: An example of a valid tunnel (green) following the supplied auxiliary guiding path (blue) and an invalid tunnel (red) reaching the free space outside the protein molecule (gray).

To emulate this behavior, we have employed the technique of scaling down the Van der Waals atomic radii. This technique is widespread in the protein planning field and has been successfully utilized, e.g., in the MoMA–LigPath planner [13].

The shrinking is performed in complex parts of the environment, when the planner is in the stagnation phase (see *Figure 3.7*). If the planning proceeds well, the radii are not scaled at all. If the planner gets stuck, the radii are slowly scaled down, according to the number of iterations, which have failed to reach the goal node. This continues, until the temporary goal node is reached, or until a certain scaling factor is reached (in our case 0.5). Once a goal node is reached, the scaling factor is reset to the initial value (in our case 0.8) and planning continues.

3.4.3 Auxiliary Guiding Path Distance Limiting

When finding a path using the RRT-Path algorithm, the *goal bias* parameter $g_b \in [0, 1]$ determines the probability of creating a new sample inside the auxiliary guiding path. A sample can be created anywhere in the available configuration space with the probability of $(1 - g_b)$. This ensures that the algorithm can select a better path when stuck in a local minimum (a bug trap scenario). This however, leads to the possibility of finding a path completely different from the guiding path, when the task is particularly difficult. See *Figure 3.4* for an example of this behavior.

To ensure this scenario doesn't occur, a condition was devised, which checks every newly created sample. If the surface of a ligand atom nearest to the auxiliary guiding path is farther than a preset parameter δ_{tunnel} (in our case 1,5 Å), the sample is discarded. This eliminates paths which don't follow the auxiliary guide, while giving the algorithm some wiggle room to escape from local bug trap type minima.

3.4.4 Receptor Atoms Culling

The auxiliary guiding path effectively restricts the configuration space of the ligand molecule. Since any samples, which are farther than a certain distance from the CAVER tunnel are discarded, receptor atoms located in this these "disabled" regions cannot possibly collide with the ligand molecule. This motivates us to reduce the complexity of collision detection by decreasing the number of atoms participating in it. When the planner is being initialized, distances of all the atoms are checked. Those which are farther than twice the parameter δ_{tunnel} from a CAVER tunnel's surface, are discarded. *Figure 3.5.a,b* shows the process. In the illustrated example, the amount of atoms has been reduced more than 10 times.

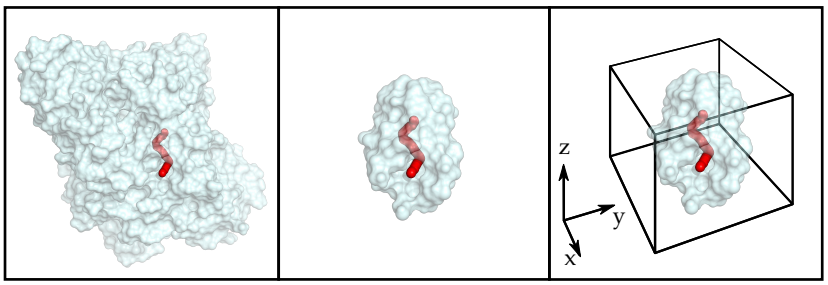


Figure 3.5: An example of a culled receptor protein molecule. Subfigure a) *shows the original protein molecule (gray), consisting of 8300 atoms, with a guiding tunnel denoted in red. As is apparent, the path does not span the entirety of the molecule. A majority of receptor atoms are therefore irrelevant for planning a path inside the tunnel.* Subfigure b) *shows the culled protein molecule, consisting of 750 atoms, with only the atoms relevant to the tunnel denoted red.* Subfigure c) *displays the final axis–aligned bounding box, used for sampling.*

After this, an axis aligned bounding box is created around the protein molecule. This serves as a sampling boundary, since ll samples created outside the guiding path must lie in this bounding box. See *Figure 3.5.c*

The combination of these two approaches speeds up the implementation of collision detection roughly 1.25 times, both due to the speedup in collision detection[4] and the limiting of the amount of invalid samples created.

3.4.5 Precomputed Ligand Conformations

Flexible ligand molecules can more easily navigate much smaller and more complex environments, when compared to rigid molecules. However, flexibility comes with the cost of a significantly expanded configuration space. Each degree of freedom of the ligand molecule presents an additional dimension which we need to sample, thereby slowing down the search. To mitigate this issue a compromise has been conceived. The planner has two states, a *normal* state, for when the search is progressing normally, and

[4]Although the computing time saved in one collision detection might seem negligible, this function can be called millions of times for one planning task and therefore presents a significant speedup

a *stagnation* state, when a tree growth stagnation has been detected and the search tree has stopped growing (see *Section 3.4.1* for details regarding growth stagnation detection).

When the planner is initialized, several molecular conformations are precomputed:

- The **lowest energy** conformation (*Figure 3.6.a*), which is found in the potential energy function space by random sampling. This conformation is used during the *normal* planner phase in order to avoid slowdowns caused by waiting for results from the AutoDock Vina library wrapper.
- The **narrowest** conformation (*Figure 3.6.b*), also created by sampling ligand angle values, but chosen by selecting a conformation with the smallest distance between two most distant atoms and an atom most distant to a line created by the two most distant atoms. This conformation was chosen to fit the potential narrow corridors, which the most compact configuration might be too wide to fit in.
- The most **compact** conformation (*Figure 3.6.c*), created by sampling ligand angle values and picking the most compact ligand spatial arrangement.

Figure 3.8 shows the a ligand molecule depicted in the previous three conformations.

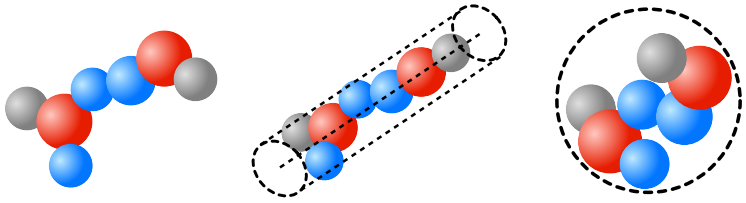


Figure 3.6: Illustration of different types of conformations of a ligand molecule. a) *The lowest potential energy conformation, used during the normal planner phase.* b) *The narrowest conformation, used during tree stagnation. The illustrated cylinder represents the utilized metric. Two most distant atoms are found and a line is created between them. A conformation, which has the shortest distance from that line to any atom, is then picked.* c) *The most compact conformation, with the shortest maximum distance, between any two atoms, also used during tree stagnation*

Upon initialization, the planner starts in the *normal* state and only utilizes the *lowest energy* conformation to avoid unnecessary slowdowns. When a particularly difficult part of the environment is encountered and search tree growth stagnation is detected (see *Section 3.4.1*), the planner switches to the *stagnation* state. During this state, the configuration space is expanded by the degrees of freedom of the ligand molecule. A parameter $p_{altConf}$ denotes the probability of choosing one of the two precomputed ligand conformations (*compact* and *narrowest*, each with probability $\frac{p_{altConf}}{2}$). If neither of the two is randomly chosen, new conformation is generated, using the Transition–RRT algorithm (see *Section 3.4.7*) with regard to the potential energy function. A workflow of the entire process is illustrated in *Figure 3.7*.

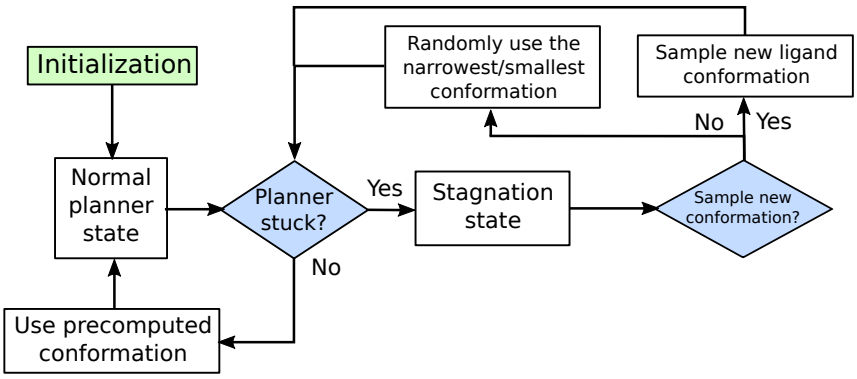


Figure 3.7: Workflow diagram of the conformation sampling procedure. The planner consists of two states, the normal *state, which utilizes the default precomputed ligand conformation, and the* stagnation *state, which occurs when the search tree growth has stopped. During the stagnation state the planner can use one of the two precomputed conformations, or sample an entirely new conformation, depending on the parameter* $p_{altConf} \in [0, 1]$

This significantly improves the planner progress rate while also maintaining the benefits of flexible ligand planning and avoiding unnecessary overhead processing costs.

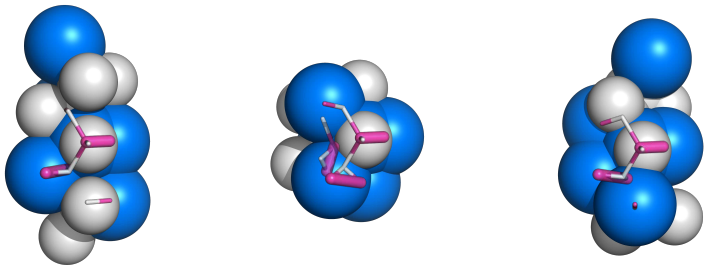


Figure 3.8: Conformations of the m040 ligand molecule. a) *depicts the lowest potential energy conformation,* b) *the most compact conformation and* c) *the narrowest conformation.*

3.4.6 Ligand Flexibility Limiting

The degrees of freedom present in the form of rotatable bonds are decoupled from the rigid six degrees of freedom of the ligand. To prevent large sudden conformation changes of the ligand, we have implemented a limiting function. When a new sample is created, its molecular angles are compared to that of its nearest neighbor. If they differ by more than a preset parameter Δ_α (0.25 rad in our case), they are clamped so that the difference is equal to the value of the parameter.

This process ensures smoother transitions between conformations and allows for faster planning since the number of rejected trajectories is significantly reduced.

3.4.7 Ligand Potential Energy Limiting

Intramolecular ligand energy is a function of the ligand's internal angles, i. e., its conformation. When planning with a flexible ligand, we assume that the molecule can take on any conformation, regardless of its potential energy. Thus, the found trajectory might actually not be feasible in a real scenario. Taking potential energy of the ligand molecule into account enables us to further judge the feasibility of the found pathway.

The Transition–RRT algorithm [20] combines the rapid exploration ability of the RRT algorithm with stochastic optimization methods through transition tests (explained in the section below) to accept or reject sampled conformations. The planner utilizes the notion of a *minimal work path*, which enables a way of comparing path costs. It also implements a self tuning parameter, which controls its exploratory behavior, of preferring low cost valleys and saddle points of the cost space, and thus avoiding high–cost configurations. An example of the resulting tree graph is available in *Figure 3.9*. Pseudocode of the algorithm is available in *Algorithm 4*.

Transition Test

The transition test function is presented in *Algorithm 4*. The function filters transitions which would result in the tree searching a high cost region, that might otherwise be avoided. The function first computes a probability of transition (referred to as the Boltzmann probability) as follows:

$$p_{i,j} = \begin{cases} exp\left(-\frac{\Delta c^*_{i,j}}{K \cdot T}\right) & \text{if } \Delta c^*_{i,j} n \geq 0 \\ 1 & \text{otherwise.} \end{cases}$$

where $\Delta c^*_{i,j} = \frac{c_j - c_i}{d_{i,j}}$, is the slope of the cost (the cost difference divided by the node distance). This ensures that downhill transitions are automatically accepted, and the steepest uphill transitions have the lowest chance of acceptance. K is a normalization constant, in our case $K = 2$. Finally, T is an adaptive parameter representing current *temperature* (and consequently transition probabilities).

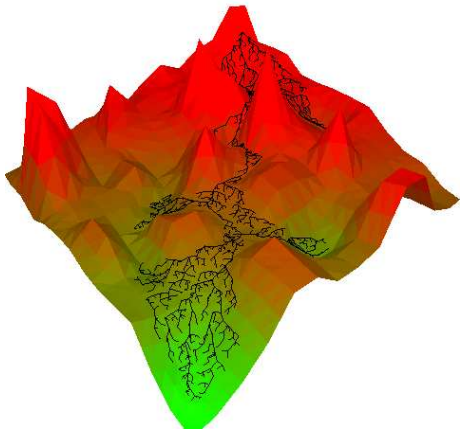


Figure 3.9: An example of a solution originating from the T-RRT algorithm. Note how the resulting graph avoids high–cost slopes, while still being able to climb out of local minima. Image courtesy of [20]

The temperature parameter changes during the planning to reflect the current state of tree growth. Low temperature limits the expansion to low cost regions of the cost space, while high temperatures allow the planner to reach high cost slopes, when stuck in a local minimum. The adaptive tuning algorithm is presented in *Algorithm 5*.

Analysis of the results of the algorithm shows that although the planner is not mathematically guaranteed to find the optimal solution, a great amount of performed experiments has shown, that T–RRT provides significantly more optimal solutions from the potential energy standpoint.

```
1  Function BuildTRRT
       input : Initial configuration q_init, configuration space C, goal region G and the
               cost function f_c
       output: TRRT tree graph (V,E)
2      V ← q_init;
3      E ← ∅;
4      while ¬goalReached() do
5          q_rand ← createRandomSample(C);
6          q_near ← nearestVertex(q_rand, V);
7          if not Extend(V, q_rand, q_near, q_new) then
8              Continue;
9          end
10         if TransitionTest(f_c(q_near), c(q_new), d_near−new) and
             MinExpandControl(V, q_near, q_rand) then
11             V ← V ∪ q_rand;
12             E ← E ∪ (q_near, q_rand);
13         end
14     end
15     return (V, E);
```

Algorithm 4: Pseudocode of the T-RRT algorithm. Function *goalReached* checks whether a vertex of the search tree lies within the goal region G. Function *createRandomSample* samples the configuration space for a new state. Function *nearestVertex* returns a nearest neighbor of the supplied vertex, located in the search tree. Function *Extend* checks whether the newly sampled state lies within a preset δ parameter away from the nearest tree node and returns a new parameter q_{new} which satisfies this condition. The Function *TransitionTest* filters parameters which do not satisfy the cost limits imposed by the planner (this is further explained in *Section 3.4.7*). Finally, function *MinExpandControl* forces to planner to maintain a minimal exploration rate. Similarly to the RRT algorithm, the search graph is a tree, so the final path can easily be extracted by iterating over parent nodes of the node in the goal region, once it is reached.

3.5 Unsuccessful Modifications

During our attempts to improve the performance of our planner, we have devised several improvements which, although seemingly promising ideas, did not convey any significant advantages. This section briefly describes the ideas and problems faced when implementing them.

When a tree is successfully expanded, the newly added sample can be used to increase the growth of the tree, by creating similar samples which only differ in certain preset steps. In our implementation, the node was moved forward and backward along each axis by the largest possible growth step. The new samples were then tested for collisions and, if eligible, inserted into the tree. See *Figure 3.10* for details.

```
1  Function Adaptive Tuning
       input  : Initial cost c_i, final cost c_j, node distance d_{i,j}
       output: Boolean transition feasibility
2      if c_j ≤ c_i then
3          return False;
4      end
5      p ← exp((−c_j − c_i)/d_ij / K·T);
6      if Rand(0,1) ≤ p then
7          T ← T/α;
8          nFail ← 0;
9          return True;
10     end
11     else
12         if nFail ≥ nFail_max then
13             T ← T · α;
14             nFail ← 0;
15         end
16         else
17             nFail ← nFail + 1;
18         end
19         return False;
20     end
```

Algorithm 5: Parameter T adaptive tuning algorithm. The algorithm alters the parameter throughout the search process. When the planner is initialized, T is set to a low value. This filters out all states except for those with the lowest costs. During the search, the variable $nFail$ keeps track of failed attempts at creating a transition. When this value crosses a preset threshold $nFail_{max}$ (in our case $nFail_{max} = 100$), the temperature is multiplied by α. Correspondingly, after a new transition is successfully added to the tree, temperature is divided by the same factor α. This ensures that the temperature automatically adapts to the current state of the planner.

During performance testing, it was revealed that although the forced expansions stimulated tree growth, the resulting performance was significantly slower, when compared to the original implementation. This was partly due to the increased frequency of collision checking and partly due to a much larger amount of nodes in the search tree. This resulted in increased collision detection times and consequently about 1,5 times worse performance overall than without forced node expansions.

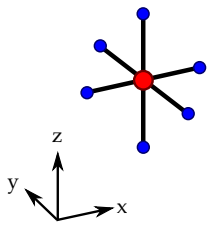


Figure 3.10: An illustration of forced node expansion. The original node (red) is used to create new samples (blue) by moving along each axis.

In the original implementation, when a ligand molecule is loaded by the Vina AutoDock wrapper, it is automatically centered in its coordinate system. When planning in tight spaces, this might cause problems, since even a slight rotation can place the molecule in a colliding configuration. We have devised a way

of eliminating this, by placing the molecular coordinate system origin on the "front" end of the molecule, i.e. the end closer to the exit. See *Figure 3.11* for illustration. The theory was that this gives the molecule a higher chance of a collision–free configuration.

In reality this caused a significant slowdown. The planner struggled to find collision–free configurations, despite our attempts to slow down rotation of the molecule during planning steps. The overall search time was nearly two times worse than without this modification.

3.6 Final Implementation

The planner was implemented in C++11. The Boost library [10] was utilized for runtime parameter parsing and other miscellaneous tasks. The final implementation employed a number of other libraries, described in the following sections. The GLM library [16] was utilized in order to speed up vector, matrix and quaternion computations.

Two variants of the algorithm have been developed. The first variant with single auxiliary guiding path support, for performance demanding tasks, and a second variant, with multiple guiding paths support, for special case scenarios (see *Section 3.7* for details).

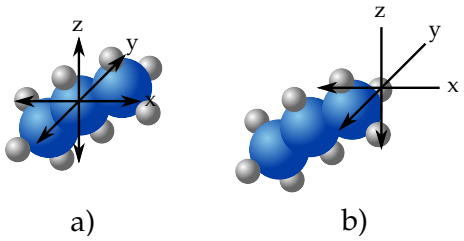


Figure 3.11: An illustration of ligand endpoint coordinate system origin. a) *shows the standard implementation and* b) *the proposed change.*

3.6.1 Nearest Neighbor Search

During a single run of a path planning algorithm a nearest neighbor search is performed at every iteration of the planner. This makes the task of searching the closest node and the speed of its implementation crucial to the overall performance of the resulting implementation. To ensure the best possible performance, we have utilized the excellent *MPNN library* [7], which utilizes kD-trees and supports n-dimensional manifolds[5]. The library offers low memory footprint while being easy to integrate and use.

Because ligand flexibility is adaptively toggled during path searching and ligand angle changes do not significantly alter the geometry of the molecule, when combined with ligand angle change limiting, the cost of considering ligand angles in the distance metric would unnecessarily slow the planner down with little to no benefits to overall performance. The distance metric is as follows:

$$d(n_i, n_j) = \sqrt{||v_i - v_j|| + 10||q_i - q_j|| + \sum_k (|a_{i,k} - a_{j,k}|)}$$

where $||\cdot||$ is an L2 vector norm, n_i, n_j are tree nodes, v_i, v_j are the respective coordinate vectors of the vertices, q_i, q_j are quaternions representing spatial orientations of

[5] Cartesian products of Euclidean one-space, circle, and three-dimensional rotation group SO(3)

the two corresponding graph nodes and a_i, j is the k–th intramolecular angle of the i–th ligand. The number 10 was chosen experimentally and represents the weight of rotating the molecule, to limit excessive rotations when exploring the complex and narrow environments of protein cavities.

3.6.2 OZCollide Library

Protein molecules can reach significant complexity, with atom counts ranging in the orders of tens of thousands of atoms. Each of these atoms presents a geometric primitive, which has to be virtually represented to facilitate the possibility of detecting collisions. There exist several approaches of representing the molecules ranging from creating a 3D model to representing each of the atoms as a sphere and manually checking the molecules for collisions, each of which are employed by existing solutions ([1], [2], more details available in [18]). The results of our previous work [41] have shown that the most effective approach consists of utilizing the OZCollide [23] external library and representing the atoms as geometric primitives inside the library.

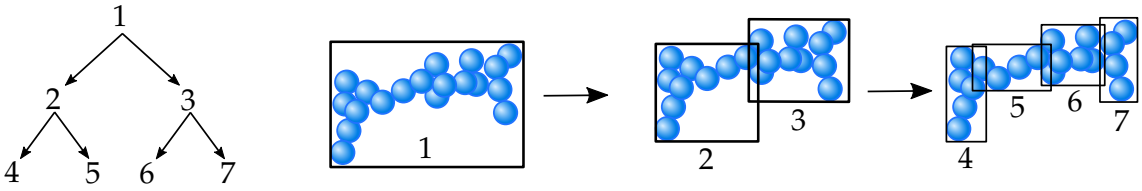


Figure 3.12: Illustration of the AABB trees structure. AABB trees function by checking collisions using several layers of geometric primitives. When the tree is being constructed, the structure is iteratively split into smaller and smaller segments. These segments are then hierarchically organized so that two smaller segments cover the same area as the parent larger segment. The tree on the left illustrates the hierarchy of the structure. When checking collisions first the root bounding box is checked, if it is not colliding, no part of the more complex object contained inside it can possibly be colliding either and the query is resolved in a significantly swifter manner. If it is colliding, its children are checked for collisions too. If any of the children are colliding their children are recursively checked until we reach a leaf node, at which point we check for collision with the contained object itself. The structure offers a significant speedup in non–colliding scenarios at the cost of a small overhead in colliding scenarios.

The OZCollide offers speed improvements for collision checking consisting of e.g. automatic axis–aligned bounding box trees building (see *Figure 3.12* for an illustration of the structure). Our previous work has shown that it performs nearly orders of magnitude better than some of the existing alternatives in the protein path planning scenario. Although the original website for its distribution is no longer active, it has been archived in a github repository[6] and is still available for use today. One feature which we were unable to implement was collision checking between two AABB trees. We have therefore resorted to iterating over all ligand atoms and checking whether each of them is colliding with the receptor molecule. Since the complexity of checking collisions in AABB trees

[6] https://github.com/jslee02/OZCollide (accessed 28-04-2018).

is $log(n)$, the resulting complexity is $m \cdot log(n)$, where m is the number of ligand atoms and n is the number of receptor atoms. By using receptor atom culling from *Section 3.4.4*, n can be further decreased.

3.7 Multiple Auxiliary Paths Algorithm

CAVER 3.0 is capable of exporting multiple tunnels, originating from a given active site, when analyzing the supplied receptor molecule. To utilize this fact, a modification of the RRT–Path algorithm, capable of searching using multiple guiding paths, was devised. The algorithm performs analogously to the single guiding path variant, but employs multiple temporary goal nodes, which are chosen at random when sampling new nodes. This allows the algorithm to consider all possible tunnels when finding a path. When the algorithm starts, multiple guiding paths are loaded. When creating a new sample from a temporary goal node, all paths are considered using a roulette wheel algorithm, according to their weights (see *Section 3.7.1* for details). This approach ensures that most suitable tunnel is considered with a high probability, at the cost of a slight slowdown. The algorithm is similar to the original RRT–Path algorithm, except for the function *createRandomSample*, which is modified to accommodate the possibility of multiple auxiliary paths (see *Algorithm 6* for details).

1 **Function** *createRandomSampleMultiple*
  input : Set of temporary goal nodes G_t, set of path weights W_p
  output: New temporary goal node
2 $p \leftarrow \mathbf{Rand}(0,1)$;
3 $index \leftarrow \mathbf{getPathIndex}(p)$;
4 **return** $G_t[index]$;

Algorithm 6: Multiple auxiliary paths sampling function. First, a random number $p \in [0,1]$ is generated. Then, a guiding path is chosen according using the roulette wheel function *getPathIndex*, which selects one of the paths using their weights as a distribution function. Then, the temporary goal node, belonging to the respective path is returned.

After this algorithm is run multiple times, a statistical analysis of the results is performed. The probabilities of finding a trajectory using a certain tunnel are computed and the tunnels are ordered. We can then use the tunnels with the highest probabilities in the single auxiliary guiding path algorithm, thus avoiding planning with unsuitable tunnels.

3.7.1 Dynamic Auxiliary Path Weighting

When the tunnels are loaded during the algorithm's initial phase, they are assigned weights according to the following formula:

$$w_i = \left(\frac{1}{n} + \frac{2}{||tgn_i - init_i|| + 1} + minWidth_i \right) \cdot \nu$$

where n is the number of tunnels, tgn_i is the location of the i-th temporary goal node, $init_i$ is the location of the tunnel's initial node, $minWidth$ is the size of the tunnel's bottleneck and ν is the normalization constant, satisfying $\sum_i w_i = 1$.

This solution ensures that all tunnels are sampled with a non–zero probability, while the more suitable wider tunnels are sampled more often. Once any of the temporary goal nodes is moved, the weights are recomputed, to better reflect the situation. The second fraction ensures that tunnels with slow progress are sampled more often.

During experiments, the dynamic weighting proved to speed up the planner progress. Despite this, the planner is still slower than the original single auxiliary path implementation. This variant should therefore only be used for the initial assessment, to aid the decision of which tunnels should be utilized for sampling.

3.8 List of Symbols

The resulting implementation uses a number of parameters, affecting its performance. A list of the most important parameters, with their units and meanings, has been provided in *Table 3.1*.

Symbol	Meaning [Units]	Name in the program	Chosen value
$p_{altConf}$	Probability of choosing one of the precomputed ligand conformations $[-]$	PROBABILITY_OF_ ALTERNATIVE_CONF_ WHEN_STUCK	0,15
d_α	Maximum difference of intramolecular angles between two ligand conformations $[radians]$	MAXIMUM_DOF_CHANGE	0,25
g_b	Temporary goal node sampling bias $[-]$	RRT_PATH_FOCUS_ PROBABILITY	0,9
δ_{tunnel}	Maximum distance of tree nodes from the auxiliary guiding path [Å]	RRT_PATH_MAX_DISTANCE_ CAVER_TRAJECTORY	1,5
I_s	Number of iterations without progress for the planner to become stuck $[iterations]$	ITERATIONS_UNTIL_STUCK	1000

Table 3.1: A list of symbols used throughout this thesis

4 Experiments

The final implementation was tested on three supplied protein receptor molecules, 4L2L, DHA and DHA–AWT. The state–of–the–art algorithm MoMA–LigPath was used as a reference for comparison of the algorithm's performance. Default parameter values were used for running MoMA–LigPath.

Each of the algorithms was tested on 100 conformations of the receptor molecules. For each of the receptor conformations, up to three most suitable tunnels (found by CAVER 3.0, with a probe size of 0,9 Å) were used for planning. The supplied ligand molecules were 1-Chloropropane (11 atoms, 2 DOF, m003), 1,2-Dichloroethane (8 atoms, 1 DOF, m037t), 1,3-Dichloropropane (11 atoms, 2 DOF, m038t), 1,5-Dichloropentane (17 atoms, 4 DOF, m040), 2-Chloropropane (14 atoms, 3 DOF, m056), 1,2,3-Trichloropropane (11 atoms, 2 DOF, m080) and Propionyl–chloride (10 atoms, 1 DOF, m113r). See *Figure 4.1* for illustrations.

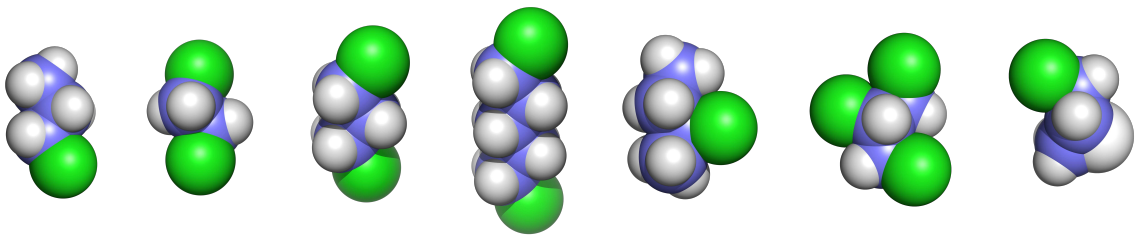


Figure 4.1: Tested ligand molecules. From left to right, 1-Chloropropane, 1,2-Dichloroethane, 1,3-Dichloropropane, 1,5-Dichloropentane, 2-Chloropropane, 1,2,3-Trichloropropane and Propionyl–chloride.

Our P–RRT–Path algorithm can in principle detect pathways both from inside and from outside of the protein molecule.However, to offer a fair comparison to MoMA–LigPath, only pathways from inside the molecules were used, since that is the only type supported by MoMA–LigPath. Our implementation was allowed to scale the atoms of both the ligand and protein from $0,8$ to $0,5$ times the original atomic radius. The tests were rated according to their success rates, i.e., the percentages of successfully finding a path over all the receptor conformations and the repeated runs for each conformation. Each of the ligand molecules was used for planning in each of the supplied tunnels

for each of the conformations of the three protein molecules 10 times. Therefore, for each receptor, up to $100^{(conformations)} \cdot 7^{(ligands)} \cdot 10^{(runs)} \cdot 3^{(tunnels)} = 21000$ runs were required. The tests were performed on the *MetaCentrum*[1] computing cluster, using 8 CPU cores and 4 GB of RAM. Maximum runtime was limited to 10 minutes.

4.1 Planner Parameter Space Exploration

The proposed P–RRT–Path planner uses several parameters, affecting its performance (available in *Table 3.1*. We have tested our implementation to determine the optimal parameter values, providing the best overall performance. The planner was run 10 times for each parameter value using the *DHA* receptor and the *m080* ligand.

4.1.1 Goal Bias

The goal bias determines the probability of choosing the temporary goal node (and subsequently the guiding path) to create new samples. Higher values mean the temporary goal node will be chosen with a higher probability.

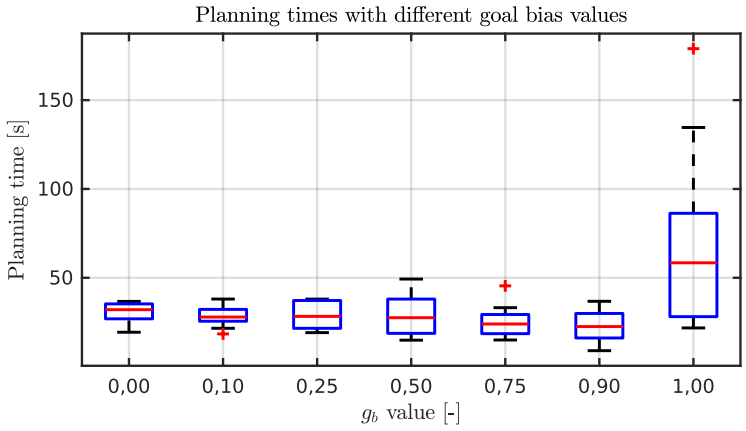


Figure 4.2: Planning times with respect to different goal bias g_b values. The mean values slowly decrease, as we g_b increases from 0 to $0,9$. This is expected, since the auxiliary paths represent additional knowledge about the void spaces. When goal bias $= 0$, the RRT-Path algorithm degenerates into a simple RRT, since it doesn't utilize any knowledge about the environment. When goal bias $= 1$, all samples are generated inside the current temporary goal node. This causes the algorithm to slow down, since it is unable to overcome bug–trap–type local minima as easily.

The results in *Figure 4.2* show that the optimal value lies in the region of $0,9$, which is the value selected in the original RRT–Path implementation [43] and the value selected in our experiments.

[1] https://www.metacentrum.cz/en/ (accessed 28-04-2018).

4.1.2 Precomputed Conformations Probability

The parameter $p_{altConf}$ denotes the probability of selecting a precomputed conformation (the most compact or the narrowest) instead of sampling an entirely new random one. The higher the probability, the less likely the algorithm is to sample a new conformation.

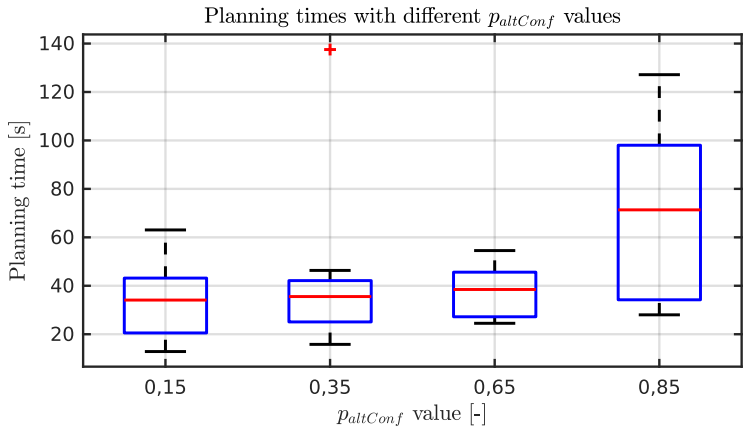


Figure 4.3: Planning times with respect to different probabilities of choosing alternative pre–computed configurations. The mean values of planning times slowly increase as we increase the probability.

Figure 4.3 shows that the planning times increase as the probability increases. This might be attributed to the narrow spaces in the protein tunnels, which results in increased collision rates for the precomputed conformations. We have therefore selected the probability $p_{altConf}$ value of $0,15$.

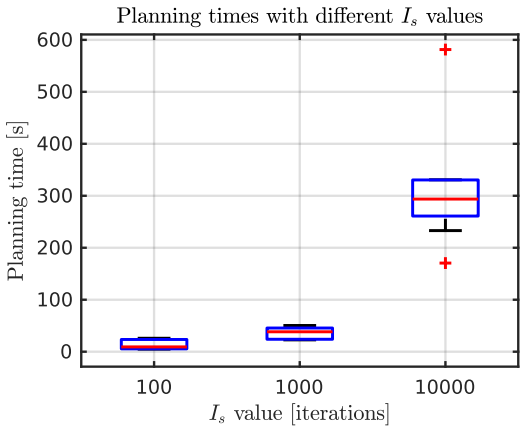
4.1.3 Planner Stuck Detection Threshold

Our implementation detects when the search tree has stopped progressing, and is able to mitigate this by countermeasures, which include decreasing the scale of atomic radii. The threshold I_s denotes how many iterations without progress are necessary for the tree to be considered stuck.

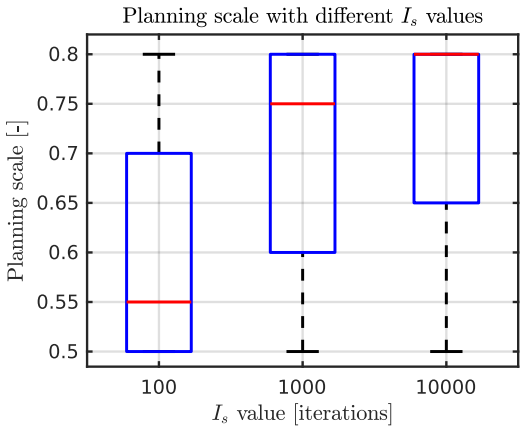
The results in *Figures 4.4.a* and *4.4.b* show the magnitude with which this parameter can affect the planner performance. Low values significantly speed up the performance, while decreasing the scale with which the planner operates. This decreases the accuracy and feasibility of the found trajectories. Higher values on the other hand significantly improve the accuracy of the found trajectories, with the cost of severely reduced performance by nearly 100 times. We have selected a value of 1000 iterations, which balances these two characteristics, offering accurate planning scale with good performance.

4.2 4L2L

The A4 hydrolase (PDB ID 4L2L) receptor consists of 9666 atoms. Its structure is depicted in *Figure 4.5*. The tCONCOORD tool [38] was used to generate 100 conformations of the protein. The initial PDB structure was minimized by GROMACS 4.0.7 [17],



(a) Planning times with different thresholds of iterations.



(b) Planning scales with different thresholds of iterations.

Figure 4.4: Planning times with different thresholds of iterations. Although lower thresholds bring significantly faster planning times, they also denote much lower atomic scales and thus less accuracy and trustworthiness of the found trajectories.

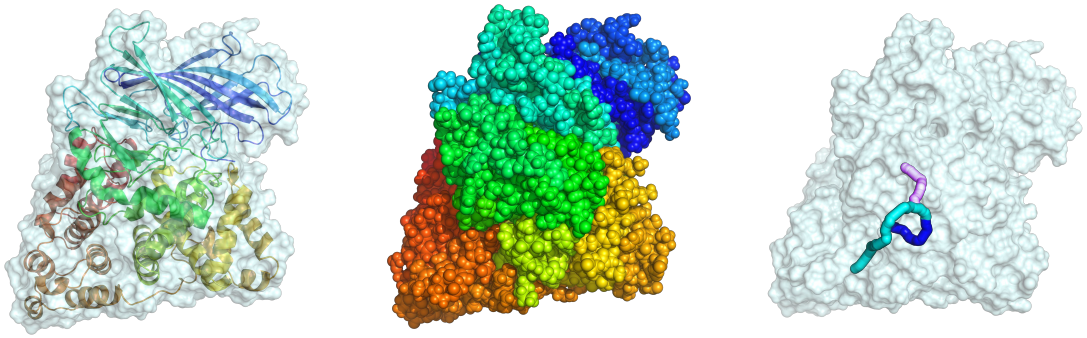
using the input parameters recommended on the example page of tCONCOORD [37]. The protein conformations were generated using the default settings for tCONCOORD in the PERT mode which only perturbs the starting configuration of atoms instead of complete randomization. The tunnels leading to the active site defined by residues 134 (Alanine) and 311 (Phenylalanine) were computed by CAVER 3.0 in each conformation, using the probe 0.9 Å. Unlike in the other two receptor molecules, three tunnels are present in each of the conformations.

The characteristics of the tunnels are shown in *Figure 4.6*. According to bottleneck sizes, there is a substantial amount of tunnels which are too narrow to pass the molecule, and will either be considered packed or require substantial atom downscaling. There is also a substantial amount of tunnels surpassing the length of 20 Å, which require long planning times to find the pathway.

The results for our P-RRT-Path algorithm are presented in *Table 4.1*, the results for MoMA–LigPath in *Table 4.2* and *Figures 4.7* and *4.8*.

As we can see, our P-RRT-Path algorithm surpassed MoMA–LigPath in nearly all aspects and scenarios. The reported amount of pathways is fairly substantial, with one ligand surpassing 70% of tunnels reported as feasible for planning. Furthermore, the numbers are quite consistent over the three tunnels, which implies that the rest of the receptor conformations were unsuitable for planning. The number of found paths overall is quite high, considering the fact that the tunnels were not tested for ligand traversability, and a majority of them may in fact be packed.

Our results contrast with the results of the MoMA–LigPath algorithm, which failed to find any pathways at all for 5 of the 7 ligands. It succeeded to find a path for the *m003* ligand, but the reported amount of pathways is lower than in our algorithm. One contrasting scenario is the *m113r* ligand molecule, for which our algorithm failed to find



a) b) c)

Figure 4.5: Illustration of the 4L2L protein molecule. Subfigure a) *shows the surface and ribbon model.* Subfigure b) *shows the spherical space–filling model and* Subfigure c) *shows an example of the tunnels in relation to the protein surface.*

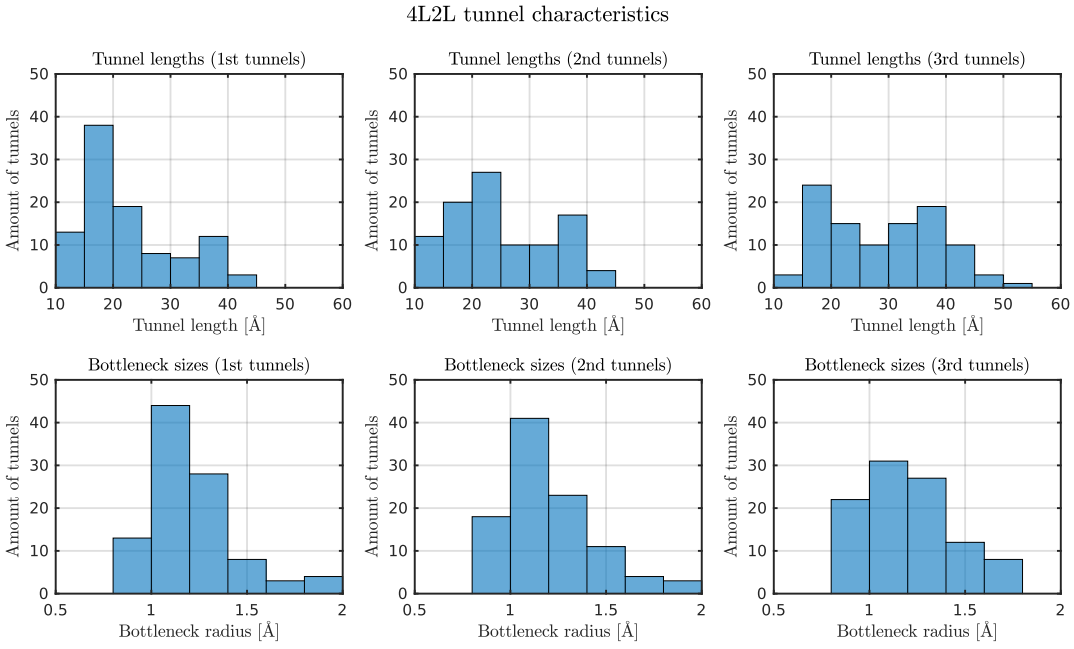


Figure 4.6: Tunnel characteristics for the 4L2L receptor molecule. We can see a large amount of secondary and tertiary tunnels reaching tunnels lengths over 20 Å and also a bottleneck radius smaller than 1,5 Å. These tunnels will require long planning times and significant atom downscaling, in order to be used to find a pathway.

P-RRT-Path success rates in the 4L2L receptor [%]							
	Ligands						
Tunnels	m003	m037t	m038t	m040	m056	m080	m113r
1st tunnels	42,80	72,70	36,70	16,30	29,40	27,20	0,00
2nd tunnels	38,40	65,80	33,30	16,30	28,40	27,20	0,00
3rd tunnels	42,20	72,00	39,30	21,80	33,10	32,70	0,00

Table 4.1: Overall percentages of successfully found pathways for P-RRT-Path in the 4L2L receptor. Our algorithm exhibits a fairly consistent performance over the three most feasible tunnels. With the m037t *ligand reaching 72,7% of successfully found paths. This is contrasting to the performance for the* m113r *ligand, for which our algorithm failed in all cases. This may be attributed to the fact that the* m113r *molecule is more spherical than others and thus may require wider tunnels to successfully find a path.*

MoMA–Ligpath success rates in the 4L2L receptor [%]							
	Ligands						
Tunnels	m003	m037t	m038t	m040	m056	m080	m113r
1st tunnels	3,80	0,00	0,00	0,00	0,00	0,00	5,90
2nd tunnels	3,60	0,00	0,00	0,00	0,00	0,00	7,30
3rd tunnels	3,40	0,00	0,00	0,00	0,00	0,00	5,40

Table 4.2: Overall percentages of successfully found pathways for MoMA–LigPath in the 4L2L receptor. As we can see MoMA–LigPath exhibits a surprisingly poor performance, when compared to our algorithm. It successfully found a path for only 2 ligands out of the 7, m003 *and* m113r*. For the* m003 *ligand, the rates are substantially lower than for our algorithm. For the* m113r *MoMA–LigPath has succeeded, where our algorithm has failed. This might be the result of the different nature of MoMA–LigPath, which is able to alter the positions of the receptor atoms to facilitate the passage of the ligand.*

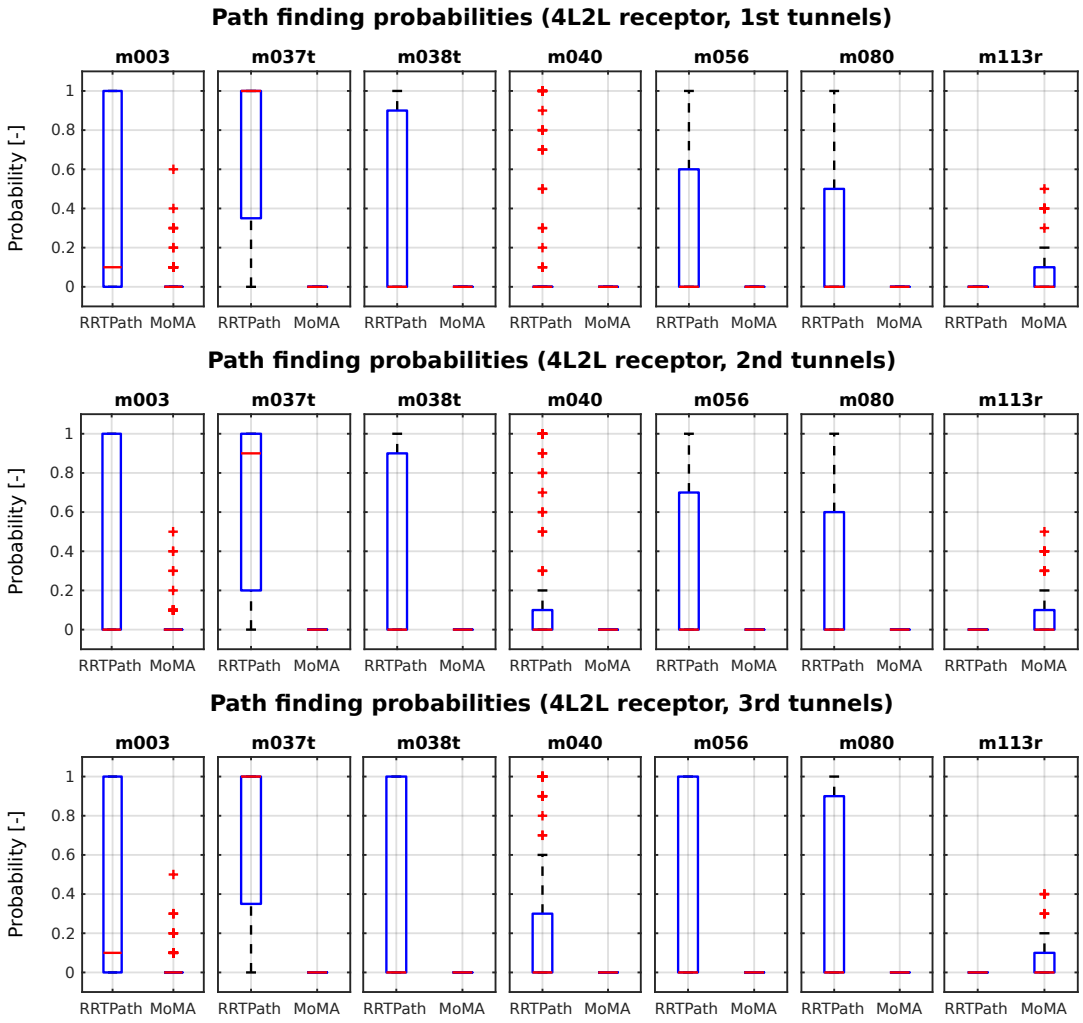


Figure 4.7: Path finding probabilities for 4L2L receptor tunnels. As we can see, our algorithm surpasses MoMA–LigPath in almost all scenarios, the exception being ligand m113r*. The possible reason is explained in* Section 4.2*. The results are consistent over all tunnel types, which suggests a high percentage of the tunnels, for which the pathways were not found, might be packed.*

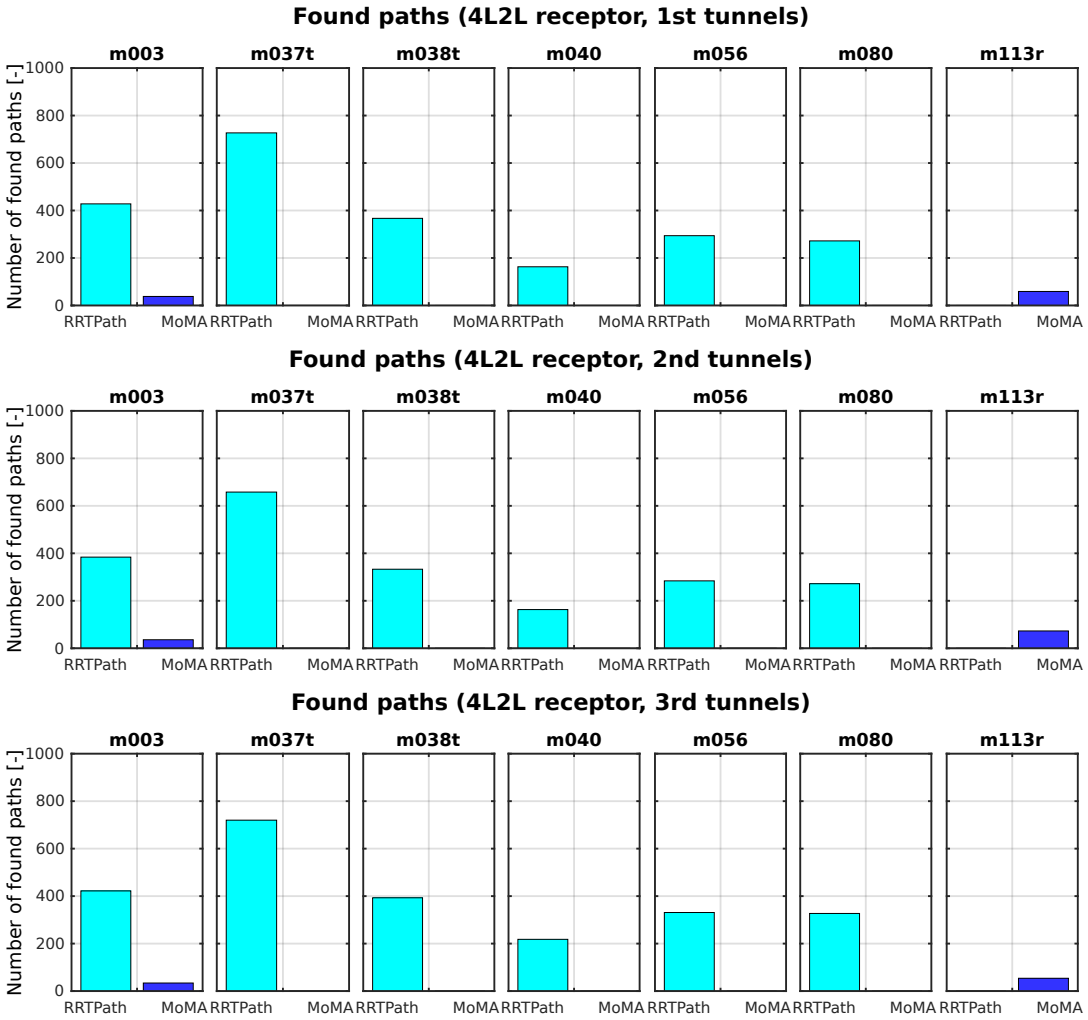


Figure 4.8: Number of found paths for 4L2L receptor tunnels over all receptor conformations. The results confirm the previous conclusions, with our algorithm surpassing MoMA–LigPath in all cases, except for the m113r *ligand. The numbers are fairly consistent over all tunnel types.*

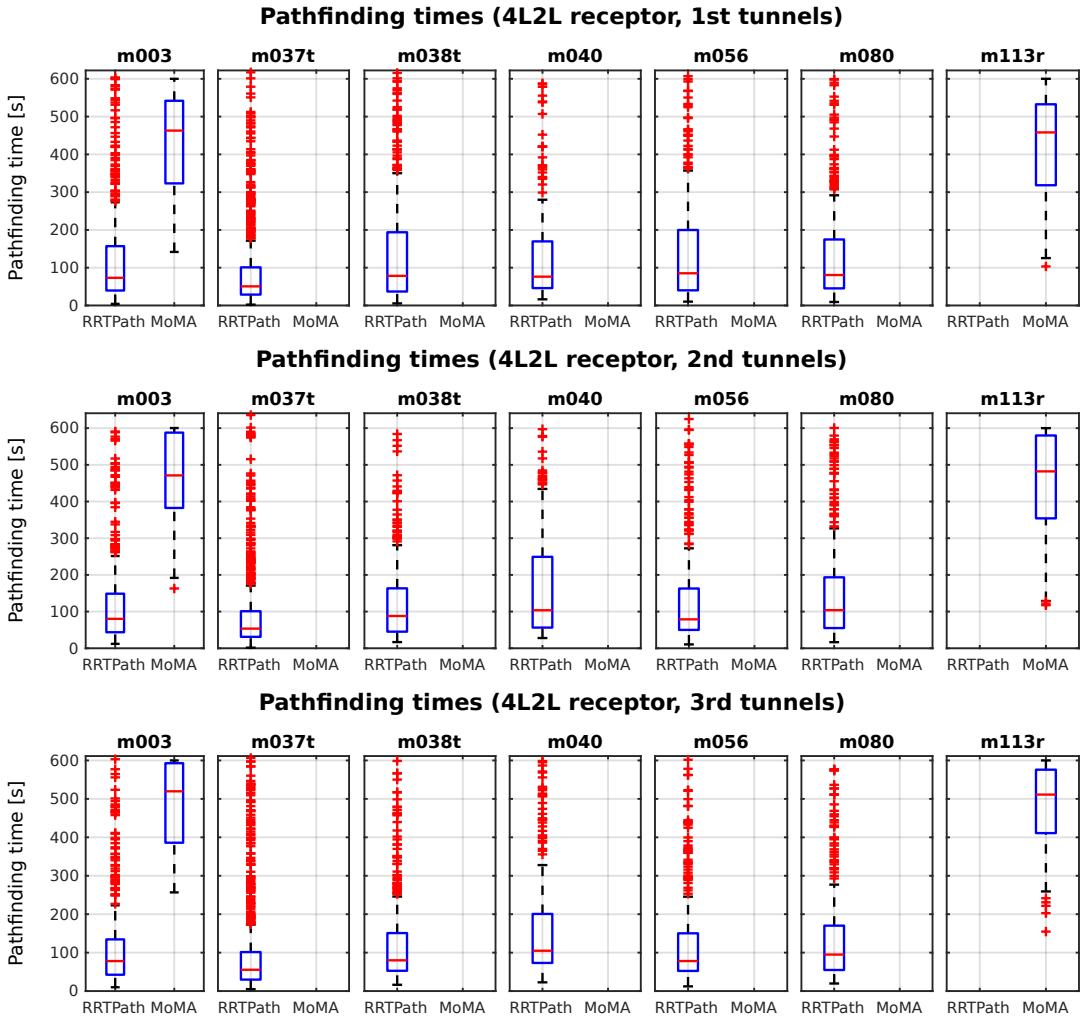


Figure 4.9: Path planning times in the 4L2L receptor over all possible tunnel types. Our algorithm clearly surpasses MoMA–LigPath, with a majority of P-RRT-Path planning times not exceeding 300 seconds, with the average planning time being around 100 seconds. MoMA–LigPath planning times lie mostly above 300 seconds, with none being below 100 seconds.

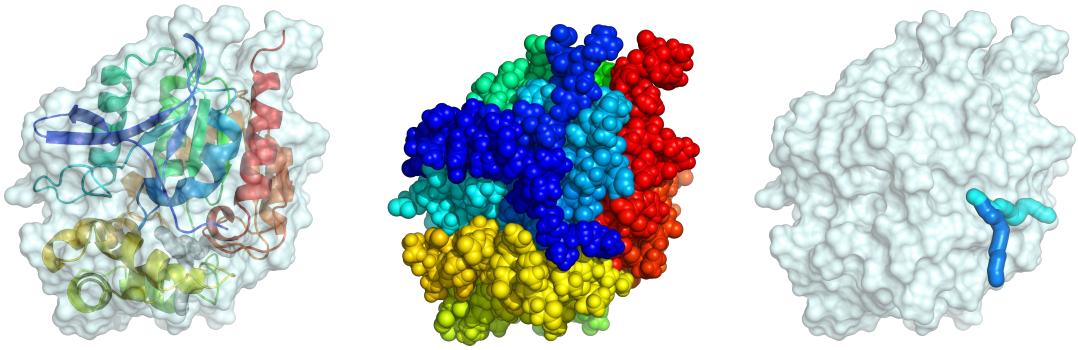
any pathways at all. This may be caused by the spherical nature of the molecule, which exhibits greater width than others and can require wider tunnels. MoMA–LigPath is able to alter the positions of receptor atoms. This may have been a contributing factor, which is something our algorithm emulates using atom downscaling.

The planning times are depicted in *Figure 4.9*. For our P-RRT-Path algorithm, the planning times rarely surpass 300 seconds, with the average path planning time being about 100 seconds in all of the cases. MoMA–LigPath, however, required nearly the entire available time amount in all of the found paths, and its pathfinding times were never lower than 100 seconds.

4.3 DHA

The DHA receptor consists of 4650 atoms. Its structure is depicted in *Figure 4.10*. The conformations were acquired using molecular dynamics simulations. The characteristics of the tunnels are shown in *Figure 4.11*. Compared to the 4L2L receptor, the amount of tunnels with lengths surpassing 20 Å is much lower. However, a majority of the tunnels is much narrower than in the 4L2L receptor. This might result in a reduced amount of successfully found pathways mainly for the secondary and the tertiary tunnels.

The results for our P-RRT-Path algorithm are presented in *Table 4.3*, the results for MoMA–LigPath in *Table 4.4* and *Figures 4.12* and *4.13*.



a) b) c)

Figure 4.10: Illustration of the DHA protein molecule. Subfigure a) *shows the surface and ribbon model.* Subfigure b) *shows the spherical space–filling model and* Subfigure c) *shows the tunnels in relation to the protein surface.*

Our P-RRT-Path algorithm surpassed MoMA–LigPath in all cases when planning in primary tunnels, with one ligand surpassing **92%** of tunnels reported as feasible for planning. It however, lacked in performance for the secondary and tertiary tunnels, where MoMA–LigPath managed to find more pathways for some of the tunnels (see *Figure 4.14*). MoMA–LigPath, however, once again failed to find any pathways for 5 of the 7 ligand molecules.

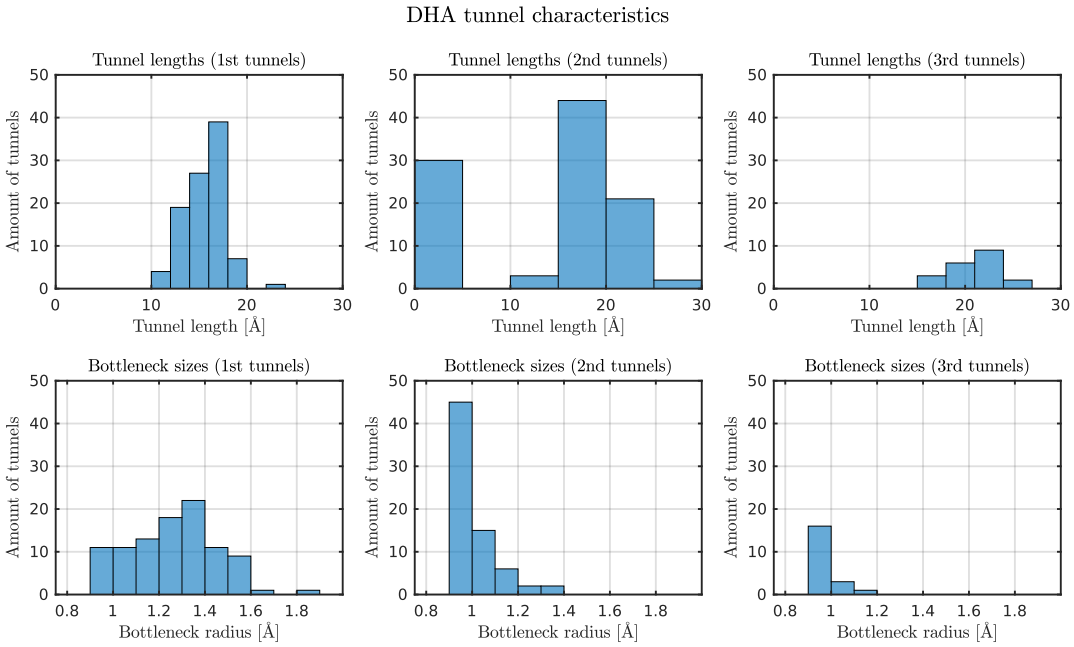


Figure 4.11: DHA receptor tunnel characteristics. We can see fewer tunnels with length surpassing 20 Å, when compared to the 4L2L receptor. However, the amount of narrow tunnels is much larger, which might result in a higher number of tunnels being reported as packed.

P–RRT–Path success rates for the DHA receptor							
	Ligands						
Tunnels	m003	m037t	m038t	m040	m056	m080	m113r
1st tunnels	68,25	92,37	62,58	26,60	53,00	54,95	89,48
2nd tunnels	6,86	22,43	4,57	1,43	3,00	3,29	31,57
3rd tunnels	0,50	11,50	1,50	0,00	0,00	0,00	9,50

Table 4.3: Overall percentages of successfully found pathways for P–RRT–Path in the DHA receptor. The success rates are fairly high for primary tunnels, though much less consistent than in the 4L2L receptor. Our algorithm exhibits a high degree of robustness, by being able to find tunnels in nearly all cases. There were no paths found for tertiary tunnels using ligands m040, m056 *and* m080. *This might be caused by the increasing complexity of tunnels, with the tertiary tunnels being the least suitable for ligand traversability. However, it might also suggest that a high amount of tunnels was incorrectly labeled as packed.*

MoMA–LigPath success rates for the DHA receptor							
	Ligands						
Tunnels	m003	m037t	m038t	m040	m056	m080	m113r
1st tunnels	16,29	0,00	0,00	0,00	0,00	0,00	32,16
2nd tunnels	17,29	0,00	0,00	0,00	0,00	0,00	31,14
3rd tunnels	10,00	0,00	0,00	0,00	0,00	0,00	25,00

Table 4.4: Overall percentages of successfully found pathways for MoMA–LigPath in the DHA receptor. Just like in the 4L2L receptor, MoMA exhibits great specificity, by being able to find pathways for primary tunnels using ligands m003 *and* m113r*. However, it completely failed to find any paths for all other ligands. Furthermore, the found tunnel rates diminish fairly quickly, which suggests a large amount of tunnels might have been incorrectly reported as packed.*

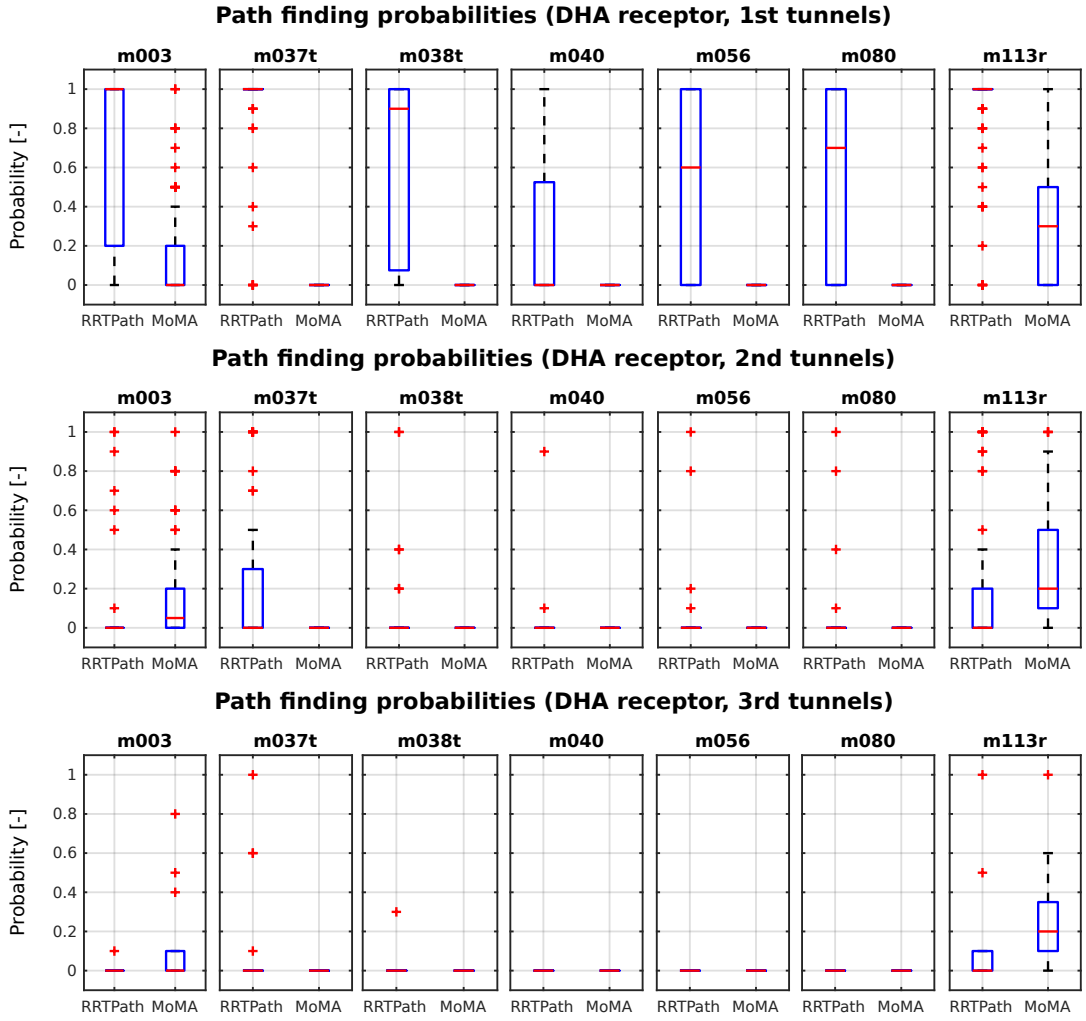


Figure 4.12: Path finding probabilities for DHA receptor tunnels over all receptor conformations. Our algorithm surpassed MoMA–LigPath in all primary tunnels scenarios. MoMA–LigPath, however, showed a greater degree of robustness for 2 of the 7 ligands in secondary and tertiary tunnels, where our algorithm found a significantly lower amount of pathways.

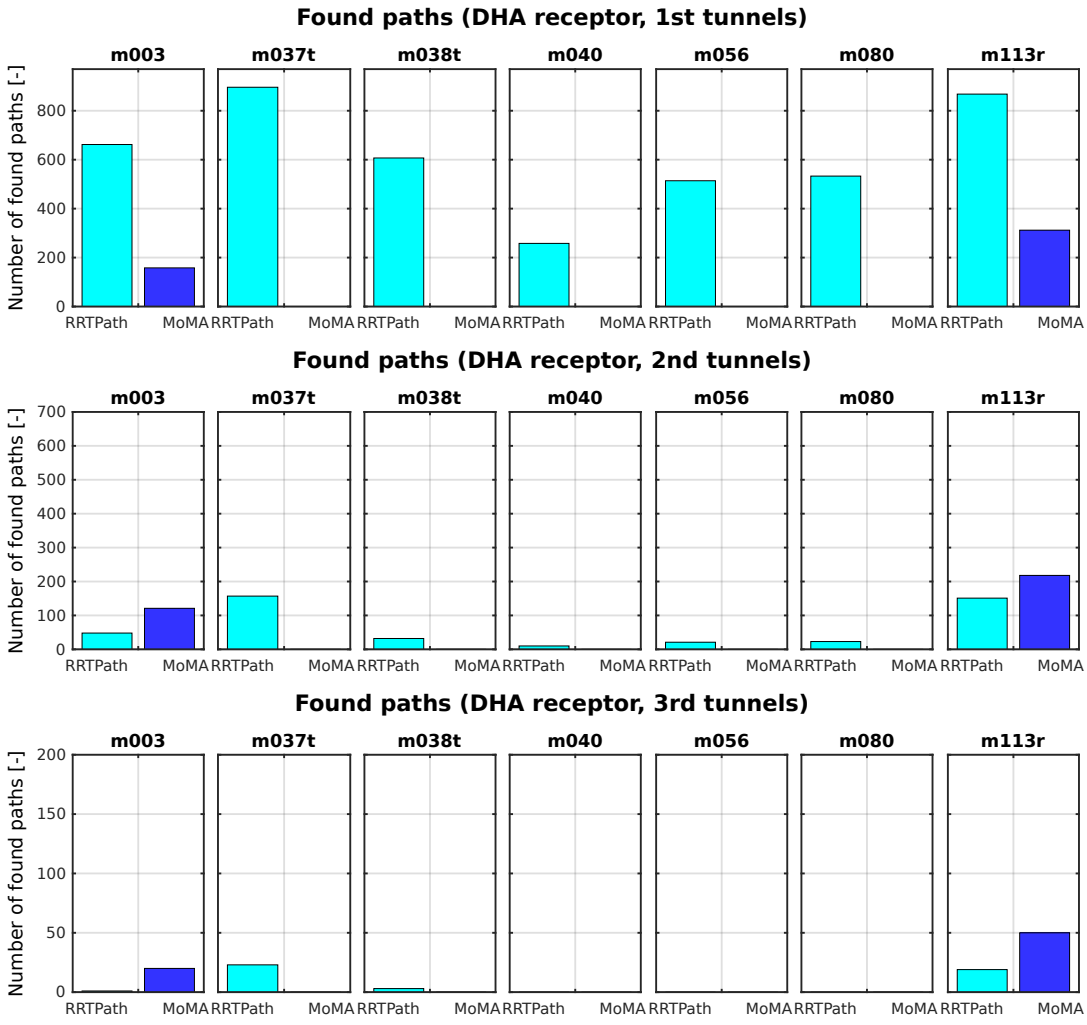


Figure 4.13: Number of found paths for DHA receptor tunnels over all receptor conformations. The results are consistent with previous figures, where our algorithm surpassed MoMA–LigPath in primary tunnels, but lacked in performance in secondary and tertiary tunnels.

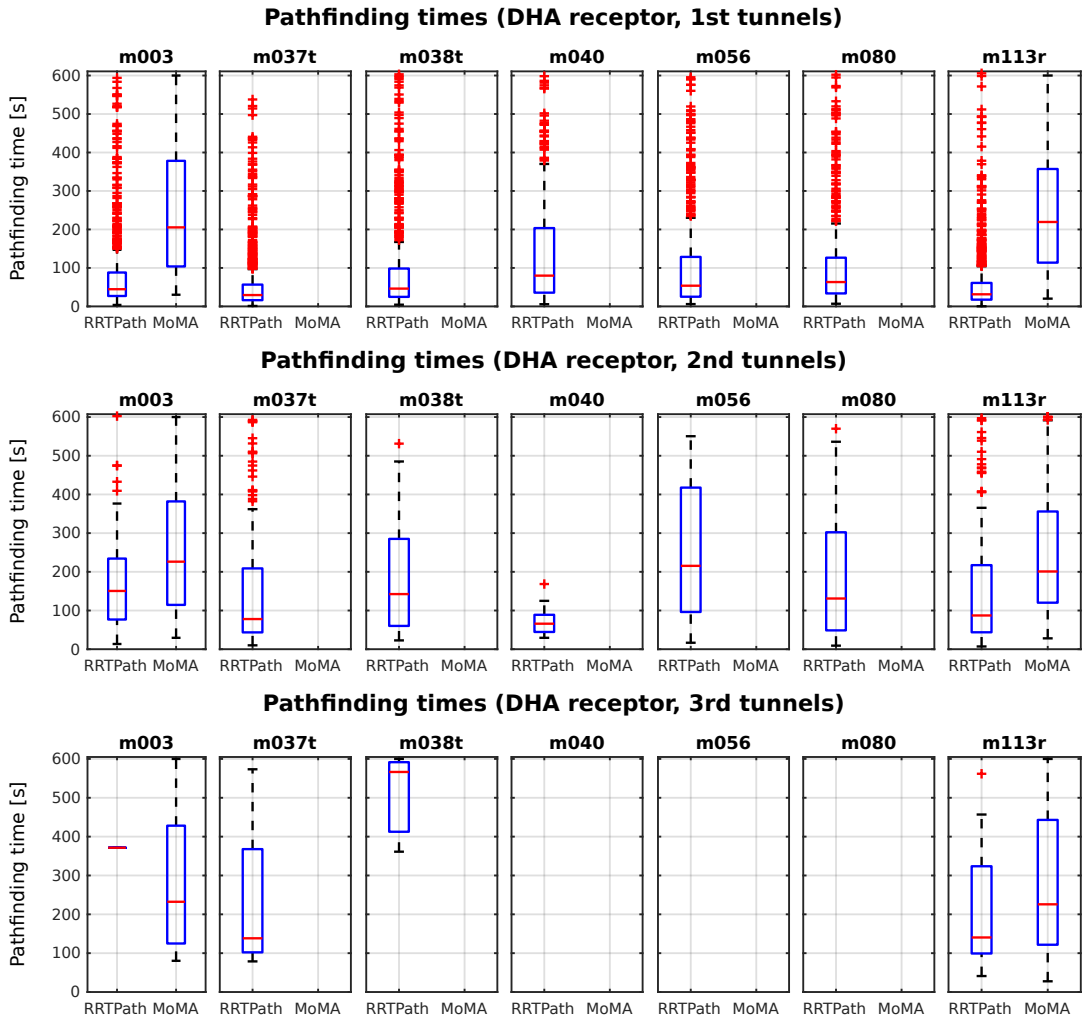


Figure 4.14: Path planning times in the DHA receptor over all receptor conformations. MoMA–LigPath exhibits a slightly worse performance in primary tunnels, with the average planning time being around 200 seconds. Our algorithm's average once again lies around 100 seconds. In secondary and tertiary tunnels however, the performances are much more similar, in cases where both algorithms managed to find pathways.

Our P–RRT–Path algorithm exhibited greater performance, with significantly shorter planning times for the primary tunnels, and slightly shorter planning times for secondary and tertiary tunnels.

4.4 DHA–AWT

The DHA–AWT (PDB ID 4E46) receptor consists of 4650 atoms. Its structure is depicted in *Figure 4.10*. The molecular dynamics simulations was computed using AMBER 12 (details are described in [31]). It is a modification of the DHA receptor, with tunnels slightly more suitable for planning.

The characteristics of the tunnels are shown in *Figure 4.16*. Similar to the DHA receptor, the amount of tunnels with lengths surpassing 20 Å is much lower. Again a majority of the tunnels is much narrower than in the 4L2L receptor. This might result in a reduced amount of successfully found pathways mainly for secondary and tertiary tunnels.

The results for our P-RRT-Path algorithm are presented in *Table 4.5*, the results for MoMA–LigPath in *Table 4.6* and *Figures 4.12* and *4.13*.

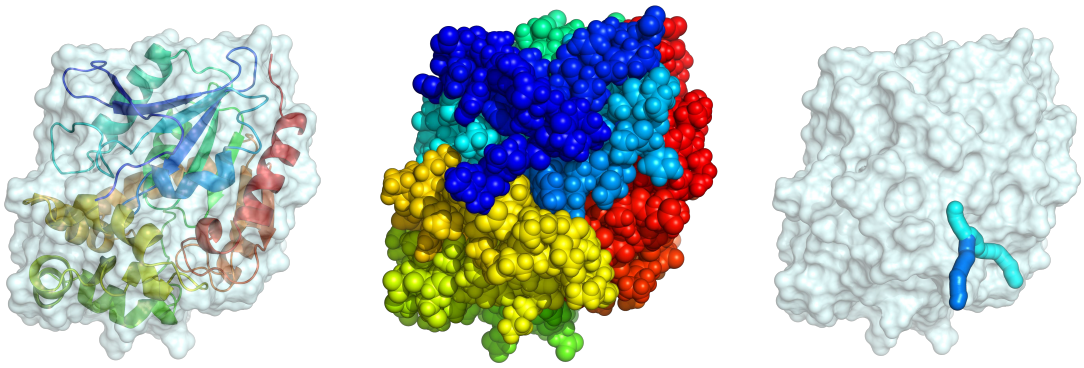


Figure 4.15: Illustration of the DHA–AWT protein molecule. Subfigure a) *shows the surface and ribbon model.* Subfigure b) *shows the spherical space–filling model and* Subfigure c) *shows the tunnels in relation to the protein surface.*

MoMA–LigPath surpassed our algorithm in some of the scenarios which used ligands *m003* and *m113r*. The performances of the algorithms were similar in primary tunnels, with MoMA–LigPath exhibiting shorter planning times (see *Figure 4.19*) in secondary and tertiary tunnels. Our algorithm exhibited robustness, in the form of the ability to find tunnels in all of the scenarios. MoMA–LigPath again failed to find any pathways for 5 of the 7 ligand molecules.

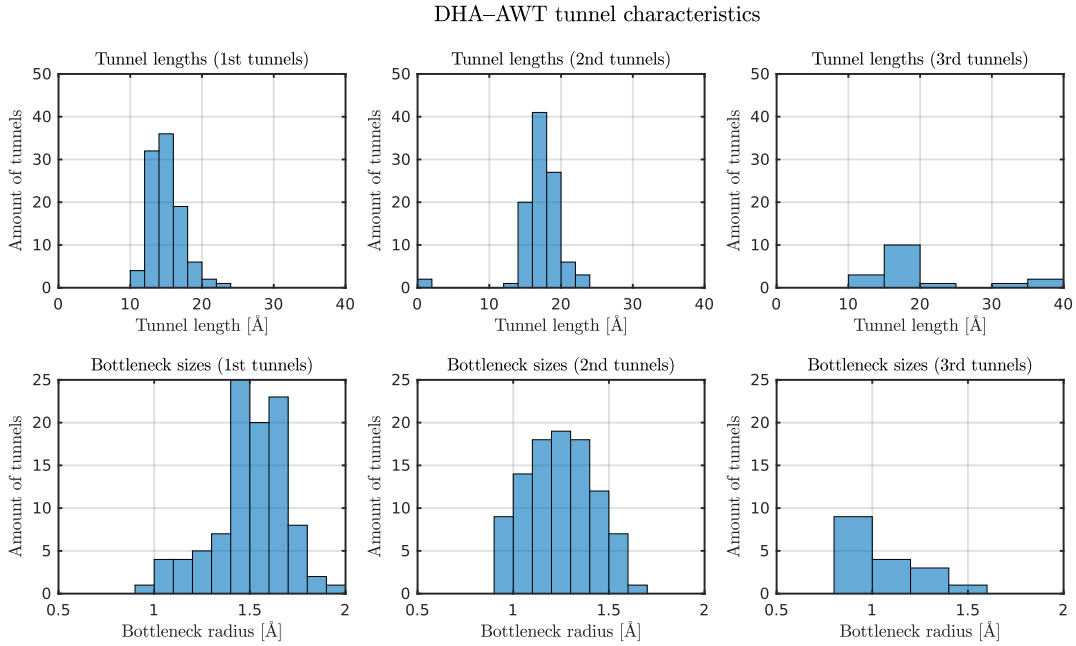


Figure 4.16: DHA–AWT receptor tunnel characteristics. The tunnels are fairly similar to those in the DHA receptor. The primary tunnels exhibit the biggest feasibility, with the shortest lengths and widest bottlenecks, which diminishes with secondary and tertiary tunnels.

P–RRT–Path success rates for the DHA–AWT receptor							
	Ligands						
Tunnels	m003	m037t	m038t	m040	m056	m080	m113r
1st tunnels	67,9	74,7	64,8	56,2	64,0	63,5	74,7
2nd tunnels	11,02	12,96	9,39	7,04	8,78	8,88	12,04
3rd tunnels	10,00	17,65	10,00	4,12	5,29	8,24	16,47

Table 4.5: Overall percentages of successfully found pathways for P-RRT-Path. Similarly to the DHA receptor, the success rates are fairly high for the primary tunnels and quickly diminish for the secondary and tertiary tunnels. This may be attributed to the decreasing bottleneck radii, as seen in Figure 4.16. *Our algorithm again exhibits a high degree of robustness, with the lowest number of found tunnels in all of the scenarios being 4,12%.*

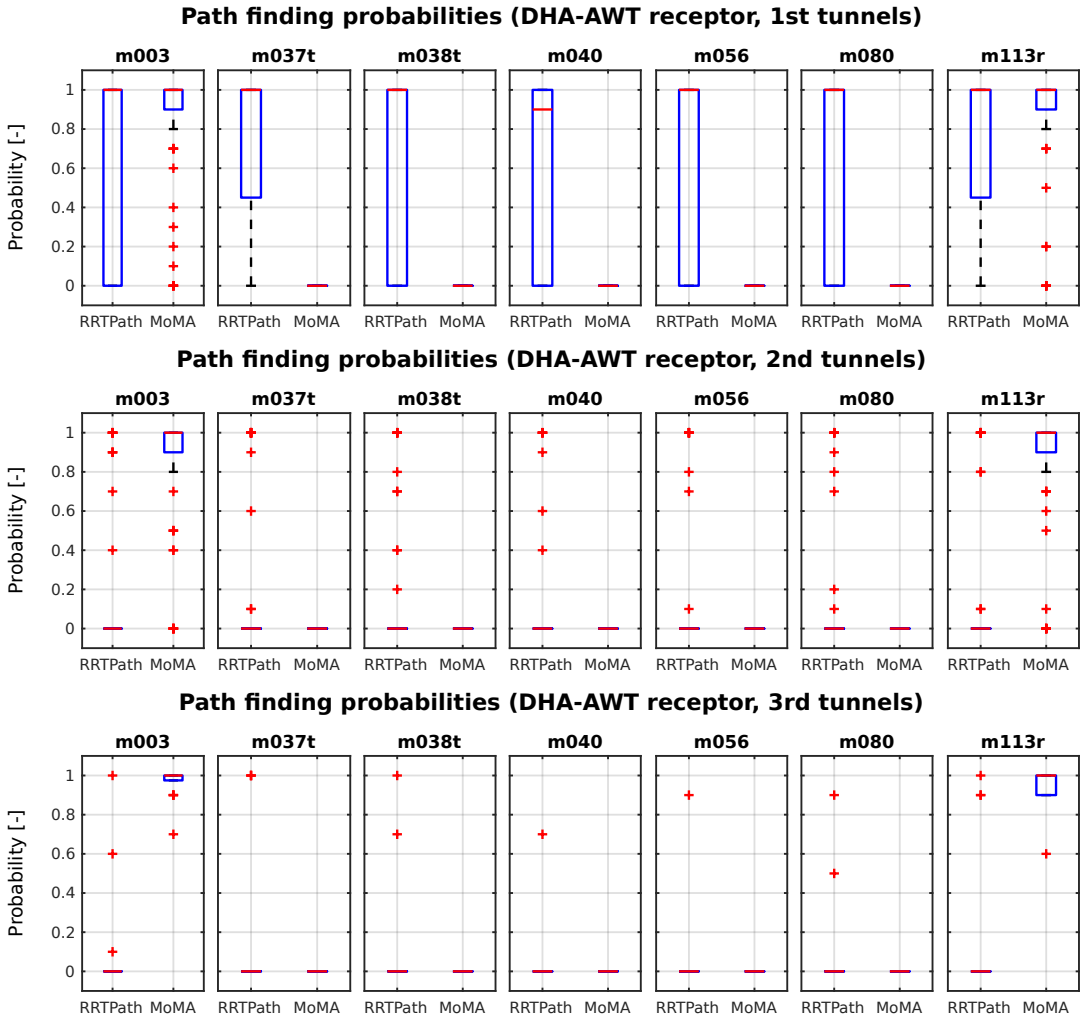


Figure 4.17: Path finding probabilities for DHA–AWT receptor tunnels over all receptor conformations. Our algorithm exhibits high success rates for all of the primary tunnel scenarios, which quickly diminish for secondary and tertiary tunnels. MoMA–LigPath surpassed our P–RRT–Path algorithm in all scenarios using ligands m003 *and* m113r*, with probabilities of finding a path nearly always reaching 100%. It however completely failed to find any pathways for all other ligands.*

MoMA–LigPath success rates for the DHA–AWT receptor							
	Ligands						
Tunnels	m003	m037t	m038t	m040	m056	m080	m113r
1st tunnels	88,3	0,0	0,0	0,0	0,0	0,0	88,9
2nd tunnels	87,14	0,0	0,0	0,0	0,0	0,0	88,57
3rd tunnels	96,47	0,0	0,0	0,0	0,0	0,0	95,29

Table 4.6: Overall percentages of successfully found pathways for MoMA–LigPath in the DHA–AWT receptor. MoMA–LigPath was able to find nearly all pathways for primary tunnels using ligands m003 *and* m113r*. Furthermore, it exhibited a paradoxical behavior, with the success rates increasing as tunnel feasibility decreased. However, once again it completely failed to find any paths for all other ligands.*

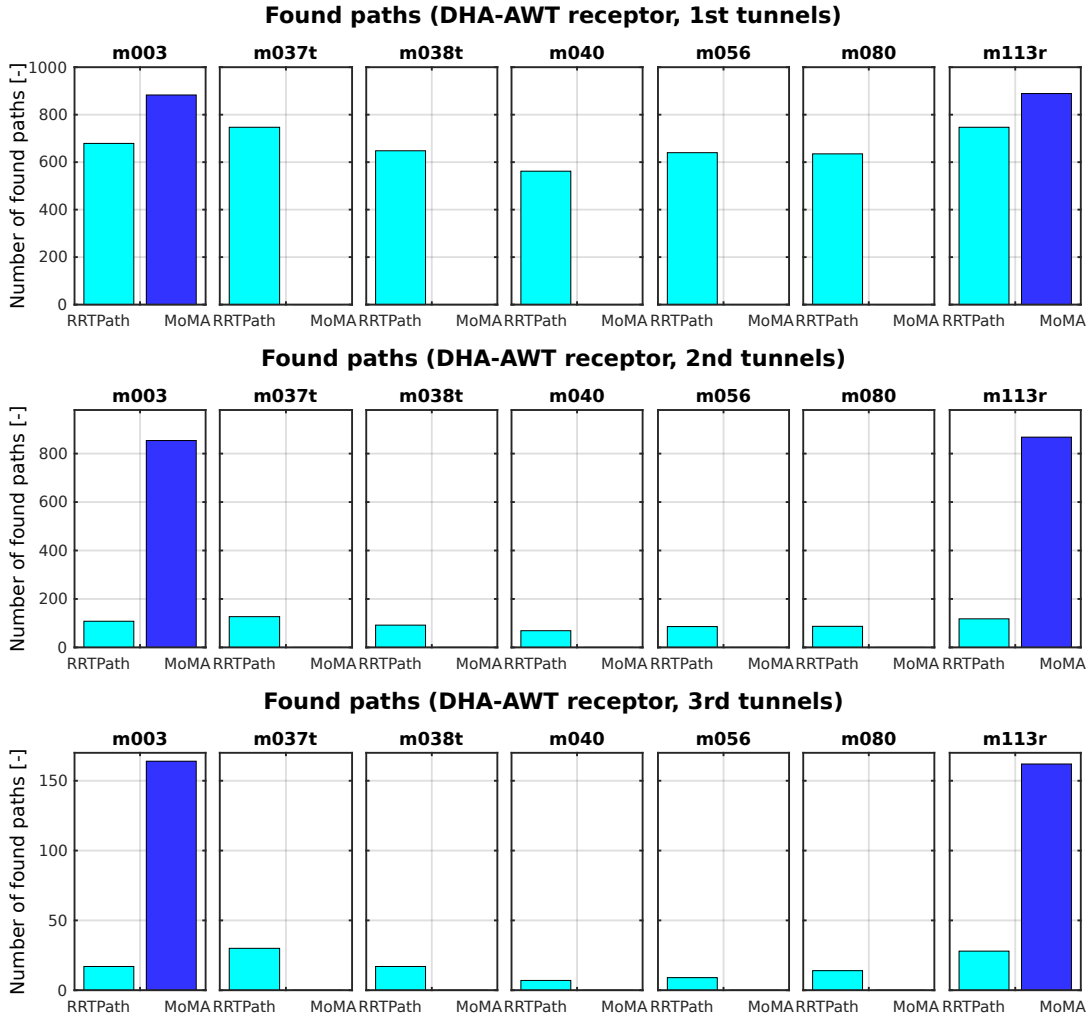


Figure 4.18: Number of found paths for DHA–AWT receptor tunnels over all receptor conformations. The results are consistent with previous figures, where our algorithm's performance is on par with MoMA–LigPath in primary tunnels in terms of specificity, surpassing MoMA–LigPath in robustness, but lacked in performance in secondary and tertiary tunnels.

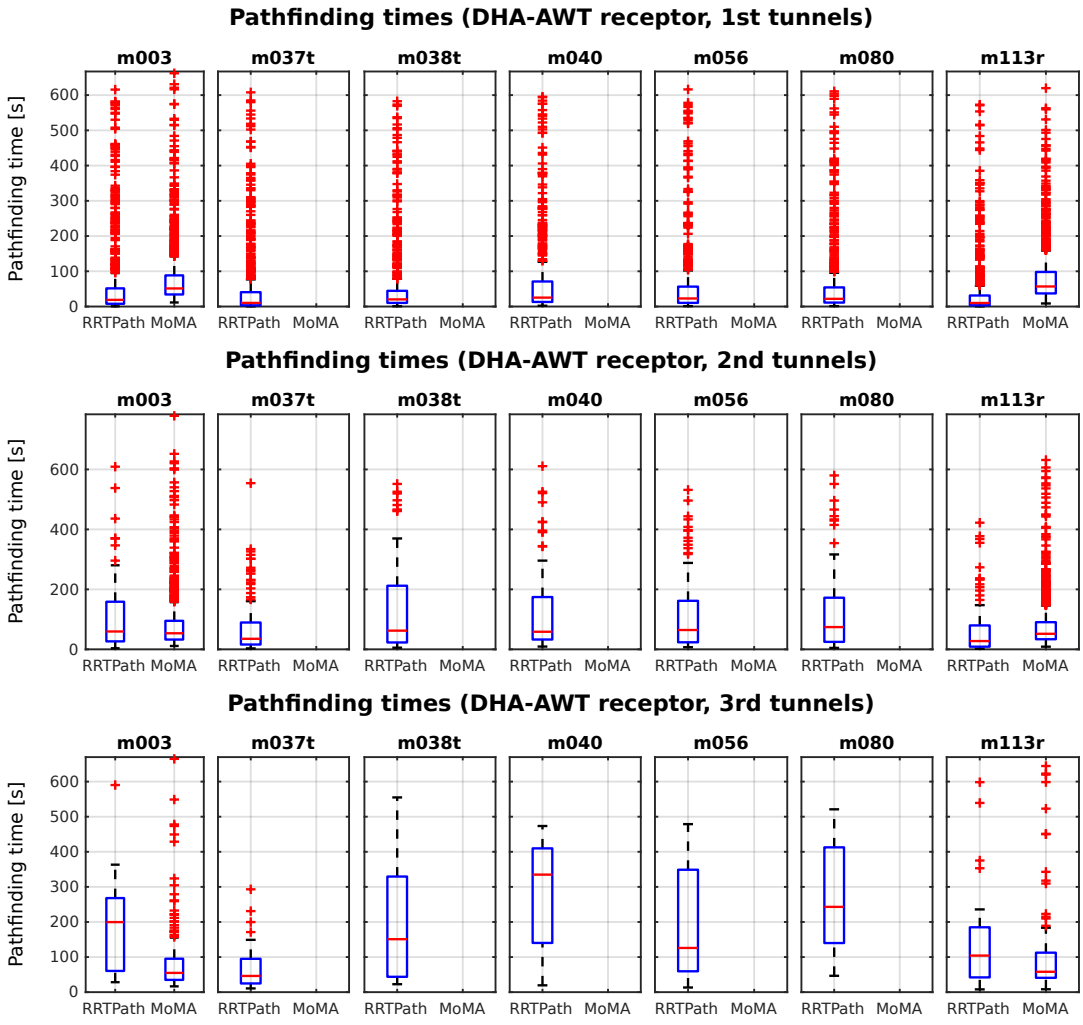


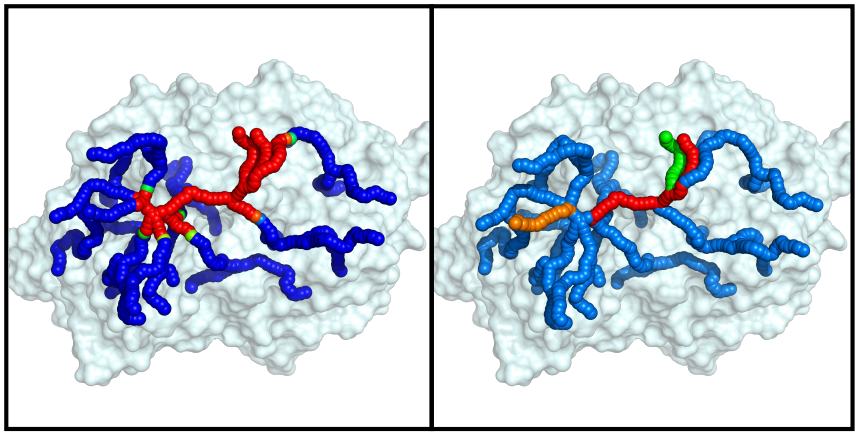
Figure 4.19: Path planning times in the DHA–AWT receptor over all receptor conformations. Our algorithm's performance is slighly better in the primary tunnel scenarios, but, for the first time, its performance was surpassed by MoMA–LigPath in secondary and tertiary tunnels.

4.5 Multiple Guiding Paths

CAVER orders the exported protein tunnels according to their length, bottleneck size, and a number of other factors. This ordering reflects the possibilities of passing a ligand molecule through the tunnel. During the testing of our implementation, we have found that this ordering does not always coincide with the actual scenario. This is caused by the fact that CAVER doesn't take the shape of the ligand molecule into account and rates the tunnels using a spherical probe. Due to the nature of the RRT-Path algorithm, when we select an improper tunnel for path planning, the algorithm may run excessively long, or may not find a pathway at all, despite the fact that there may exist a better suited tunnel inside the molecule, which could facilitate ligand passage.

To mitigate this issue, we have devised a version of our algorithm which utilizes multiple auxiliary paths at once. A statistical analysis of its output results can be performed and the tunnels ordered according to the probabilities of being found with the actual ligand molecule used for planning.

The algorithm was tested on the *1MXTa* receptor molecule with the *m003* ligand. The planning was performed 100 times.



a) b)

Figure 4.20: Multiple auxiliary paths in 1MXTa receptor. Subfigure a) *shows all of the tunnels found by CAVER, with unused tunnels colored in blue, and tunnels, which were used to find pathways colored in red.* Subfigure b) *shows Tunnel 1 (orange), which is the most important tunnel according to CAVER, and tunnels 3 (red) and 5 (green), which are actually the most suitable for planning with the* m003 *ligand molecule.*

As we can see in *Figure 4.20*, the feasibility of tunnels reported by CAVER does not actually reflect the real situation. A pathway was never found using the first two most suitable tunnels. The tunnels number 3 and 5 were actually utilized to find all of the 100 pathways.

After performing the statistical analysis and selecting the best suited protein tunnel we now have the ability to run the single auxiliary path implementation without any unnecessary slowdowns, which results in an improved performance.

4.6 Improved MoMA–LigPath Conditions

The results of our tests show that the MoMA–LigPath algorithm performs exceedingly poorly for 5 of the 7 provided ligands. To try and mitigate this issue, we have performed further test runs of the algorithm, with computing resources consisting of 8 CPU cores, 16 GB of RAM and a 1 hour time limit. The tests were run on the first 10 conformations of the DHA–AWT receptor, on primary tunnels only.

MoMA–LigPath success rates with improved conditions							
	Ligands						
Tunnels	m003	m037t	m038t	m040	m056	m080	m113r
1st frame	30	0,0	0,0	0,0	0,0	0,0	60
2nd frame	30	0,0	0,0	0,0	0,0	0,0	80
3rd frame	0	0,0	0,0	0,0	0,0	0,0	70
4th frame	0	0,0	0,0	0,0	0,0	0,0	30
5th frame	90	0,0	0,0	0,0	0,0	0,0	90
6th frame	0	0,0	0,0	0,0	0,0	0,0	30
7th frame	0	0,0	0,0	0,0	0,0	0,0	10
8th frame	40	0,0	0,0	0,0	0,0	0,0	80
9th frame	10	0,0	0,0	0,0	0,0	0,0	40
10th frame	0	0,0	0,0	0,0	0,0	0,0	10
Average	20	0	0	0	0	0	50

Table 4.7: Success rates for MoMA–LigPath in improved conditions in the DHA–AWT receptor. MoMA–LigPath successfully found pathways for ligands m003 *and* m113r*. However, once again it completely failed to find any paths for all other ligands, despite the increased time limit and computing resources.*

As is apparent from *Table 4.7*, MoMA–LigPath once again failed to find any pathways for ligands other than *m003* and *m113r*, despite the additional computing resources and the increased time limit. A majority of the runs using ligands *m003* and *m113r* failed due to timeouts. Other runs either failed due to timeouts as well, or MoMA–LigPath reported unrepairable collisions in the initial conformations and exited, desbite the fact that both our implementation and the Pymol visualization tool reported no collisions.

4.7 Intramolecular Potential Energy

The implementation of the Transition–RRT algorithm (see *Section 3.4.7*) allows our implementation to rapidly explore the configuration space, while maintaining a low potential energy ligand state throughout the entire trajectory.

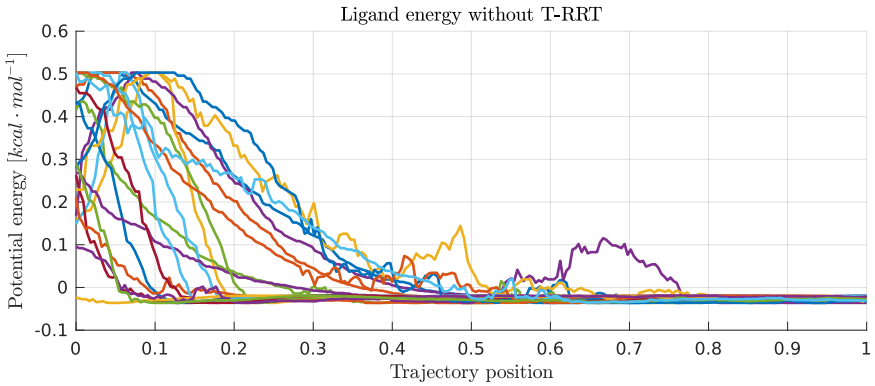


Figure 4.21: Trajectory intramolecular potential energy for the 1MXTa *receptor and* m080 *ligand with the T–RRT algorithm disabled. As we can see in the middle part of the plot, the energy function exhibits a significant amount of peaks.*

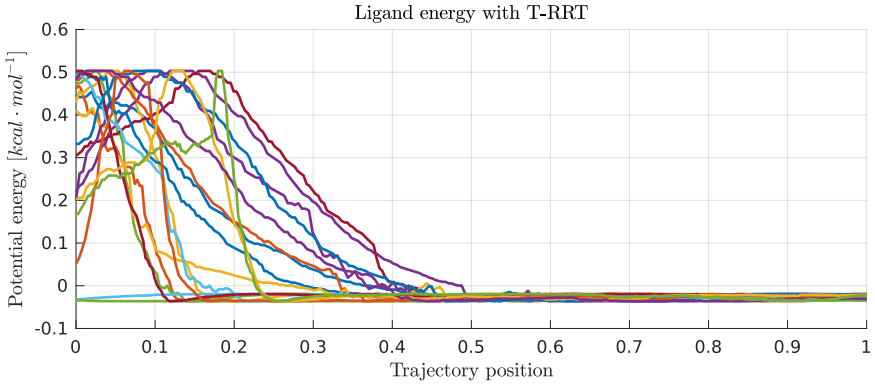
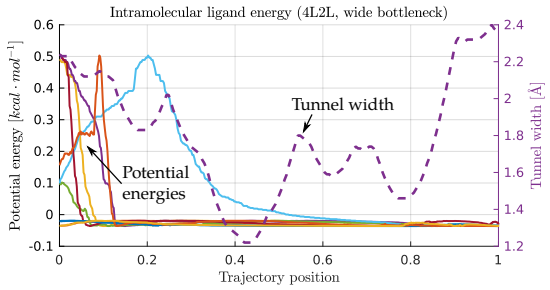
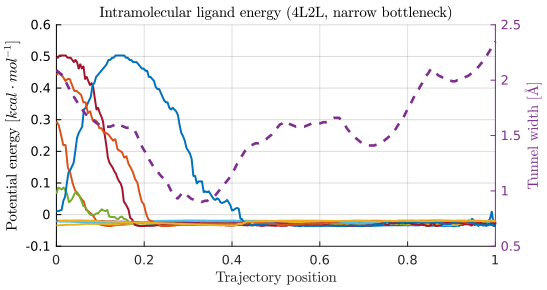


Figure 4.22: Trajectory intramolecular potential energy for the 1MXTa *receptor and* m080 *ligand with the T–RRT algorithm enabled. The function peaks, which are present in* Figure 4.21 *have been eliminated. However, the T–RRT algorithm is still flexible enough to facilitate high–energy ligand conformations when necessary.*

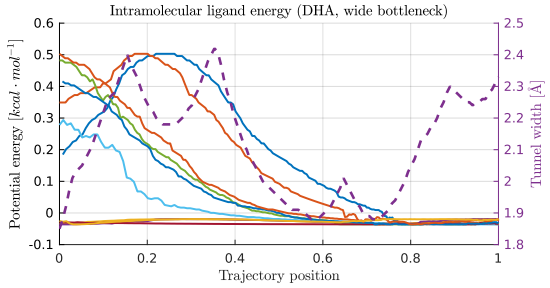
To verify this ability, the algorithm was tested on the *1MXTa* receptor molecule, using the *m080* ligand. The tests were performed 20 times with T–RRT enabled and 20 times with T–RRT disabled. The resulting energy costs of the trajectories are presented in *Figures 4.21* and *4.22*. The energy units are kcal·mol^{-1}. As we can see, the algorithm successfully eliminated the peaks present in trajectories found without using the T–RRT algorithm. Furthermore, the algorithm was still able to quickly adjust the transition test to allows higher energy states, when a bottleneck in the tunnel required a less compact ligand conformation in order to facilitate ligand passage through the tunnel. The algorithm has also been tested on selected conformations of all the receptors, namely those with the widest bottlenecks and those with small bottleneck, which still allowed the passage of ligand molecules. The algorithm was run 10 times for each receptor conformation, with the T–RRT algorithm enabled. See *Figure 4.23* for results. Our implementation managed to eliminate a vast majority of the energy fluctuations, keeping the ligand molecule at a steady low–energy state, only reaching higher energy states when required.



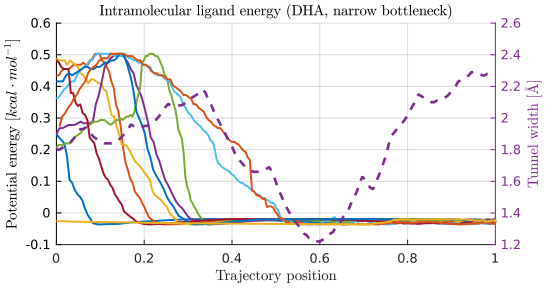
(a) Trajectory potential energy for the 4L2L receptor (wide bottleneck).



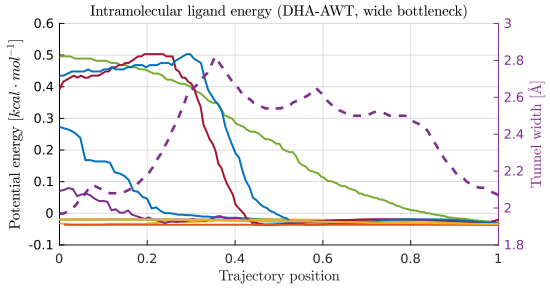
(b) Trajectory potential energy for the 4L2L receptor (narrow bottleneck).



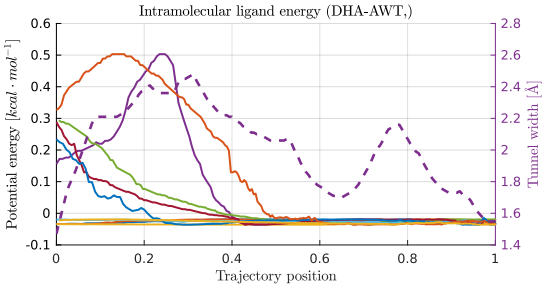
(c) Trajectory potential energy for the DHA receptor (wide bottleneck).



(d) Trajectory potential energy for the DHA receptor (narrow bottleneck).



(e) Trajectory potential energy for the DHA–AWT receptor (wide bottleneck).



(f) Trajectory potential energy for the DHA–AWT receptor (narrow bottleneck).

Figure 4.23: Intramolecular potential energies for receptor molecules with varying bottleneck sizes. As we can see, the T–RRT algorithm has successfully managed to eliminate fluctuations of potential energies of the ligand molecule, while still maintaining enough flexibility to reach the surface of the receptor. The ligand stays at a low energy state throughout a majority of the trajectories and only reaches higher energy states when required.

4.8 Intermolecular Potential Energy

The AutoDock Vina library offers the possibility of computing intermolecular potential energy of a ligand–receptor system. Similarly to the atoms in the ligand molecule, the two molecules can react with repulsive forces. The forces affect the potential energy of the current molecular configuration, which in turn changes the feasibility of traversing the selected pathway.

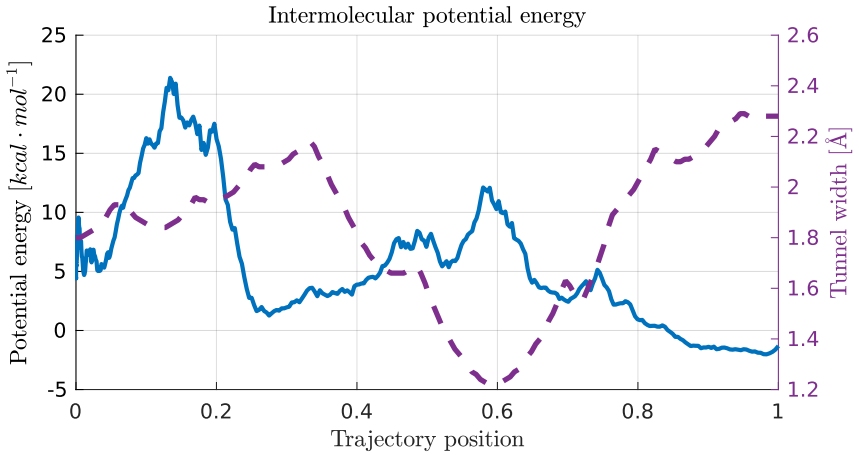


Figure 4.24: Computed intermolecular potential energy for a single pathway generated in the DHA receptor using the m113r *ligand. The purple striped line represents the width of the tunnel. Note how the potential energy function peaks in the tunnel's bottleneck.*

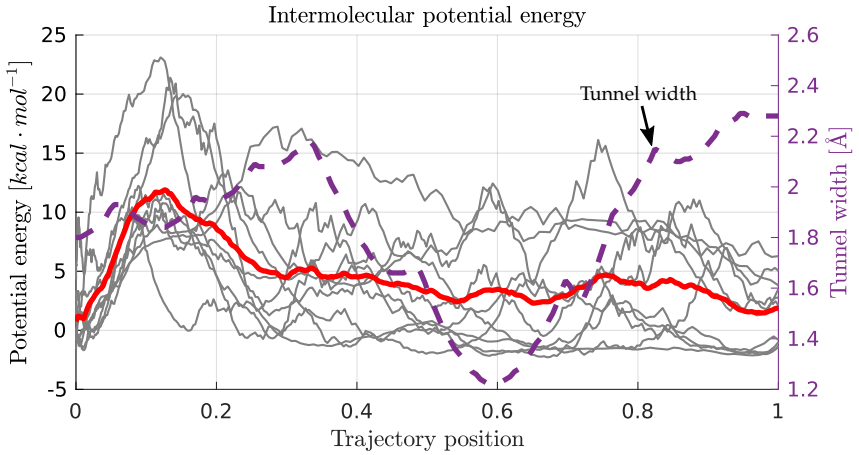


Figure 4.25: Computed intermolecular potential energy for 10 pathways generated in the DHA receptor using the m113r *ligand. The purple striped line represents the width of the tunnel. The thick red line represents the average value over all potential energies. Note the slight peak in the tunnel's bottleneck. This is consistent with the expectations, since the potential energy in a narrow space should be higher than in a wide space.*

Figure 4.24 depicts a single potential energy function though the generated pathway. The function exhibits a noticeable peak in the tunnel's bottleneck, which is consistent

with the expectations, since the narrowest point of the protein tunnel should have higher potential energy than other locations in the tunnel.

However, *Figure 4.25* shows that the data is not always this consistent. Some of the functions do not peak in the bottleneck at all. Therefore, to correctly interpret the results and avoid invalid results, statistical analysis of multiple runs is needed. Although the potential energy does not reach its highest value in the tunnel's bottleneck, it still exhibits a noticeable peak even after averaging over all 10 runs.

5 Discussion of the Results

The experimental data have shown that the P–RRT–Path algorithm outperformed the MoMA–LigPath algorithm in a majority of the testing cases.

5.1 4L2L

In the *4L2L* receptor (*Section 4.2*), our algorithm managed to find pathways with a higher probability than the MoMA–LigPath algorithm in 18 of the 21 tested cases. Our algorithm failed to find any paths for the *m113r* ligand. This may be attributed to its shape, which is more spherical than other ligands. MoMA–LigPath failed to find any pathways for five of the seven ligands. Our algorithm also exhibited a high amount of consistency, with the pathways numbers staying similar for all of the three tunnel types.

In the scenarios, where MoMA–LigPath managed to detect a pathway, our algorithm also performed better in terms of planning speed, with the median value of planning duration being up to 6 times lower than that of MoMA–LigPath.

5.2 DHA

In the DHA receptor (*Section 4.3*) the P–RRT-Path algorithm found pathways with a greater probability than MoMA–LigPath in 15 of the 21 tested cases. Our algorithm struggled with finding pathways for the tertiary tunnels, where we can see a significantly decreasing percentages. MoMA–LigPath, however, once again failed to find any tunnels for five of the seven ligands.

Overall planning times were again better with our algorithm.

5.3 DHA–AWT

When planning in the DHA–AWT receptor (*Section 4.4*), our algorithm outperformed MoMA–LigPath in 15 of the 21 tested cases. P–RRT–Path managed to detect some pathways in all of the receptor conformations and tunnels. MoMA–LigPath however

once again failed for five of the seven ligands. It has however exhibited significantly high percentages of success in the other two ligands.

In the cases where MoMA–LigPath succeeded in detecting pathways, the planning times were comparable to those of our algorithm.

As we can see, MoMA–LigPath has consistently failed to detect any pathways for five of the seven supplied ligands. To try and mitigate this issue, we have tried running MoMA–Ligpath with improved conditions, i.e., more CPU cores, more RAM and more computing time. Even then, it has still failed to produce any paths for the same 5 ligands. All of the failed ligands have more than 1 DOF, which might be the cause of the problem. This expands the configurations space the algorithm has to search through and possibly breaks MoMA–LigPath.

5.4 Multiple Paths

We have devised a modification of our algorithm, which enables us to utilize all of the provided CAVER tunnels at once, to try and determine which ones are the most feasible for pathways detection. We have successfully demonstrated the functionality in *Section 4.5*, where the rating system of CAVER did not accurately reflect the actual situation. The first and second tunnels were unsuitable for planning and instead, the third and fifth tunnels were used.

5.5 Ligand Potential Energy

The inclusion of AutoDock Vina and the T–RRT algorithm allowed us to implement a feature previously unavailable in any of the current state-of–the–art tools, which is sampling ligand conformations, while taking ligand intramolecular potential energy into account. *Section 4.7* demonstrates the effectiveness of this method, and the final results of planning in individual receptor molecules.

We have demonstrated the ability to compute intermolecular potential energies for a receptor–ligand system. The results are consistent with expectations.

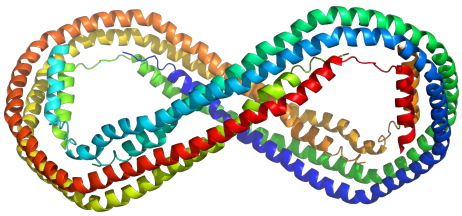
5.6 Future work

A publication of a technical paper with the presented results is planned in the near future. Results have been published in the international POSTER 2018 conference, under the title *Pathways searching for flexible ligands in proteins*. Future work could further explore the possibilities of pathways detection while taking intermolecular (ligand–receptor) potential energy into account. Further optimizations and parameter space exploration could also yield more optimal settings for pathways detection.

6 Conclusion

This thesis has explored the possibilities of applying state–of–the–art algorithms from the field of robotics to the receptor tunnel traversability problem (finding whether a ligand molecule can pass through a protein tunnel), originating in the field of biochemical engineering. The first chapter established the problem and the terminology necessary to describe it. The second chapter outlined several possible methods of approaching this issue and selected the most suitable one. The third chapter described our implementation of a novel algorithm Protein–RRT–Path, the modifications required to achieve it and the improvements, coined by us. The algorithm is able to detect pathways both from inside and outside of a protein receptor molecule. The fourth chapter presented results of experiments and compared our algorithm to the current state–of–the–art tool, MoMA–LigPath. The fifth chapter summarized the achieved results.

The performance of our P–RRT–Path algorithm has surpassed that of the MoMA–LigPath tool in all of the tested receptor molecules, while also presenting a feature previously unavailable in any of the currently used tools, which allows us to take intramolecular ligand potential energy into account when planning with flexible ligands, as well as computing intermolecular potential energy of the receptor–ligand system for the generated pathway.



Bibliography

[1] Bullet Physics Library: A physics engine for soft and rigid body dynamics simulation. http://bulletphysics.org/. Accessed: 2018-05-01.

[2] CGAL, Computational Geometry Algorithms Library. http://www.cgal.org/. Accessed: 2018-05-01.

[3] RCSB: Protein Data Bank. http://www.rcsb.org/pdb/.

[4] ROCO Chapter 2: Space-filling Models. http://www.reed.edu/chemistry/roco/Geometry/space_filling.html. Accessed: 2017-12-22.

[5] Sampling–Based Planning Under Differential Constraints. from Steven M. LaValle, 2006, cambridge university press. http://planning.cs.uiuc.edu/node713.html. Accessed: 2018-04-20.

[6] C.B. Anfinsen. The formation and stabilization of protein structure. *Biochem*, 1972.

[7] Yershova Anna and LaValle Steven M. MPNN: Nearest neighbor library for motion planning, 2006.

[8] Priyadarshi Bhattacharya and Marina L. Gavrilova. Voronoi diagram in optimal path planning. In *4th International Symposium on Voronoi Diagrams in Science and Engineering (ISVD 2007)*, pages 38–47, July 2007.

[9] Harbury P.B. Boas F.E. Potential energy functions for protein design. *Current Opinion in Structural Biology*, 17(2):199–204, 2007.

[10] Boost Project. Boost C++ Libraries. http://www.boost.org/. Accessed: 2018-04-06.

[11] Tyson Brochu, Essex Edwards, and Robert Bridson. Efficient geometrically exact continuous collision detection. *ACM Trans. Graph.*, 31(4):96:1–96:7, July 2012.

[12] Eva Chovancová, Antonín Pavelka, Petr Beneš, Ondřej Strnad, Jan Brezovský, Barbora Kozlíková, Artur Gora, Vilém Šustr, Martin Klvaňa, Petr Medek, Lada Biedermannová, Jiří Sochor, and Jiří Damborský. CAVER 3.0: A Tool for the Analysis of Transport Pathways in Dynamic Protein Structures. *Pathways in Dynamic Protein Structures, PLoS Computational Biology 8: e1002708*, 2012.

[13] Didier Devaurs, Léa Bouard, Marc Vaisset, Christophe Zanon, Ibrahim Al-Bluwi, Romain Iehl, Thierry Siméon, and Juan Cortés. MoMA-LigPath: a web server to simulate protein–ligand unbinding. *Nucleic acids research*, page gkt380, 2013.

[14] Kavraki Lydia E, Švestka Petr, Latombe Jean-Claude, and Overmars Mark H. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on*, 12(4):566–580, 1996.

[15] M. Elbanhawi and M. Simic. Sampling-based robot motion planning: A review. *IEEE Access*, 2:56–77, 2014.

[16] G-Truc Creation. OpenGL Mathematics. `http://glm.g-truc.net/`. Accessed: 2018-04-07.

[17] B. Hess, C. Kutzner, D. Van Der Spoel, and E. Lindahl. Gromacs 4: algorithms for highly efficient, load-balanced, and scalable molecular simulation. *Journal of chemical theory and computation*, 4(3):435–447, 2008.

[18] S. Hoher, P.Neher, and Alexander Verl. Analysis and evaluation of collision detection libraries for collision monitoring of multi-channel control applications. 2013.

[19] Trott O., Olson A. J. Autodock vina: improving the speed and accuracy of docking with a new scoring function, efficient optimization and multithreading. *Journal of Computational Chemistry*, 31(11):455–461, 2010.

[20] Cortes J. Jaillet L. and Simeon T. Transition-based rrt for path planning in continuous cost spaces. *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 17(2):2145–2150, 2008.

[21] H. Kaluđer, M. Brezak, and I. Petrović. A visibility graph based method for path planning in dynamic environments. In *2011 Proceedings of the 34th International Convention MIPRO*, pages 717–721, May 2011.

[22] Frazzoli Emilio Karaman Sertak. Sampling-based algorithms for optimal motion planning. *International Journal of Robotics Research*, 30(7):846–894, 2011.

[23] Igor Kravtchenko. Ozcollide: Advanced and fast collision detection library. `https://github.com/jslee02/OZCollide`. Accessed: 2018-03-05.

[24] M. Krone, B. Kozlíková, N. Lindow, M. Baaden, D. Baum, J. Parulek, H.-C. Hege, and I. Viola. Visual analysis of biomolecular cavities: State of the art. *Computer Graphics Forum*, 35(3):527–551, 2016.

[25] LaValle Steven M. Rapidly exploring dense trees. `http://planning.cs.uiuc.edu/node230.html`. Accessed: 2018-02-13.

[26] Tuffery P. Le Guilloux V, Schmidtke P. Fpocket: An open source platform for ligand pocket detection. *BMC Bioinformatics*, 10(168), 2009.

[27] Lindow Norbert, Baum, Daniel, Hege Hans–Christian. Ligand excluded surface: A new type of molecular surface. *Visualization and Computer Graphics, IEEE Transactions on*, 20(12), 2014.

[28] T. Lozano-Perez. Spatial planning: A configuration space approach. *IEEE Transactions on Computers*, C-32(2):108–120, Feb 1983.

[29] LaValle Steven M. and Kuffner Jr. James J. Rapidly-exploring random trees: Progress and prospects. 2000.

[30] Lehninger A., Nelson David L., Cox Michael M. *Principles of Biochemistry*. W. H. Freeman and Company, 2008.

[31] S. M. Marques, Z. Dunajová, Z. Prokop, R. Chaloupková, J. Brezovský, and J. Damborský. Catalytic cycle of haloalkane dehalogenases toward unnatural substrates explored by computational modeling. *Journal of Chemical Information and Modeling*, 57(8):1970–1989, 2017.

[32] Neidigh J.W., Fesinmeyer R.M., Andersen N.H. Representing proteins in silico and protein forward kinematics. `https://www.rcsb.org/pdb/explore/explore.do?structureId=1L2Y`. Accessed: 2017-12-21.

[33] Neidigh J.W., Fesinmeyer R.M., Andersen N.H. Voronoi Diagram. from MathWorld–a Wolfram Web Resource. `https://www.rcsb.org/pdb/explore/explore.do?structureId=1L2Y`. Accessed: 2017-12-21.

[34] Jia Pan and Dinesh Manocha. Efficient configuration space construction and optimization for motion planning. *Engineering*, 1(1):046 – 057, 2015. Accessed: 2018-03-25.

[35] Richardson, Jane S. Schematic drawings of protein structures. *Methods in Enzymology*, 1985.

[36] P.J. Russell. *iGenetics 3rd ed.*, 2012.

[37] D. Seeliger. tCONCOORD examples. http://www.user.gwdg.de/ dseelig/tconcoord.html.

[38] D. Seeliger, J. Haas, and B. L. de Groot. Geometry-based sampling of conformational transitions in proteins. *Structure*, 15(11):1482–1492, 2007.

[39] B.A. Sela. Titin: some aspects of the largest protein in the body. *Harefuah*, 2002.

[40] Kavraki Lydia Stamati Heman, Shedu Amarda. Computing forward kinematics for protein-like linear systems using denavit-hartenberg local frames. 2007.

[41] Jankovec Tom. Path Planning in Protein Structures, Bachelor Thesis. 2016.

[42] Vonásek Vojtěch. *A guided approach to sampling-based motion planning*. PhD thesis, Czech Technical University in Prague, 2016.

[43] Vojtěch Vonásek, Jan Faigl, Tomáš Krajník, and Libor Přeučil. RRT-Path–a guided rapidly exploring random tree. In *Robot Motion and Control 2009*, pages 307–316. Springer, 2009.

[44] Vonásek, Vojtěch and Faigl, Jan and Krajník, Tomáš and Přeučil, Libor. A Sampling Schema for Rapidly Exploring Random Trees Using a Guiding Path. In *Proceedings of the 5th European Conference on Mobile Robots*, pages 201–206, Örebro, 2011. AASS Research Centre.

[45] Schneider G. Weisel M, Proschak E. Pocketpicker: analysis of ligand binding-sites with shape descriptors. *Chemistry Central Journal*, 1(7), 2007.

[46] Worldwide Protein Data Bank. wwPDB: File Format. `http://www.wwpdb.org/documentation/file-format`. Accessed: 2018-03-07.

[47] Worldwide Protein Data Bank. wwPDB: File Format version 3.3: Coordinate Section. `http://www.wwpdb.org/documentation/file-format-content/format33/sect9.html#ATOM`. Accessed: 2018-03-07.

[48] Kavraki E. Lydia Zhang Ming. A new method for fast and accurate derivation of molecular conformations. *Journal of Chemical Information and Modeling*, 42(1):64—70, 2001.

7 Addenda

7.1 Contents of the Attached Disc

The attached disc contains:

- **Input_Data** folder - Contains all the input data used to generate the presented results. The file names should be self–explanatory.
- **Protein-RRT-Path** folder - Contains source codes to the provided final implementations.
- **Results_Example** folder - Contains a small part of the final results. Sadly, due to the immense size of the generated data (>500 GB). We have not been able to provide the data alongside this thesis. The rest of the data is available on the *Metacentrum* server upon request.
- **Thesis** folder - Contains a pdf file with this thesis.

7.2 Usage Instructions

7.2.1 Installation Guide

The program was compiled using CMake 3.5.1, Make 4.1 and g++ 5.4.0 for Ubuntu 16.04. The compilation requires a number of external libraries to be installed. While we have tested the program with these specific versions, other versions may work as well:

- Boost C++ library version 1.58.0
- GLM library version 0.9.7.2-1
- Eigen library version 3.0
- PThread library version 0.3.4

Other libraries are provided alongside the programs and require no further steps.

The programs are located in the Protein-RRT-Path folder. Follow these steps to compile the program using terminal in the extracted folder on a Linux operating system:

Parameter	Meaning
-h (--help)	Writes out program description and a list of possible parameters
-r (--receptor)	Path to the PBD receptor file
-p (--pdbqtReceptor)	Path to the PBDQT receptor file (optional, single guiding path planner only)
-l (--ligand)	Path to the PBDQT ligand file
-t (--tunnel)	Path to the PBD tunnel file

Table 7.1: Runtime parameters

```
cd Protein-RRT-Path
mkdir build
cd build
cmake ..
make all -j 4
```

Figure 7.1: Compilation commands

7.2.2 Running the program

The compiled programs can be run via the Linux terminal. The list of required and available runtime parameters can be found in *Table 7.1*.
Upon completion, the program generates a number of PDB files, as well as an XML file:

- **<receptor name>_<ligand name>_<tunnel name>.goal.pdb** – PDB file with ligand in the initial position.
- **<receptor name>_<ligand name>_<tunnel name>.init.pdb** – PDB file with ligand in the initial position.
- **<receptor name>_<ligand name>_<tunnel name>.protein.pdb** – PDB file with just the receptor.
- **<receptor name>_<ligand name>_<tunnel name>.traj.xml** – XML file containing all information about the trajectory, including the internal ligand angles, planning scale and intramolecular and intermolecular energies.
- **<receptor name>_<ligand name>_<tunnel name>.traj<number>.pdb** – PDB files with interpolated ligand positions throughout the pathway.
- **<receptor name>_<ligand name>_<tunnel name>.iterations.txt** – Text file with the number of iteration necessary to complete the planning task.
- **<receptor name>_<ligand name>_<tunnel name>_time_ms.txt** – Text file with planning time duration.

Since MoMA–LigPath requires a specifically tailored PDB files, alongside the two presented programs (*protein_planning_flexible_ligand* and *protein_planning_flexible_ligand_multiple_pathways*) is also provided a helper program *moma_exporter*, used for generating MoMA–LigPath compatible PDB files.

An XML file with task definitions has to be passed to the program using the "-t" parameter, alongside the necessary pdb files. An example of such XML file is found in *Figure 7.2*

```
<?xml version="1.0" encoding="UTF-8"?>
<moma>
    <task>
        <molecule name="1BL8.pdb"/>
        <ligand name="m001.pdbqt"/>
        <tunnel>
            <location x="82.561" y="26.438" z="46.280"/>
        </tunnel>
    </task>
    <task>
        <molecule name="4L2L.pdb"/>
        <ligand name="m113r.pdb"/>
        <tunnel>
            <location x="72.850" y="27.137" z="22.147"/>
            <location x="68.224" y="25.813" z="24.235"/>
        </tunnel>
    </task>
 </moma>
```

Figure 7.2: An example program utilizing the AutoDock Vina library wrapper.

7.2.3 AutoDock Vina Wrapper Usage

The wrapper was created for the AutoDock Vina library version 1.2.1. It however does not rely on a specific version and, in theory, should work with if the necessary functions are present. We have found several bugs in the implementation regarding the usage of quaternions. We therefore recommend the usage of the provided version, in order to prevent any further bugs from occuring.

An example program, which utilizes the wrapper, is found below in *Figure 7.3*.

```
#include <iostream>
#include <vector>
#include <stdio.h>
#include "parse_pdbqt.h"
#include "MoleculeWrapper.h"

using namespace std;
int main(int argc, char *argv[]) {

    string name = "m111.pdbqt"; // Name of the ligand PDBQT file
    MoleculeWrapper a(name);  // Initialize the wrapper

    printf("Loaded the model\n");
    printf("DOF: %zu\n", a.get_DOF());

// Fill the angles vector according to the number of DOF
    vector<double> angles;
    for(int i=0;i<a.get_DOF(); i++){
        angles.push_back(0);
    }

// Get coordinates of the individual atoms in the provided conformation
    vector<vector<double> > coords = a.get_coordinates(angles);

// Print the coordinates
    printf("Coordinates:\n");
    for(size_t i=0; i<coords.size(); i++) {
        printf("   %1.3f, %1.3f, %1.3f\n", coords[i][0], coords[i][1], coords[i][2]);
    }

    // Get intramolecular energy of the provided conformation
printf("Energy:\n %.9f J\n", a.calculate_ligand_energy(angles));

// Print energies for multiple angles
    for(int i=1;i<6;i++) {
        for (int j = 0; j < a.get_DOF(); j++) {
            angles[j] = 3.141592 * ((double) i/2.0);
        }
        coords = a.get_coordinates(angles);
        std::printf("Energy:\n %.9f J\n", a.calculate_ligand_energy(angles));
    }

// Export to PDBQT file
    a.export_to_pdbqt(angles, "NewFile.pdbqt");

    string receptor = "m111_rigid.pdbqt"; // Name of the ligand PDBQT file
    string ligand = "m111.pdbqt"; // Name of the receptor PDBQT file

// Initialize the wrapper with both the ligand and receptor
    MoleculeWrapper b(ligand,receptor);

    // Calculate intra + intermolecular energy of both the ligand and the receptor molecules
    for(int i=0;i<6;i++) {
        for (int j = 0; j < a.get_DOF(); j++) {
            angles[j] = 3.141592 * ((double) i/2.0);
        }
        qt quat = qt(0, i, i, 1);
        quaternion_normalize(quat);
        printf("Energy:\n %.9f J\n",
         b.calculate_intermolecular_energy(angles, Vec3D(i,0,0), quat));
    }
}
```

Figure 7.3: An example program utilizing the AutoDock Vina library wrapper.

7.3 Further Implementation Details

7.3.1 PDB File Parsing

All of the information regarding ligand and receptor molecules from the x–ray crystallography data is provided in the *PDB* file format (*pdb.org*). This file contains specified amount of fixed-width columns and can easily be parsed, using a manual provided for the file format [46]. One section particularly interesting for our purposes is the detailed description of "ATOM" records, which provides all the required details for parsing molecular geometric data, available in *Figure 7.4*.

COLUMNS	DATA TYPE	FIELD	DEFINITION
1 - 6	Record name	"ATOM "	
7 - 11	Integer	serial	Atom serial number.
13 - 16	Atom	name	Atom name.
17	Character	altLoc	Alternate location indicator.
18 - 20	Residue name	resName	Residue name.
22	Character	chainID	Chain identifier.
23 - 26	Integer	resSeq	Residue sequence number.
27	AChar	iCode	Code for insertion of residues.
31 - 38	Real(8.3)	x	Orthogonal coordinates for X in Angstroms.
39 - 46	Real(8.3)	y	Orthogonal coordinates for Y in Angstroms.
47 - 54	Real(8.3)	z	Orthogonal coordinates for Z in Angstroms.
55 - 60	Real(6.2)	occupancy	Occupancy.
61 - 66	Real(6.2)	tempFactor	Temperature factor.
77 - 78	LString(2)	element	Element symbol, right-justified.
79 - 80	LString(2)	charge	Charge on the atom.

Figure 7.4: Details of the "ATOM" record format. Courtesy of Worldwide Protein Data Bank [47]

Functionality for *PDB* file parsing is provided in class "*PDBParser*", located in file "*pdb_parser.cpp*". Functionality for *PDBQT* file parsing is provided in the form of AutoDock Vina library.

7.3.2 Pymol Exporting

We have extensively utilized the program *Pymol* (a tool for biochemical visualizations, mainly focused on protein molecules), mainly for the purposes of image rendering and data processing. *Pymol* is capable of visualizing both static and dynamic molecules, with a wide array of options for data visualization.

Functionality for exporting into a *Pymol* compatible format is provided in the class "*TrajectoryExporter*" in the file "*protein_trajectory_export.cpp*".

7.4 Utilized Pymol Commands

Below is attached a list of commands utilized during data visualization and processing:
show spheres — turns on spherical atom representation
frame — switches to frame n
mview store — saves the camera position to the current animation sequence
mview reinterpolate — interpolates saved camera positions in the current animation sequence
cmd.get_object_matrix(object, state) — return transformation matrix of the camera
setView() — set the transformation matrix of the camera
util.color_objs("all") -color each object with a different color
fetch — downloads the protein file and builds a biological unit from a protein fragment
png — saves an image file
set sphere_transparency, 0.5, < name > — alters sphere transparency
fetch 2ci2, type = pdb1, multiplex = 1, async = 0 — downloads and displays the selected protein
set orthoscopic, off — disables orthoscopic rendering

Change atomic radii according to the temperature factor:
load tunnel.pdb, tun
alter tun, vdw = b
rebuild

Color atoms according to their temperature factors and render an image:
spectrum b, rainbow, minimum = 0, maximum = 1
set transparency, 0.5, < name >
ray 2400, 2400
png file, dpi = 300