Bachelor Project

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Control Engineering

Sampling-based tunnel detection in protein dynamics

Ronald Krist

Supervisor: Ing. Vojtěch Vonásek, Ph.D.
Field of study: Cybernetics and Robotics
Subfield: Systems and control
January 2018

Acknowledgements

Děkuji ČVUT, že mi je tak dobrou *alma mater*.

Declaration

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškerou použitou literaturu.

V Praze, 6. January 2018

Abstract

The issue of protein tunnel detection is an up-to-day problem with many real world applications, such as drug design or protein engineering. Many tools based on different principles already exist, with approaches based on computational geometry being the most popular. However, alternative methods can offer solutions for specifications of the task that are more difficult for the current methods. These specifications can be the detection of tunnels in molecular dynamics or finding tunnels for ligands of a specific shape. An alternative approach based on algorithms used in robotics, specifically on the sampling-based motion planning algorithm RRT is explored in this thesis. The goal is to take an already existing solution, the TOM-RRT and explore its possible extension for the tunnel detection in the protein dynamics.

Keywords: protein tunnel detection, sampling-based motion planning, protein dynamics

Supervisor: Ing. Vojtěch Vonásek, Ph.D.

Abstrakt

Klíčová slova: slovo, klíč

Contents

1 Introduction **1**

1.1 Motivation 1

1.2 Biochemical Background 2

1.3 Problem definition 4

1.4 Thesis Objectives 7

2 State-of-the-art **9**

2.1 Grid methods 9

2.2 Voronoi diagram based methods 10

3 Sampling-based motion planning methods **13**

3.1 Introduction 13

3.2 Differences 15

3.2.1 Branch collision effect 16

4 Algorithm description **17**

4.1 Overview 17

4.2 Probe and goal region detection 18

4.3 Sampling 18

4.4 Sample connection 19

4.5 Tunnel analysis 22

4.6 Multiple tunnel detection 23

4.7 Tunnel similarity analysis 25

4.8 Molecular dynamics 29

4.8.1 Pruning 29

4.8.2 Clustering 30

4.8.3 Centering 31

4.8.4 Smoothing 32

4.9 Implementation details 33

5 Experimental results **35**

5.1 Static tunnel detection 35

5.1.1 1TQN 36

5.1.2 1MXTa 39

5.1.3 1AKD 42

5.1.4 Comparison with the TOM-RRT 46

5.1.5 Dynamic search 46

6 Conclusions **47**

6.1 Test — this is just a little test of something in the table of contents 47

6.1.1 Yes, table of contents 47

7 Bibliography **49**

Index **51**

A Project Specification **53**

Figures

1.1 Protein structures illustration [from https://courses.lumenlearning.com/boundless-chemistry/chapter/protein-structure/] 3

1.2 Protein cavities illustration, taken from [1]. 4

1.3 Illustration of tunnel definition, taken from [8]. 6

1.4 Illustration of tunnel bottleneck, taken from [9]. 6

1.5 The blue line represents centerline of the tunnel. The tunnel is stuck in frame n, however the changing conformations of tunnel atoms allow it to continue in the next frame. The dotted part symbolizes jump through discretized time into the same spatial position in the next frame, tracing tunnel from frame n into frame $n + 1$. 7

2.1 Illustration of construction of the weighted graph. At first, Delaunay trinagualation is constructed(a). Then the Voronoi diagram is constructed from it(b). Edge e can then be evaluated by the maximum radius r, which does not cause the sphere to collide(c). Taken from [1]. 10

2.2 Visual comparison of different tools based on voronoi diagrams. A shows tunnel found in two dimensional environment found using the Voronoi diagram. B presents how compared tools deal with constructing the Voronoi diagram from variable atom sizes. C illustrates principles according to which the edge weight is computed. Taken from [10]. 11

3.1 Illustration of the RRT expansion, taken from [13] 15

3.2 Illustration of the branch collision effect. We see branches (red and black), which could form new tunnels, the black by reaching exit 1 and the red by reaching exit 2. However, a tree based method with already developed tree at this state will probably not detect them. Joining the red and the black branch is not possible, it would invalidate the tree structure. Due to the properties of the nearest neighbor search, it is virtually impossible for the black branch to reach exit 1 and for the red branch to reach exit 2. 16

4.1 Algorithm flow chart 17

4.2 The large empty space around the left down corner will result in faster expansion of the green branch. That is caused by all samples in the empty space having their nearest neighbors in the green branch. The result is lower chance of finding the tunnel formed by the blue branch. 19

4.3 Illustration of new sample connection. Light blue node represents sample, red nodes denote interpolation steps, which are not added to the tree. Green nodes represent interpolation steps, which are added to the tree. Dark blue node represents the nearest neighbor. The dotted circle represents the parameter d. Current node is added to the tree if no node closer to the nearest neighbor collides. 20

4.4 Illustration of tunnel centering and usage of blocking spheres. In a) we see an unoptimized tunnel. In b) we see the tunnel centered tunnel and blocked with a red colored blocking sphere. In c) we see the state of tree before deleting branches inside the blocking sphere, in d) after. The green colored part represents the edges belonging to the tunnel. 25

4.5 Illustration of breaking tunnels down to interval representations using two intervals. The lighter nodes represent the original tunnel, the darker the interval representation. the navy blue nodes belong to the first interval, the lighter ones to the second one. The pink ones are not used for the computation. Take into account that this is not a realistic situation, the nodes are for readability too far from each other, in real situations the interval representations copy the shape of the tunnel closer. 28

4.6 Each node in a new frame is checked for collision in the same spatial position. The red nodes collide in the new frame and are no longer considered. Dotted connections represent connections between individual frames. The light blue connections show a dynamic tunnel, which opens in frame $n + 1$ and closes in frame $n + 2$ 29

4.7 Arrows represent connections from child to parent. The doted arrows represent actual connections between nodes, the full arrows their representations in the data structures. The dotted nodes are not added into any data structure. The node farthest on the right represents new node connected to the tree. Node below which there is not any else is the one first frame one. The crossed nodes are have been pruned and set inactive in their frame. Inactive nodes collide and are no longer considered in the pruning procedure or used for nearest neighbor search. 30

4.8 The tunnel ending with purple endpoint is considered similar to the light blue tunnel and assigned to the cluster generated by it in the previous frame. 31

4.9 Illustration of centering in dynamic environment 32

4.10 Constructed matrix M. Interpolation step represents shift by w. 33

4.11 Illustration of the breadth first search. 33

Tables

5.1 Tunnels found by CAVER 3.0 in the 1TQN molecule. 36

5.2 Some tunnels found by SilVy RRT in the 1TQN molecule. 36

5.3 Some tunnels found by SilVy RRT in the 1AKD molecule with probe radius 0.6. 42

5.4 Takhle to vypada pro 50 frames a sondu 1 Å. 46

5.5 A tenhle bordel je pro pro 50 frames a sondu 0.7 Å. 46

Chapter 1

Introduction

This chapter presents the goals of this thesis, explains the motivation behind them and gives a basic overview of its structure.

1.1 Motivation

Proteins are macromolecules composed of atomic chains of amino acids. They are essential components of living organisms, for example, they make over eighty percent of human tissue [2]. Proteins have numerous functions in living organisms. They work as transport medium for other molecules, they catalyze metabolic chemical processes, they are important for immune system and play important role in DNA replication among other functions [1][2].

Proteins are able to interact with surrounding molecules or they can let other molecules such as water, ions or ligands penetrate into their structure. These molecules then can reach the so-called active sites or binding sites, where the chemical reactions between the penetrating molecule and the protein takes place. These reactions can change the properties of the penetrating molecule or the properties of the protein. The active sites can be positioned on the protein surface, but they are often located in the cavities deep inside the protein, therefore the study of tunnels leading to them can give us insights about the properties of these proteins and how to modify them to acquire the desired behavior [1].

[1] Uses two biochemical case studies to illustrate the importance of these tunnels. The first concentrated on the interaction of the DhaA haloalkane dehalogenase Rhodococcus rhodochrous protein with 1,2,3 - trichloropropane

(TCP). Researchers' task was mutating the amino acids of the protein so that it would accept the TCP molecule into its active site. Molecular simulations discovered that the main problem lies in dense concentration of water molecules near the active site. These water molecules prevented the TCP molecule to enter the active site. The goal of the mutation design was to narrow the main tunnel. The best design had all mutated amino acids along the protein's tunnels and increased the reactivity of the protein to TCP 32-fold.

The second study dealt with low resistance to common temperatures and some environments. It was done with the same protein from the previous study. Mutating few amino acids along the access tunnel increased the half-life of the protein in 40% dimethyl sulfoxide from minutes to weeks and increased its melting temperature by 19°C [3].

Due to intramolecular atomic forces and influence of the surroundings of the protein, the proteins are constantly changing their inner structure, which leads to constant rearrangement of their inner void spaces. Some tunnels may close for a short period of time, the others may open only for a moment. Study of temporal development of their cavities offers substantial advantage in understanding the behavior of the molecule, providing important insights about the stability of the tunnel in time and therefore we aim to incorporate this aspect into our tool [4][5][7].

Most state-of-the-art tools are based on computational geometry. In this thesis we aim to present approach based on different paradigm and evaluate how well it performs compared to the standard.

1.2 Biochemical Background

This section explains some basic properties of proteins. It aims to provide an understanding of the protein representations used in our problem definition and some basic information about the protein void spaces.

The processes during the protein folding give rise to the protein structure, which can be represented in different ways:

- Primary structure is given by the order of amino acids in the polypeptide chain of the protein.
- Secondary structure is determined by local folded structures in the polypeptide chain.
- Tertiary structure is the overall geometric shape of the polypeptide which

can be described by atomic coordinates.

- Quaternary structure is defined for proteins consisting of more than one polypeptide chains. It defines spatial arrangement of their polypeptide chains [2][9].

For the scope of this thesis, the tertiary structure representation of a protein is used. A visual example of these protein structure representations is given in Figure 1.1.

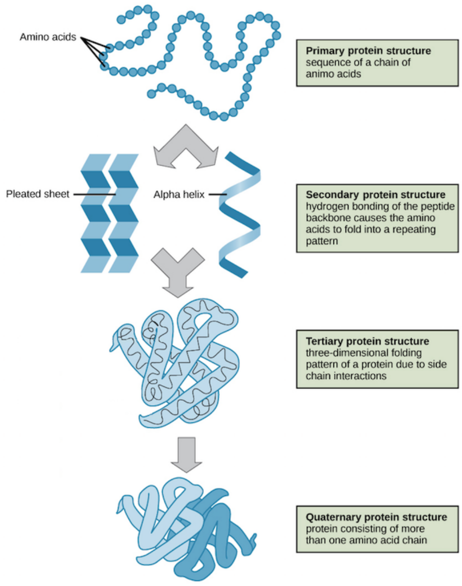


Figure 1.1: Protein structures illustration [from https://courses.lumenlearning.com/boundless-chemistry/chapter/protein-structure/]

Several types of void spaces inside the protein can be identified. A tunnel is represented by the void space between the atoms of the protein, which forms a pathway between an active site located deep inside the protein and the outer environment of the protein, called solvent. A molecule small enough to fit into the void space can use it to get to the active site.

Intramolecular tunnels allow transport of reaction intermediates between two distinct active sites inside the protein. Pores or channels are pathways leading through the protein molecule without interruption by inner cavity. They transport molecules across biological membranes. Cavities are empty places inside the protein which are not directly accessible from the protein's surface. On the other hand, pockets are spots on the protein surface which are easily accessible by ligand [1]. Both can also contain active sites. Examples of all mentioned structures are in figure 1.2.

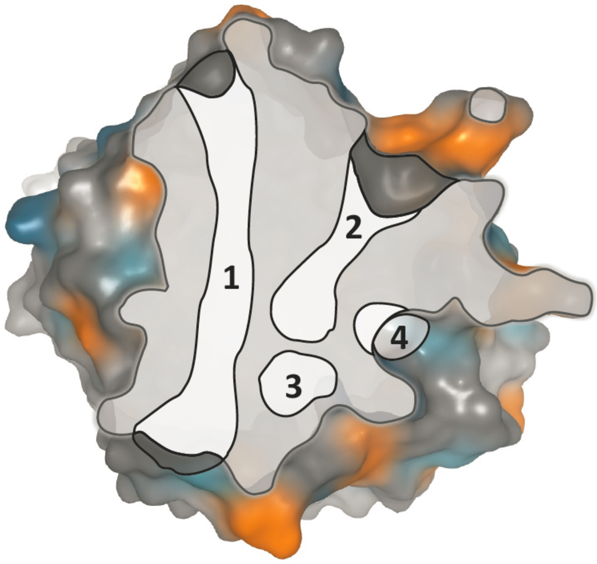


Figure 1.2: Protein cavities illustration, taken from [1].

Molecular dynamics are obtained by molecular dynamics simulations. These simulations work by approximating the molecular forces acting on each atom, moving each atom according to these forces and advancing the simulation time. As the output, we get series of snapshots describing position of each atom in each time step. The time step is usually 1 or 2 fs. In the scope of this thesis, these frames are used as the basis for detecting tunnels in molecular dynamics [15].

1.3 Problem definition

The task of a protein tunnel detection tool is to find a collision-free path for a ligand molecule leading from a point inside a protein to a point on its surface. For the scope of this thesis, we consider only spherical ligands. The search takes place inside an ensemble of molecular dynamics.

Formally, a tunnel was defined in [8]. We are expanding that definition for search in frames representing the molecular dynamics. Using the tertiary protein structure makes an approximation of the protein structures by a sequence of spheres possible [1]. Let the set of all spheres be denoted as S and let the set of spheres representing atoms of a protein in a particular frame be denoted A. Sphere s is defined as $s(p, r)$, where $p(c, t)$ is a time-space point defined by a spatial coordinate $c \in \mathbb{R}^3$ representing its center and a time coordinate t representing time, in which the point is located. Since the frame representation of the molecular dynamics leads to discretized time, $t \in \mathbb{N}$ represents a frame number instead of an actual time record. The frames are sorted ascendantly by the time taken. $r \in \mathbb{R}$ represents the radius of the sphere.

Let $c(x)$ return a 3D position of a sphere or a point and let $t(x)$ return the time coordinate. Let $r(x)$ return the radius of the input sphere x. Let $D(c, s)$ compute distance of a spatial coordinate and a sphere:

$$D(c, s) = \|c - c(s)\| - r(s). \tag{1.1}$$

Where c is a 3D position and a is a sphere.
Let $R(x)$ compute the shortest distance of a time-space point towards the nearest atom in the protein structure:

$$R(p) = min\{D(c(p), a) | a \in A, t(a) - t(p) = 0\}. \tag{1.2}$$

Now we can formulate the intuitive concept of the tunnel. In [8], tunnel is defined by its centerline and volume. The centerline a_t is a continuous time-space curve leading from a point inside the protein to a point on the protein surface. For the purpose of this thesis, the tunnel is discretized and represented by a sequence of spheres $s_1 ... s_n$ with centers positioned on the centerline, ordered by their succession towards the point on the protein surface. We call the sphere s_1 the root node and the s_n endpoint. Since we can move through time only into the future, the following must be true for a valid tunnel centerline. For all spheres s_i on the centerline except for the root node and the endpoint:

$$t(s_{i-1}) \leq t(s) \leq t(s_{i+1}). \tag{1.3}$$

s_{i-1} is not defined for the root node and s_{i+1} for the tunnel endpoint. Only the defined parts of the condition must be true for them. Volume V is the union of the spheres belonging on the centerline with radius equal to the distance towards the closest atom. Let $p(s)$ return a time-space point of a sphere. Formally, a tunnel is defined as:

$$T = \bigcup_{s \in V} s, \quad r(s) = R(p(s)), \quad R(p(s)) > 0. \tag{1.4}$$

((Moc nevim jak tohle spravne zapsat, takhle se tunel = volume v podstate))

Illustration of a static tunnel is given in Figure 1.4.

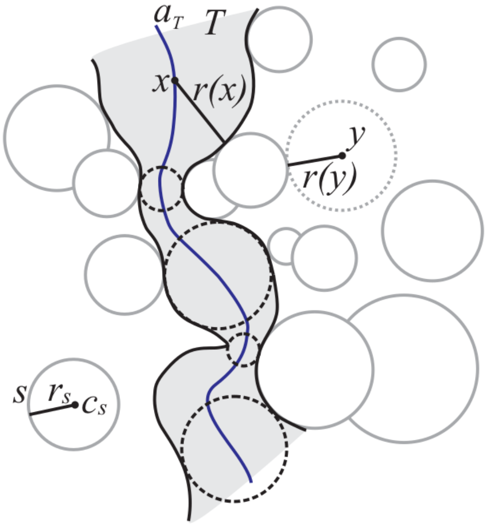


Figure 1.3: Illustration of tunnel definition, taken from [8].

Among the most useful characteristics of a tunnel is the size of its bottleneck, the narrowest part of the tunnel [9]. Illustration is provided is Figure 1.3.

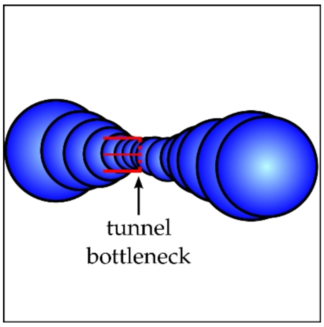


Figure 1.4: Illustration of tunnel bottleneck, taken from [9].

The algorithm should be able to compute the length of the tunnel and its bottleneck. To find the bottleneck, we have to develop a procedure, which will approximately put the center of every sphere composing the tunnel into a line going through center of the tunnel, so that the sequence of spheres composing the tunnel would have the largest possible radii. Bottleneck is then defined as the minimum value of radius of the spheres composing the tunnel.

$$l = \min_{s \in T} r(s).. \tag{1.5}$$

The tunnel length is defined as the sum of distances of all consequent spheres.

$$v = \Sigma_{s_i \in T \setminus s_n} \|s_{i+1} - s_i\|. \tag{1.6}$$

Where s_n is the tunnel endpoint.

Our approach is different from the state-of-the-art geometric approaches, which detect tunnels in individual frames and cluster them by their similarity. We are tracing the tunnel through individual frames, as can be seen in Figure 1.5. Tunnels located only in one frame are called static tunnels, tunnels spanning through multiple frames dynamic tunnels.

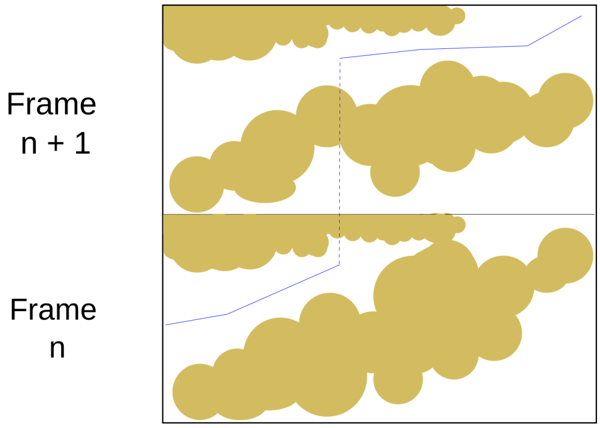


Figure 1.5: The blue line represents centerline of the tunnel. The tunnel is stuck in frame n, however the changing conformations of tunnel atoms allow it to continue in the next frame. The dotted part symbolizes jump through discretized time into the same spatial position in the next frame, tracing tunnel from frame n into frame $n + 1$.

1.4 Thesis Objectives

The goal of this thesis is to present a tool for detecting tunnels in the protein dynamics based on the sampling-based motion planning methods. The tool takes pdb files representing the protein dynamics, starting position of the tunnels and a set of parameters as an input. PDB stands for protein tunnel databank. At the end of 2014, pdb repository stored 105 400 protein structures, which made it the home of majority of experimentally determined structures by the time being [6].

The output consists of all found tunnels represented as sequences of spheres. The user should be able to filter tunnels which are too similar to each other that they do not provide any useful information. Tunnels spanning through molecular dynamics should be put into clusters based on their spatial similarity. The output also contains information about the length and bottleneck of found tunnels. The output tunnels are saved in pdb files and can be visualized in a protein visualization software such as PyMOL.

Our tool will be compared against protein tunnel detection program CAVER 3.01. The criteria include the speed of the detection, memory complexity, the number of found tunnels. We also analyze various theoretical differences in the search mechanisms with the current state-of-the-art approaches.

Our solution is based on already existing algorithm, called the TOM-RRT [9]. The TOM-RRT is an algorithm based on sampling-based motion planning with capability of detecting tunnels in the static environment. Our main task is to extend its capabilities for the dynamic search.

Chapter 2 provides brief overview of the state-of-the-art approaches. Chapter 3 explains random-sampling based motion planning methods. Chapter 4 explains the workings of our tool. Finally, Chapter 5 evaluates the results.

Chapter 2

State-of-the-art

This chapter presents a review of the state-of-the-art tools used. Numerous new tools and methods for tunnel detection and classification have already been developed. Typical approaches forming core of those tools include grid based methods and using the Voronoi diagram [9].

2.1 Grid methods

Methods based on the grid approach work by bounding the entire protein by a boundary box and rasterizing the bounded space using regularly distributed grid in the three dimensional space. Modeling the proteins by spheres allows checking grid nodes for collision. Grid composed of non-colliding nodes then forms a graph, in which paths can be found using any standard graph-traversing algorithm, such as the Dijkstra's algorithm or A*.[9][7]
Tools representing this approach are for example HOLLOW and 3V [1].

Major disadvantages coming with the grid based approaches are dependency of the accuracy on the resolution of the grid. Also, CPU and memory requirements can limit grid methods only to smaller systems [10].

2.2 Voronoi diagram based methods

To overcome the major disadvantages in the grid based methods, an approach based on the Voronoi diagrams was developed [1].

The Voronoi diagram is a geometric structure created by splitting an n-plane into convex shapes so that every point inside the shape is closer to the point generating the shape than to any other point in the plane.

In practice, Voronoi diagram construction exploits its duality with the Delaunay triangulation, which is constructed from the set of points denoting centers of the atoms. A weighted graph is constructed from the Voronoi diagram. Nodes of the graph are represented by the Voronoi vertices. An edge of the graph is created if two Voronoi vertices are connected. The weight of the edge is set by an evaluation function, which can take into account the distance to the nearest atom and the length of the edge. The resulting graph is then traversed by graph search algorithm to discover connections.

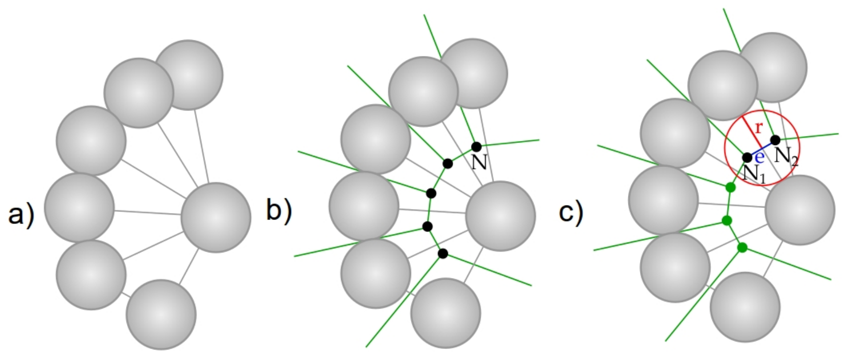


Figure 2.1: Illustration of construction of the weighted graph. At first, Delaunay trinagualation is constructed(a). Then the Voronoi diagram is constructed from it(b). Edge e can then be evaluated by the maximum radius r, which does not cause the sphere to collide(c). Taken from [1].

Complications arise from differences between Van der Waals radii of the individual atoms in the structure, causing the distances in the resulting diagram to be distorted when constructed by above described way. This error is either considered insignificant enough and ignored (Mole)[10], or approximation using a number of smaller spheres is implemented (Caver, Molaxis).

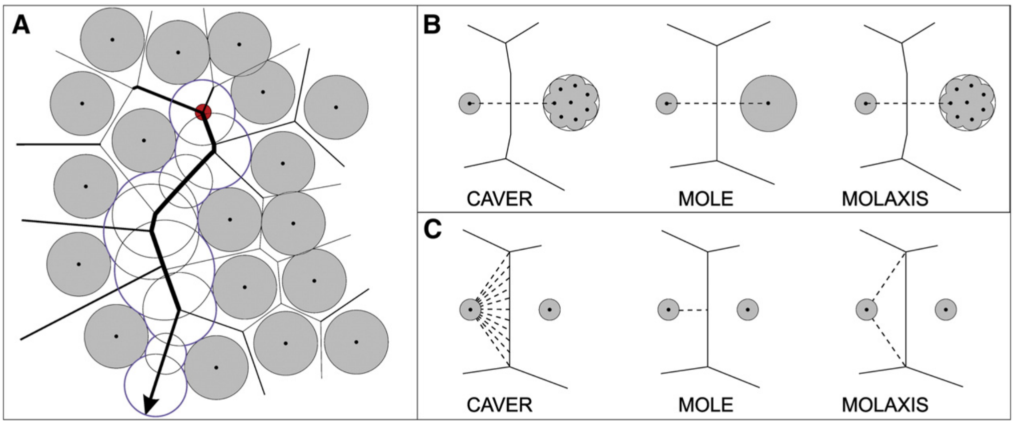


Figure 2.2: Visual comparison of different tools based on voronoi diagrams. A shows tunnel found in two dimensional environment found using the Voronoi diagram. B presents how compared tools deal with constructing the Voronoi diagram from variable atom sizes. C illustrates principles according to which the edge weight is computed. Taken from [10].

Molecular dynamics are addressed (Caver, Mole) by clustering algorithms. Tunnels from different snapshots are assembled into clusters by some function evaluating their similarity. Molaxis uses split distance parameter limiting the distance from the last common node [11].

Chapter 3

Sampling-based motion planning methods

3.1 Introduction

Motion planning is a term most commonly used in robotics. In robotics motion planning involves a task of getting a robot to automatically determine how to move while avoiding collisions with obstacles.

The basic path planning problem is summarized as follows: Given an initial placement of the robot, compute how to gradually move it into a desired goal placement so that it never touches the obstacle region. Consider the task in terms of input and output of the algorithm. The input consist of an initial placement of the robot, a desired goal placement, and a geometric description of the robot and an obstacle region. The output is a description of how to move the robot gradually from its initial placement to the goal placement while never touching the obstacle region. The output description will be a path through the set of all intermediate transformations of the robot, from start to finish. [12]

It is possible to describe the problem of tunnel detection from this perspective. The obstacle region is defined as the set of balls representing the protein molecule, the ligand molecule is represented as the robot for which the motion is planned, the start is some place inside the protein and the goal is defined as a location lying outside of the molecule. With this definition, after some modifications, one of the well-established algorithms can be used to find trajectory representing the tunnel.

In general case the ligand molecule has six degrees of freedom. In the scope of this thesis only spherical ligands are assumed, therefore only three degrees of freedom are needed, leaving out the rotations of the ligand molecule. Let the spherical ligand be called probe. Let the configuration space C be space of all available spatial positions in three dimensional space inside the boundary enclosing the protein molecule in some frame. Single configuration is defined as $q \in \mathbb{R}^3$. Let the atoms representing the protein in some frame A be the C_{obs}, then the free space can be defined:

$$C_{free} = \{q \in C | s \bigcap A \in \varnothing\}, \tag{3.1}$$

where $s \in S$ represents a sphere with radius of the probe with $c(s) = q$. Let C_{out} represent the part of the configuration space which is considered to be outside of protein molecule. Then the goal region is equal to C_{out}. The concrete definition of this region is left to the needs of the specific implementation of the algorithm. It could for example be the space of all configurations lying outside the α shape of the protein.

Main approaches used for solving the motion planning problems are based on combinatorics and sampling. Main idea of the sampling-based methods is to avoid explicit construction of the obstacle configuration space. This idea offers important advantage in problems with thousands or even millions of geometric primitives, such problems would be practically impossible to solve with methods involving explicit construction of the obstacle configuration space. [12]

One of the algorithms based on sampling is the RRT (Rapidly Exploring Random Tree). Its main idea is taking random samples from the configuration space and connecting them to a tree structure until a path to place in some arbitrary distance from goal is found. [13]

Let's assume that C is a metric space. The tree structure used by the RRT is a topological graph, $G(E, V)$. The exploration algorithm begins by initiating the first vertex at start position, q_{start}. Then, for k iterations, C is randomly sampled according to some sampling strategy. When a new sample q_{rand} is produced, the algorithm tries to connect it to its closest node in the tree, q_{near}. The path to closest point is checked for collision by a vertex interpolating function extend q_{near} towards q_{rand} using some collision detection function and connected to the closest point from the first point, after which the path to the closest point is considered as collision free. The closest point can be found analytically or an approximation by creating intermediate vertices along tree's line segments and searching for the nearest vertice instead can be used. Pseudocode is given in Algorithm 1 and illustration of tree expansion is in figure 3.1.

Algorithm 1 Original RRT. [16]

$Tree.add(q_{start})$
while $iteration < k$ **do**
  $q_{rand} = random\ configuration$
  $q_{near} = nearest\ neighbor\ in\ tree\ T\ to\ q_{rand}$
  $q_{new} = extend\ qnear\ toward\ q_{rand}$
  if q_{new} $can\ connect\ to$ q_{near} **then**
    $Tree.addVertex(q_{new})$
    $Tree.addEdge(q_{near}, q_{new})$
  end if
  if $\varrho(q_{new}, q_{goal}) < distanceToGoalTh$ **then**
    $break$
  end if
end while

45 iterations   2345 iterations

Figure 3.1: Illustration of the RRT expansion, taken from [13]

3.2 Differences

Unlike methods based on geometry used in mentioned standard approaches, sampling-based motion planning algorithms are incomplete. That means, it is not assured that the algorithm will decide in a finite time whether a solution exists. They instead rely on notion of probabilistic completeness, meaning that the probability of finding a solution converges to one with enough samples, given that the solution exists, but they can run forever if the solution does not exist [13].

The usage of random sampling makes sampling-based methods nondeterministic, therefore different outcome is expected from individual runs of a tool based on them. The algorithm can find different tunnels with the same inputs. From the functional perspective, the nature of motion planning paradigm offers two important distinctions: Firstly, they were developed for planning motion of robots with many degrees of freedom. That means they are naturally extensible for planning with non-spherical ligand composed from more atoms. Secondly, by using four dimensional configuration space, $\mathbb{R}^3 \times Time$ they allow direct search in molecular dynamics without need for clustering between tunnels from individual snapshots.

3.2.1 Branch collision effect

All of the mentioned state-of-the-art methods form a graph, which is then traversed by a graph search algorithm. However, motion planning methods can use a tree structure instead. Tree structure prevent detection of some valid tunnels due to what we call the branch collision effect. Illustration is given in Figure 5.2.

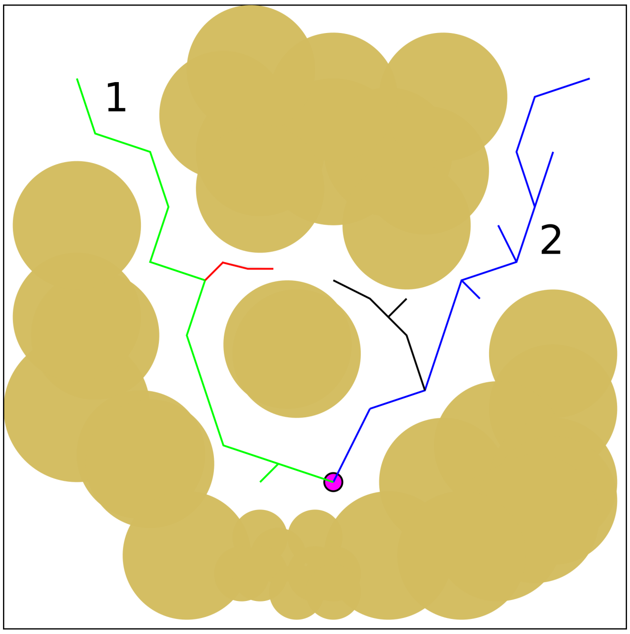


Figure 3.2: Illustration of the branch collision effect.
We see branches (red and black), which could form new tunnels, the black by reaching exit 1 and the red by reaching exit 2. However, a tree based method with already developed tree at this state will probably not detect them. Joining the red and the black branch is not possible, it would invalidate the tree structure. Due to the properties of the nearest neighbor search, it is virtually impossible for the black branch to reach exit 1 and for the red branch to reach exit 2.

Chapter 4
Algorithm description

4.1 Overview

The goal of this thesis is to provide an algorithmic solution for dynamic tunnel detection based on the original RRT algorithm. The original RRT needs extensions such as search for multiple tunnels, processing important information about individual tunnels and support for search in the molecular dynamics. Algorithm presented is inspired by the TOM-RRT algorithm described in [9]. TOM-RRT offers required functionality for detecting protein tunnels in the static environment, therefore our main task is to extend its functionality with the ability to detect protein tunnels in the dynamic environment. This algorithm was chosen because it already offers RRT based solution for some needed functionality.

Our tool, called SilVy RRT, was implemented from scratch, testing various different approaches. The reason is to test some modifications of the original version, which could perform better for our purpose. The basic outline of SilVy RRT is described in Figure 4.1.

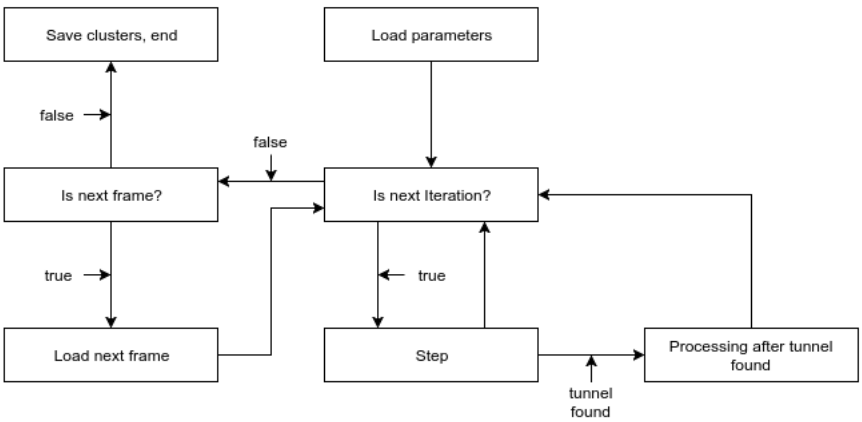


Figure 4.1: Algorithm flow chart

4.2 Probe and goal region detection

The probe serves the function of the robot in motion planning task. The probe in our tool is limited to a spherical object with user defined radius. This approach allows detection of tunnels with specified minimal bottleneck.

Goal state is defined by a new node lying outside of the protein structure, in C_{out}. To validate this condition, our algorithm uses procedure similar to rolling probe method used in 3V[14]. A probe called testing sphere with radius defined by u is placed into the center of the node and tested for collision. If the test fails, the node is considered to lie outside of the protein. The parameter u is defined by the user. The user should keep in mind that too small value can lead to mistakingly identifying inner cavities of the protein as the actual empty space around the protein. Reasonable value can be between 4-5Å. For collision detection an external library is used. We have used OZCollide as in the TOM-RRT.

4.3 Sampling

The sampling region is the area from which the random samples are taken. It is necessary to stretch it over the whole protein molecule and some space around it. If not, we can end in a situation where the tunnel endpoint is missing because none of its potential locations lie in the sampling region.

The sampling region is found automatically by function analyzing atom coordinates in pdb files and locating atoms lying on the edges of the molecule. They are selected by extreme values of their Euclidean coordinates. A cubical area with vertices defined by the extreme values shifted by u multiplied by a parameter m towards the outside of the protein is used. This is the same approach as in the TOM-RRT.

The sampling region does not exactly follow the shape of the protein molecule. That can lower the probability of finding tunnels with endpoints lying near smaller empty spaces behind them. Figure 4.2 offers visual explanation.

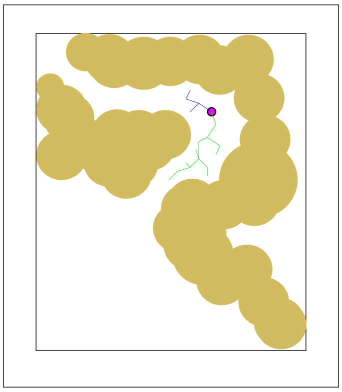


Figure 4.2: The large empty space around the left down corner will result in faster expansion of the green branch. That is caused by all samples in the empty space having their nearest neighbors in the green branch.
The result is lower chance of finding the tunnel formed by the blue branch.

Two approaches for sampling probability distribution were tested. One is a simple uniform distribution. The second one is designed to reduce the effect of the large empty spaces outside the protein. It is based on lowering the probability of sampling from the outside of the protein. That could also allow more time to be spent on developing the tree in more difficultly reachable regions rather than reaching to new tunnels in the already explored parts.

The sampling function tries to put a sphere with radius $u$inside the generated sample. If the testing sphere does not collide, the sample is marked as lying in outer environment. Parameter b is used for quantification of this bias. It describes the percentage of samples required to lie inside the protein. For every sample, a random number in interval(0, 1) is generated. If it lies under the threshold defined by b, the sample is required to lie inside the protein. If it does not, another sample is created. If the random number crosses the threshold, the sample can lie anywhere in the configuration space.

Testing revealed that increase in b value leads to larger count of found tunnels and longer computing times. Experimental data are presented in Chapter 5. Generally, while the increase in computing times remains the same, the increase in number of found tunnels is less significant for larger parameter value. After the experiments, the default value was set to 0.75. User can use whichever value he/she wants to by editing the config file of the application.

4.4 Sample connection

When a new sample is generated, we use external library MPNN to find its nearest neighbor node in the tree. TOM-RRT uses the same library. If the distance between the new sample and its nearest neighbor is larger than parameter d, it is moved in a straight line so the distance is equal to d. Then we position the probe incrementally towards the nearest neighbor. The probe is moved on the straight line by w parameter distance and checked for collision until the distance is smaller than w. The tree edge begins on the first place after which the probe does not collide. Tree node is added at the beginning of the edge and at every spot after it, which was checked for collision. The process is depicted in Figure 4.3. The pseudocode for connecting samples is given in Algorithm 3 and for interpolation of sample in the Algorithm 4. Algorithm 4 checks interpolated coordinates from the sample to the nearest neighbor, adds them to temporary container and saves container index for the first valid node, than goes from the nearest neighbor to first valid node and adds nodes to tree.

Algorithm 2 mainStep

```
parameters ⟵ readParameters()
counter ⟵ 0
while isNextFrame() do
    while counter < parameters.iterations do
        sample ⟵ createSample()
        nearestNeighbor ⟵ findNearestNeighbor(sample)
        connectSample(sample, nearestNeighbor)
        counter++
    end while
end while
```

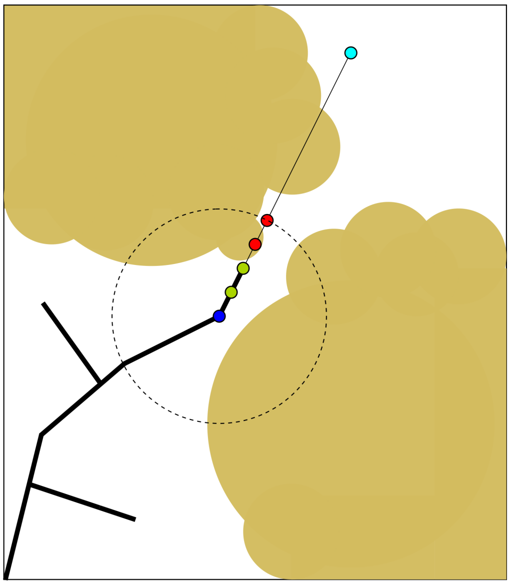


Figure 4.3: Illustration of new sample connection.
Light blue node represents sample, red nodes denote interpolation steps, which are not added to the tree. Green nodes represent interpolation steps, which are added to the tree. Dark blue node represents the nearest neighbor. The dotted circle represents the parameter d. Current node is added to the tree if no node closer to the nearest neighbor collides.

Algorithm 3 connectSample(sample, nearestNeighbor)

```
distance ⟵ computeDistance(sample, nearestNeighbor)
if distance > maxStepDistance then
    moveNodeToAnother(sample, nearestNeighbor, distance - maxStepDistance)
end if
return interpolateSegment(sample, nearestNeighbor)
```

Algorithm 4 interpolateSegment(fromCoordinates, toCoordinates, distance)

```
curCoordinates ⟵ fromCoordinates
isPreviousColliding ⟵ true
counter ⟵ 0
while distance > step do
    moveCoordinates(curCoordinates, toCoordinates, step)
    isCurColliding ⟵ cur.isColliding(probeRadius)
    if !isCurColliding && isPreviousColliding then
        firstNodeIndex ⟵ counter
    end if
    isPreviousColliding ⟵ isCurColliding
    distance ⟵ distance - step
    tempNodeContainer.add(curCoordinates)
    counter++
end while
if isPreviousColliding then
    return
end if
index ⟵ tempNodeContainer.indexAtLastElement
break ⟵ false
while !break do
    curCoordinates ⟵ tempNodeContainer.get(index)
    addNodeToTree(curCoordiantes)
    if areInGoalRegion(coordinates) then
        afterTunnelFoundProcedure()
        return
    end if
    if index == firstNodeIndex then
        break ⟵ true
    end if
    index−−
end while
```

4.5 Tunnel analysis

Raw tunnels found by the RRT algorithm do not provide ideal results to work with. Their nodes are placed randomly in the tunnel instead of in its center. This leads to loss of important information about the bottleneck. Therefore, our algorithm should be able to process the raw tunnel to make it more suitable for the analysis. Our algorithm implements two processes for the tunnel optimization: centering and smoothing.

The tunnel centering procedure ideally puts every node into its center position in the tunnel. The center position of the node can be described using a normal plane defined by a direction vector pointing from the parent of the centered parent to the centered node itself. The center position is then the position in this plane with the longest distance to the surface of nearest neighbor atom.

However, computing the exact position is computationally expensive [9]. We instead approximate the position using method similar to the one in TOM-RRT. We divide the circle around centered node in the normal plane into number of angles defined by parameter e and try to move the node in the angle's direction by f distance. For the circle division we use function

$$(i * 2 * \pi)/(e - 1). \tag{4.1}$$

The angle is defined in radians and i is integer ranging from 0 to e - 1. Than we use function approximating the largest non-colliding radius. The pseudocode for radius approximation is in Algorithm 5. Finally, the node is moved in the direction which achieved the largest increase and f is decreased. This cycle continues until the f falls beyond some threshold value.

The centering procedure is applied to every node of the tunnel until the tunnel ends or some radius of some node after the procedure is larger than the testing sphere. If that happens, we consider the currently centered node lying outside of the protein and delete the rest of the tunnel. If a node was centered before, it is simply copied. Section 4.10 describes how an already centered node is recognized. After the procedure, the user of the algorithm can analyse properties of the tunnel dependent on width of the tunnel, such as the size of its bottleneck.

We found situations when the tunnel even after the centering procedure is too far outside the protein, lying closely on its surface. To get rid of those parts, we run the centering procedure again, this time from the endpoint. The centered nodes are not saved and the f is higher. The procedure runs until it reaches the root node or runs into an already centered node. The part of the tunnel after last found node with radius larger than the testing sphere is deleted. The difference between uncentered and centered tunnel is illustrated in Figure 4.4 in Section 4.7. The difference between TOM-RRT is in using a deterministic procedure instead of simulated annealing and omitting the move of neighbors of the centered node. Also the cutting procedure is not used in the TOM-RRT. (Obrazek k centrovani node jeste dodam)

Algorithm 5 approximateRadius(node, startApproximationStep, precision)

```
approximationStep = startApproximationStep
curRadius = node.radius
lastValidRadius = node.radius
while approximation_step > precision && curRadius < testSphereRadius do
    testRadius ⟵ curRadius + approximationStep
    hasCollided ⟵ isColliding(node.coordinates, curRadius + approximationStep)
    if !hasCollided then
        curRadius = testRadius
        lastValidRadius = curRadius
    else
        approximationStep /= 2
        curRadius -= approximationStep
    end if
end while
return lastValidRadius
```

To make the tunnel shorter and less jagged, we use the smoothing function, which take two neighbors of a node and tries to connect them in a straight line. The path segment between them is checked for collision. If the path segment does not collide, the middle node is removed. It is the Optimization-PhaseOneAlgorithmTwo from TOM-RRT, run over every triple of the tunnel. This procedure is repeated until the percentage of successful attempts falls below the parameter k. It is also not applied on nodes which are already too far away from themselves. By this we prevent having empty spaces in tunnel visualization.

The centering is required to take place in tunnel search algorithm. That is because locating the actual end of the tunnel depends on having center positions of tunnel nodes available. However, the smoothing procedure is run after all the tunnels are found. That allows us to save time by omitting smoothing of duplicate paths. The TOM-RRT runs the smoothing in the search phase and OptimizationPhaseOneAlgorithmOne is omitted here.

4.6 Multiple tunnel detection

The original RRT algorithm was not designed for multiple paths search. It terminates after the first path is found. Extending it to continue its search after finding the first path leads to the same path being quickly rediscovered because the tree is already in the area near the C_{out}.

A possible solution is based on the concept of "disabled areas", described in the TOM-RRT algorithm [9]. Disabled areas are parts the of configuration space which serve as obstacles for the search part of the algorithm. They are used to block the area around which the end of current path was found, therefore blocking the tree growth in it and preventing rediscovery of the same path.

Spheres with the radius of endpointcalled blocking spheres are added to C_{block}. C_{block} is considered as separate part of the configuration space. The reasoning is that it works as C_{obs} only in sample connection step. If a tunnel was found to be similar, the radius of its blocking sphere gets multiplied by parameter g multiplied by count of already found similar tunnels with same one. This increase is limited by some threshold value. Similar tunnels are defined in section 4.7.

The blocking spheres invalidate part of the tree lying in the C_{block}. The tree after the centering procedures can span areas after the blocking sphere. Two approaches for this complications were tried:
The first one is deleting the whole tree and starting again from scratch with modified configuration space. This approach was used in the TOM-RRT [9]. The second one should remove every branch of the tree lying inside the blocking sphere. However, we found that just removing the part of the tree spanned by tunnel node before the blocking sphere works better. Going through the whole tree was too time-consuming.

Experiments showed different results for each of these approaches. Deleting the whole tree works better in environments with many shorter tunnels, while the second approach performed better in finding longer tunnels. Deleting the whole tree also leads to shorter running times in some of the experiments, especially in the ones with high number of found tunnels. The comparison of results of these methods is presented in Chapter 5.

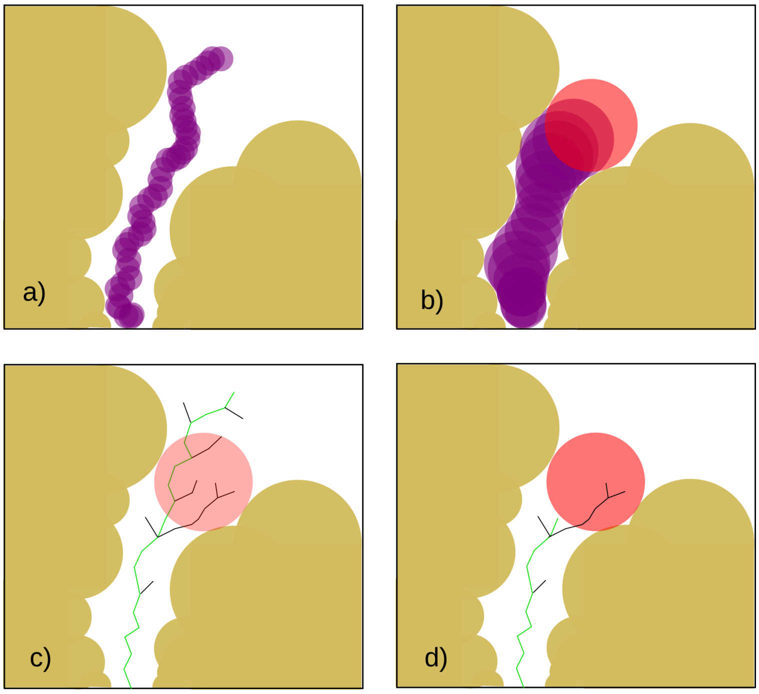


Figure 4.4: Illustration of tunnel centering and usage of blocking spheres. In a) we see an unoptimized tunnel. In b) we see the tunnel centered tunnel and blocked with a red colored blocking sphere. In c) we see the state of tree before deleting branches inside the blocking sphere, in d) after. The green colored part represents the edges belonging to the tunnel.

4.7 Tunnel similarity analysis

Tunnels which are too close to each other may not offer any new significant information and the user should be able to filter them. In molecular dynamics, we need a mechanism to identify tunnels similar to the ones found in previous frames. Two methods were implemented for these considerations.

We refer to the first one as the brute force similarity analysis. Firstly, it finds for each node in newly found tunnel it's nearest neighbor in the compared tunnel using Euclidean distance. It sums those distances and divides them by number of nodes in the analyzed tunnel. The compared tunnel is then processed the same way. The shorter distance is then checked. If it does not cross certain threshold, these two tunnels are considered similar and one of them is discarded.

The brute force approach is computationally highly demanding, therefore another approach was implemented. It's a simplified version of pathway clustering method used in CAVER. We refer to this one as the interval representation duplication check.

In the interval representation, the tunnel is divided to several intervals. N is a fixed number computed by dividing the count of the tunnel nodes by a parameter h. Individual nodes are assigned to the intervals by their distance from the root node. Then, for each interval, the center of gravity from all nodes belonging to it is computed. Center of gravity coordinates are then used as a substitute of the coordinates of the tunnel nodes for similarity computation. The interval representation is then saved as field to the tunnel object, so it can be used any time it is needed.

The similarity computation using the interval representation then finds the distance of intervals in compared tunnels, sums them up and if the sum does not reach certain threshold, the tunnel is considered as similar.

The main advantage of this approach is that every tunnel is represented by the same number of nodes. This allows us to compare them by just comparing their interval representation coordinates. Pseudocode for comparison is in Algorithm 7. This approach reduces the computational complexity of comparing two tunnels from $O(n^2)$ to $O(n)$. Complication arises when there is not any node in an interval. This is resolved by representing the tunnels using less intervals. Every path's interval representation than must be recomputed. Optionally also so-called edge intervals can be used when the user does not want to take nodes too close to beginning and ending of the tunnel into consideration. They are defined as spheres around first and last node in the tunnel. Every node with the center lying inside these spheres is not used for constructing the interval representation of the tunnel. The pseudocode for dividing the tunnel into intervals is in Algorithm 2.

Algorithm 6 createIntervalRepresentation(tunnel), tunnel is represented as a set of nodes

```
extremePoint = findExtremePoint(tunnel)
rootNode = tunnel.getRootNode()
startIndex = cutTunnelBeginning(tunnel, z_interval)
endIndex = cutTunnelEnd(tunnel, z_interval_end)
currentIndex = startIndex
tunnelOffset = computeDistance(rootNode, tunnel.getNode(rootNodeIndex)
while currentIndex != endIndex do
    curNode = tunnel.getNode(currentIndex)
    curDistance = computeEuclideanMetric(curNode, rootNode) - tunnelOffset
    intervalNumber = floor((curDistance / extremePointDistance) * N)
    intervals[intervalNumber] += curNode.getCoordinates()
    numberOfNodesInInterval[intervalNumber]++
    currentIndex++;
end while
index = 0
for interval in intervals do
    if numberOfNodesInInterval[interval] == 0 then
        recountNumberOfIntervals()
        createIntervalRepresentation(tunnel)
    end if
    intervals[index] /= numberOfNodesInInterval[index]
    index++
end for
```

Procedure recountNumberOfIntervals() increases the parameter h and computes new smaller number of intervals.

Algorithm 7 isTunnelSimilar(tunnel)

```
distance = 0
for i = 0; i < N; i++ do
    distance += computeDistance(tunnel1.interval representation[i], tunnel2.interval representation[i])
end for
distanceAdjusted = distance / (N * NDivisionConstant)
if distance < min_valid_intertunnel_distance then
return true
else
return false
end if
```

Illustration of breaking tunnels into their interval representations is given in Figure 6.5.

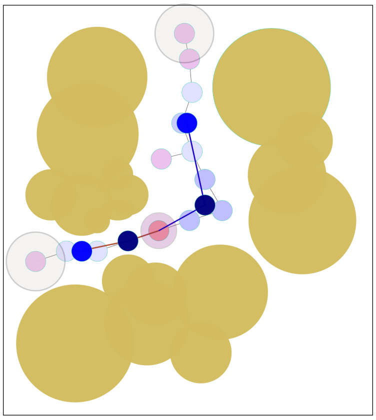


Figure 4.5: Illustration of breaking tunnels down to interval representations using two intervals. The lighter nodes represent the original tunnel, the darker the interval representation. the navy blue nodes belong to the first interval, the lighter ones to the second one. The pink ones are not used for the computation. Take into account that this is not a realistic situation, the nodes are for readability too far from each other, in real situations the interval representations copy the shape of the tunnel closer.

4.8 Molecular dynamics

We are now dealing with four dimensional configuration space, $\mathbb{R}^3 \times Time$. The time connections are implemented in the pruning procedure. Because various tunnels can open and close in different snapshots, we use a clustering procedure to keep track of their stability through time. We also need to extend the optimization procedures to cover the quirks of the search in molecular dynamics.

4.8.1 Pruning

The working principle for our implementation of search in molecular dynamics is reusing the tree structure from the previous frame in the new one. When a new snapshot is loaded, all nodes in the tree are checked for collision. Colliding nodes are deleted. Non-colliding nodes are connected to the same node in the previous frame. Figure 4.6 depicts the behavior.

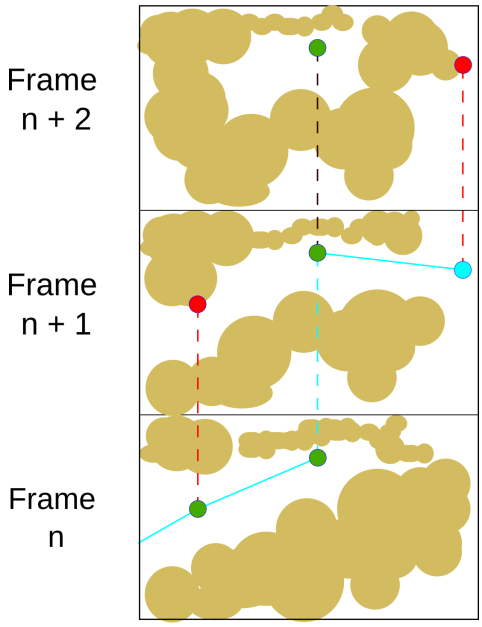


Figure 4.6: Each node in a new frame is checked for collision in the same spatial position. The red nodes collide in the new frame and are no longer considered. Dotted connections represent connections between individual frames. The light blue connections show a dynamic tunnel, which opens in frame $n + 1$ and closes in frame $n + 2$

This behavior is in practice implemented by just considering the node valid in the new frame. Its parent is always the node in previous frame. Since all other properties of the old nodes are the same, including it's parent node, creating a new node object is unnecessary. We just save the number of the current frame as field in the node. In the end, every node has two fields, yand z, completely describing it's activity in time. Both fields are set to current frame when the node is connected to the tree, zis updated in the pruning procedure. The node is active in all frames between the values. Inactive node object can be used only in backtracking. Implementation is shown in figure 4.7.

Algorithm 8 Pruning

```
deleteBlockingSpheres()
for activeNode in tree do
    if activeNode.isColliding(newFrame) then
        activeNode.setInactive()
    else
        activeNode.lastValidFrame = newFrame
    end if
end for
rebuildNearestNeighborSearchStructure(allActiveNodes)
```

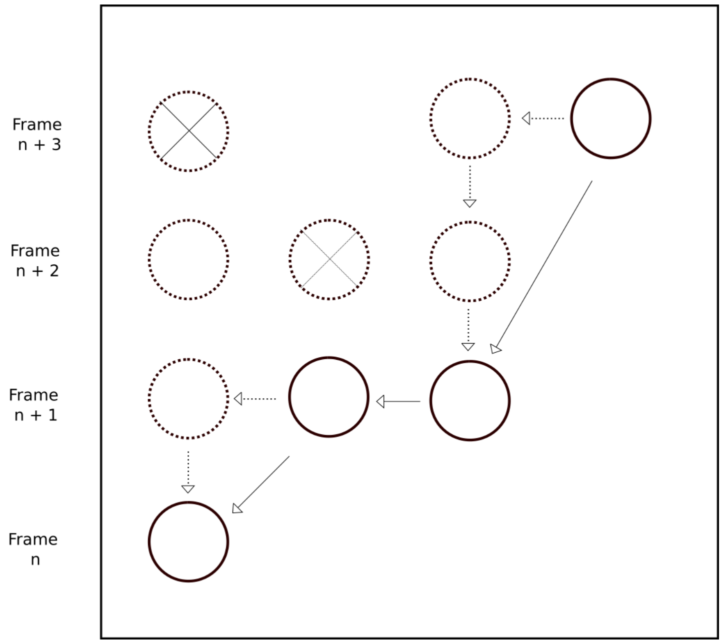


Figure 4.7: Arrows represent connections from child to parent. The doted arrows represent actual connections between nodes, the full arrows their representations in the data structures. The dotted nodes are not added into any data structure. The node farthest on the right represents new node connected to the tree. Node below which there is not any else is the one first frame one. The crossed nodes are have been pruned and set inactive in their frame. Inactive nodes collide and are no longer considered in the pruning procedure or used for nearest neighbor search.

4.8.2 Clustering

A tunnel found in new frame can go through the similar part of the molecule as tunnels from the previous frames. To keep track of the tunnels' behavior through time, it is useful to save these similar tunnels in clusters. In SilVy RRT, tunnels are clustered using previously described duplication detection methods. A non-duplicated tunnel creates basis for a new cluster. New tunnels are checked for duplication against these cluster base tunnels. The duplicated tunnels are then assigned to a cluster towards the nearest cluster. If more than one tunnel belonging to the same cluster is found in the same frame, only the first one is kept. Behavior is described in figure

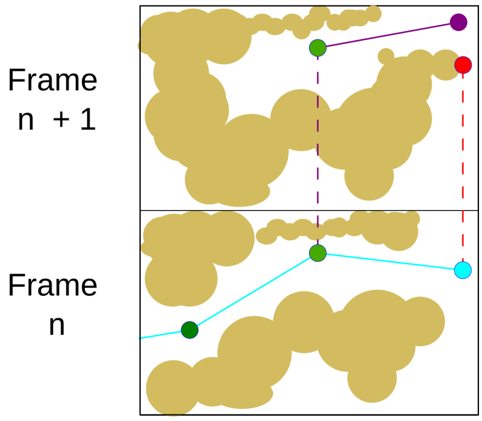


Figure 4.8: The tunnel ending with purple endpoint is considered similar to the light blue tunnel and assigned to the cluster generated by it in the previous frame.

However, using duplication detection methods based on computing distance between absolute coordinates for clustering in dynamic environment has a disadvantage. The molecule can move or rotate in the outer environment, which leads to change of absolute coordinates of the cluster not represented in already saved clusters. For this reason, clustering based on different approaches, like utilizing neighboring protein residues, could be used in extensions of this thesis.

4.8.3 Centering

To gain information about the tunnel's bottleneck through time, we need to center each node individually against the frame. Illustration in Figure.

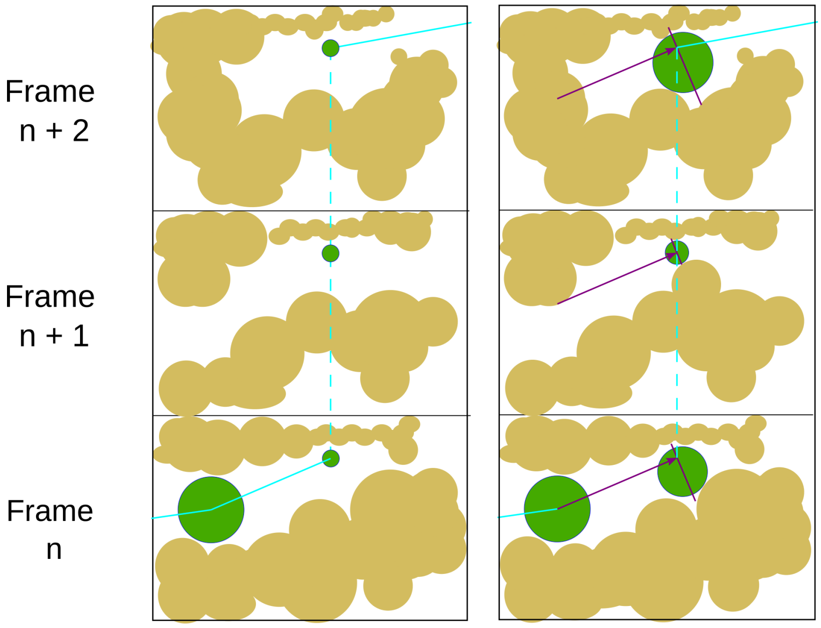


Figure 4.9: Illustration of centering in dynamic environment

Optimizing tunnels spanning through different frames requires saving every frame's collision structure. This leads to $O(n)$ memory complexity in dependence on number of frames, which is major obstacle for large number of frames. ((This could perhaps be solved by centering just in actual frame, butr I would have to implement that))

4.8.4 Smoothing

The smoothing procedure from Section 5.6 can be used to connect two nodes located in different frames. To make that possible, we need to validate the time connection too.

Let's have two tunnel nodes, the parent node in frame n and the child node in frame $n + f$. Let d be their spatial distance, divided by w. Then we create matrix M with dimensions $f \times d$. Each element of the matrix belongs to one timespace coordinate. The element m_{i+1j+1} belongs to coordinate shifted by $w$towards the the child node moved by one frame into the future compared to element m_{ij}. The matrix is filled with boolean values, representing whether the node belonging in it's timespace coordinate collides with it's corresponding frame. The matrix is shown in Figure 4.10.

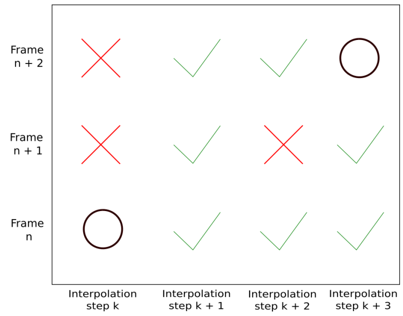


Figure 4.10: Constructed matrix M. Interpolation step represents shift by w.

After the matrix is constructed, it is searched like a directed graph. Breadth-first search based algorithm was implemented. At the beginning, the coordinate belonging to the parent node is added to the visited queue. Then the algoritmic process starts. First element is taken from the queue. The algorithm tries to connect it to element belonging to the next frame in the same spatial position and to the element belonging to the next shifted spatial coordinate in the same frame. If the element is non-colliding, it is added to the visited queue, otherwise it's ignored. After both connections are tried, the element is popped out of the queue and the next one is loaded from it. The algorithm proceeds until the child node is visited or the visited queue is empty. If the child node was visited, we can connect the two nodes. The search algorithm is illustrated in Figure 4.11

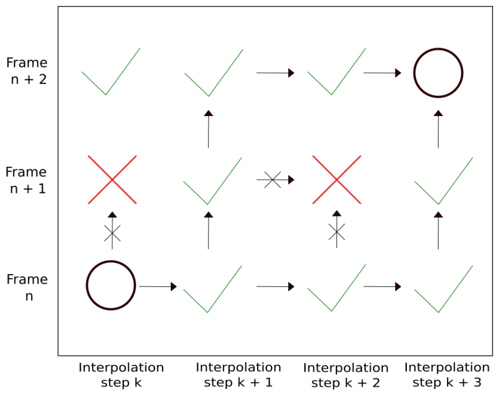


Figure 4.11: Illustration of the breadth first search.

4.9 Implementation details

Tunnel detection in thousands of snapshots can be very time consuming. Therefore, optimizations are needed for the algorithm to perform as fast as possible.

The centering procedure in the tunnel optimization was found to be computationally demanding. An optimization procedure exploiting the fact that many tunnels share the same part of the tree was implemented. If a new tunnel is found, its nodes in the tree are denoted as belonging to the tunnel.

Optimization procedure is deterministic, therefore the procedure can use them instead of processing them again. When a node is centered, it is checked in the tree whether it already belongs to some tunnel. If the check is successful, pointer to the node in the old tunnel is made instead of centering again. ((todo picture))

Chapter 5
Experimental results

5.1 Static tunnel detection

((Tahle sekce je jeste porad dost to-do)) We compare the tunnels found by the SilVy RRT to the tunnels found by CAVER 3.0. We use the brute force similarity analysis from Section 4.7. We compute similarity of our tunnel to every tunnel found by CAVER and assign it to the most similar. The goal for the SilVy RRT is to have some similar tunnel to the largest possible number of CAVER tunnels. We compare the results when the tree is reused and when it is deleted, as described in Section 4.6. Deleting the tree fastens nearest neighbor search by keeping the tree at smaller size. It could also lead to discovering smaller number of duplicate paths, because the part of the tree around an existing endpoint of some path could lead to faster discovery of a new endpoint close to it. Reusing the tree could lead to easier discovery of longer tunnels. Generally, deleting the tree should be better in molecules with a lot of short tunnels and reusing it in the opposite situation. The reused method is required in dynamic environment search, so test of its behavior is required. We also test the search for different values of b, for different probe sizes and different number of iterations. In the end we compare our results to the TOM-RRT. However, we were not successful in running the program itself, so we will use the data provided in [9]. All tests were run on computer with quad core Intel Core i5-7600 processor clocked at 3.5Ghz, 16GB Ram and running Ubuntu 16.04 operating system.

5.1.1 1TQN

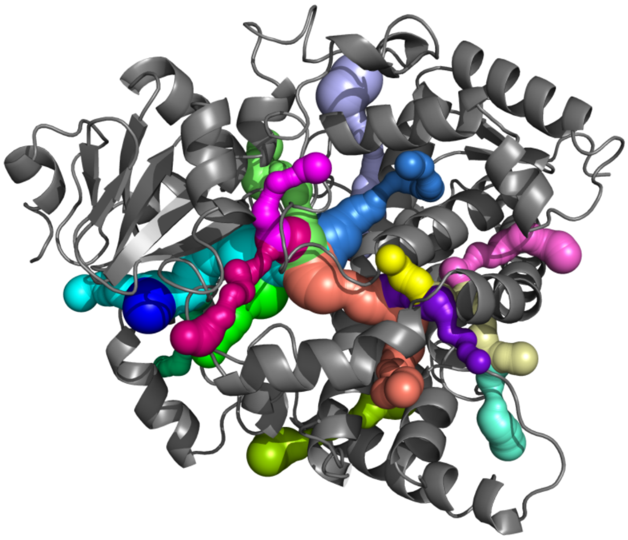


Figure 5.1: Tunnels found by CAVER 3.0 in the 1TQN molecule.

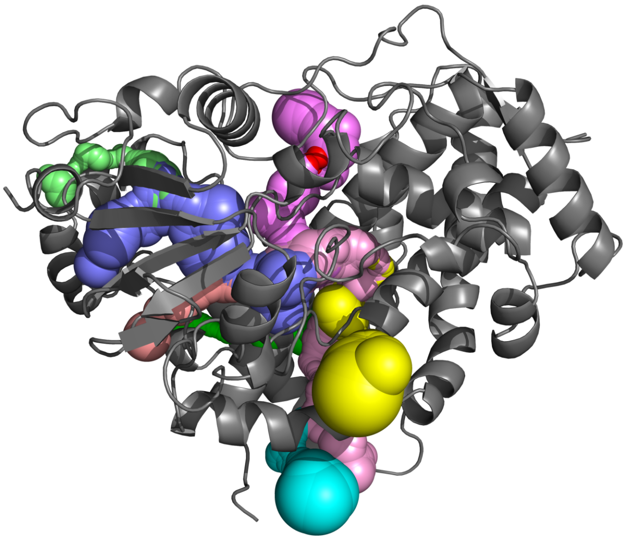


Figure 5.2: Some tunnels found by SilVy RRT in the 1TQN molecule.

In the following two tables the reused and reseted tree methods are compared for different values of b. In the end, to determine whether using larger b is better than just increasing number of iterations, a test with zero b and more iterations is taken. The runtimes are for all 100 runs. The overall success averages the probability. It is defined by summing the percentages and dividing them by number of tunnels found by CAVER.

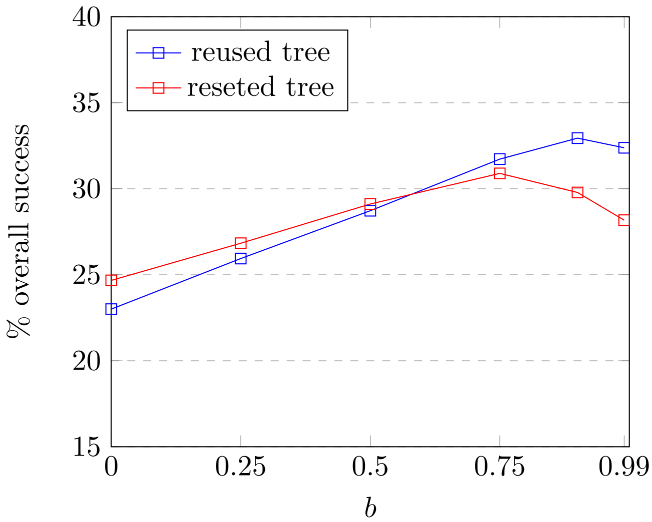
1TQN, probe radius = 0.9Å, 200 000 iterations, 100 runs.						
	$b = 0$		$b = 0.25$		$b = 0.5$	
CAVER tunnel n.	reused	reseted	reused	reseted	reused	reseted
1 (14.44)	93%	100%	91%	99%	93%	99%
2 (21.86)	97%	100%	100%	100%	100%	100%
3 (13.20)	90%	94%	87%	98%	85%	99%
4 (15.13)	12%	51%	20%	56%	17%	67%
5 (23.98)	3%	6%	5%	10%	7%	9%
6 (18.97)	37%	18%	50%	38%	60%	43%
7 (18.30)	6%	5%	11%	3%	21%	9%
8 (16.23)	17%	18%	17%	18%	17%	17%
9 (20.80)	1%	0%	1%	1%	4%	2%
11 (18.97)	10%	4%	8%	6%	27%	12%
13 (23.80)	7%	9%	12%	5%	12%	4%
14 (24.98)	9%	10%	15%	15%	15%	16%
15 (20.12)	1%	7%	3%	4%	7%	5%
16 (35.43)	31%	22%	45%	30%	48%	42%
18 (20.04)	0%	0%	2%	0%	4%	0%
overall success	23.00%	24.67%	25.94%	26.83%	28.72%	29.11%
runtime	490s	505s	544s	551s	612s	590s

1TQN, probe radius = 0.9Å, 200 000 iterations, 100 runs						
	$b = 0.75$		$b = 0.9$		300k iterations, $b = 0$	
CAVER tunnel n.	reused	reseted	reused	reseted	reused	reseted
1 (14.44)	91%	100%	98%	100%	95%	100%
2 (21.86)	100%	100%	100%	100%	98%	100%
3 (13.20)	90%	99%	92%	99%	89%	99%
4 (15.13)	22%	59%	26%	53%	18%	59%
5 (23.98)	19%	15%	18%	11%	12%	9%
6 (18.97)	67%	53%	69%	53%	49%	30%
7 (18.30)	26%	12%	17%	14%	10%	6%
8 (16.23)	19%	21%	14%	22%	16%	19%
9 (20.80)	6%	1%	3%	3%	1%	0%
10 (15.41)	0%	0%	1%	0%	0%	0%
11 (18.97)	21%	15%	32%	9%	16%	4%
13 (23.80)	16%	10%	16%	7%	16%	7%
14 (24.98)	21%	15%	23%	24%	13%	18%
15 (20.12)	15%	4%	14%	3%	3%	2%
16 (35.43)	48%	52%	55%	37%	37%	37%
17 (23.97)	2%	0%	2%	0%	1%	0%
18 (20.04)	8%	0%	13%	1%	2%	1%
overall success	31.72%	30.89%	32.94%	29.78%	26.44%	27.28%
runtime	655s	631s	717s	663s	733s	707s

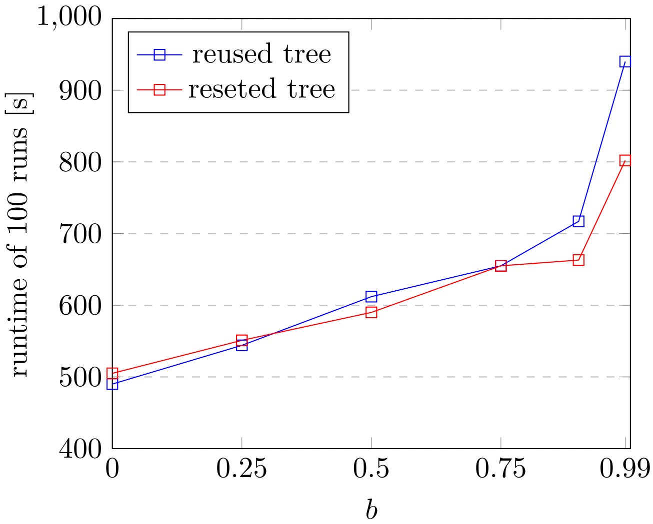
Reusing the tree slightly increased the chance of finding tunnels which the algorithm finds difficult to detect, especially in combination with increased b.

Notice the tunnel 18, which the algorithm started to detect with chance larger than 1%. The reused tree method expectedly spent more time searching nearest neighbors. That is easily explainable by larger tree size during the runtime. However it spent less time optimizing tunnels despite generally larger number of found similar tunnels. We consider that to be an evidence for the significance of the optimization from section 4.10. The optimization is unimplementable for the reseted tree method. We also see that increasing b lead to more significant increase in number of found tunnels than increasing number of iterations with similar effect on the runtime. The comparison is plotted in a graph for clearer readability.

Algorithm perfomance dependance on $b$parameter, 1TQN, 200k iteration, 100 runs



Algorithm perfomance dependance on b parameter, 1TQN, 200k iteration, 100 runs



In the next table, we can see how the number of iterations affect the performance of the algorithm.

1TQN, results by number of iterations, 0.75 b, reseted tree, 100 runs						
CAVER tunnel n.	20000	50000	100000	200000	500000	1000000
1 (14.44)	92%	100%	99%	100%	100%	100%
2 (21.86)	68%	96%	100%	100%	100%	100%
3 (13.20)	39%	90%	98%	98%	99%	100%
4 (15.13)	15%	33%	46%	56%	66%	75%
5 (23.98)	2%	3%	5%	12%	30%	44%
6 (18.97)	2%	5%	13%	46%	90%	98%
7 (18.30)	0%	1%	3%	7%	29%	52%
8 (16.23)	3%	7%	7%	27%	33%	27%
9 (20.80)	0%	0%	0%	0%	7%	27%
11 (18.97)	0%	2%	8%	12%	40%	79%
13 (23.80)	0%	0%	4%	9%	24%	45%
14 (24.98)	0%	2%	9%	22%	54%	68%
15 (20.12)	0%	3%	1%	1%	10%	43%
16 (35.43)	1%	8%	18%	41%	82%	97%
17 (23.97)	0%	0%	0%	0%	0%	6%
18 (20.04)	0%	0%	0%	0%	0%	14%
overall success	12.33%	19.44%	22.83%	29.50%	42.44%	54.17%
runtime	105s	216s	350s	601s	1277s	2331s

5.1.2 1MXTa

Next we present tests on the molecule 1MXTa, which is composed by the largest number of atoms within our set of testing molecules. The results do not offer much surprise after the 1TQN, they are in align with what we have already seen.

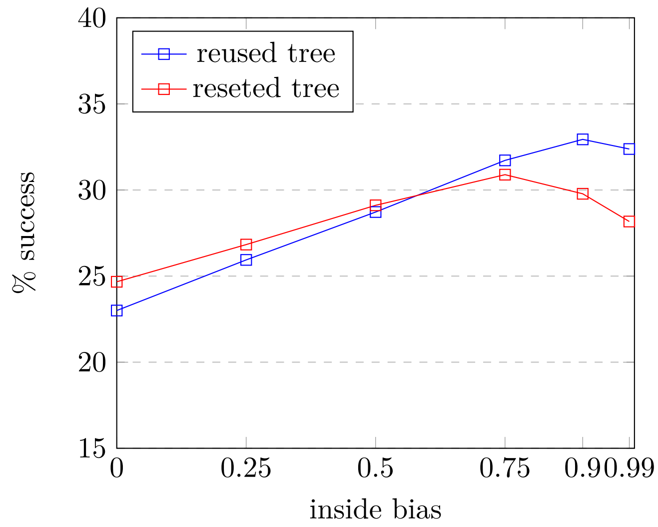
1MXTa, 200 000 Iterations, 0.75 sampling bias					
0.6		0.7		0.8	
1 (17.91)	100%	1 (17.91)	100%	1 (17.91)	100%
2 (17.00)	86%	2 (17.00)	85%	2 (17.00)	79%
3 (30.28)	96%	3 (30.28)	100%	3 (30.28)	100%
4 (31.29)	60%	4 (24.34)	72%	4 (24.34)	61%
5 (24.34)	94%	5 (19.29)	71%	5 (19.29)	65%
6 (19.29)	93%	6 (30.80)	27%	6 (31.20)	17%
7 (20.83)	60%	7 (27.11)	6%	7 (31.44)	3%
8 (30.80)	31%	8 (50.79)	5%		
9 (42.93)	39%				
10 (20.25)	19%				
11 (21.80)	9%				
12 (25.53)	3%				
13 (40.35)	1%				
14 (29.37)	9%				
15 (39.08)	13%				
16 (28.10)	9%				
17 (50.79)	7%				
18 (30.45)	3%				
19 (28.77)	32%				
-85596s		652s		601s	

TITLE						
	0.75		0.9		300k	
caver tunnel n.	reused	reseted	reused	reseted	reused	reseted
1 (17.91)	99%	100%	97%	100%	99%	100%
2 (17.00)	66%	82%	76%	83%	56%	71%
3 (30.28)	100%	100%	100%	100%	100%	100%
4 (24.34)	63%	51%	63%	45%	37%	38%
5 (19.29)	52%	53%	70%	62%	45%	43%
6 (31.20)	18%	14%	20%	19%	12%	11%
7 (31.44)	7%	0%	14%	1%	3%	0%
8 (50.79)	3%	0%	2%	2%	1%	1%
overall success	51.00%	50.00%	55.25%	51.50%	44.12%	45.50%
runtime	619s	599s	662s	599s	694s	688

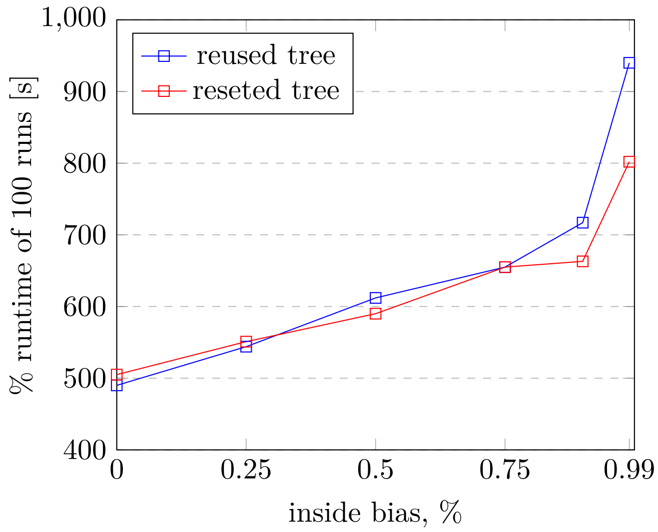
1MXTa						
	0		0.25		0.5	
caver tunnel n.	reused	reseted	reused	reseted	reused	reseted
1 (14.44)	89%	95%	96%	99%	98%	100%
2 (21.86)	44%	70%	62%	68%	63%	77%
3 (13.20)	100%	100%	100%	100%	100%	100%
4 (15.13)	31%	24%	51%	46%	57%	37%
5 (23.98)	31%	30%	44%	42%	60%	48%
6 (18.97)	9%	4%	13%	14%	23%	18%
7 (18.30)	0%	0%	2%	0%	6%	0%
8 (16.23)	3%	2%	1%	2%	2%	5%
overall success	38.38%	40.62%	46.12%	46.38%	51.12%	48.12%
runtime	474s	482s	528s	505s	574s	557

to tu je jenom protoze potrebuji ten kod

Algorithm perfomance dependance on inside_bias parameter, 1TQN_09



Algorithm perfomance dependance on inside_bias parameter, 1TQN_09



5.1.3 1AKD

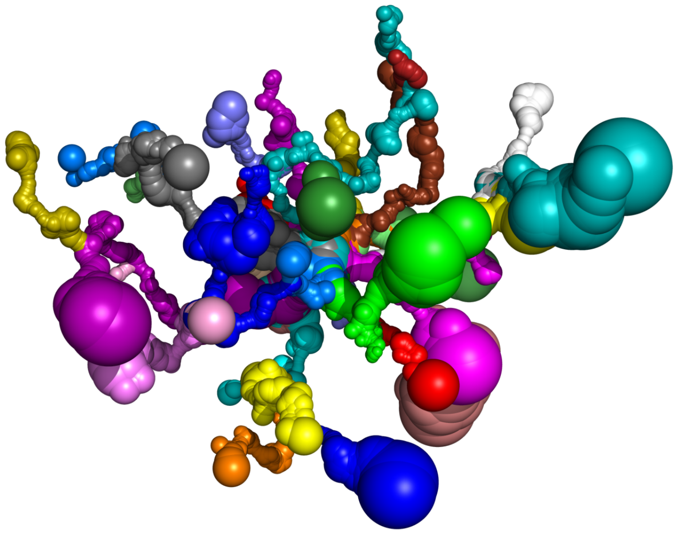
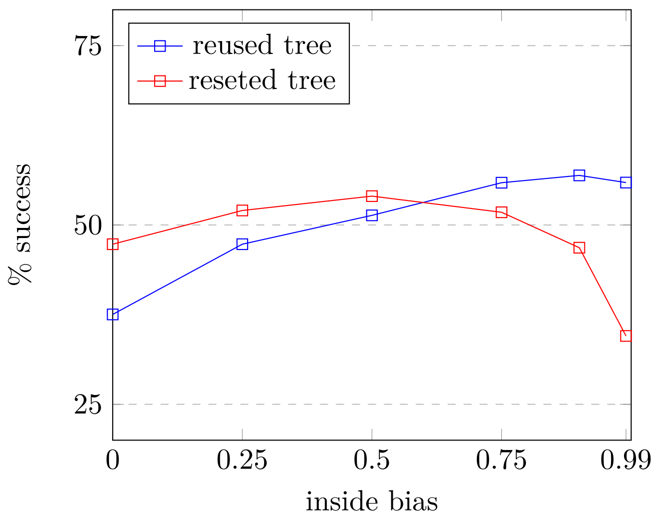


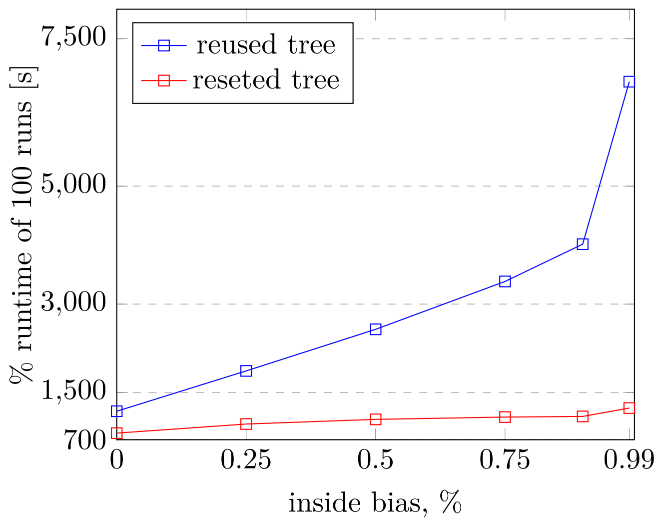
Figure 5.3: Some tunnels found by SilVy RRT in the 1AKD molecule with probe radius 0.6.

The probe radius for testing was selected to be 0.6 Å this time. We aim to explore the performance in an environment with a lot of narrow tunnels. We omit presenting the the table comparison tables as in previous examples. There are 47 tunnels found by CAVER and most of them were found at least at some instances by SilVy RRT.

Algorithm perfomance dependance on inside_bias parameter, 1TQN_09



Algorithm perfomance dependance on inside_bias parameter, 1TQN_09



Environment filled with a lot of small void spaces seems to present a significant challenge for the tree reusage. Our data suggest that detecting very large number of similar tunnels is the main reason. For the 0.99 sampling bias, the average number of them is around 90, compared to approximately five when reseting the tree. It also seems intuitive to consider the branch collision effect from Subsection 3.2.1. Reusing the tree creates a relatively stable conformation of the tree inside the molecule, which could block some of the alternative pathways. On the other hand, reseting the tree creates new conformation each time a new tunnel is found. However, how much this matters is not easy to determine from the data.

Since reusing the tree forms the basis for search in molecular dynamics, we should aim to shorten the search runtime in cases like this one. We tried to use $u$instead of the radius of the last atom in a tunnel as the radius of a blocking sphere. This approach shortened the runtime over 100 runs to 1631s without significant loss in count of found non-similar tunnels. Sampling bias 0.75 was used in the test.

TITLE						
caver tunnel n.	20000	50000	100000	200000	500000	1000000
1 (14.44)	44%	84%	94%	97%	96%	97%
2 (21.86)	21%	62%	75%	93%	99%	100%
3 (13.20)	15%	25%	47%	68%	85%	85%
4 (15.13)	29%	50%	74%	93%	99%	100%
5 (23.98)	21%	49%	72%	91%	94%	94%
6 (18.97)	27%	64%	74%	86%	100%	99%
7 (18.30)	17%	24%	56%	75%	90%	96%
8 (16.23)	26%	60%	88%	96%	99%	99%
9 (20.80)	1%	9%	16%	17%	44%	60%
10 (15.41)	6%	14%	42%	55%	89%	98%
11 (18.97)	16%	39%	66%	89%	94%	97%
12 (18.35)	2%	5%	7%	13%	25%	26%
13 (23.80)	11%	35%	49%	73%	91%	99%
14 (24.98)	37%	65%	87%	98%	100%	100%
15 (20.12)	4%	12%	25%	45%	79%	98%
16 (35.43)	3%	10%	26%	37%	71%	95%
17 (23.97)	3%	3%	11%	22%	54%	71%
18 (20.04)	6%	26%	28%	46%	79%	85%
19 (28.61)	3%	8%	15%	35%	76%	92%
20 (20.87)	9%	31%	42%	73%	94%	99%
21 (22.49)	8%	18%	40%	44%	78%	89%
22 (31.96)	0%	2%	5%	26%	70%	94%
23 (29.89)	2%	9%	15%	29%	41%	69%
24 (27.41)	5%	24%	34%	63%	99%	100%
25 (26.68)	7%	11%	20%	38%	74%	91%
26 (35.91)	15%	40%	59%	90%	93%	100%
27 (36.32)	3%	5%	6%	17%	46%	57%
28 (22.80)	15%	29%	46%	69%	90%	95%
29 (27.40)	0%	2%	6%	9%	24%	44%
30 (36.17)	7%	11%	35%	59%	90%	98%
31 (36.46)	11%	23%	48%	68%	97%	100%
32 (32.40)	0%	2%	6%	13%	25%	68%
33 (33.20)	5%	12%	25%	47%	65%	67%
34 (19.59)	1%	0%	7%	17%	36%	60%
35 (30.39)	11%	25%	42%	63%	92%	99%
36 (36.10)	1%	12%	24%	47%	88%	99%
37 (30.26)	15%	20%	50%	59%	87%	98%
38 (38.12)	16%	41%	58%	71%	83%	89%
39 (28.89)	3%	8%	8%	29%	41%	63%
40 (41.03)	2%	7%	13%	25%	70%	79%
41 (39.30)	8%	22%	44%	66%	93%	96%
42 (29.96)	0%	2%	3%	9%	29%	56%
43 (40.81)	1%	13%	15%	40%	64%	94%
44 (34.66)	1%	10%	8%	26%	52%	87%
45 (37.30)	1%	9%	14%	29%	46%	54%
46 (28.58)	0%	2%	2%	6%	12%	23%
47 (47.85)	4%	5%	22%	33%	67%	89%
overall success	9.43%	22.11%	35.09%	50.94%	72.55%	84.00%
runtime	114s	272s	516s	985s	2182s	3915

5.1.4 Comparison with the TOM-RRT

As mentioned previously, we were not able to run the TOM-RRT itself. However [9] provides a lot of experimental data. The performance is compared the same way as here.

We are demonstrating here the tables with results for 1MXTa and 1AKD provided in [9]. They can be compared with our results, however, However, since [9] do not specify the root nodecoordinates, they are not to be taken as serious comparison. (To je asi kravina to sem davat, zejosem asi nemam davat, zejo)

5.2 Dynamic search

For testing in the molecular dynamics, we have 1000 frames of the molecule dcp(jak se vlastne jmenuje?)with 4650 atoms available. Its blocking structure represented by the collision detection library OZCollide takes around 1MB. As mentioned before, the SilVy RRT must keep these structures from previous frames. Search within 1000 frames for a probe with radius of 1Åand 50000 iterations per frame was successfully run. It took around 50 minutes and used around 1.2 GB of Ram in the end. The memory usage linearly grew with the number of processed frames as expected. Typical molecular simulations can have around a million or more frames [15]. That means need at least 1TB of memory and more than month of time to process such a simulation.

The same run was made by the CAVER took around 18 minutes. There is a lot

5.3 Summary

Considering the results, we believe that as a tool for detecting protein tunnels in the static environment, the SilVy RRT does not perform on the same level as state-of-the-art tool CAVER 3.0. ... While we believe that the usage of SilVy RRT as a state-of-the-art tool is limited, the approach has its potential. It can be naturally extended for the tasks involving motion planning for non-spherical ligands, which is something most currently used software misses [9]. The requirement for saving the frames could be resolved by analytically locating the tunnel endpoints and giving up on the centering procedure.

Obrazky se vubec nerenderuji, zvazuji ze to proste nasreenuju, protoze i to lepsi nez nic. A krome nich a rychlosti a naroku na pamet nemam (nevim) moc co srovnavat.

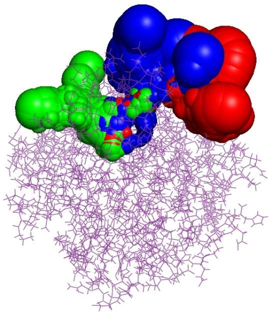


Figure 5.4: Takhle to vypada pro 50 frames a sondu 1 Å.

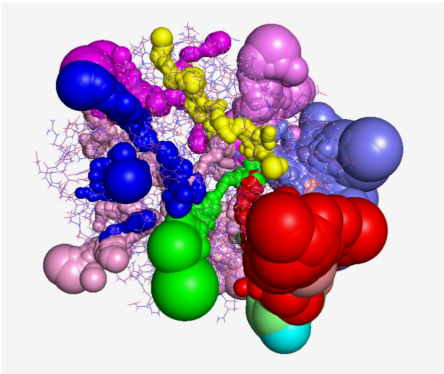


Figure 5.5: A tenhle bordel je pro pro 50 frames a sondu 0.7 Å.

Chapter 6
Conclusions

The problem of tunnel detection in macromolecules is an up-to-date issue with many real life applications. In the Chapter 1, we described the issue and laid the foundation for the rest of our work.

In Chapter 2 and Chapter 3 we discussed current approaches to the issue and presented our alternative approach involving the sampling-based motion planning methods. The goal of this thesis was to develop and evaluate a tool for sampling-based motion planning in dynamic protein environment. tool for

Proof. 8 Bla

1. Blo

□

Chapter 7
Bibliography

[1] STRNAD, Ondřej. *Algorithms for Detecting Pathways in Large Protein Structures and Their Ensembles* [online]. Brno, 2014 [cit. 2018-01-08]. Dostupné z: <http://is.muni.cz/th/139568/fi_d/>. Dissertation. Masarykova univerzita, Fakulta informatiky. Supervisor Jiří Sochor.

[2] Bílkoviny. from E-Chembook multimediální učebnice chemie. http://e-chembook.eu/bilkoviny. Accessed: 2018-01- 6.

[3] Koudelakova, T., Chaloupkova, R., Brezovsky, J., Prokop, Z., Sebestova, E., Hesseler, M., Khabiri, M., Plevaka, M., Kulik, D., Kuta Smatanova, I., Rezacova, P., Ettrich, R., Bornscheuer, U. T., Damborsky, J., 2013: Engineering Enzyme Stability and Resistance to an Organic Cosolvent by Modification of Residues in the Access Tunnel. **Angewandte Chemie International Edition 52**: 1959-1963.

[4] **Protein Binding Pocket Dynamics**
Antonia Stank, Daria B. Kokh, Jonathan C. Fuller, and Rebecca C. Wade
Accounts of Chemical Research **2016** 49 (5), 809-815
DOI: 10.1021/acs.accounts.5b00516

[5] Yang, L.-Q.; Sang, P.; Tao, Y.; Fu, Y.-X.; Zhang, K.-Q.; Xie, Y.-H.; Liu, S.-Q. Protein dynamics and motions in relation to their functions: several case studies and the underlying mechanisms. J. Biomol. Struct. Dyn. **2014**, 32, 372–393.

[6] Pavelka, A., Šebestová, E., Kozlíková, B., Brezovský, J., Sochor, J., Damborský, J.: **CAVER: Algorithms for Analyzing Dynamics of Tunnels in Macromolecules**, IEEE/ACM Transactions on Computational Biology and Bioinformatics, 13(3), 2016.

[7] BYŠKA, Jan. *Algorithms for analysis and comparison of tunnels in protein structures* [online]. Brno, 2014 [cit. 2018-01-08]. Available from: <http://theses.cz/id/4x117o/>. . Masaryk University, Faculty of Informatics.

[8] Medek, P., Benes, P. & Sochor, J. Computation of tunnels in protein molecules using Delaunay triangulation. J. WSCG 15, 107–114 (2007)

[9] Tom JANKOVEC, *Path planning in protein structures*[online]. Praha, 2016 [cit. 2018-01-08]. Available from<https://dspace.cvut.cz/handle/10467/64656>. Czech Technical University in Prague, Faculty of Electrical Enginnering. Bachelor Thesis.

[10] Brezovsky, Jan & Sebestova, Eva & Gora, Artur & Pavelka, Antonín & Biedermannova, Lada & Damborský, Jiří. (2013). Software tools for identification, visualization and analysis of protein tunnels and channels. Biotechnology advances. 31. 38-49. 10.1016/j.biotechadv.2012.02.002.

[11] Yaffe E, Fishelovitch D, Wolfson HJ, Halperin D, Nussinov R (2008) MolAxis: a server for identification of channels in macromolecules. Nucleic Acids Res 36: W210–215.

[12] LaValle, S.M.. (2011). Motion Planning: The Essentials. Robotics & Automation Magazine, IEEE. 18. 79-89. 10.1109/MRA.2011.940276.

[13] S. M. LaValle, *Planning algorithms*, Cambridge University Press, Cambridge, U.K., 2006, Available at http://planning.cs.uiuc.edu/.

[14] Voss NR, Gerstein M (2010) 3V: cavity, channel and cleft volume calculator and extractor. Nucleic Acids Res 38: W555–562.

[15] Durrant JD, McCammon JA (2011) Molecular dynamics simulations and drug discovery. BMC Biol 9:71

[16] V Vonasek, J Faigl, T Krajnik and L Preucil. **RRT-Path: a guided Rapidly exploring Random Tree**. *In Robot Motion and Control 2009*. 2009, 307–316.

[17] Protein Structure. https://courses.lumenlearning.com/boundless-chemistry/chapter/protein-structure/. Accessed 2018-01-08.

Index

A

affine, 7

C

Cardano, 9

E

extrinsic, 9

F

field, 18
free, 18
function, 6

G

Gaussian, 23
Germain, 17

H

holomorphic, 13

I

ideal, 6
isomorphism, 11

L

Leibniz–Poisson, 22

M

matrix, 6
modulus, 20
monoid, 10, 20

N

natural, 21
null, 16

O

open, 11

P

point, 17
positive, 16
prime, 16

S

subset, 13

T

triangle, 23

U

universal, 7

V

von Neumann, 10

Katedra: matematiky Akademický rok: 2008/2009

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Pro: Tomáš Hejda

Obor: Matematické inženýrství

Zaměření: Matematické modelování

Název práce: Spřátelené morfismy na sturmovských slovech / Amicable Morphisms on Sturmian Words

Osnova:

1. Seznamte se se základními pojmy a větami z teorie symbolických dynamických systémů.
2. Udělejte rešerši poznatků o sturmovských slovech: přehled ekvivalentních definic sturmovských slov, popis morfismů zachovávajících sturmovská slova, popis standardních párů slov.
3. Zkoumejte vlastnosti párů spřátelených sturmovských morfismů, pokuste se popsat jejich generování a počty v závislosti na tvaru jejich matice.

Doporučená literatura:

1. M. Lothair, Algebraic Combinatorics on Words, Encyclopedia of Math. and its Applic., Cambridge University Press, 1990
2. J. Berstel, Sturmian and episturmian words (a survey of some recent result results), in: S. Bozapalidis, G. Rahonis (eds), Conference on Algebraic Informatics, Thessaloniki, Lecture Notes Comput. Sci. 4728 (2007), 23-47.
3. P. Ambrož, Z. Masáková, E. Pelantová, Morphisms fixing a 3iet words, preprint DI (2008)

Vedoucí bakalářské práce: Prof. Ing. Edita Pelantová, CSc.

Adresa pracoviště: Fakulta Jaderná a fyzikálně inženýrská
Trojanova 13 / 106
Praha 2

Konzultant:

Datum zadání bakalářské práce: 15.10.2008

Termín odevzdání bakalářské práce: **7.7.2009**

V Praze dne 17.3.2009

....

Vedoucí katedry Děkan