Master thesis on Intelligent Interactive Systems

Universitat Pompeu Fabra

Inverse Optimal Control for Modeling Virus Mutations in SARS-CoV-2

Ilse Meijer

Supervisor: Vicenç Gómez

Co-Supervisor: Mario Ceresa

Co-Supervisor: Antonio Puertas Gallardo

July 2022

Contents

Acknowledgement

I would like to express my sincere gratitude to my supervisor, Vicenç Gómez, for guiding me throughout this project and to my second supervisor, Mario Ceresa, for this same guidance and for preparing a useful state representation of the data. I would also like to thank my co-supervisor Antonio Puertas Gallardo and his team Gabriele Leoni and Jessica Gliozzo for providing me with the necessary preprocessed data. Lastly I would like to thank my family, boyfriend and friends for their support.

Abstract

Inverse Optimal Control (IOC) deals with the problem of recovering an unknown cost function in a Markov decision process from expert demonstrations acting optimally. In this thesis we apply IOC to SARS-CoV-2 data. For our application we use the (mutated) sequences found in SARS-CoV-2 data as the expert demonstrations. We present a way to learn useful state representations for this data, and successfully apply IOC on a special class of Markov decision processes which allow for an efficient computation of the value and cost functions of the states, and informative 2D representations of the state.

Keywords: Inverse Reinforcement Learning; Inverse Optimal Control; OptV; LMDP; SARS-CoV-2; Mutation;

Chapter 1

Introduction

RNA viruses use different mechanisms of genetic variation for their survival and with these mechanism they are among the most prevalent parasites for humans. One of the characteristics that helps viruses achieve this is their high mutation rate. Their offspring on average differs by one or two mutations from their parent. This combined with their short replication times, results in a complex virus swarm [1]. Their ability to change their genome this easily makes them able to appear in new hosts and circumvent immunity obtained by vaccination [2].

In order to fight these ever-changing viruses there has to be a continuous development of new vaccines. Still only few viruses can be effectively suppressed with vaccination/antiviral therapy [1]. There are several limitations to current vaccine technologies. One of them being the long production and distribution time of vaccines. Many researchers devote their time to make these production times shorter and make the vaccines more effective [3]. Some researchers now also focus on prepandemic vaccines, that can be administered at the beginning of or before a pandemic [4]. In this work we aim to contribute to the detection of regions of concern by developing an inverse optimal control (IOC) method that can help the prediction of future virus mutations.

Our contribution will be to frame the problem of learning from SARS-CoV-2 data as a problem of inverse optimal control (IOC). IOC considers an agent (in this case the

virus) acting optimally in an environment and deals with the problem of learning the (unknown) cost from state trajectories (RNA sequences from different patients at different times) generated by its optimal behavior or policy (a probability over RNA mutations).

For that, we will consider a particular framework for optimal control, called linearly-solvable optimal control [5] or also Kullback-Leibler control [6]. We will use an algorithm named OptV [7], which allows to compute the optimal cost optimized by the virus and the optimal policy efficiently. We will also design a learning algorithm that discovers a low-dimensional state representation from the high-dimensional data amenable for use in the inverse optimal control methodology. This is a challenging task, since the virus RNA consists of $3 \cdot 10^4$ bases and each mutation only differs in a few bases [8]. With this method and representation, we aim to extract the cost function that the virus uses to reproduce. Furthermore we aim to use this to model the behavior of the virus seen in data and lay the groundwork to later predict the possible strains of concern needed for the detection of viruses. When such a strain of concern is detected this should give some kind of alert, helping to spot new variants.

Our research is motivated by a recent study in which inverse reinforcement learning (IRL) has been successfully applied to real-world data from colorectal cancer patients [9]. In our case we focus on a particular class of control problems which allow for efficient computation under certain assumptions [5, 10]. Similarly as for colorectal cancer there exist many parallels between IOC/IRL and virus mutations.

Our approach is motivated by the following insights. First of all, the mutations of viruses follow a very complex behaviour. Since non-beneficial mutations within viruses will not help their survival and sometimes even cause them to go locally extinct [2], we can consider the viruses found in patients as optimally acting experts. Their successful mutations can be considered an optimal path. With inverse optimal control, we can derive an optimal mutation policy at every instance of the virus RNA.

Secondly, linearly-solvable optimal control problems satisfy several properties that make them interesting for this task. Contrary to more general models, these prob-

lems allow to obtain the solution of the inverse problem without having to solve the forward control problem, which is more computationally expensive. This computational advantage comes from the fact that this class of optimal control problems assume that the possible actions (in our case applied by the virus) are defined as a probability distribution between two different states (in our case, two different RNA profiles, or equivalently, a set of mutations). This assumption appears natural in our type of data, as we do not aim to characterize directly the intrinsic actions performed by the virus, but rather to have a more macroscopic description in terms of changes in the RNA mutations.

The sources of uncertainty in the Sars-Cov-2 data are multiple. First, it is not always clear which virus originated from where, since we collect the data from different individuals and do not take into account who infected who. Second, the VCF files used for variant prediction have a certain accuracy, which shows that we can not be absolutely sure about every mutation. Last, in order to sequence the virus RNA we use ARTIC. For this we need some sample virus RNA to make a virus-specific library [1]. Thus the virus can already have undergone multiple mutations before we discover and document it. Even if we have a library for some variant of the virus, a mutation in this variant can already disrupt the process.

Finally, we can look for useful information encoded in the obtained solutions (both the optimal policy and the underlying cost function) to find an interpretation that could be used for predictive and mechanical modelling.

1.1 Objectives

Our objectives are:

1. To learn a state representation from genomic data, in particular Sars-Cov-2 data that is amenable for IOC.

[1] Library Prep Methods for SARS-CoV-2 Sequencing for Surveillance and Research. https://sequencing.roche.com/en-us/science-education/education/articles/library-prep-for-sarscov2-research.html.

2. To implement an IOC algorithm for LMDPs (OptV).

3. To apply OptV to Sars-Cov-2 data and analyze the obtained results.

1.2 Structure of the Report

In the second chapter we will discuss the data and state representation. We will first go over the used methods to extract and preprocess the data and create the state representation. Next we will outline the experiments we use to test the quality of the state representations. Finally we will present the results of these experiments.

In the third chapter we will explain the OptV algorithm used for IOC, and show experimental results on a small grid-like problem.

In the fourth chapter we apply the OptV algorithm on SARS-CoV-2 data and present the results. In the fifth chapter we will draw conclusions and give directions for future work.

Chapter 2

Data and State Representation

2.1 Methods

In this section we will first discuss the methods used to extract and preprocess our data and create a suitable state representation. Then we will outline the experiments we apply to test the quality of the created state representation.

2.1.1 Data

We downloaded raw FASTQ data of 865 samples from the Covid-19 Data Portal accessed at April 2022 [1]. In FASTQ files DNA-sequences can be stored [2]. FASTQ files are created from the combination of base calling files, each of which stores the letter of the DNA in one particular position of the DNA. FASTQ files are text files that contain information about the sequencing process, the found sequence and the quality of each base in the sequence.

Samples were selected according to the following filters:

- Country: Italy,
- Library selection: RT-PCR,
- Instrument platform: ILLUMINA,

[1] Covid-19 Data Portal. Viral Sequences. `https://www.covid19dataportal.org/`

[2] `https://support.illumina.com/bulletins/2016/04/fastq-files-explained.html`.

- Library strategy: Amplicon.

In general we chose these filters to obtain the lowest amount of possible confounding factors, whilst getting a high amount of samples. By selecting one country and specific library selection and strategy techniques we reduce biases introduced by different laboratories, sequencing methods and sample extraction methods. We chose Illumina since it is the most widely adopted technology for this purpose and moreover because Illumina generates short reads instead of long reads. Short read strategies have the advantage of having a higher quality of each base in the sequence, which improves the variant calling steps that will follow in the next section.

2.1.2 Data Preprocessing

We process the raw FASTQ data with the covigator-ngs-pipeline using default parameters [3]. This pipeline consists of multiple steps: first the sequences with low quality are removed. Next, every sequence is aligned with the reference genome [4]. After this, some more preprocessing steps are applied, such as removing duplicated sequences. Thereafter variant calling is performed, for which we use the Genome Analysis Toolkit (GATK). GATK trains machine learning algorithms on a dataset of known variants and uses them to compute a variant calibration score for every sequence of the input file [5]. It outputs the variants that pass this test in a variant calling file (VCF). After some more necessary preprocessing steps the covigator-ngs-pipeline performs variant annotation, which is a method used to get more insight in the biological impact of the variants, it predict for example if variants are harmful. Finally to retain only high confidence calls, we remove variants with read depth (DP VCF field) below or equal to 5, giving us a total of 346 samples.

Next we construct a phylogenetic tree relying on the augur toolkit using the SARS-CoV-2 reference genome sequence as root (WuhCor1, MN908947.3) [11]. The augur toolkit was designed to transfer sequence data (such as our vcf files) into phylogenetic

[3] https://github.com/TRON-Bioinformatics/covigator-ngs-pipeline.

[4] https://www.basepairtech.com/knowledge-center/introduction-to-variant-calling-qc-alignment-deduplication-variant-annotation/.

[5] https://gatk.broadinstitute.org/hc/en-us/articles/360035531612-Variant-Quality-Score-Recalibration-VQSR-.

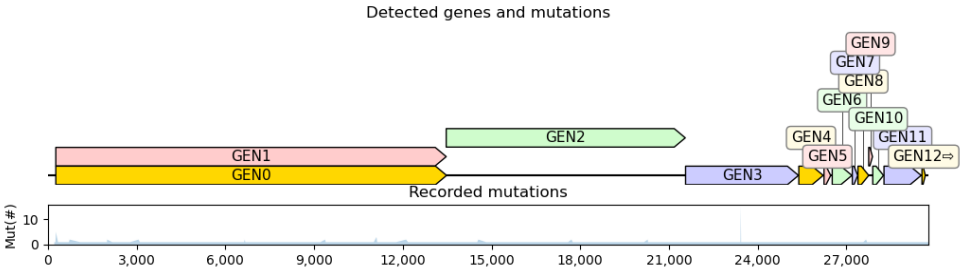


Figure 1: Observed mutations

trees. A phylogenetic tree is a tree that shows the lines of evolutionary descent of genes starting from their common ancestor [12]. The resulting phylogenetic tree is composed by leafs, which represent viral genomic sequences that were observed in patients, and nodes, that represent the viral genomic sequences of their common ancestors. The nodes connect the leafs to the root, which contains the common ancestor of every leaf. Every split in the tree represents one or more mutations events. In figure 1 we show an overview of the mutations found in these samples and in which of the 13 genes it occurred.

2.1.3 State Representation

To create the state representation we first transform the phylogenetic tree into a graph. Some nodes in the phylogenetic tree did not mutate compared to their parent node. If this is the case we remove this node and if the node has successors, we create an edge between its parent and each of its successors. Because nodes without any mutation are the same as the parent node, we add the probability of going to this node, to the probability of the parent to stay in the same node. This way we still use the information of the deleted nodes.

Figure 2 shows the resulting graph. We use this graph to create two different state representations. The first state representation is one dimensional: we consider every node of the graph as a state. The second state representation is derived from the RNA sequence corresponding to each node. This RNA sequence is first transformed into a 9,984 dimensional vector: 13 vectors of size 768. Each of the 13 vectors represent a gene of the RNA with over 10,000 bases per gene. We create this full

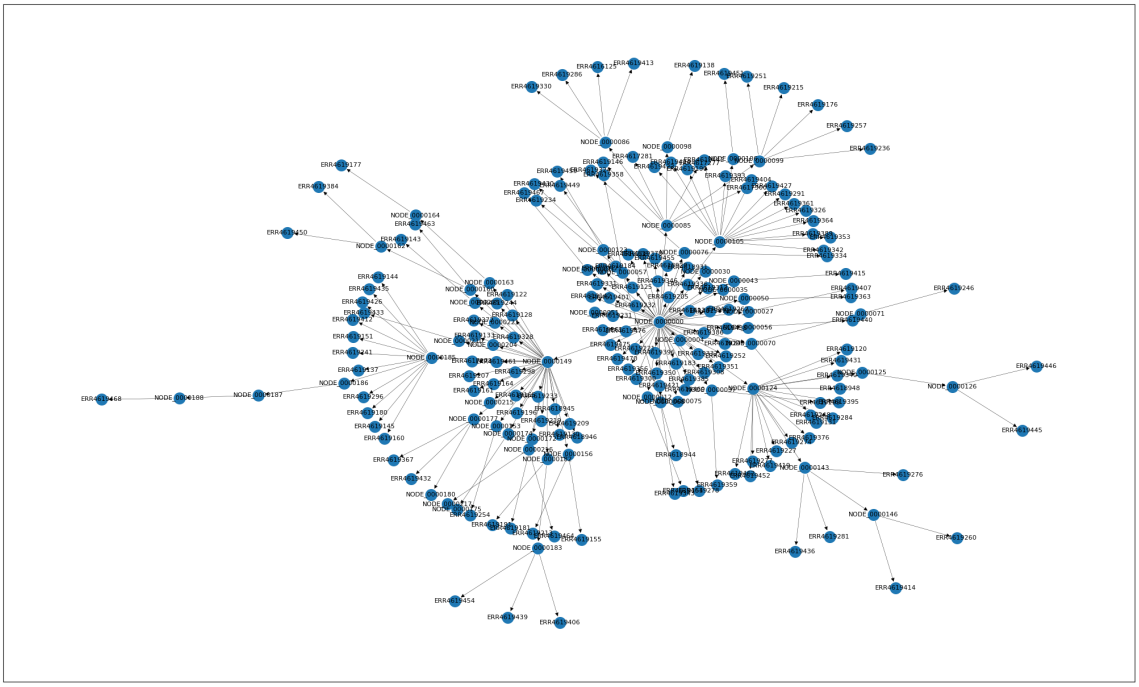


Figure 2: Graph constructed from phylogenetic tree.

vector using DNABERT, which combines a BertForMaskedLM architecture and a pretrained DNA tokenizer [13]. BERT is a transformer that was trained for natural language processing purposes [14]. DNABERT thus looks at the DNA as a language composed of k letter words formed with the letters A, C, T and G. Our version is configured using six letter chunks of RNA (e.g. AACTAG). We apply our BERT model to every of the 13 genes in each of the nodes in the phylogenetic tree. Each gene is divided into 768 chunks of 512 letters and these chunks are separately given as input to BERT. Each position in the resulting 786 dimensional vector represents one of these chunks. We then use the Potential of Heat-diffusion for Affinity-based Trajectory Embedding (PHATE) to transform this into a 2D vector [15]. PHATE is a dimensionality reduction model that aims to preserve both local and global distances between feature-vectors. The resulting state representation thus contains a 2D coordinate for every state, representing the RNA sequence corresponding to that state.

Because the one dimensional state representation is discrete, we can later apply our OptV algorithm directly. For the 2D case this is a bit more complicated, since the states consist of two continuous numbers between -1 and 1. Therefore we have to

discretize the states into bins before applying our OptV algorithm.

2.1.4 Experiments

In this section we evaluate the choice of state representation in terms of similarity between the real-world data corresponding to similarity in the state representations. In this thesis we consider two state representations: every node in the phylogenetic tree is a state (1D) and a 2D vector representing the RNA sequence of every node in the phylogenetic tree.

The one dimensional state representation is directly taken from our original environment: the phylogenetic tree and is thus a reliable representation.

To test our 2D state representation we have to perform two comparisons:

- Are the full 9,984D vectors a good representation of the phylogenetic tree?
- Are the 2D vectors a good representation of the full vectors?

Full state representation

To answer the first question we use the State Representation Learning (S-RL) toolbox [16], which wrote about ways to evaluate an reinforcement learning state representation. According to them the most practical way to test both representations is to see if the corresponding RL algorithm can perform its task. However there is no clear baseline performance for this task yet and we do not know the reward function for the problem, so this evaluation is not straightforward. Therefore we perform two of the alternative tests mentioned: the k nearest neighbours mean squared error (KNN-MSE) and the mantel test. Both of these methods need a measure of distance. For full vectors we use the Euclidean distance. For the phylogenetic tree we count the number of bases that mutated between each state.

KNN-MSE We use the k nearest neighbours mean squared error (KNN-MSE) to test the local coherence between states [16]. In this metric x is the state in the learned state representation and $\tilde{x}$ is the same state in the original representation.

We calculate:

$$\text{KNN-MSE}(x,k) = \frac{1}{k} \sum_{x' \in KNN(x,k)} ||\tilde{x} - \tilde{x}'||,$$

where $\text{KNN}(x,k)$ returns the k nearest neighbours of x. Instead of implementing a machine learning algorithm to make predictions, we obtain the k nearest neighbors by looping over the state space and finding the k states that are closest to x. This is possible since the state space is small enough.

We use the KNN-MSE to test if states that are close together in the state representation, were also close together in the original environment. This is important to validate, since we want the local structure of the original environment to be maintained. Since this matrix is quite big, we report the mean, mode, minimum and maximum of the matrix. In order to draw conclusions on the size of these values, we compare it to the maximum distance in the original environment.

Mantel Test To test the overall correlation we use a two-tailed mantel test [17]. The mantel test is a statistical test to measure the correlation between two different distance matrices. The mantel test is especially suitable to test the correlation between two distance matrices created by different metrics and is thus a good tool to test the correlation between the number of mutations and the euclidean distance. To obtain the correct input we first calculate the distance matrix for each representation. We then use this test to derive the Pearson correlation coefficient ρ and test if this value is smaller than 0.05.

2D state representation

To answer the second question we use the pyDRMetrics toolbox, which is designed to analyze dimensionality reduction (DR) algorithms [18]. They use multiple general methods, which all evaluate a different aspect of the DRs performance and are independent of the DR method used. We will not use their first method, since it is not possible for our DR algorithm to reverse the DR.

Distance matrices The second method consists of comparing the two distance matrices, by comparing the heatmaps and by calculating the residual variance of Pearsons correlation coefficient. The residual variance is a value between zero and one that tells us the percentage of variation in the reduced matrix that can not be explained by the original matrix. We thus want this value to be as low as possible.

Ranking matrices The third method compares the two ranking matrices. A ranking matrix is related to a distance matrix, but instead of the distance between point i and j, the ranking matrix calculates the number of points with a distance bigger than the distance between point i and j. Formally:

$$R_{ij} = \#\{k | D_{ik} < D_{ij} \cup (D_{ik} = D_{ij} \cap k < j)\}.$$

This matrix is more suitable for pairwise comparison between states, since it takes into account the relative distance instead of the specific distance.

To compare the difference between these two ranking matrices we calculate the co-ranking matrix. The element Q_{kl} denotes the number of samples of rank k that become rank l. More formally this is:

$$Q_{kl} = \#\{(i,j) | R_{ij} = k \cap R'_{ij} = l\}$$

For this matrix we want the values on the diagonal to be high and outside the diagonal to be (close to) zero: this means that there is no change in rank from the original matrix to the reduced matrix.

To get a more in depth idea of the co-ranking matrix we calculate the trustworthiness and continuity. Trustworthiness (T) is a measure of the number of false close pairs generated by the DR. Continuity (C) is a measure of the number of previously close

points that become distant after DR. Formally they are defined as:

$$T(k) = 1 - \frac{1}{mk(m-k)} \sum_{i=k}^{m} \sum_{j=1}^{k} Q_{ij}(i-k)$$
$$C(k) = 1 - \frac{1}{mk(m-k)} \sum_{i=k}^{m} \sum_{j=1}^{k} Q_{ij}(j-k),$$

where m is the number of samples.

Local and global properties We also use the covariance matrix to compute the agreement between the k-nearest neighbors. We denote the co-k-nearest neighbor size with:

$$Q_{NN}(k) = \frac{1}{km} \sum_{i=1}^{k} \sum_{j=1}^{k} Q_{ij},$$

which is again a value between zero and one. Since we are solely interested in the nearest neighbors, we want to determine a suitable maximum for k. To accomplish this we take the argmax of the Local Continuity Meta Criterion (LCMC), which subtracts the baseline from our measure for k-nearest neighbors and is defined as:

$$LCMC(k) = Q_{NN}(k) - \frac{k}{m-1}.$$

We now define the local and global rank properties as:

$$Q_{local} = \frac{1}{k_{\max}} \sum_{k=1}^{k_{\max}} Q_N N(k)$$
$$Q_{global} = \frac{1}{m - k_{\max}} \sum_{k=k_{\max}}^{m-1} Q_N N(k).$$

A value of Q close to one means that the nearest neighbors in both the original vectors and the reduced vectors are similar, which is desirable.

2.2 Results

In this section we present the results from applying the previous experiments to the state representation.

2.2.1 Full state representation

First we compare the full state representation to the original states. In table 1 we report the mean, mode, minimum and maximum of the KNN-MSE for different values of k. Considering that there exists at least one letter mutation between every state, the mean KNN-MSE is low, considering that every state differs at least in one mutation. However the maximum value of the KNN-MSE is very high, even compared to the maximum distance between two nodes in the graph, which is 29 mutations. When looking at the entire matrix we can see that this is caused by one state. We can see this as well by the low value of the mode, which indicates that the most common value is low even when we increase the number of nearest neighbours k. This means that there exists one state that differs a lot in number of mutations from a state that has a very similar embedding. Overall the full representation is thus able to capture the original representation, but there is still room for improvement.

k	Mean	Mode	Minimum	Maximum
2	3.41	2.5	1	19
3	3.59	2.3	1	19.7
4	3.72	2.5	1.5	20.5
5	3.82	2.8	1.4	20.8
10	4.15	2.9	1.4	21.6

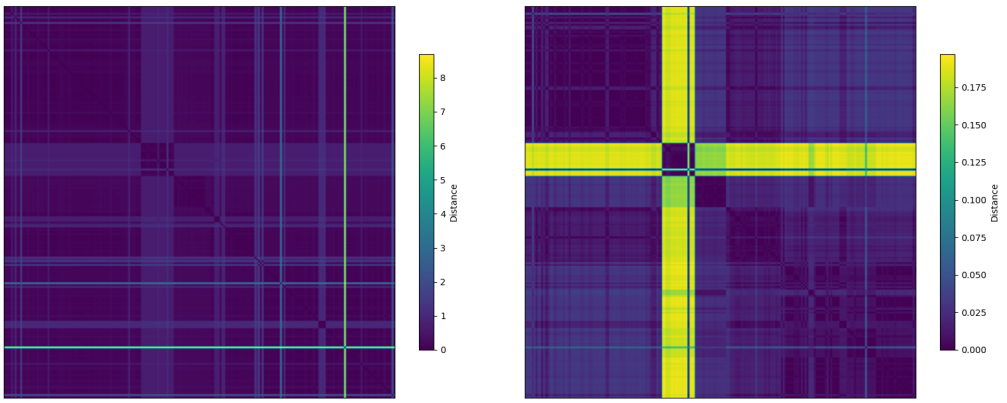
Table 1: Results KNN-MSE for different values of k

Next we calculate the mantel test. For this we obtain the value $\rho = 0.0087 < 0.05$, which means that the correlation between the two distance matrices is significant.

2.2.2 2D state representation

Now we examine the 2D state representation created from the 9,984D state representation, corresponding to the RNA sequence of a node in the phylogenetic tree.

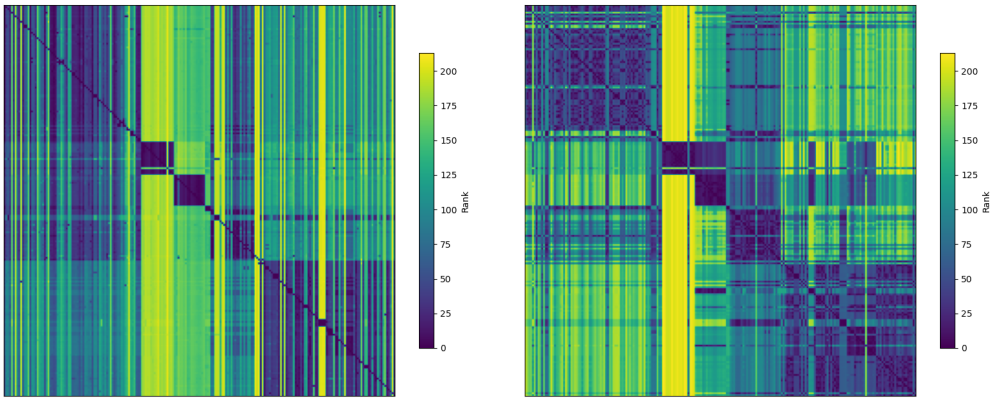
Figure 3 shows the two distance matrices. We observe some similarities in the pattern, but for the reduced version there is a cluster of nodes that is far from the remainder of nodes. This same cluster is visible in the original version, but it is not



(a) Full vector. (b) 2D vector.

Figure 3: Heatmap distance matrices.

as far away from the rest of the graph. Instead there are two isolated nodes or very small clusters that are far from the remainder of nodes in the original version.



(a) Full vector. (b) 2D vector.

Figure 4: Heatmap ranking matrices.

In figure 4 we show the ranking matrices. Here we see something similar: the general pattern is similar, with some small disturbances, but the location of the highest rank is different in the two versions. To analyze these in more detail we show the co-ranking matrix in figure 5.

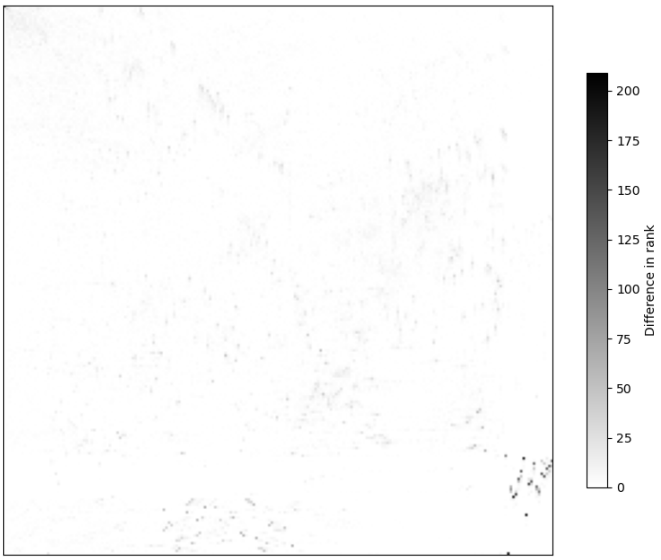


Figure 5: Heatmap co-ranking matrix

The co-ranking matrix shows the difference between two ranking matrices: a high value on the diagonal means high similarity. We can see many dots surrounding the diagonal, but they do not create a clear diagonal lines and there are also a lot of outliers. To investigate this further we take a look at figure 6, in which we plot the value of the trustworthiness and continuity for different values of k.

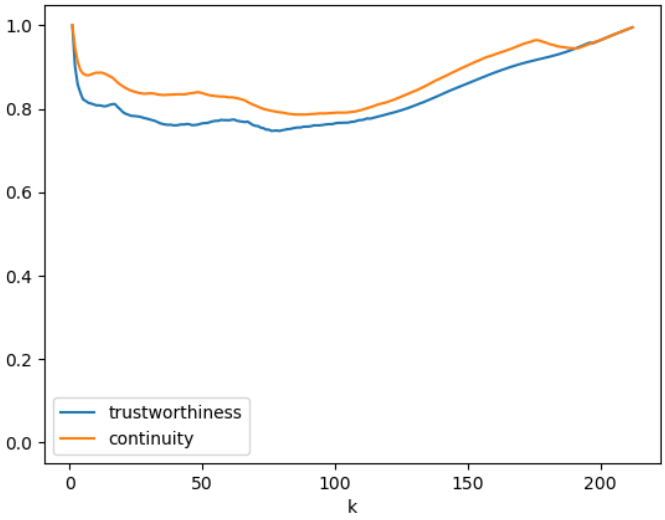


Figure 6: Trustworthiness and continuity

Especially for very low and high values of k the trustworthiness and continuity

are close to one. The continuity is always bigger or equal to the trustworthiness, meaning that there exist more distant points that were close before DR than close points that were distant before DR. Next we calculate the local and global coherence:

$$Q_{local} = 0.296$$
$$Q_{global} = 0.658.$$

Unfortunately the local coherence is low and also the global coherence is not very close to 1. This can be improved for next state representations.

Chapter 3

Inverse Optimal Control

Inverse optimal control is a type of inverse reinforcement learning used to recover the underlying cost function of a Markov decision process. In our case, we focus on linearly-solvable MDPs (LMDP).

3.1 Methods

In this section we will first give an explanation about LMDPs, after this we will explain the inverse optimal control algorithm we used called OptV. Finally we will outline the experiments we will execute to show the performance of the algorithm.

3.1.1 Linearly-Solvable MDPs

Linearly-Solvable MDPs (or LMDPs) were introduced by Todorov [19].

The main difference from an LMDP compared to a standard MDP is that LMDPs do not consider a discrete action space $\mathcal{A}$ for the agent, nor a transition probability distribution $P(x'|x,a)$ between state x and x', as resulting from applying action $a \in \mathcal{A}$ in state x. Instead, LMDPs model directly an action as a transition probability between two consecutive states, effectively replacing the traditional symbolic actions $a \in \mathcal{A}$ with continuous probability distributions u that model the probability to go from a state x to a state x' directly, i.e., $u(x'|x)$, and no explicit modeling

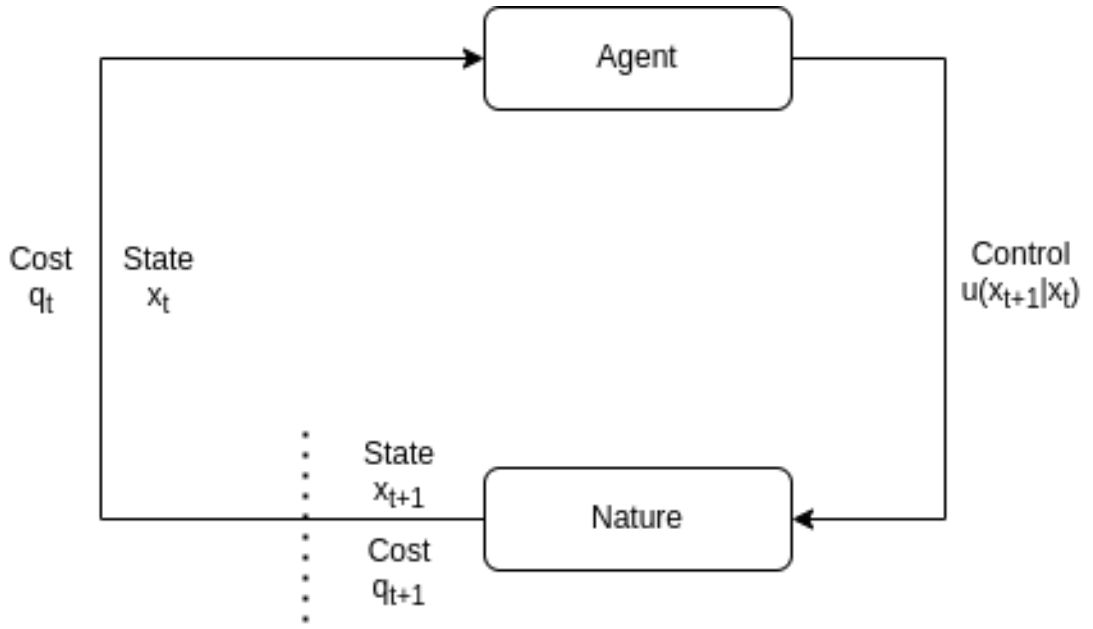


Figure 7: Schematic representation of a linearly-solvable MDP (LMDP).

of discrete actions a is considered [5]. Further, LMDPs assume the existence of a base distribution $p(x'|x)$ that defines how the state evolves in the absence of control. Figure 7 shows a schematic representation of an LMDP.

Such uncontrolled dynamics dictate what transitions are allowed or prohibited by the controls, i.e., $p(x'|x) = 0$ implies $u(x'|x) = 0$, preventing the agent to jump between states that are not connected.

Immediate Cost

Instead of an immediate cost $c(x, a)$ depending on the current state x and the taken action a, as in traditional MDPs, LMDPs define the immediate cost as the sum of two terms: a state-dependent term $q(x)$, and a control-dependent term that quantifies how much the controlled distribution u deviates from the uncontrolled dynamics p in an information theoretical sense, using the Kullback-Leibler (KL)-divergence between u and p [20]:

$$c(x, u) = q(x) + \sum_{x'} u(x'|x) \log \frac{u(x'|x)}{p(x'|x)}. \tag{3.1}$$

The KL-divergence is well defined, because we previously demanded that $p(x'|x) = 0$ implies $u(x'|x) = 0$.

The standard objective to optimize in a MDP is the cost-to-go (or value function) $v(x)$, which is defined recursively via the so-called (optimal) Bellman equation, which in the case of LMDPs becomes [5]:

$$v(x) = \min_u \{q(x) + E_{x' \sim u(\cdot|x)} \left[\log \frac{u(x'|x)}{p(x'|x)} \right] + E_{x' \sim u(\cdot|x)} v(x')\}. \tag{3.2}$$

We will follow [5, 7] to show that, under the following transformation of v

$$z(x) = \exp(-v(x)), \tag{3.3}$$

the Bellman equation (3.2) becomes linear in z.

The function (3.3) is also named *desirability* function, as it takes high values when the cost to go is low. Since v is now expressed as minus the logarithm of z, this will give us some convenient properties when inferring the optimal value function analytically.

Optimal transition function

Using the Bellman equation and Eqs. (3.1),(3.3), the optimal transition function (or optimal control) $u^*(x'|x)$ can be obtained in closed-form:

$$u^*(x'|x) = \frac{p(x'|x)z(x')}{\mathcal{G}[z](x)}. \tag{3.4}$$

Linear Bellman Equation

Substituting back u^* in the formula for $-\log(z(x))$, the KL-divergence becomes equal to zero, thus we get:

$$-\log(z(x)) = q(x) - \log(\mathcal{G}[z](x)). \tag{3.5}$$

Taking the expectation now results in:

$$z(x) = \exp(-q(x))\mathcal{G}[z](x),$$

which is linear in z.

In addition to leading to a linear Bellman equation, LMDPs enjoy several computational advantages and theoretical properties, such as compositionality of optimal control laws [21, 22, 23]. LMDPs have been applied in numerous settings for robotics [24, 25, 26], multi-agent systems [27, 28, 29], or for finding policies in other complex scenarios such as power grids [30], online forums [31], crowd-sourcing [32], or rankings [33].

3.1.2 Inverse Optimal Control using OptV

The property we exploit in this work is the convexity of the inverse optimal control problem, which leads to an efficient algorithm called OptV [7]. In figure 8 we have visualized the main steps the OptV algorithm consists of. It takes as input the system dynamics in the absence of control $p(x'|x)$ discussed before and a set of transition pairs (instead of trajectories that are usually needed for IRL [10]). Using this we want to estimate the optimal value function v, the desirability function z, the state cost function q and the optimal control law u^*. Fortunately once we have v, we can infer z from equation 3.3, q from 3.5 and u^* from 3.4. In order to estimate v we can calculate the maximum likelihood estimator taken from n independently distributed equations 3.4. The likelihood function is:

$$\begin{aligned} L(z[.]) &= \prod_{i=1}^{n} u^*(x'_i|x_i) \\ &= \prod_{i=1}^{n} \frac{p(x'_i|x_i)z(x'_i)}{\mathcal{G}[z](x_i)} \\ &= \prod_{i=1}^{n} \frac{p(x'_i|x_i)z(x'_i)}{\sum_{x' \in X} p(x'|x_i)z(x')} \end{aligned}$$

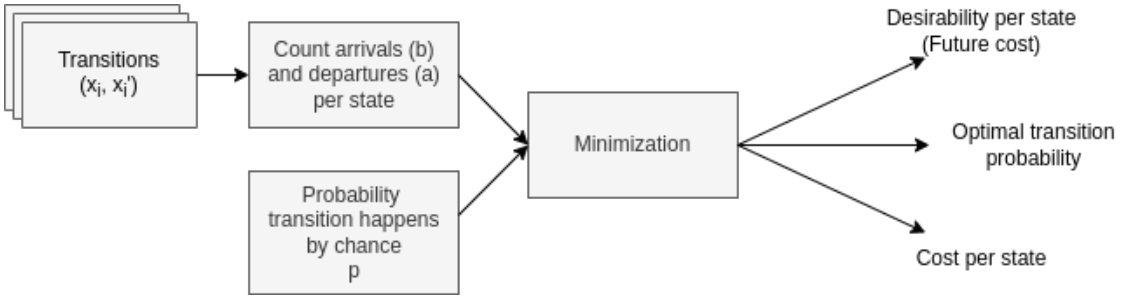


Figure 8: Main concept OptV

When we take the negative log-likelihood, we obtain:

$$\mathcal{L}(z[.]) = \sum_{i=1}^{n} \left(-\log(p(x_i'|x_i)) - \log(z(x_i')) + \log \left(\sum_{x' \in X} p(x'|x_i) z(x') \right) \right).$$

Since our method uses minimization of $\mathcal{L}$ to obtain the optimal value of z we can remove the terms that do not depend on z. This gives:

$$\mathcal{L}(z[.]) = -\sum_{i=1}^{n} \log(z(x_i')) + \sum_{i=1}^{n} \log \left(\sum_{x' \in X} p(x'|x_i) z(x') \right).$$

In terms of v we thus have:

$$\mathcal{L}(v[.]) = \sum_{i=1}^{n} v(x_i') + \sum_{i=1}^{n} \log \left(\sum_{x' \in X} p(x'|x_i) exp(-v(x')) \right).$$

Which we can simplify using the visitation counts $a(x')$ and $b(x)$ representing the number of times $x_i' = x'$ and $x_i = x$, respectively:

$$\mathcal{L}(v[.]) = \sum_{x'} a(x') v(x') + \sum_{x} b(x) \log \left(\sum_{x' \in X} p(x'|x_i) exp(-v(x')) \right). \tag{3.6}$$

We use the quasi-newton method to find our estimate of v [34].

3.1.3 Experiments

We will first replicate the experiment from the original paper [7]. Our implementation uses the LMDP described before with a goal in the bottom left corner and two

walls. From this we sample M trajectories of length T and store each trajectory in state transitions. Subsequently we extract the visitation counts for each state needed for our likelihood function 3.6. We create a matrix p for the system dynamics in the absence of control which consists for each state of a uniform probability to go to each of its neighbours or itself. Next we run our OptV model. We analyze plots of the resulting value function and compute the difference in control matrices, comparable to the original paper. Moreover we compare the cost-values of the original LMDP and the inferred cost. Besides this we analyze the result with different numbers of sample trajectories M, to test how many trajectories we need for the OptV to converge.

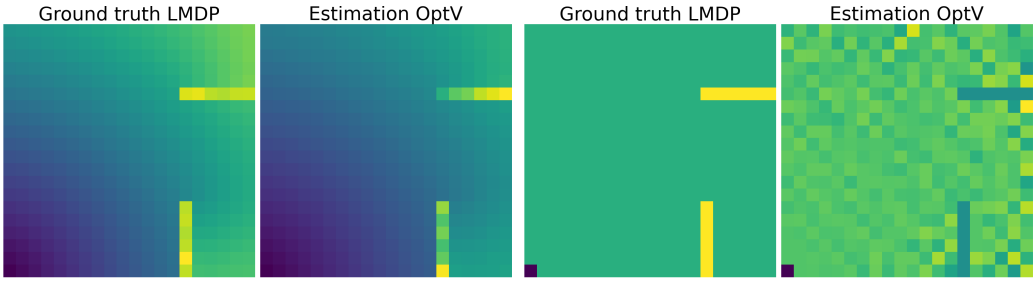
Thereafter we show the results of the binned version of OptV we implemented. We do this by aggregating neighbouring states into bigger bins and applying OptV to these bins. Later on, we will need this to deal with the continuous values of the 2D covid state representation. For the binned case we only analyze the plots of the value and cost function, since our control matrices have different sizes.

3.2 Results

In this section we will discuss the results from the previously mentioned experiments.

3.2.1 Replication GridWorld

In this section we show that we can replicate the results of the original paper [7]. We use the same 20x20 GridWorld with two sets of obstacles and show both the value and the cost function in figure 9. These results were obtained by running OptV using $M = 1e4$, where M is the number of trajectories. In every figure of this section, the left plot is the ground truth and the right plot is the estimation with OptV. We analyze the control function estimated by OptV by comparing it to the system dynamics in the absence of control of the GridWorld. The Euclidean distance between the two is 7.14. Thus for this GridWorld the model has an error of 1.79% per pixel.



(a) Value function. (b) Cost function.

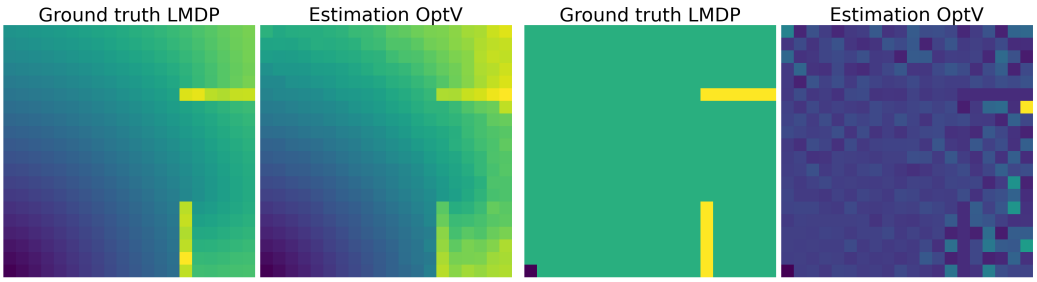
Figure 9: Results OptV 20x20 GridWorld with walls.

Tuning hyperparameters

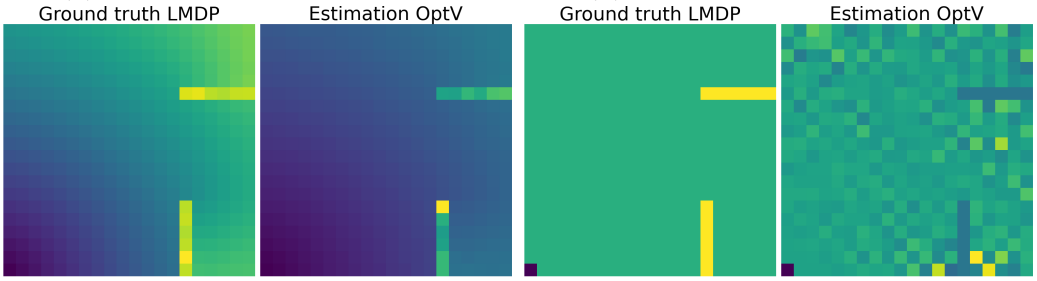
Next we test for which number of trajectories M we have convergence. In figure 10 we show results for some different values of v. As you can see in figure 10a and 10b for $M = 1e3$ there is no convergence yet. There are some spots in the value function, especially close to the obstacles that differ a lot from the ground truth results. Furthermore the cost function is unable to distinguish the goal clearly and does not recover the location of the walls. For $M = 5e3$ in figure 10c and 10d we can already see improvements, the value function is now slightly overestimating the quality of the grid compared to a part of the wall, but we can see a big improvement in the cost function: the walls and goals are clearly visible. We can see the best result in figure 9a and 9b, where we used $M = 1e4$.

3.2.2 Binned GridWorld

In this section we will discuss the results for binned states. For this we use a 10x10 GridWorld, which we analyze in 2x2 bins, creating a 5x5 GridWorld. We consider two different types: a goal in the bottom left corner and a 2x2 goal in the middle of the GridWorld. This results in a value function that is quite accurate. However the cost function overestimates the cost right before reaching the goal. It does so in both cases, but in the case with multiple goals in the middle the difference with the surrounding cost is less big. The most important part though is that it is able to estimate where the lowest cost is, so that the agent will try to reach this low cost state.

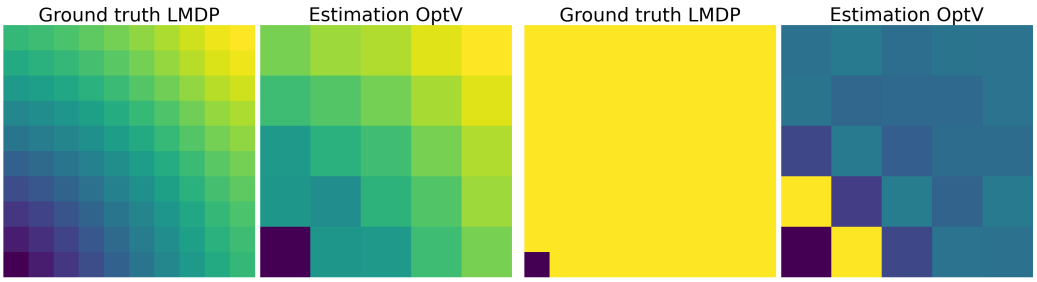


(a) Value function for $T = 1e3$. (b) Cost function for $T = 1e3$.

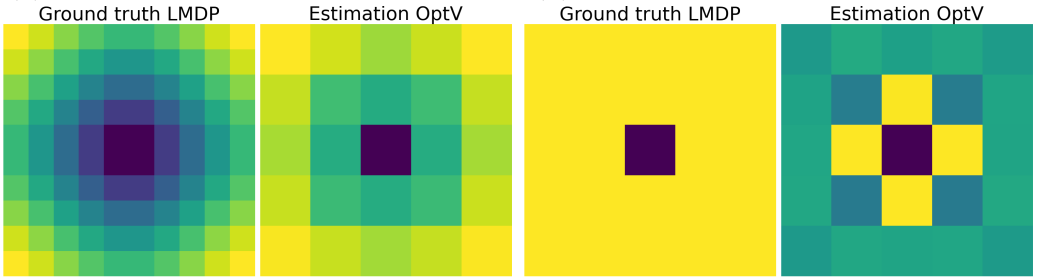


(c) Value function for $T = 5e3$. (d) Cost function for $T = 5e3$.

Figure 10: Results OptV for different values of hyperparameter T.



(a) Value function of 10x10 GridWorld. (b) Cost function of 10x10 GridWorld.



(c) Value function of 10x10 GridWorld. (d) Cost function of 10x10 GridWorld.

Figure 11: Results OptV binned GridWorld.

Chapter 4

IOC using SARS-CoV-2 data

4.1 Experiments

In this section we outline the experiments we perform to analyze the performance of our algorithm on the SARS-CoV-2 data. Just like for the GridWorld case, we traverse the graph obtained from the phylogenetic tree M times to sample T state transitions. For the one dimensional case we store the state names, for the two dimensional case we store the 2D coordinates of every state. We traverse the tree using the probabilities we observed in the data of the virus. For this we use the augur toolkit used to generate the phylogenetic tree, with which we can extract the frequencies with which every node in the tree was encountered. We normalize the frequencies over the entire tree and calculate the conditional probabilities between every pair of parent, successor nodes.

4.1.1 The one dimensional case

For the 1D case we can immediately extract the visitation counts from the trajectories mentioned before. After this we again create a system dynamics in the absence of control matrix p, this time consisting of the probability to randomly mutate, and thus make a transition from one state to the other, which is a lot smaller than the probability with which we observe transitions in the virus. With the visitation

counts and the system dynamics in the absence of control we run our OptV model.

For the analysis we will take the following steps:

- Run the experiment twice and compare the difference between both models.
- Plot and analyze the cost and value function of both models.
- Closely examine the extrema in the cost and value functions.
- Project the cost and value function to the 2D positions of the nodes.
- Plot and analyze the control function.
- Compare the control dynamics to the passive and virus probabilities. Examine to what degree it differs from both.

4.1.2 The two dimensional case

For the 2D case we have to first determine the edges of the bins. We do this by splitting the x-axis in 80 bins and the y-axis in 95 bins. In areas with more nodes, the edges of the bins are closer together and the other way around. This way we obtain a maximum of 7 nodes per bin. Similar to the 1D case we create the system dynamics in the absence of control matrix p based on the probability to randomly mutate. For every bin we sum the probabilities to move from one bin to another, since some bins contain multiple nodes. Still these probabilities will be very small. With the visitation counts and the system dynamics in the absence of control we run our OptV model.

We repeat the analysis steps for the 1D case.

4.2 Results

In this section we will discuss the results of applying OptV to the covid data. In order to understand the analysis it is important to note that all the nodes starting with *ERR* are leaf nodes and nodes starting with *NODE* are (by augur inferred) intermediate states. Since we removed some states that did not have any mutation, some states submerged a leaf state, while others turned into a leaf state completely. For more clarity see figure 2 and table 2. The nodes in the left column are the

nodes that turned into a leaf node. The nodes in the right column are the ones that nothing happened to: they are still intermediate nodes. The remaining nodes are intermediate nodes that submerged a leaf node and thus were seen in real life, but also mutated further.

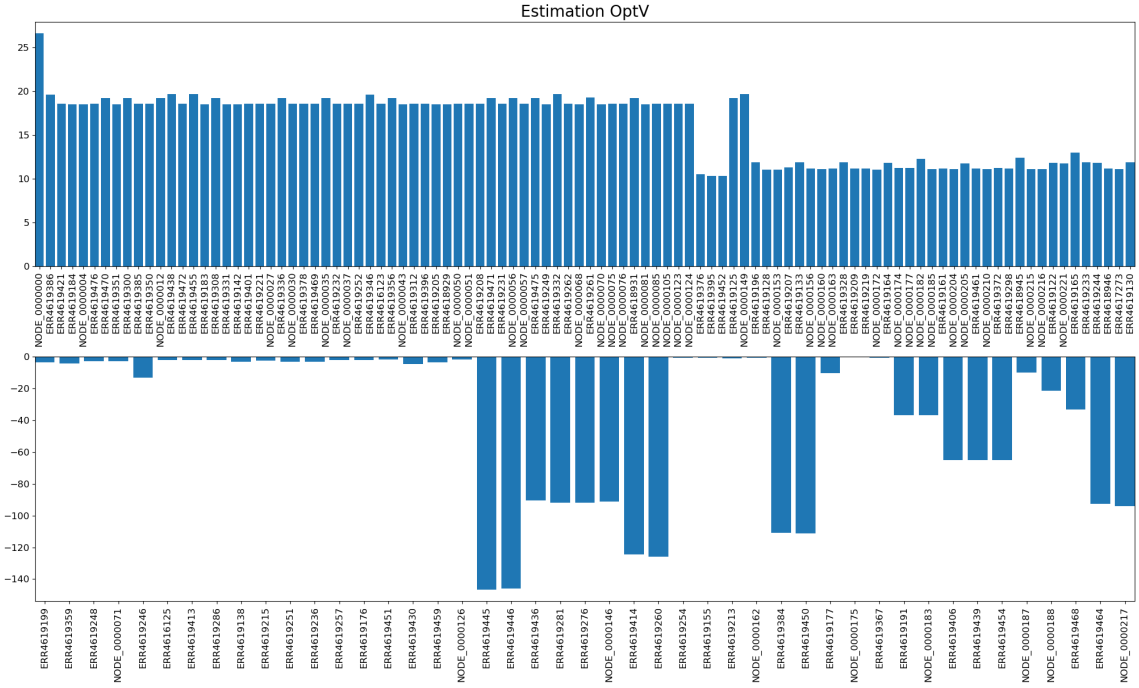
Leaf nodes	Intermediate nodes
NODE_0000004	NODE_0000037
NODE_0000030	NODE_0000070
NODE_0000035	NODE_0000123
NODE_0000051	NODE_0000126
NODE_0000175	NODE_0000143
NODE_0000180	NODE_0000146
NODE_0000204	NODE_0000162
NODE_0000215	NODE_0000182
NODE_0000217	NODE_0000183
NODE_0000221	NODE_0000216

Table 2: Node kinds

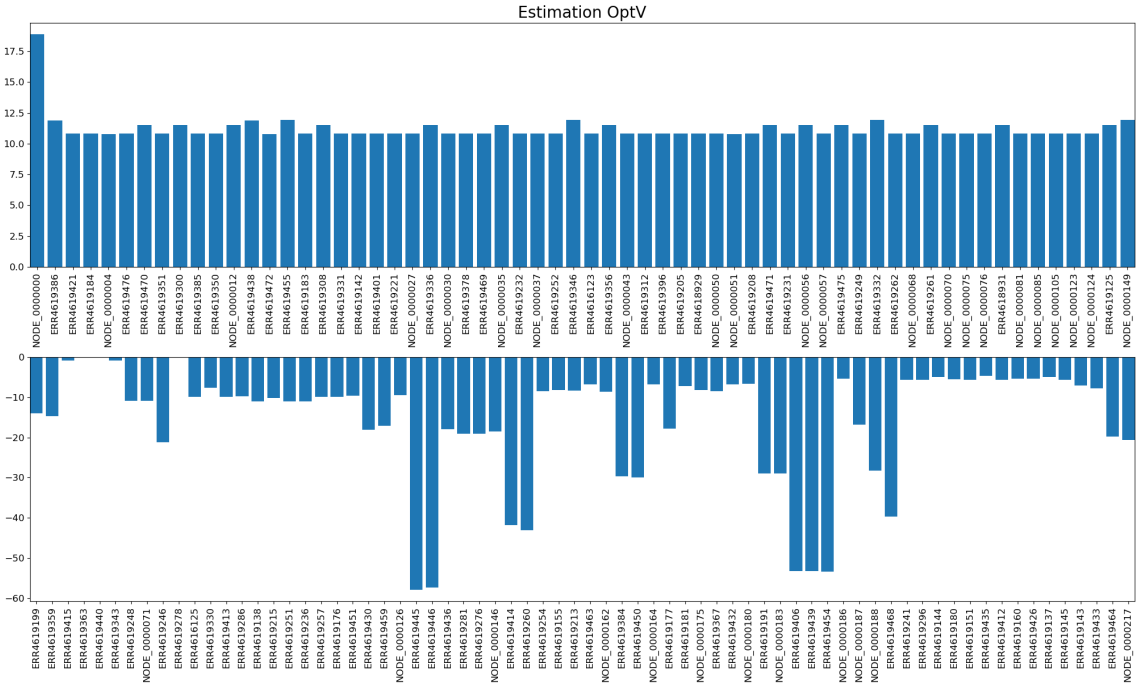
4.2.1 The one dimensional state representation

In this section we will analyse the results of the first state representation: the states are nodes in the graph extracted from the phylogenetic tree. We run the experiment twice to see the difference per run. Unlike in the GridWorld case, for the covid case, there are multiple successful states and therefore the result can differ each time. In figure 12 we plot the nodes with a value function lower than 0 or higher than 10, since we are mainly interested in the extreme cases. Recall that the value function represents the expected cost to go. In figure 13 we show the state cost value for nodes with a state cost higher than zero. All the other nodes have cost equal to zero.

By looking at the plot of the value function (figure 12), we can conclude that the size of the value function differs between experiments. For the positive value function we can see a number of different levels, of which the value differs per experiment, but the nodes in both experiments are exactly the same for each level. Every level represents exactly each layer of successors in the graph. Remember that we only show the nodes



(a) Experiment 1.

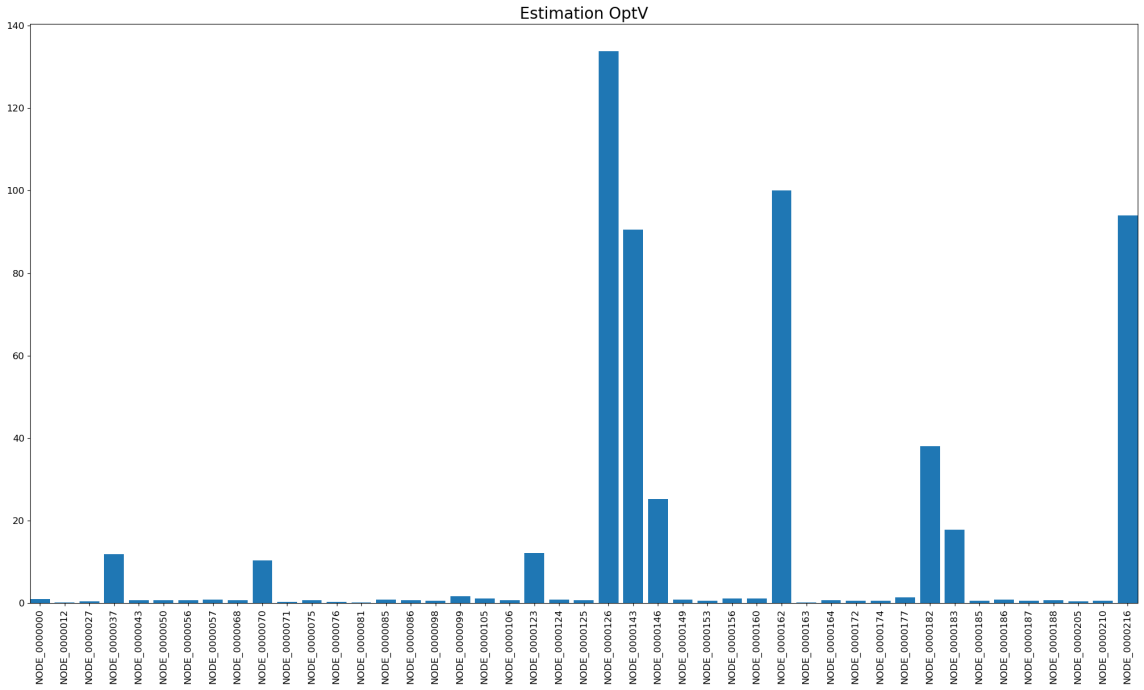


(b) Experiment 2.

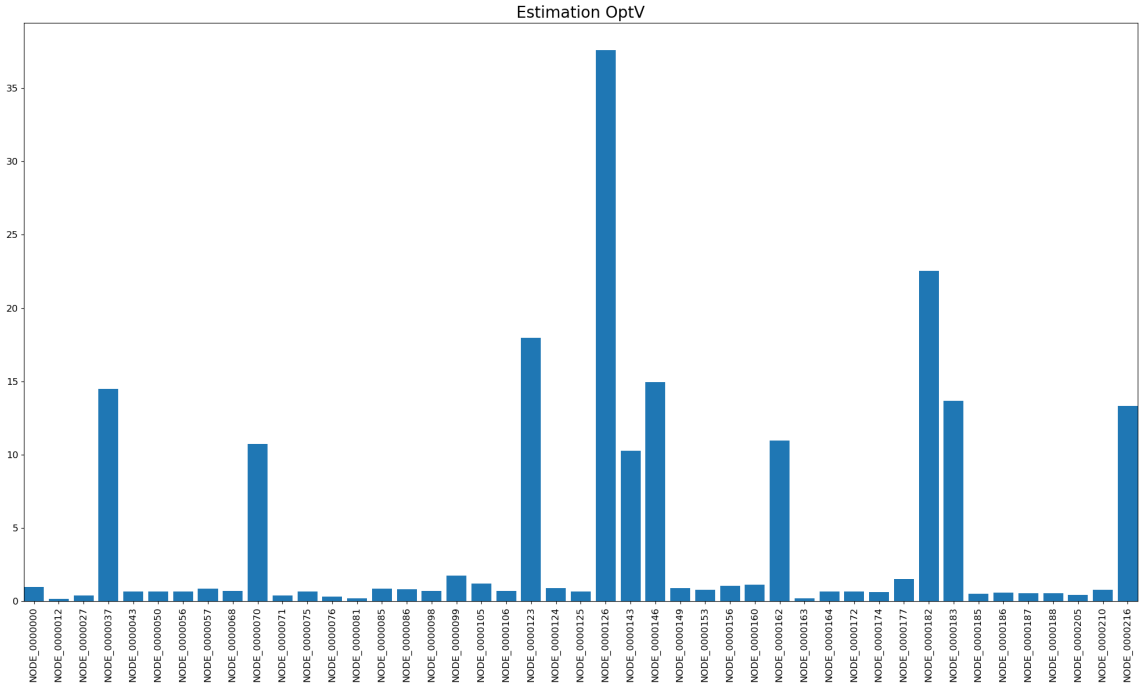
Figure 12: Value functions 1D states SARS-CoV-2.

with values higher than 10. For this reason the third level is not visible at the second experiment, but it exists there too. For the negative values, which are the beneficial states, we can see a similar pattern between the two experiments as well. The cost function is even more stable: in every experiment the same nodes have a positive

value and the others a value of 0 (figure 13). The exact value of the positive cost fluctuates, but not as much as that of the value function.



(a) Experiment 1.



(b) Experiment 2.

Figure 13: Cost functions 1D states SARS-CoV-2.

We will now analyze the value and cost function in more depth. In table 3 we show the ten nodes with smallest values of the value function. In table 4 we do the same

for the biggest of the cost function. In these tables we can see what we mentioned before: the top ten nodes of the value function do not always appear in the both experiments, while the top ten nodes of the cost function are exactly the same. It is noticeable that the maxima in the cost function are exactly the intermediate nodes in the phylogenetic tree that were not observed but inferred, while the minima in the value function are only leafs that did not mutate further in the phylogenetic tree. This is a sign that the model is modelling the data correctly, since the leafs are the latest version of the virus sequences found in real life and are therefore more desirable.

Experiment 1	Experiment 2	Both
ERR4619281	ERR4619406	ERR4619260
ERR4619276	ERR4619439	ERR4619384
ERR4619464	ERR4619454	ERR4619414
NODE_0000217	ERR4619468	ERR4619445
		ERR4619446
		ERR4619450

Table 3: Nodes with the smallest value function

Both
NODE_0000037
NODE_0000070
NODE_0000123
NODE_0000126
NODE_0000143
NODE_0000146
NODE_0000162
NODE_0000182
NODE_0000183
NODE_0000216

Table 4: Nodes with the biggest cost function

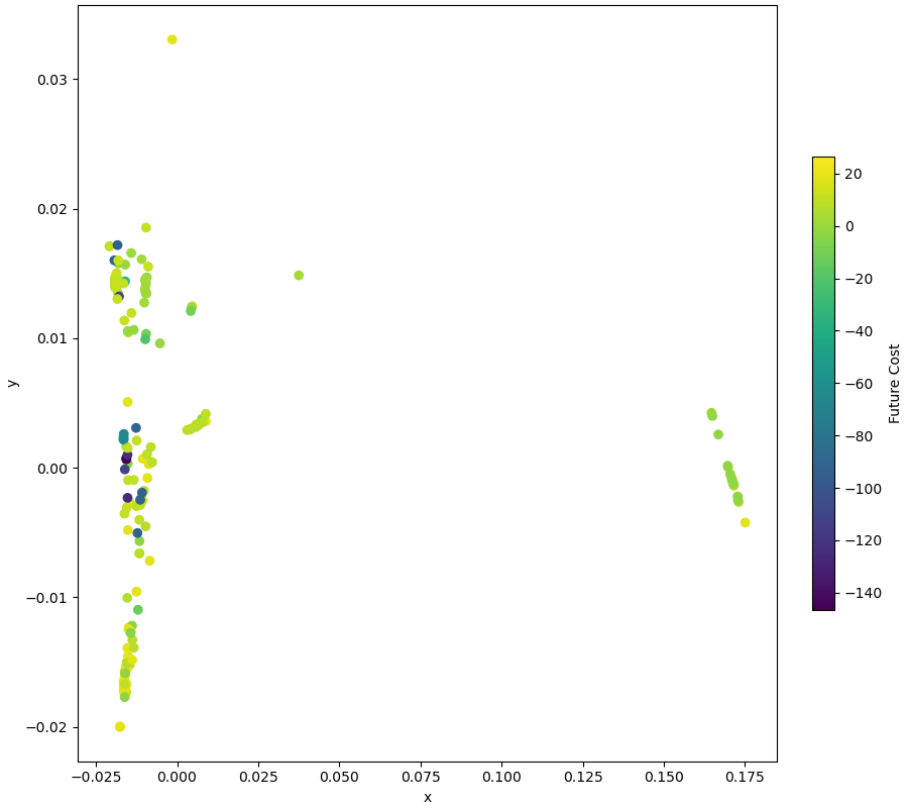
To get a better idea of the meaning of the states we project the value function we obtained with the one dimensional OptV on the 2D positions of the nodes. The results can be viewed in figure 14 in which the blue represents low values and yellow high values. We do the same for the cost function in figure 15. All nodes with a cost larger than one are enlarged. Notice that the highest cost values are located mainly on the left middle and top clusters and that the every high value node seem

to have a cluster of low value nodes surrounding it. We will later compare these to our 2D OptV results.

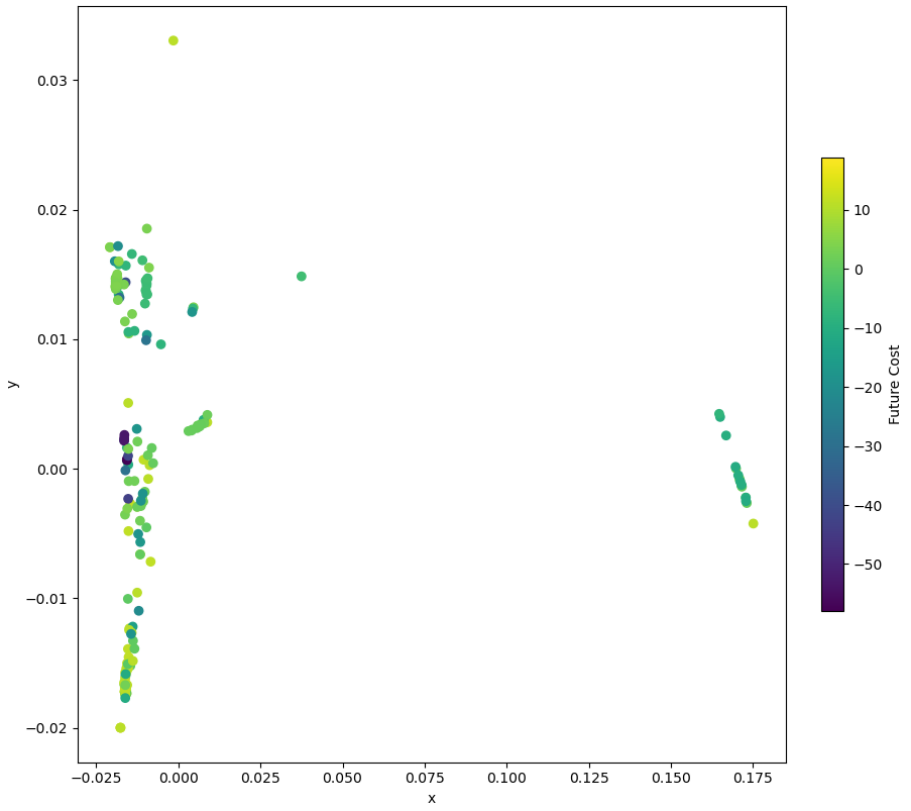
Finally we will look at the control function that was estimated by our model (figure 16). Since this is a very large matrix, we first make it more compact. We exclude from the rows all nodes that go with probability one to themselves. Next we calculate the sum over each column (thus the sum of nodes that arrive at the corresponding node, excluding the ones from the already deleted rows). If this sum is smaller than 0.1, we delete this column. This way we have a more compact representation of the control function. On the y-axis in the plot of the control function (figure 16) we can see the nodes that we move away from and on the x-axis the nodes that we move towards. The color of each positions indicates the probability with which this move is made. We can see that all paths in the graph are possible, since only the leaf nodes have probability one to themselves and are excluded from the y-axis. Every intermediate node is either mapped to a leafnode, to itself or to another intermediate node. Especially intermediate nodes that were seen in real life have a high probability to stay at the same node. Every node has the possibility to reach a leafnode eventually, although this might take a big number of steps.

Furthermore we want to know to which degree the control function differs from the virus dynamics. In table 5 we include all nodes with their successors of which one transition has a probability that differs at least 0.05 from the virus probability.
A difference of 0.05 is very low and still only four nodes satisfy this demand for the first experiment and only two for the second experiment. From this we can conclude that the model follows the virus probability very close. We will discuss the biggest divergences one by one:

NODE_0000162 has equal probability to go to either one of the leaf nodes for both the virus and passive probabilities, however for the passive probabilities the chance is very low. The model completely ignores the system dynamics in the absence of control preference to stay in the current state and instead has a high probability to go to either one of the leaf states. In doing so it has a preference for ERR4619450. Why exactly this happens we are not sure, it is true that the value function of this

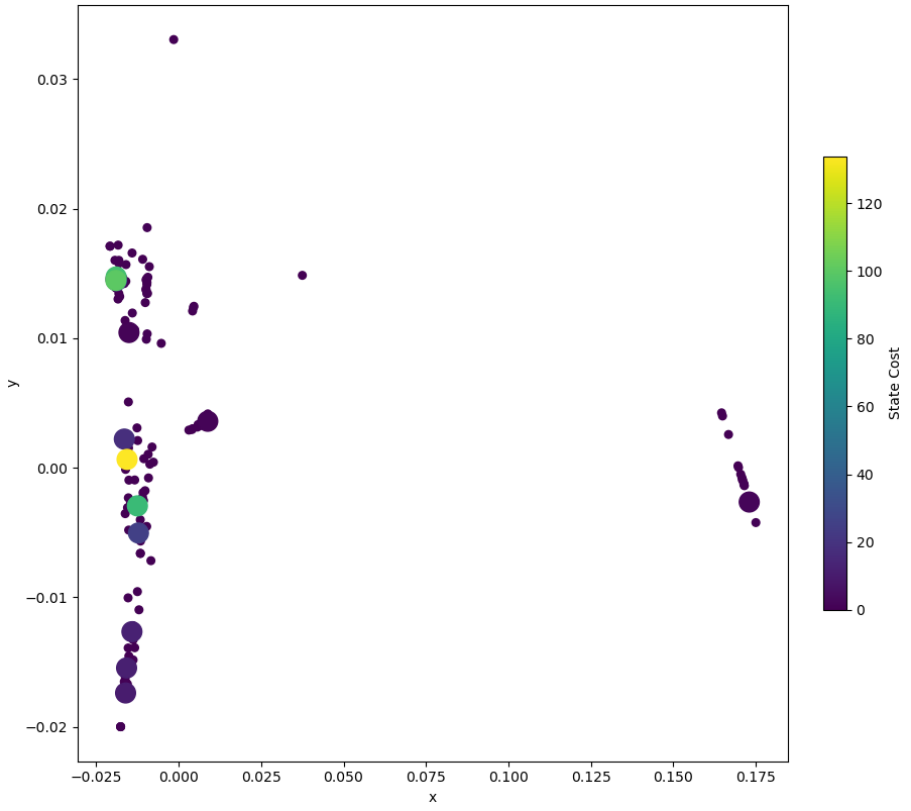


(a) Experiment 1.

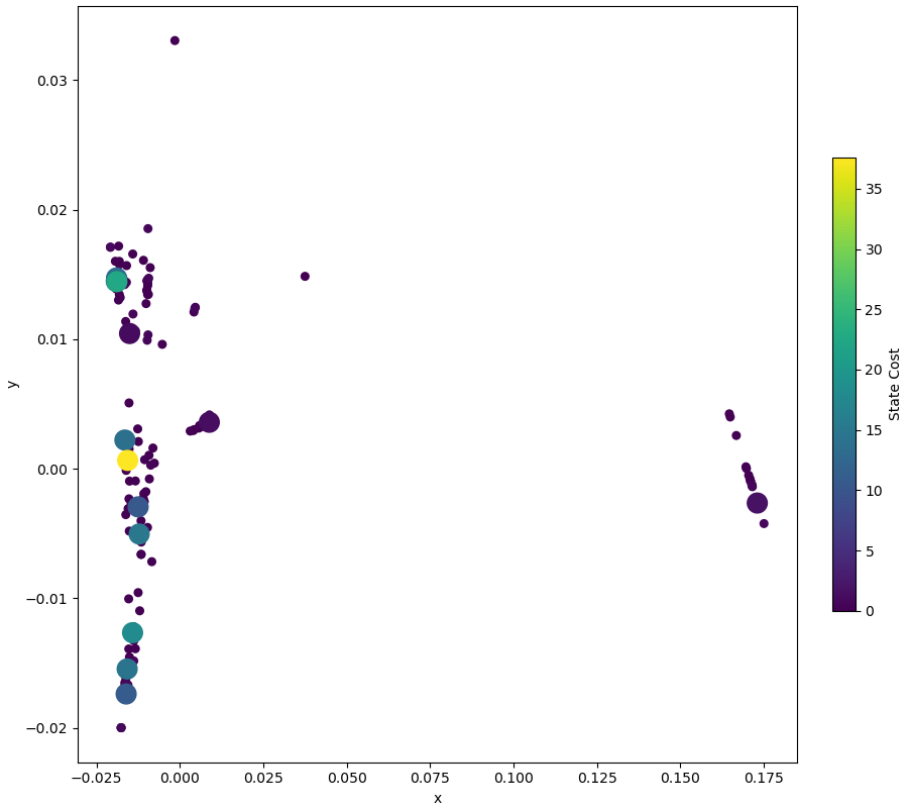


(b) Experiment 2.

Figure 14: 1D Projection Value Function

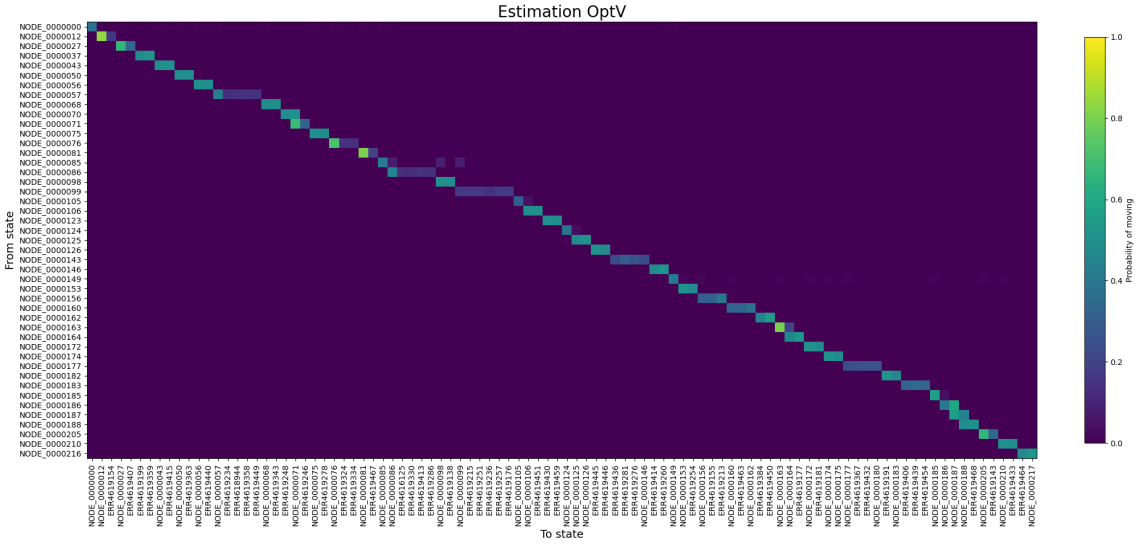


(a) Experiment 1.

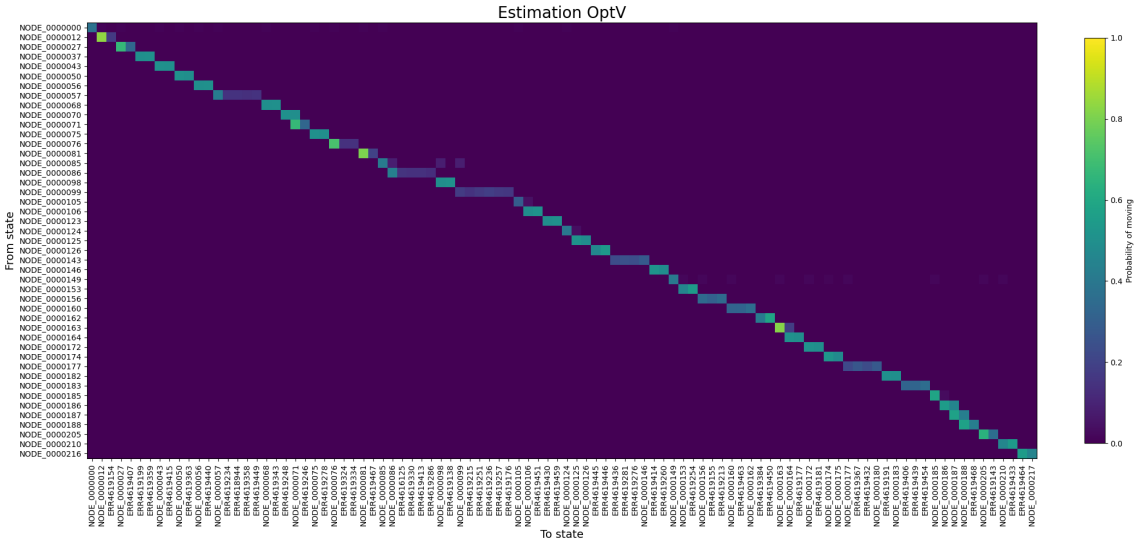


(b) Experiment 2.

Figure 15: 1D Projection Cost Function



(a) Experiment 1.



(b) Experiment 2.

Figure 16: Control functions 1D states SARS-CoV-2.

node is bigger than the other node, but this difference is very minimal.

For NODE_0000187 and NODE_0000188 the preference for staying in the same state is clearly caused by the combination of the high positive passive probability

From	To	Control u	p_virus	p_passive
NODE_0000162	ERR4619384	0.4153	0.5	1.6e-05
NODE_0000162	NODE_0000162	0.0	0.0	0.999966
NODE_0000162	ERR4619450	0.5846	0.5	1.6e-05
NODE_0000187	NODE_0000188	0.4293	0.5	8e-06
NODE_0000187	NODE_0000187	0.5706	0.5	0.999991
NODE_0000188	ERR4619468	0.4284	0.5	8e-06
NODE_0000188	NODE_0000188	0.5715	0.5	0.999991
NODE_0000216	ERR4619464	0.5501	0.5	2.5e-05
NODE_0000216	NODE_0000216	0.0	0.0	0.999966
NODE_0000216	NODE_0000217	0.4498	0.5	8e-06

(a) Experiment 1

From	To	Control u	p_virus	p_passive
NODE_0000186	NODE_0000187	0.5949	0.5	8e-06
NODE_0000186	NODE_0000186	0.405	0.5	0.999991
NODE_0000187	NODE_0000188	0.4403	0.5	8e-06
NODE_0000187	NODE_0000187	0.5596	0.5	0.999991

(b) Experiment 2

Table 5: Transitions in which the control dynamics differ from the virus dynamics

and the indifference of the virus probability. For experiment 2 something similar happens with NODE_0000187.

NODE_0000216 has a preference for ERR4619464 even though the other leaf node (NODE_0000217) has a slightly more appealing value function. This is probably caused by the higher passive probability for this state.

In experiment 2 NODE_0000186 has a difference of almost 0.1 with the virus probability and 0.6 with the passive probability. This can be explained by the low negative value its successor NODE_0000187 has.

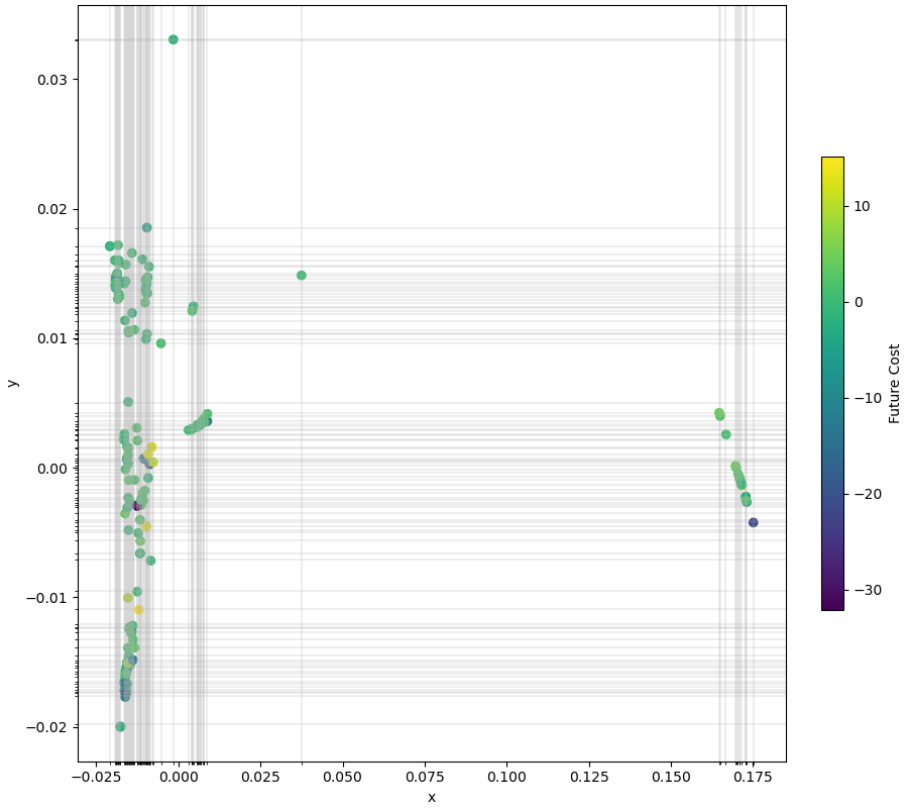
As can be seen in these tables, the inferred control function differs a lot from the system dynamics in the absence of control. We do not include all the examples for which this is the case, since there are so many. Some examples can be seen in table 5.

4.2.2 The two dimensional case

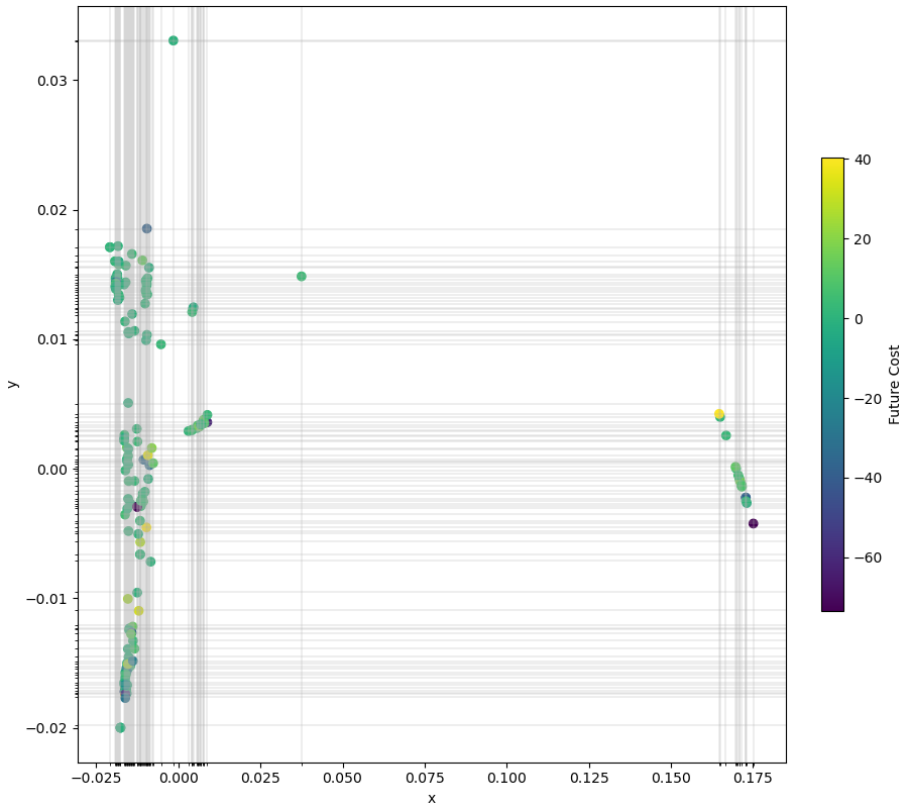
In this part we will perform a similar analysis of the second state representation: the states are a 2D representation of the RNA sequence of every node. We again run the experiment twice to see the difference per run. In figure 17 we show the resulting value function for every node (in which blue is a low value and yellow is a high value). The bins to which those nodes correspond are visualized by the grid covering the plot. We show the same for the cost function in figure 18, in which again nodes with cost higher than one are enlarged.

Looking at the plots of the two experiments we can see quite similar results, there are some small changes in colors for some nodes, but the general pattern is the same. For the cost function we see a change in one of the top left nodes, the remainder seems similar. If we compare it to the 1D case (figure 14 and 15), the result seems a bit off: the 1D case has more high cost states than the 2D case and the high cost states in the 1D case seem to be more centered in a cluster of low cost states, while the high cost states in the 2D case are located on the border and only in the left side of the plot. Furthermore the value function of the 2D plot seems to have more sparse minima and maxima, while in the 1D plot we can see a more gradual change.

In figure 19 we plot the control function learned from the covid data. Since this is a very large matrix, we first make it more compact in the same way as for the 1D case, but then comparing bins instead of nodes. As we can see in the plot, most states do not make transitions to states other than themselves. This happens because the system dynamics in the absence of control of the system make it very likely to stay in the same state (and thus the probability of a mutation is low). However we want the model to learn that it is desirable to make these mutations and thus change its probabilities despite the additional cost of deviating from the system dynamics in the absence of control. In table 6 we take a closer look at the few times the model does traverse to a state other than itself. This happens when the virus probability to stay in the same state is lower than the system dynamics in the absence of control and the virus probability to move to some states is higher than the system dynamics in

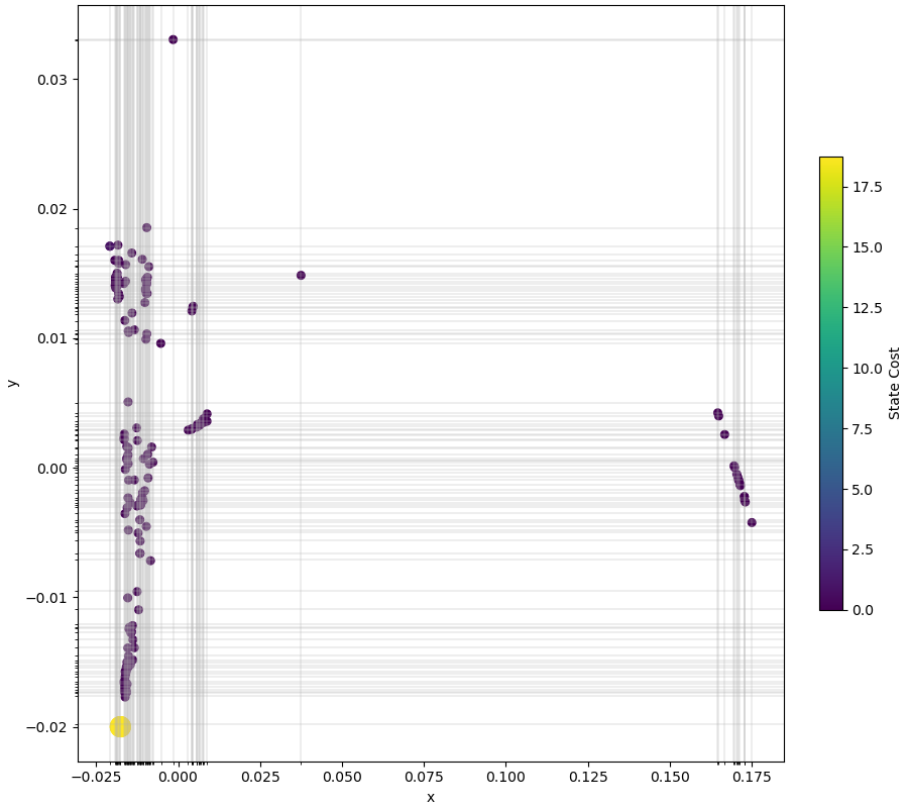


(a) Experiment 1.

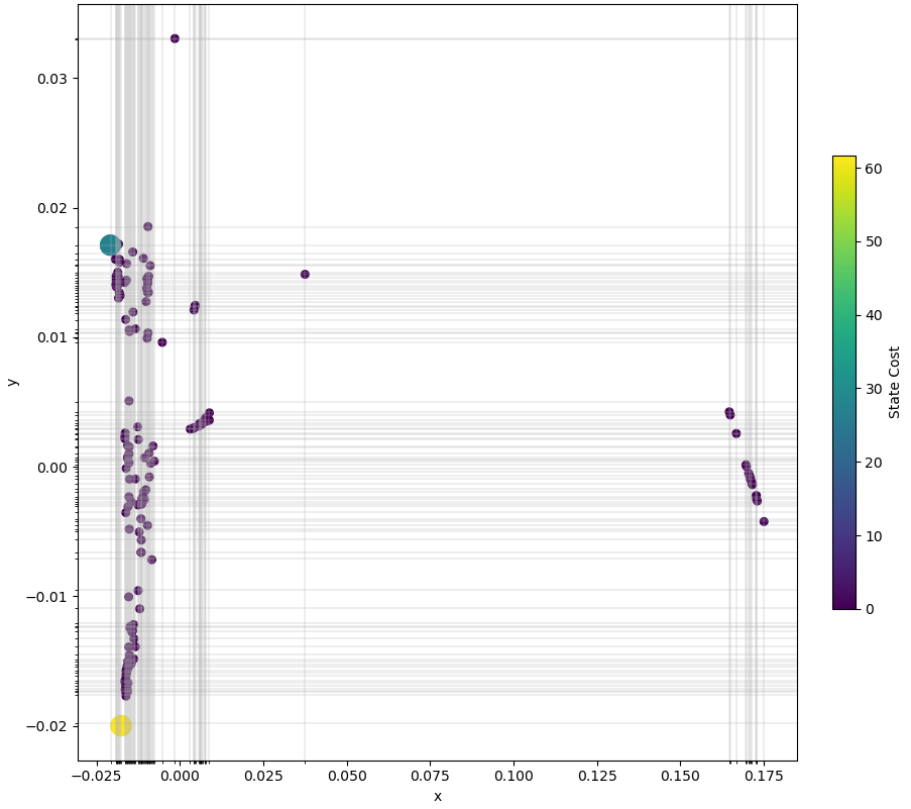


(b) Experiment 2.

Figure 17: Value Function

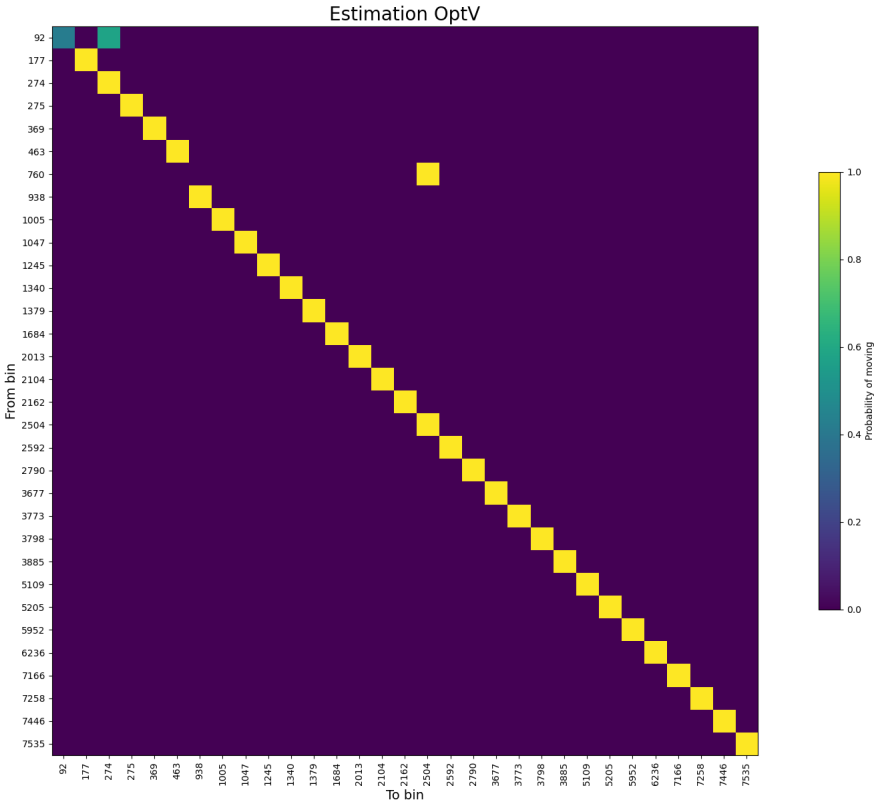


(a) Experiment 1.

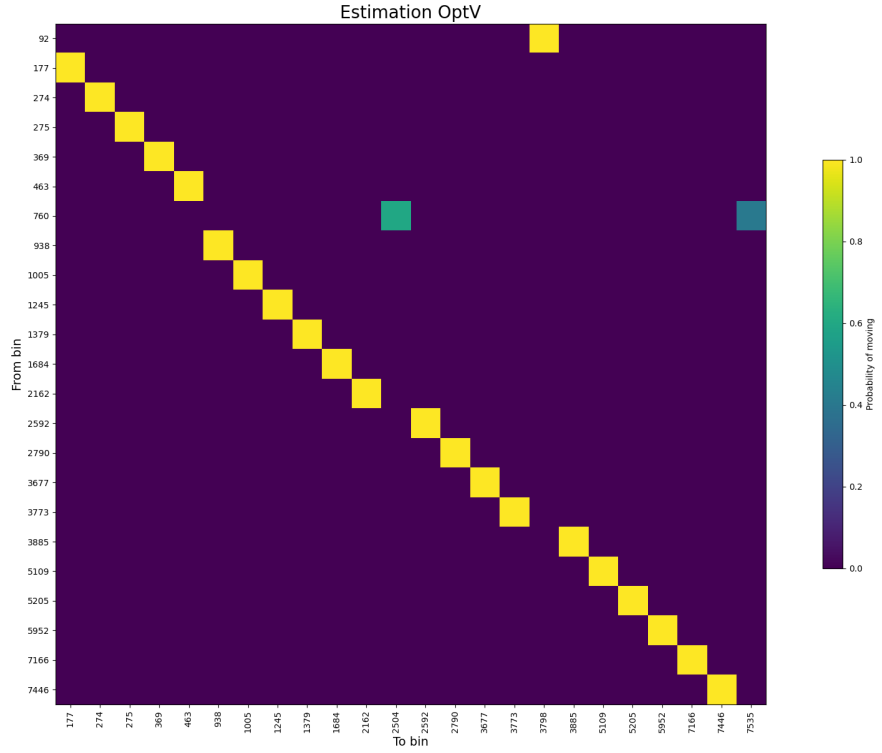


(b) Experiment 2.

Figure 18: Cost Function



(a) Experiment 1.



(b) Experiment 2.

Figure 19: Control functions 2D states SARS-CoV-2.

the absence of control. However since this only happens rarely we can conclude that this model does not perform well. We have tried with different methods of binning, including changing the binning method to uniform bins, changing the density of the bins in areas with more nodes and changing the number of bins. Unfortunately these changes did not improve the result. In some cases this made the performance even worse, by creating an optimal policy that always stayed in the same state.

From	To	Control u	p_virus	p_passive
92	274	0.5808	0.1060	8.42e-05
92	92	0.4188	0.4237	0.9995534
760	2504	0.9999	0.0102	8.4e-06
760	760	0.0	0.0646	0.9992753

(a) Experiment 1

From	To	Control u	p_virus	p_passive
92	3798	0.9999	0.0169	8.4e-06
92	92	0.0	0.4273	0.9995534
760	2504	0.5939	0.0102	8.4e-06
760	760	0.0	0.0646	0.9992753
760	7535	0.406	0.0102	8.4e-06

(b) Experiment 2

Table 6: Transitions in which the 2D control dynamics differ from the virus dynamics

Chapter 5

Discussion

From the results we can conclude that the 2D state representation was designed successfully from the full state representation. Most of the states maintain their relative local distances to each other and their overall correlation. However they can still be improved further. For the full state representation we come to the same conclusion. In future research we can finetune the state representation. Another option is to use inverse optimal control with adaptive bases (OptVA) instead of OptV [7]. This is a continuous variant of the OptV algorithm we used, where the feature vectors are represented by Gaussians. This algorithm optimizes the value function, while simultaneously optimizing the position and shape of the Gaussians.

From the results of the one dimensional model we can conclude that this model learns to prefer many of the states that the virus finds desirable. We conclude this based on the control function we presented in the results, in which the model deviated from the system dynamics in the absence of control when the virus dynamics are different for that state. In the positive part of the value function we were even able to recognize the pattern of the layers in the graph. The negative part of the value function fluctuates and shows the flexible possibilities for the virus to be successful. For future research it can be interesting to use a multiple rewards function, just like was done for the previous paper that applies IRL to cancer [9]. Besides the optimal control and value functions, we were also able to retrieve the underlying

cost function. The retrieved cost function consists of a large group of states with the same minimum cost and therefore does not give any extra information about which states are more desirable than others. It does tell us which ones are the least desirable: the intermediate states not seen in real life.

With the two dimensional representation we managed to apply OptV to the data, but unfortunately it was not able to learn from the data. We base this conclusion on the fact that the model showed a strong preference for the system dynamics in the absence of control. We expected imperfect results for this case, since the optimal way to deal with large feature representations is the continuous method described in the OptV paper [7]. In this method they use a linear combination of features with unknown weights as the value function. In this paper the weights are then optimized using a similar method to ours. Unfortunately we did not have enough time to implement this method and we leave this improvement to future research.

In this thesis we wanted to lay the groundwork for future prediction of possible strains of concern needed for the detection of viruses. If the designed feature vectors are used in the method described above, this might be possible. We believe that this is the case, because the method will not only predict the values of the current nodes (like in the 1D case), but also areas surrounding the known nodes. We picture this as follows: suppose we have a mutation of which we want to check if its potentially dangerous. Encode this mutation to a 2D vector, using DNABert the same way as before, and recover the value and cost function corresponding to that GridCell. If its value and cost function is very low, the mutation should be marked as dangerous.

In future research we also recommend to apply both methods to an even bigger dataset. In this thesis we used a graph with approximately 300 nodes, but we have already prepared a graph with 10000 nodes as well. This increase in data might give more interesting and complex results.

In our research we treated the data as static, since the differences in virus probabilities between states over time only changed slightly. It might as well be interesting to analyze the difference in performance of the model over time in more depth.

5.1 Conclusions

We successfully achieved our aims and objectives. We were able to create a fitting feature representation for genomic data. Moreover we successfully implemented OptV and applied the one dimensional state representation to SARS-CoV-2 data. For the two dimensional representation, this was not yet successful, but in the discussion we outline a method to achieve better results here as well. Lastly for the one dimensional case we extracted the cost function that the virus used to reproduce. This cost function can give us more insights in understanding the underlying reasons for which the virus chooses a certain path as the optimal path. A model like ours is especially useful for circumstances like these, when there exist a lot of states and possibilities, which make it difficult to discover patterns. This cost and value function might eventually help us to predict new strains of concern.

List of Figures

List of Tables

Bibliography

[1] Domingo, E. & Holland, J. J. Rna virus mutations and fitness for survival. *Annual Review of Microbiology* **51**, 151–178 (1997).

[2] Duffy, S. Why are rna virus mutation rates so damn high? *PLOS Biology* **16**, 1–6 (2018).

[3] Ostrowsky, J. *et al.* Tracking progress in universal influenza vaccine development. *Current Opinion in Virology* **40**, 28–36 (2020).

[4] Leroux-Roels, I. & Leroux-Roels, G. Current status and progress of prepandemic and pandemic influenza vaccine development. *Expert Review of Vaccines* **8**, 401–423 (2009).

[5] Todorov, E. Efficient computation of optimal actions. *Proceedings of the National Academy of Sciences* **106**, 11478–11483 (2009).

[6] Kappen, H. J., Gómez, V. & Opper, M. Optimal control as a graphical model inference problem. *Machine learning* **87**, 159–182 (2012).

[7] Dvijotham, K. & Todorov, E. Inverse optimal control with linearly-solvable mdps. 335–342 (2010).

[8] Cao, C. *et al.* The architecture of the sars-cov-2 rna genome inside virion. *Nature Communications* **12** (2021).

[9] Kalantari, J., Nelson, H. & Chia, N. The unreasonable effectiveness of inverse reinforcement learning in advancing cancer research. *Proceedings of the AAAI Conference on Artificial Intelligence* **34**, 437–445 (2020).

[10] Ab Azar, N., Shahmansoorian, A. & Davoudi, M. From inverse optimal control to inverse reinforcement learning: A historical review. *Annual Reviews in Control* **50**, 119–138 (2020).

[11] Hadfield, J. *et al.* Nextstrain: real-time tracking of pathogen evolution. *Bioinformatics* **34**, 4121–4123 (2018).

[12] Kliman, R. M. Reading a phylogenetic tree: The meaning of monophyletic groups (2009).

[13] Ji, Y., Zhou, Z., Liu, H. & Davuluri, R. V. DNABERT: pre-trained Bidirectional Encoder Representations from Transformers model for DNA-language in genome. *Bioinformatics* **37**, 2112–2120 (2021).

[14] Devlin, J., Chang, M., Lee, K. & Toutanova, K. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR* **abs/1810.04805** (2018).

[15] Moon, K. R. *et al.* Visualizing structure and transitions for biological data exploration. *Nature Biotechnology* **37**, 120378 (2019).

[16] Raffin, A. *et al.* S-RL Toolbox: Environments, Datasets and Evaluation Metrics for State Representation Learning. In *NeurIPS 2018 Workshop on "Deep Reinforcement Learning"* (2018).

[17] Szubert, B., Cole, J. E., Monaco, C. & Drozdov, I. Structure-preserving visualisation of high dimensional single-cell datasets. *Scientific Reports* **9** (2019).

[18] Zhang, Y., Shang, Q. & Zhang, G. pydrmetrics - a Python toolkit for dimensionality reduction quality assessment. *Heliyon* **7**, e06199 (2021).

[19] Todorov, E. Linearly-solvable markov decision problems. In *Advances in neural information processing systems*, 1369–1376 (2007).

[20] Joyce, J. M. *Kullback-Leibler Divergence*, 720–722 (Springer Berlin Heidelberg, Berlin, Heidelberg, 2011).

[21] Todorov, E. Compositionality of optimal control laws. In *Advances in Neural Information Processing Systems*, 1856–1864 (2009).

[22] Jonsson, A. & Gómez, V. Hierarchical linearly-solvable markov decision problems. In *26th International Conference on Automated Planning and Scheduling*, ICAPS'16, 193–201 (AAAI Press, 2016).

[23] Saxe, A. M., Earle, A. C. & Rosman, B. Hierarchy through composition with multitask LMDPs. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, 3017–3026 (JMLR. org, 2017).

[24] Williams, G., Aldrich, A. & Theodorou, E. A. Model predictive path integral control: From theory to parallel computation. *Journal of Guidance, Control, and Dynamics* **40**, 344–357 (2017).

[25] Gómez, V., Kappen, H. J., Peters, J. & Neumann, G. Policy search for path integral control. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 482–497 (Springer, 2014).

[26] Matsubara, T., Gómez, V. & Kappen, H. J. Latent Kullback Leibler control for continuous-state systems using probabilistic graphical models. *30th Conference on Uncertainty in Artificial Intelligence* (2014).

[27] Van Den Broek, B., Wiegerinck, W. & Kappen, B. Graphical model inference in optimal control of stochastic multi-agent systems. *Journal of Artificial Intelligence Research* **32**, 95–122 (2008).

[28] Gómez, V., Thijssen, S., Symington, A. C., Hailes, S. & Kappen, H. J. Real-time stochastic optimal control for multi-agent quadrotor systems. In *26th International Conference on Automated Planning and Scheduling* (2016).

[29] Wan, N., Gahlawat, A., Hovakimyan, N., Theodorou, E. A. & Voulgaris, P. G. Cooperative path integral control for stochastic multi-agent systems. *arXiv preprint arXiv:2009.14775* (2020).

[30] Chertkov, M., Chernyak, V. Y. & Deka, D. Ensemble control of cycling energy loads: Markov decision approach. In *Energy Markets and Responsive Grids*, 363–382 (Springer, 2018).

[31] Thalmeier, D., Gómez, V. & Kappen, H. J. Action selection in growing state spaces: control of network structure growth. *Journal of Physics A: Mathematical and Theoretical* **50**, 034006 (2016).

[32] Abbasi-Yadkori, Y., Bartlett, P., Chen, X. & Malek, A. Large-scale Markov decision problems with KL control cost and its application to crowdsourcing. In *International Conference on Machine Learning*, 1053–1062 (2015).

[33] Tabibian, B., Gómez, V., De, A., Schölkopf, B. & Gomez Rodriguez, M. On the design of consequential ranking algorithms. In *Proceedings of the 36th Conference on Uncertainty in Artificial Intelligence*, vol. 124, 171–180 (2020).

[34] Dennis, J. & Moré, J. Quasi-Newton Methods, Motivation and Theory. *SIAM Review* **19**, 46–89 (1977).