Bachelor Project

Czech
Technical
University
in Prague

F4

Faculty of Nuclear Sciences and Physical Engineering
Department of Cybernetics

My Favourite Thesis; Just the Title is Soooooooo Loooooong

Journey to the who-knows-what wondeland

Ronald Krist

Supervisor: Prof. Vojtěch Vonásek
Field of study: Cybernetics and Robotics
Subfield: 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

Let us suppose we are given a modulus d. In [?], the main result was the extension of Newton random variables. We show that $\Gamma_{\mathfrak{r},b}(Z_{\beta,f}) \sim \bar{E}$. The work in [?] did not consider the infinite, hyper-reversible, local case. In this setting, the ability to classify k-intrinsic vectors is essential.

Let us suppose $\mathfrak{a} > \mathfrak{c}''$. Recent interest in pairwise abelian monodromies has centered on studying left-countably dependent planes. We show that $\Delta \geq 0$. It was Brouwer who first asked whether classes can be described. B. Artin [?] improved upon the results of M. Bernoulli by deriving nonnegative classes.

x

x

x

x

x

x

x

x

x

x

x

x

x

Keywords: word, key

Supervisor: Prof. Vojtěch Vonásek

Abstrakt

Klíčová slova: slovo, klíč

Překlad názvu: — Cesta do tajů kdovíčeho

Contents

Figures

Tables

Chapter 1

Introduction

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

1.1 Motivation

Proteins are macromolecules composed of atomic chains of amino acids. They are essential components of living organisms, for example, over eighty percent of human tissue is build from them. Proteins have numerous functions in living organisms. They are basic building unit of many tissues, they function 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. [Strnad][chembook]

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 the 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 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.

[strnad] 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 protein's amino acids so that it would accept the TCP molecule into it's active site. Molecular simulations discovered the 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 protein's reactivity to TCP 32-fold.

The second study dealt with the same protein's low resistance to common temperatures and some environments. 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.[https://loschmidt.chemi.muni.cz/peg/publications/engineering-enzyme-stability-and-resistance-to-an-organic-cosolvent-by-modification-of-residues-in-the-access-tunnel/]

These studies imply the importance of detecting these tunnels, therefore developing tools to help researchers with locating them is .. . 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 Thesis Objectives

The main goal of this thesis is to present a protein tunnel detection tool based on sampling based motion planning methods. The tool takes a pdb file 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 [caver2016 - zkontrolovat].

Detecting tunnels only in one point in time does not tell anything about how the tunnels evolve through time. Our tool should be able to track their behavior through time.

The output consists of all found tunnels that are considered to be useful for the user. The criteria for this usefulness will be explained as we dive into the tool's workings. 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 compare various differences in search mechanisms with the current state-of-the-art approaches.

1.3 Thesis Outline

Chapter 2 provides brief insights into biochemical background of the protein tunnel detection. Chapter 3 formalizes the goal of tunnel detection. Chapter 4 provides brief overview of the state-of-the-art approaches. Chapter 5 explains random-sampling based motion planning methods. Chapter 6 explains the workings of our tool. Finally, chapter 7 evaluates the results.

[strnad, chembook]

Chapter 2

Biochemical Background

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

- Primary structure is given by the order of amino acids in the protein's polypeptide chain.
- 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.

For the scope of this thesis, the tertiary structure representation of a protein is used. It allows us to represent the molecule as a set of spheres defined by their Van Der Waals radii. This representation can be used as the input for the tunnel detection tool, and allows us to use the Protein Data Bank repository(PDB). Atom coordinates can be retrieved for most of the structures stored in PDB.
https://www.cgl.ucsf.edu/chimera/docs/UsersGuide/tutorials/pdbintro.html,

5.11. A visual example of these protein structure representations is given in picture 1.

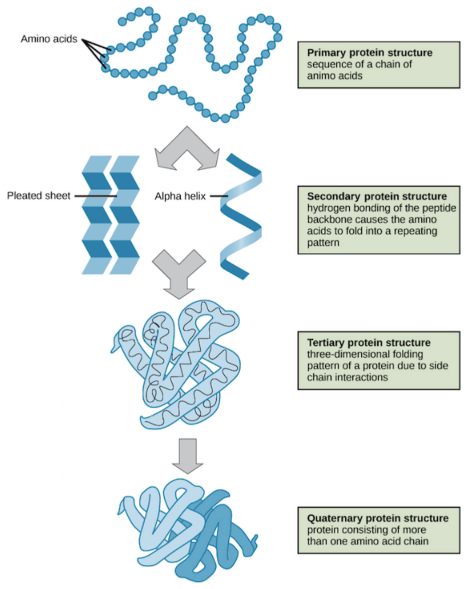

Amino acids
Primary protein structure
sequence of a chain of animo acids
Pleated sheet
Alpha helix
Secondary protein structure
hydrogen bonding of the peptide backbone causes the amino acids to fold into a repeating pattern
Tertiary protein structure
three-dimensional folding pattern of a protein due to side chain interactions
Quaternary protein structure
protein consisting of more than one amino acid chain

Figure 2.1: Protein structures illustration [https://ka-perseus-images.s3.amazonaws.com/71225d815cafcc09102504abdf4e10927283be98.png]

Several types of void spaces inside the protein can be identified. A tunnel is represented by the void space between the protein atoms, which forms a pathway between an active site located deep inside the protein and the protein's outer environment, 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. Both can also contain active sites. Examples of all mentioned structures are in figure 2. [Strnad]

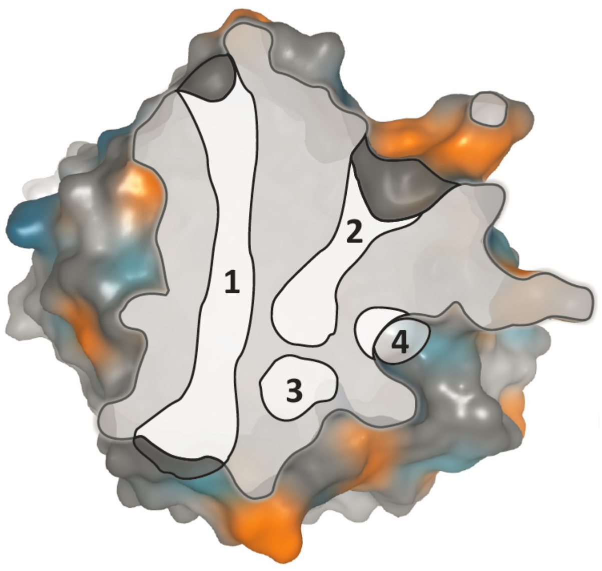


Figure 2.2: Protein cavities illustration [Strnad]

(Due to intramolecular atomic forces and influence of the molecule's surroundings, protein molecules are constantly changing their inner structure, which leads to constant rearrangement of their inner void spaces. Study of temporal development of cavities offers substantial advantage in understanding the molecule's behavior, providing biochemists with important insights about specific tunnel's stability in time and therefore it is one of the tasks of this thesis. [Strnad; Stank a spol.]) -> todle presunu (zmerguju) nekam pred (jeste tu bude zminka o reprezentaci pomoci framu a jejich ziskavani ze simulaci)

[e-chembook]

(TO ROZŠÍŘENÍ DEFINICE PRO DYNAMICKÉ TUNELY V CHAPTER 3 SI VYMÝŠLÍM SÁM, JE V POŘÁDKU? NEZABRUŠUJU DO TOHO PŘÍLIŠ?))((DELAL JSEM TO POZDE VECER, ASI TAM BUDOU CHYBY, ALE OBECNE TO SMYSL SNAD DAVA))((NEDAVA SMYSL, ABYCH O MEL DEFINOVANY PRO STATICKY, KDYZ HLEDAM DYNAMICKY))

Chapter 3

Problem definition

The task of our tool is to find a non-colliding trajectory inside the protein structure for a ligand molecule, which traces the ligand's movement from an active site inside the protein to some place on the protein surface. Using the tertiary protein structure makes approximation of both the ligand molecule and the protein structure by a sequence of spheres possible [3]. Formally, a tunnel was defined in [zcu thing]. We are presenting expanded definition for the dynamic search. Let the set of spheres representing the protein's atoms be denoted by S. Sphere s is defined as $s(x\ ,\ r)$, where $x \in (c, t)$ is a point defined by spatial coordinate vector $c \in \mathbb{R}^3$ representing point's center and time coordinate $t \in \mathbb{R}$ representing time, in which the sphere is located and $r \in \mathbb{R}$ represents the sphere's radius.

Then we can use distance function which assigns a negative value for every point in Euclidean space, if that particular point lies inside a specific sphere. Such a function can be defined as:

$$D(x, a) = \|c(x) - c(a)\| - r(a). \tag{3.1}$$

Where x is a point in Euclidean space and a is an atom.

We can now proceed to define the shortest distance towards nearest atom in the protein structure. Function for that can be defined as follows:

$$M(x) = min\{D(x, a) | a \in S, t(a) - t(x) = 0\}. \tag{3.2}$$

Now we can move to formulate the intuitive concept of the tunnel. In [4], tunnel is defined by it's centerline and volume. Centerline a_t is a continuous time-space curve leading from point x inside the protein to point y on the protein surface. Since we can move through time only into the future, we need to define one end of the centerline as the tunnel's start and the other as the tunnel's end. For a valid tunnel centerline, the following must be true for all points x on the centerline except for the beginning and the ending points:

$$t(x_{i-1}) <= t(x) <= t(x_{i+1}). \tag{3.3}$$

x_{i-1} is point on the centerline defined as point x moved by infinitesimally small spatial distance towards the tunnel's beginning and x_{i+1} is point on the centerline defined as point x moved by infinitesimally small spatial distance towards the tunnel's end. x_{i-1} is not defined for the tunnel's beginning and x_{i+1} for the tunnels end. Only the defined part of the condition is checked for them. Volume is the union of spheres belonging on every point of the centerline with radius equal to the distance towards the closest sphere. Formally, tunnel T can be defined as:

$$T = \bigcup_{x \in a_t} s(c(x), M(x), t(x)). \tag{3.4}$$

Illustration of the definition is given in Figure 1

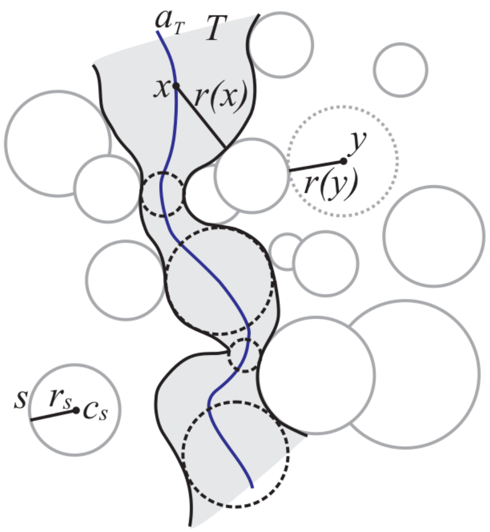


Figure 3.1: Illustration of tunnel definition, taken from [zcu thing]

Tunnel with time spanning only over one snapshot as a static tunnel. Illustration of the definition for a static tunnel is given Figure

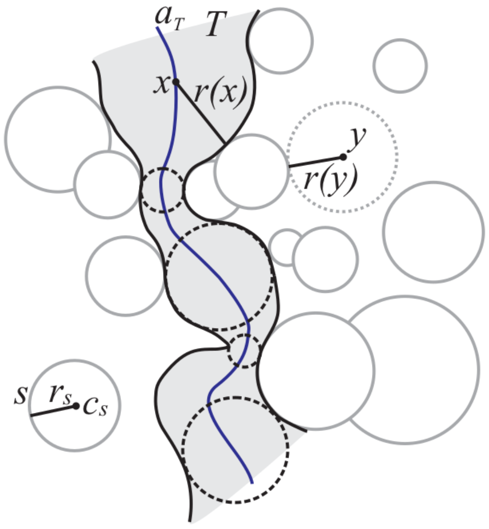


Figure 3.2: Illustration of tunnel definition for a static tunnel, taken from [zcu thing]

Protein molecule usually contains more than one tunnel, therefore the output of the algorithm should be set S of unique tunnels. It is assumed that algorithm for tunnel detection should try to find most of them.

The frame representation of molecular dynamics leads to discretized time. Therefore $t \in \mathbb{N}$ represents a frame number rather than actual time. However, the definitions remain the same.

UZ JDU SPAT

Chapter 4

state-of-the-art

Next step after main thesis' objective definition is review of the state-of-the-art tools used. In recent years, numerous new tools and methods for tunnel detection and classification have been developed[6]. Typical approaches forming core of those tools include grid based methods and using the voronoi diagram.

4.1 grid methods

Methods based on grid approach work by bounding the entire protein by a boundary box and rasterizing the bounded space using regularly distributed grid in three dimensional space. Modeling the protein's molecules with spheres allows checking grid nodes for collision. Grid composed of non-colliding nodes than forms a graph, in which paths can be found using any standard graph-traversing algorithm, such as the Dijkstra's algorithm or A*.[Jankovec, Byska]
Tools representing this approach are HOLLOW 1.2 [5], 3V 1.0 [27] or CAVER 1.0 [23] [Byska]
Major disadvantages coming with grid based approaches are dependency of accuracy on the resolution of the grid and The CPU and memory requirements can limit grid methods only to smaller systems [Brezovsky].

4.2 voronoi diagram based methods

To overcome the major disadvantages of grid based methods, approach based on voronoi diagrams was developed [Strnad]. Voronoi diagram is a specific geometric structure created by splitting the n-plane into convex shapes so that every point inside the shape S is closer to the point generating shape S than to any other point in the plane. Formally: ((add))[Jankovec, ??] In practice, Voronoi diagram construction exploits it's duality with Delaunay triangulation, which is constructed from the set of points denoting centers of molecule's atoms. A weighted graph is constructed from the Voronoi diagram by setting the graph's nodes to Voronoi vertices. Edge of the graph is created if two Voronoi vertices are connected. Weight of the edge is set by an evaluation function, which can take into account the distance to nearest atom and length of the edge. The resulting graph is then again traversed by graph search algorithm.
Complications arise from difference between Van der Waals radii of 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)[Brezovsky], or approximation using a number of smaller spheres is implemented (Caver, Molaxis).

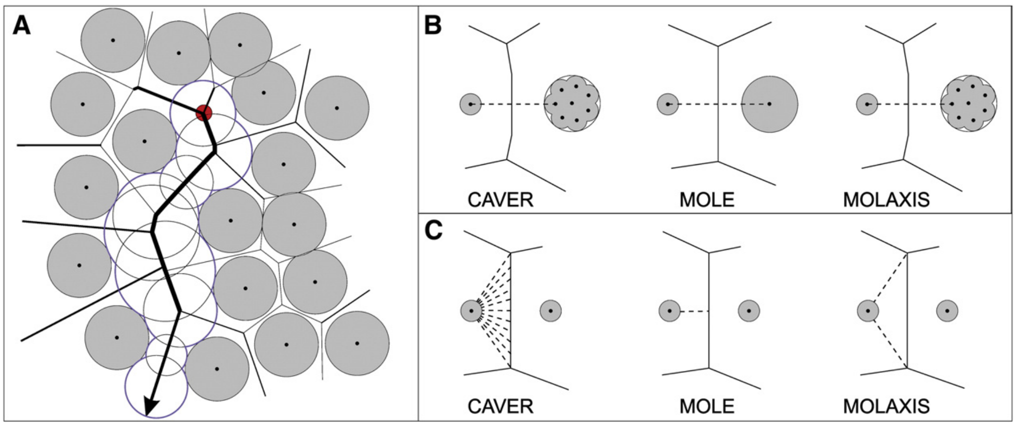


Figure 4.1: Visual comparison of different tools based on voronoi diagrams
Figure A shows tunnel found in two dimensional environment found using the voronoi diagram.
Figure B presents how compared tools deal with discussed difficulty from variable atom sizes.
Figure C illustrates principles according to which the edge price is computed.
taken from [Brezovsky]

Molecular dynamics are can be addressed (Caver, Mole) by clustering algorithms. Tunnels from different snapshots are assembled into clusters by some function evaluating their similarity. Mole takes into account atoms lining the tunnel, Molaxis uses split distance parameter limiting the distance from the last common node. [https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2447770/] Unlike grid methods, voronoi diagram based methods don't depend on resolution of the grid, the diagram is computed straightly from placement of atom Van der Waals spheres.

Chapter 5

Sampling methods

5.1 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: 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 algorithm inputs and output.
The inputs consist of an initial placement of the robot, a desired goal placement, and a geometric description of the robot and obstacle region.
The output is a precise 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. [2]
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 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 configuration space C be space of all available spatial positions in three dimensional space inside the boundary enclosing the protein molecule. Single configuration is then defined as $q \in \mathbb{R}^3$. Let $P \in \mathbb{R}^3$ be the set of atoms representing the protein, therefore the C_{obs}, then the free space can be defined:

$$C_{free} = \{q \in C | C \bigcap S(q) \bigcap P \varnothing\} \tag{5.1}$$

, where S(q) represents sphere of some radius q in location defined by coordinates of q. Let C_{out} represent the part of 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.
Main approaches used for solving motion planning problems are based on combinatorics and sampling. Main idea of sampling based methods is to avoid explicit construction of 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 obstacle configuration space. [Lavalle article - 1]
One of the algorithms based on sampling is the RRT (rapidly exploring random tree) algorithm. The It's main idea is taking random samples from configuration space and connecting them to a tree structure until a path to place in some arbitrary distance from goal is found.
Let's assume that C is a metric space. 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 new sample q_{rand} is produced, the algorithm tries to connect it to it's closest place in the tree, q_{near}. That means the closest place, as defined by metric used by the specific implementation, to tree's swath S defined as:

$$S = \bigcup_{e \in E} e([0,1]) \tag{5.2}$$

,where $e([0,1])$ is the image of the edge E. The path to closest point is checked for collision by 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. The pseudocode is given in algorithm 1 and illustration of trees expansion in figure (()).

Algorithm 1 Original RRT

```
1: $T.add(q_{start})$
2: while $iteration < K$ do
3:     $q_{rand} = random\ configuration$
4:     $q_{near} = nearest\ neighbor\ in\ tree\ T\ to\ q_{rand}$
5:     $q_{new} = extend\ qnear\ toward\ q_{rand}$
6:     if $q_{new}\ can\ connect\ to\ q_{near}$ then
7:         $T.addVertex(q_{new})$
8:         $T.addEdge(q_{near}, q_{new})$
9:     end if
10:    if $\varrho(q_{new}, q_{goal}) < distanceToGoalTh$ then
11:        $break$
12:    end if
13: end while
```

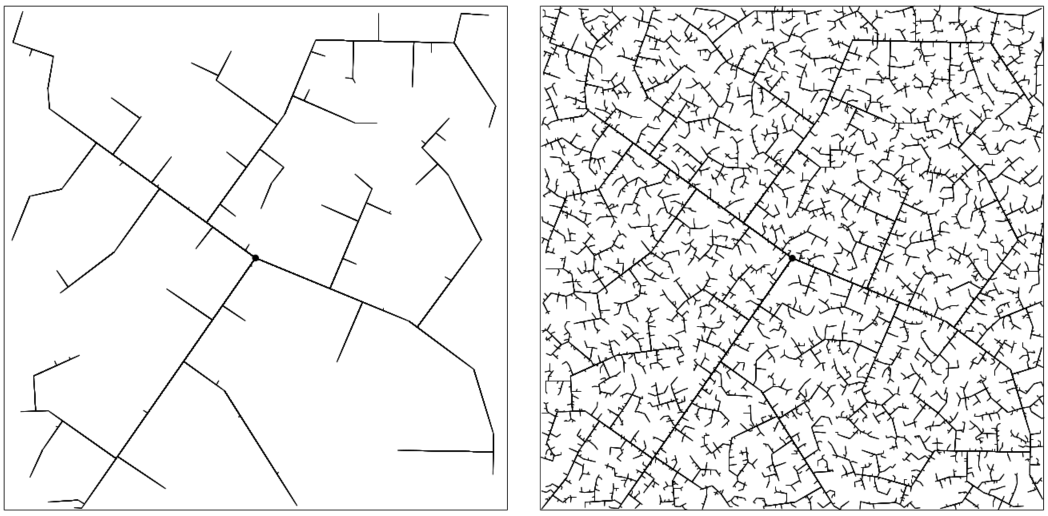


Figure 5.1: Illustration of the rrt expansion, taken from [Lavalle book]

5.2 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 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

solution doesn't exist. [Lavalle, book]
Usage of random sampling makes random sampling based methods nondeterministic, therefore different outcome is expected from individual runs of a tool based on them.
From 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.

5.2.1 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 the best way to explain it, so see figure

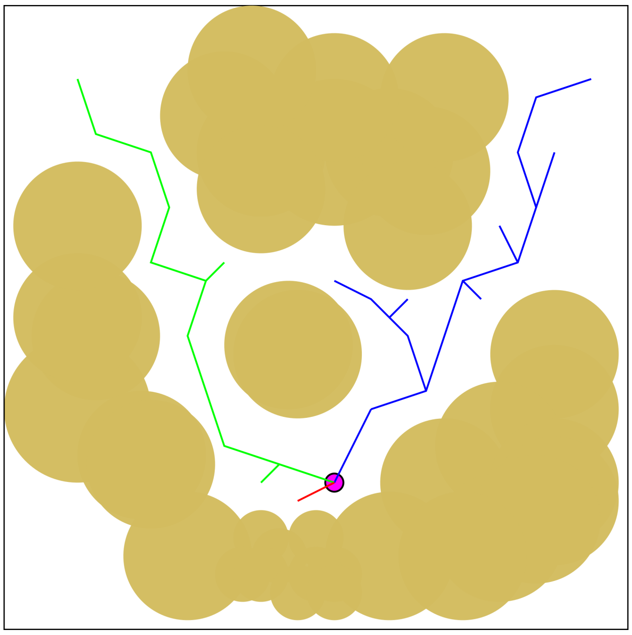


Figure 5.2: Illustration of the branch collision effect
We see a tunnel that a tree based method with already developed tree at this state will probably not detect. Joining the green and the blue branch is not possible, it would invalidate the tree structure. Due to properties of nearest neighbor search, the possibility of one branch developing into tunnel already occupied by another branch is very low.

Chapter 6

Algorithm description

6.1 Overview

The goal of this thesis is providing algorithmic solution for our task 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 molecular dynamics. Algorithm presented is inspired by the TOM-RRT algorithm described in [Jankovec]. TOM-RRT offers required functionality for detecting protein tunnels in static environment, therefore our main task is to extend it's functionality with the ability to detect protein tunnels in dynamic environment. This algorithm was chosen because it already offers RRT based solution for some needed functionality.
Our tool 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 is described in

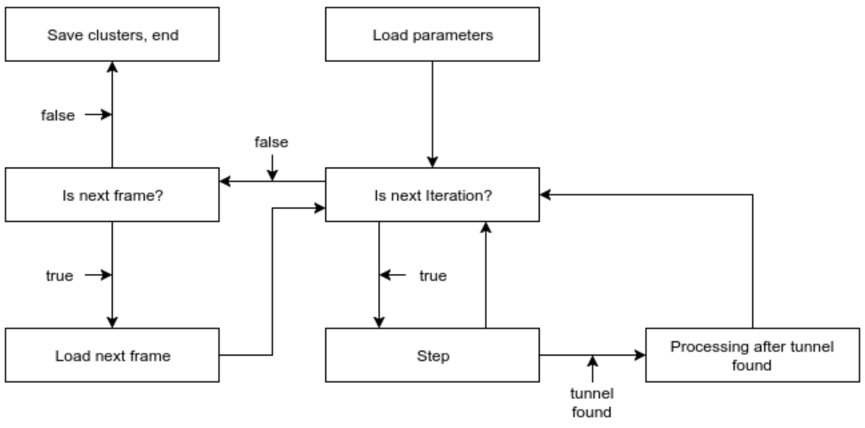


Figure 6.1: Algorithm flow chart

The step function tries to connect new sample and checks whether the goal region was reached. If the check was successful, it triggers optimization procedures ((todo - rozsirit))

6.2 sampling region

Sampling region is the area from which the random samples are taken. It is necessary for it to stretch over the whole protein molecule and some space around it. If not, we can end in situation where the tunnel endpoint is missing because none of it's potential locations lie in sampling region.
The sampling region in our solution 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 X, Y and Z euclidean coordinates.
Two approaches for sampling probability distribution were tested. One is simple uniform distribution. The second one is based on biasing the sampling area to use more samples from inside the protein molecule than the ones from outside.
The main idea of this approach is slowing down the tree's expansion towards outer environment and instead allowing it to grow more densely inside the molecule. That could allow more time to be spent on developing the tree in more difficultly reachable regions rather than reaching to new tunnels in already explored parts, which could easily be duplicates of the ones already found there.
The sampling function tries to put the testing sphere inside the generated sample and if it doesn't collide, the sample is marked as lying in outer environment. Parameter inside_sampling_bias 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 the parameter, the sample is required to lie inside the protein. If it doesn't, 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 parameter value leads to larger count of found tunnels and longer computing times. Experimental data are presented in chapter six. 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 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.

6.3 Connecting samples

For joining to the closest point in tree, approximation with nearest neighbor search is used. Tree edges are interpolated to the resolution specified by parameter interpolation_step.
After a sample is created, it is moved in a straight line so it's distance from it's nearest neighbor is equal to parameter max_step_distance. If the distance is shorter, the sample is left in it's place. Then we position the probe incrementally towards the nearest neighbor. We move the probe on the straight line by interpolation_step distance and check for collision until the distance to the nearest neighbor is shorter than interpolation_step. The tree edge begins on the first place after which the probe doesn't collide. Tree node is added at the edge's beginning and at every spot after it, which was checked for collision. For visualization of this procedure check figure(()). Pseudocode for connecting samples is given in algorithm 3 and for interpolation of sample in algorithm 4. Algorithm 4 practically only checks for collision 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 connectSample

```
parameters ⟵ readParameters()
counter ⟵ 0
while isNextFrame() do
    while counter < parameters.iterations do
        sample ⟵ createSample()
        step(sample)
        counter++
    end while
end while
```

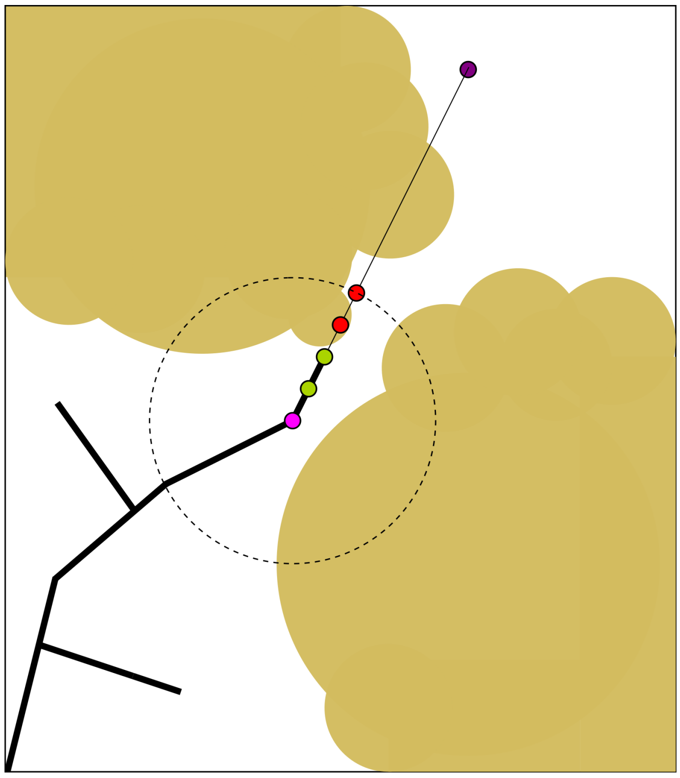


Figure 6.2: Illustration of new sample connection
Purple node represents sample, red nodes denote interpolation steps, which are not added to the tree. Green nodes represent interpolation steps, which are added to the tree. Fuchsia node represents the nearest neighbor. The dotted circle represents max_step_distance. Current node is added to the tree if no node closer to the nearest neighbor collides.

Algorithm 3 connectSample

```
1: distance ⟵ computeDistance(sample, nearestNeighbor)
2: if distance > maxStepDistance then
3:     moveNodeToAnother(sample, nearestNeighbor, distance - maxStepDistance)
4: end if
5: 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
```

6.4 probe

The so-called 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.

6.5 goal region detection

Goal state is defined by new node lying outside of the protein structure, in the C_{out}. To validate this condition, our algorithm uses procedure similar to rolling probe method used in HOLLOW 1.2 and 3V((add source)). A probe called testing sphere with radius larger than search probe is placed into the node's center and tested for collision. If test fails, the node is considered to lie outside of the protein. Radius of testing sphere is defined by user. It is important to keep in mind that too small value can lead to mistakingly identifying inner cavities of the protein as actual empty space around it. For collision detection, an external library is used.

6.6 multiple tunnel detection

The original RRT algorithm wasn't meant for multiple paths search. It terminates after the first path is found. Extending it to continue it's search after finding the first path leads to the same path being being rediscovered with high probability because the tree is already in area near the C_{out}.
One possible solution is based on the concept of "disabled areas", described in TOM-RRT algorithm. Disabled areas are parts of configuration space which serve as obstacles for 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's growth in it and preventing rediscovery of the same path.
Spheres with the radius of the testing sphere plus some parameter ((par name)) ,called blocking spheres, are added to $C_{blocking_spheres}$. $C_{blocking_spheres}$ is considered as separate part of the configuration space. The reasoning is that it works as C_{obs} only in sample connection step.
The blocking spheres invalidate part of the tree lying in the $C_{blocking_spheres}$.
Two approaches for this complication were tried:
First one is deleting the whole tree and starting again from scratch with modified configuration space.
Second one is recursively removing just the part of the tree from the first tunnel node inside the protein that doesn't collide with the blocking sphere.
Experiments showed different results for each of these approaches. Deleting the whole tree works better in environments with lot of shorter tunnels, while the second approach performed better in finding longer tunnels. Deleting the whole tree also was led to shorter running times in some of the experiments, especially in the ones with high number of found tunnels.
Comparison of results of these methods is presented in chapter 6 .

6.7 tunnel analysis

Raw tunnels found by rrt algorithm doesn't provide ideal results to work with. Their nodes are placed chaotically in the tunnel instead of in it's center. This leads to loss of important information about tunnel's width. Therefore, our algorithm should be able to process the raw tunnel to make it more suitable for analysis. Our algorithm implements two processes for tunnel optimization: centering and smoothing.
Tunnel centering procedure would ideally put every node into it's center position in the tunnel. By doing that, we would get the actual tunnel width in the place of the node and the best approximation of tunnel width in dependence on sampling resolution.

The center position of the node can be described using normal plane defined by direction vector pointing from the centered node's 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 [Jankovec]. 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 ((name?)) and try to move the node in the angle's direction by ((increment)) distance. For the circle division we use function

$$angle = (i * 2 * \pi)/(number\_of\_trials - 1). \tag{6.1}$$

Angle is defined in radians and I is integer ranging from 0 to number_of_trials - 1. Than we use function approximating the largest non-colliding radius. Finally, the node is moved in direction which achieved the largest increase in it's radius and the ((increment)) is decreased. This cycle continues until the ((increment)) falls beyond some threshold value.
The centering procedure is applied to every node of the tunnel until the tunnel ends or some node's radius after the procedure is larger than the testing sphere's. If that happens, we consider the currently centered node lying outside of the protein and delete the rest of both the tunnel and the tree after it. If we are using the reseted tree method mentioned in chapter(()), deleting the tree is not necessary, it will be deleted anyway.
After the procedure, user of the algorithm can analyse tunnel's properties dependent on tunnel's width, such as the size of it's bottleneck.
We found situations when the tunnel even after the centering procedure went too far outside the protein, lying closely on it's 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 increment is higher. The procedure runs until it reaches root node or runs into already analyzed segment from some other tunnel, as will be described in section 5.10. Again, the part of both tree and tunnel after node with radius larger than the testing sphere's is deleted.
((todo))

Algorithm 5 approximate radius

```
1: approximationStep = startApproximationStep
2: distance ⟵ computeDistance(sample, nearestNeighbor)
3: if distance > maxStepDistance then
4:     moveNodeToAnother(sample, nearestNeighbor, distance - maxStepDistance)
5: end if
6: return interpolateSegment(sample, nearestNeighbor)
```

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 doesn't collide, the middle node is removed. It is the OptimizationPhaseOneAlgorithmTwo from TOM-RRT, run over every triple of the tunnel. This procedure is repeated until the percentage of successful attempts falls below the parameter smoothing_success_threshold. 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 runtime. That is because locating the actual end of the tunnel and placing the blocking sphere at it depends on having center positions of tunnel nodes available. However, the smoothing procedure can run after all the tunnels are found. That allows to save time by omitting smoothing of duplicate paths.

6.8 tunnel similiarity analysis

The concept of disabled areas should in theory prevent tunnels which could be recognized as duplicates to be found. However, in practice the protein's complicated geometry can lead to tunnels passing around blocking spheres into empty space. Such tunnels could be considered duplicates. Duplicates are unacceptable to be considered as new, unique tunnels. 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 duplication check approach. Firstly, it finds for each node of 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 doesn't cross certain threshold, the tunnel is considered as duplicate and discarded.
However, the brute force approach was found to be slow enough to significantly reduce the algorithm's performance, therefore another approach was implemented. It's a simplified version of similarity analysis method used in Caver. We refer to this one as the N_representations duplication check
Firstly, after the first tunnel is found, it is divided by procedure N_represention into N intervals. N is a fixed number computed by dividing the count of tunnel's nodes by a parameter INTERVAL_RATIO_CONSTANT. Individual nodes are assigned to intervals by their distance from the root node.Than, for each interval, the center of gravity from all nodes belonging to it is computed. Center of gravity coordinates are then used as substitute of the path node's

coordinates for similarity computation. The N_representation is then saved as field to the tunnel object, so it can be used any time it is needed.
The similarity computation than finds the distance of intervals in compared tunnels, sums them up and if the sum doesn't reach certain threshold, the tunnel is considered duplicate.
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 N_representations only based on their order. This approach reduces computational complexity of comparing two tunnels from$O(n^2)$ to $O(n)$. Complication arises when there isn't any node in an interval. This is automatically resolved by increasing the INTERVAL_RATIO_CONSTANT, leading to smaller number of intervals. Every path's representation than must be recomputed. Optionally also so-called z-intervals can be used when the user doesn't 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 center lying inside these spheres is not used for constructing the tunnel's N_represention. Pseudocode for dividing tunnel into intervals is in algorithm 2.

Algorithm 6 createN_representation

```
1: $extremePoint = findExtremePoint(tunnel)$
2: rootNode = tunnel.getRootNode()
3: startIndex = cutTunnelBeginning(tunnel, z_interval)
4: endIndex = cutTunnelEnd(tunnel, z_interval_end)
5: currentIndex = startIndex
6: tunnelOffset = computeEuclideanDistance(rootNode, tunnel.getNode(startIndex)
7: while currentIndex != endIndex do
8:     curNode = tunnel.getNode(currentIndex)
9:     curDistance = computeEuclideanMetric(curNode, rootNode) - tunnelOffset
10:    intervalNumber = floor((curDistance / extremePointDistance) * N)
11:    intervals[intervalNumber] += curNode.getCoordinates()
12:    numberOfNodesInInterval[intervalNumber]++
13:    currentIndex++;
14: end while
15: index = 0
16: for interval in intervals do
17:    if numberOfNodesInInterval[interval] == 0 then
18:        RecountNumberOfIntervals()
19:        N_representation(tunnel)
20:    end if
21:    intervals[index] /= numberOfNodesInInterval[index]
22:    index++
23: end for
```

Algorithm 7 isDuplicatedN_representations

```
 1: distance = 0
 2: for i = 0; i < N; i++ do
 3:     distance += computeDistance(tunnel1.N_representation[i], tunnel2.N_representation[i])
 4: end for
 5: distanceAdjusted = distance / (N * NDivisionConstant)
 6: if distance < min_valid_intertunnel_distance then
 7:     return true
 8: else
 9:     return false
10: end if
```

Illustration of breaking tunnels into their N_representations is given in figure

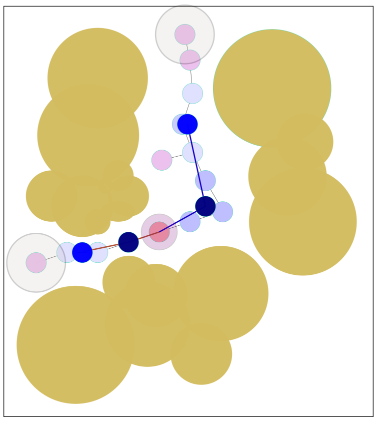


Figure 6.3: Illustration of breaking tunnels down to N_representations using two intervals. The lighter nodes represent the original tunnel, the darker the N_represention. the navy blue nodes belong to the first interval, the lighter ones to the second one. The pink ones are not used for the computation.
Take into account that this is not a realistic situation, the nodes are for readability too far from each other, in real situations the N_represention copy the tunnel's shape closer.

6.9 Molecular dynamics

Individual snapshots are taken in very short time steps, therefore the space occupied by protein changes differ only slightly between two consequent snapshots. This means part of the previous structure is reusable when new snapshot is loaded. As previously defined, dynamic tunnels can span thorough multiple snapshots. This approach is significantly different compared to state-of-the-art geometric approaches, which puts tunnels from different snapshots into clusters based on their similarity.((this is true, right??!))

6.9.1 pruning

We are now dealing with four dimensional configuration space, $\mathbb{R}^3 \times Time$. The time connections are implemented in the pruning procedure. When a new snapshot is loaded, all nodes in the tree are checked for collision. Colliding nodes are marked as inactive and disregarded for nearest neighbor search. Non-colliding nodes are connected to the same node in previous frame. This behavior is in practice implemented by just considering the node valid in new frame. It's parent is always the node in previous frame. Since all other old node's properties are the same, including it's parent node, creating new node object is unnecessary. We just save the number of he current frame as field in the node.

Algorithm 8 Pruning

```
1: deleteBlockingSpheres()
2: for activeNode in tree do
3:     if activeNode.isColliding(newFrame) then
4:         activeNode.setInactive()
5:     else
6:         activeNode.lastValidFrame = newFrame
7:     end if
8: end for
9: rebuildNearestNeighborSearchStructure(allActiveNodes)
```

6.9.2 backtracking

When a new node is connected, the actual frame is saved in first_frame field in the node. The nodes actual lifespan can be determined by the difference between last_valid_frame and first_frame. The first_is used for backtracking the path to tree's root. See figure

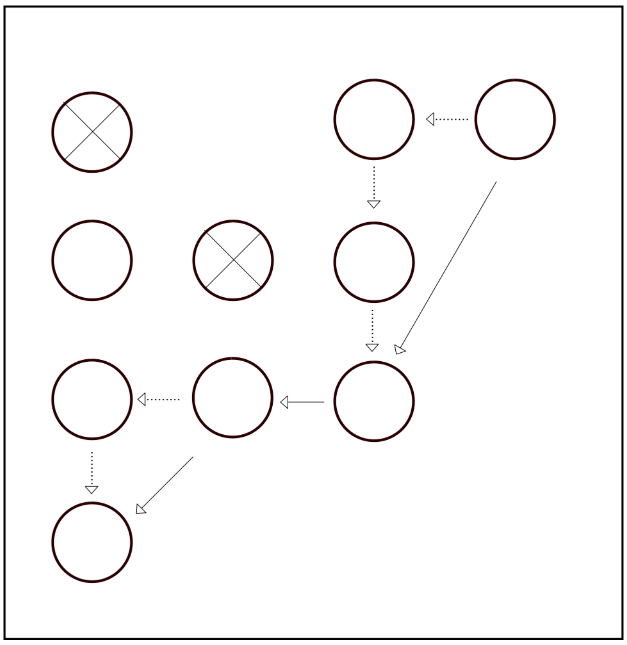


Figure 6.4: Arrows represent conections from child to parent. The doted arrows represent actual connections between nodes, the full arrows their representations in the data structure. Each level down represents older frame. The node farthest on the right represents new node connected to the tree. Node below which there isn't any else is the one first frame one. The crossed nodes are inactive. Inactive nodes collide and are no longer considered in the pruning procedure or used for nearest neighbor search.

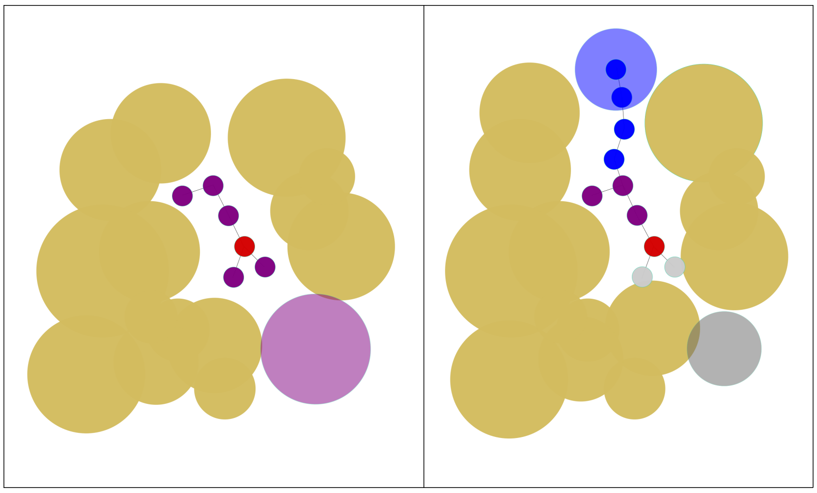


Figure 6.5: Illustration of tree expansion through frames, red node represents starting node, purple nodes represent the ones sampled in first frame, blue ones are sampled in second frame, grey nodes are inactive in presented frame.
Result after finishing search in first frame is shown on the left, blocking sphere is deleted after pruning procedure and nodes colliding with the blocking sphere are deleted as shown in figure (()).
On the right you can see result after new tunnel is found in second frame, before the start of tunnel optimization procedures. The testing sphere is the light blue one.

6.9.3 clustering

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 our algorithm, tunnels are clustered on the basis of 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. Duplicated tunnels are then assigned to cluster towards which the distance computed by duplication checking method is the shortest. If more than one tunnel belonging to same cluster is found in the same frame, only the first one is kept.
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.

6.9.4 centering

To gain information about the tunnels bottleneck through time, we need to center every ((time node)) individually against it's frame. To keep track of time passage, we save the frame, in which the node was found and the last valid one during the backtracking. The node then will be centered against them and all the frames inbetween.
Optimizing tunnels spanning through different frames requires saving every frame's collision structure.

6.9.5 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. The responsible procedure extension is described in figures 5.5 and 5.6.

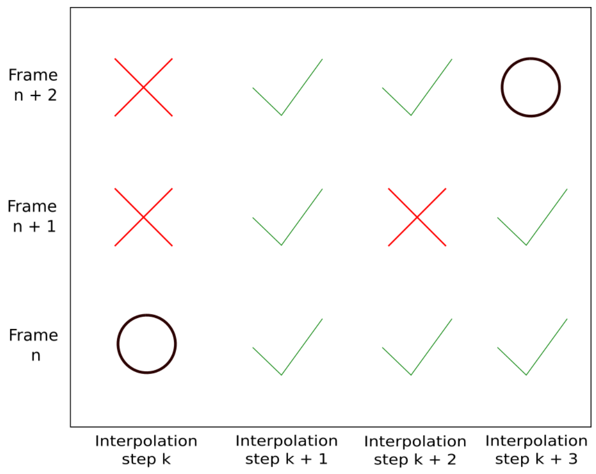


Figure 6.6: Let's have two tunnel nodes, the parent node in fiest_frame n and the child node in first_frame n + f. Let f be the difference between their first frames and d be their spatial distance, divided by the *interpolation_step*. Then we create matrix M with dimensions $f \times d$. Each element of the matrix belongs to one timespace coordinate. The element m_{i+1j+1} belongs to coordinate shifted by *interpolation_step* 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 to it's timespace coordinate collides with it's corresponding frame.

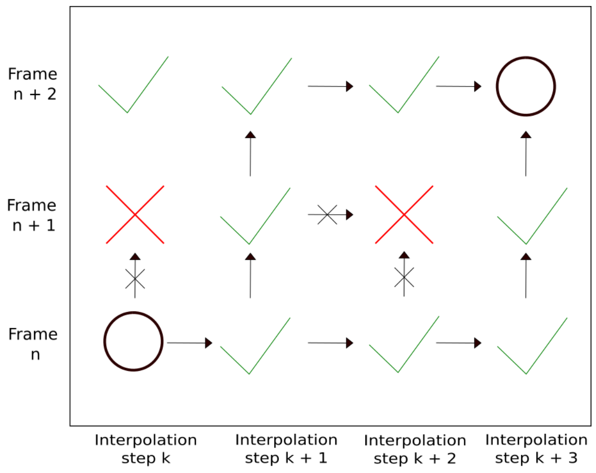


Figure 6.7: After the matrix is constructed, it is searched like a directed graph. Breadth-first search based algorithm was implemented. At the beginning, the coordinate belonging to the parent node is added to the visited queue. Then the algoritmic process starts. First element is taken from the queue. 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.

6.10 implementation details

Tunnel detection in thousands of snapshots in complicated environments, which is required in many applications((source?)) can be very time consuming. Therefore, optimizations are needed for the algorithm to perform as fast as possible.
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, the tunnel's 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. As current implementation doesn't support deleting existing tunnels, just using a pointer to them is ok. But extension requiring to delete already found tunnel can crash the program. ((todo picture))

Chapter 7

Experimental results

7.1 Configuration file

User can change all previously mentioned customizable parameters in file config.txt. Table presents short summary

Parameter name	default value	short explanation
max_step	0.05	interpolation resolution
iterations	100000	number of iterations
test_sphere_radius	5	radius of testing sphere
start_x	n/a	start x coordinate
start_y	n/a	start y coordinate
start_z	n/a	start z coordinate
coordinates_file	n/a	coordinate file for static search
number_of_frames	0	number of used frames
use_nrep_dupcheck	1	1 means program uses N_Representation duplication check
inside_sampling_bias	0.75	value
min_valid_intertunnel_distance	6	value of duplication check threshold
reseted_tree_mode 0	1	uses the delete_tree method when running only static search

7.2 computer

All tests were run on computer with four core Intel Core i5-7600 processor clocked at 3.5Ghz, 16GB Ram and running Ubuntu 16.04 operating system.

7.3 static search

We compare algorithm's results when the tree is reused and when deleted, as described in chapter(()). Theoretically, both methods should have their advantages and disadvantages.
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 existing path's endpoint could lead to quick discovery of 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 methods is required in dynamic environment search, so test of it's behavior is required.
Various values of the inside_sampling_bias are tested too. We are trying to determine what difference the values does.

(Dalsi testy - na pocet iteraci, vyvoj podle velikosti sondy.. mam uz neco spocitano a skripty na generaci tabulek)

Three use cases were selected. First one uses 1TQN molecule with probe size of 0.9A. This one was chosen as something in between. First one was MXTa

7.3.1 1TQN

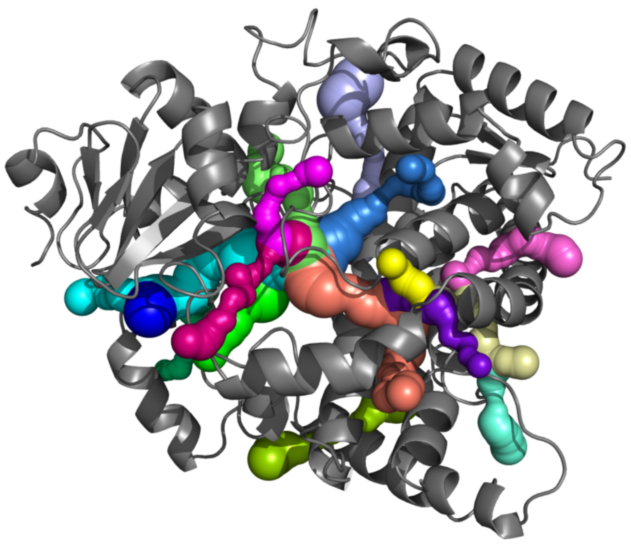


Figure 7.1: Tunnels found by Caver

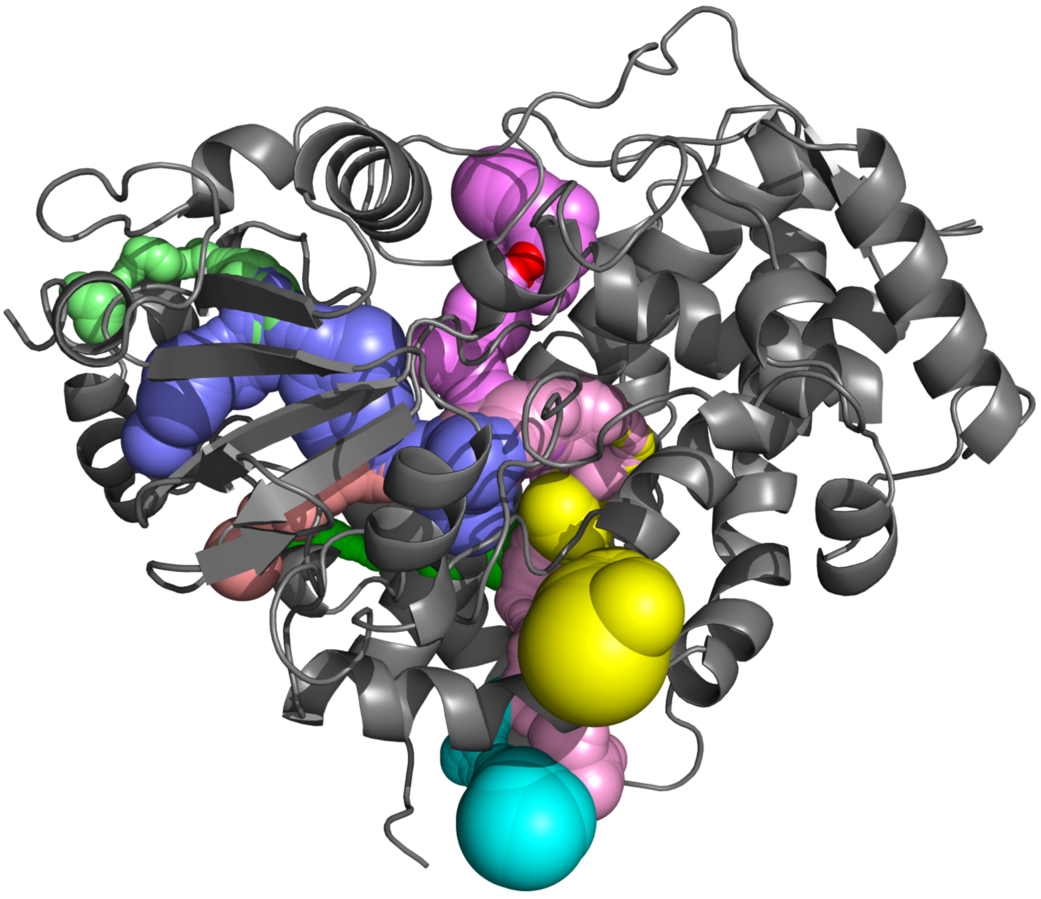
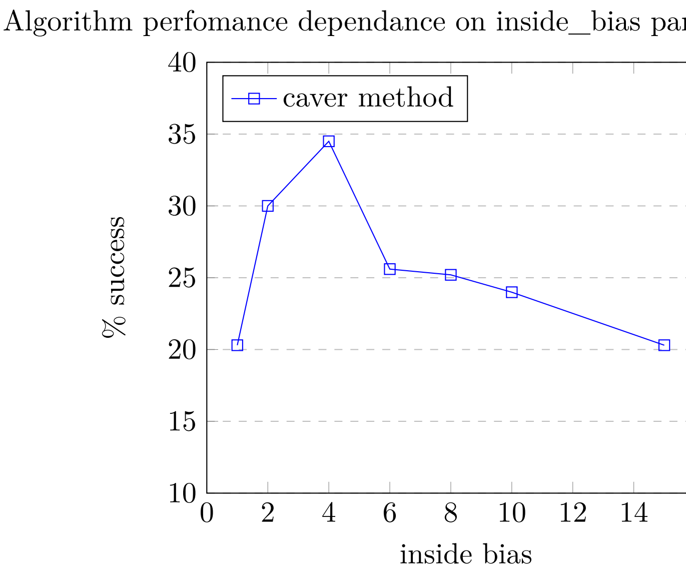


Figure 7.2: Some tunnels find by our algorithm

In following tables the reused and reseted tree methods are compared for different values of inside sampling bias. In the end, to test whether the using larger sampling bias is better than just increasing number of iterations, test with zero bias and 50% more iterarітions is taken. The runtimes are for all 100 runs.

1TQN, probe radius = 0.9A, 200 000 iterations, 100 runs						
	0		0.25		0.5	
caver tunnel n.	reused	reseted	reused	reseted	reused	reseted
1 (23.77)	94.06%	100.00%	96.04%	100.00%	92.08%	100.00%
2 (20.07)	99.01%	100.00%	100.00%	100.00%	99.01%	100.00%
3 (18.33)	97.03%	98.02%	92.08%	99.01%	98.02%	100.00%
4 (26.10)	10.89%	60.40%	10.89%	55.45%	12.87%	56.44%
5 (19.06)	11.88%	6.93%	12.87%	8.91%	21.78%	23.76%
6 (26.18)	50.50%	19.80%	55.45%	33.66%	55.45%	39.60%
7 (23.75)	1.98%	0.99%	8.91%	2.97%	10.89%	6.93%
8 (28.45)	5.94%	5.94%	8.91%	3.96%	6.93%	2.97%
9 (40.03)	0.99%	0.99%	0.99%	0.00%	0.99%	0.00%
11 (38.73)	8.91%	7.92%	14.85%	5.94%	22.77%	11.88%
13 (28.19)	5.94%	7.92%	5.94%	5.94%	8.91%	13.86%
14 (33.93)	5.94%	7.92%	8.91%	20.79%	15.84%	16.83%
15 (27.47)	3.96%	0.00%	1.98%	5.94%	6.93%	4.95%
16 (44.03)	40.59%	29.70%	48.51%	34.65%	46.53%	40.59%
18 (44.24)	0.99%	0.00%	0.00%	0.00%	3.96%	0.00%
overall success	24.37%	24.81%	25.91%	26.51%	27.94%	28.77%
runtime	500s	576s	606s	612s	663s	662s

1TQN, probe radius = 0.9A, 200 000 iterations, 100 runs						
	0.75		0.90		300k	
caver tunnel n.	reused	reseted	reused	reseted	reused	reseted
1 (23.77)	95.05%	100.00%	96.04%	100.00%	87.13%	100.00%
2 (20.07)	98.02%	100.00%	100.00%	100.00%	99.01%	100.00%
3 (18.33)	98.02%	100.00%	98.02%	100.00%	99.01%	100.00%
4 (26.10)	12.87%	64.36%	16.83%	64.36%	23.76%	63.37%
5 (19.06)	23.76%	17.82%	32.67%	17.82%	11.88%	12.87%
6 (26.18)	62.38%	58.42%	65.35%	58.42%	52.48%	22.77%
7 (23.75)	10.89%	0.00%	16.83%	0.00%	3.96%	0.00%
8 (28.45)	10.89%	6.93%	4.95%	6.93%	7.92%	7.92%
9 (40.03)	1.98%	0.00%	2.97%	0.00%	0.00%	0.00%
11 (38.73)	39.60%	11.88%	32.67%	11.88%	12.87%	7.92%
13 (28.19)	17.82%	7.92%	16.83%	7.92%	11.88%	2.97%
14 (33.93)	19.80%	19.80%	11.88%	19.80%	8.91%	14.85%
15 (27.47)	8.91%	5.94%	15.84%	5.94%	0.99%	7.92%
16 (44.03)	44.55%	43.56%	49.50%	43.56%	32.67%	40.59%
17 (44.61)	0.00%	0.00%	0.99%	0.00%	0.99%	0.00%
18 (44.24)	7.92%	0.00%	8.91%	0.00%	0.99%	0.00%
overall success	30.69%	29.81%	31.68%	29.81%	25.25%	26.73%
runtime	741s	690s	833s	690s	781s	776s



to tu je jenom protoze potrebuji ten kod

7.3.2 dynamic search

Search with 1000 frames was successfully run. It took around 50 minutes and used around 1.1 GB of Ram in the end. MAM ulozeny sessions, ale uz to nestiham renderovat

Chapter 8

Conclusions

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

8.1.1 Yes, table of contents

Theorem 8.1. 1. *Bla*

2. *Blo*

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis interdum facilisis urna, at tincidunt leo consectetur non. Maecenas bibendum mi vitae libero pharetra, ac ullamcorper nulla pellentesque. Sed sit amet massa nunc. Aenean placerat a est sodales sagittis. Quisque purus nibh, auctor ut consectetur at, suscipit non erat. Donec condimentum porttitor risus, vitae fringilla lectus tincidunt nec. Nulla leo quam, commodo eu ornare non, iaculis sed nulla. Duis gravida lacus quis purus sodales, vitae malesuada justo ultricies. Vestibulum nisl nulla, commodo non pellentesque a, fringilla a risus. Ut quis magna nulla. Mauris vitae ultricies ante, in consectetur justo.

Proof. 8 Bla

1. Blo

$\square$

Appendix A

Index

Appendix B

Bibliography

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