AdaLab: Technical document on simulator optimization

Alexandros Sarafianos

December 20, 2016

Chapter 1

Simulating growth curves for diauxic shift

1.1 Introduction

Growth as a biological process is influenced by both metabolic and regulatory effects. We propose a model, composed of two networks: a gene regulatory network and a metabolic network. These networks interact with eachother predicting growth curves given a gene knockout experiment. We will begin by explaining the two types of networks and how they are modelled individually before explaining the interaction between the two networks.

1.2 Gene Regulatory Network

1.2.1 Introduction

A gene regulatory network(GRN) is a network of regulators, such as genes, gene complexes and metabolites that interact. These interactions are symbolized with connections in the network and the network controls the gene expression levels. They play a role in the development and differentiation of organisms as well as in response to environmental changes.

1.2.2 Formal Description

The gene regulatory network is modelled as a directed graph $G(V, E)$ with parameters associated to the vertices V and edges E, respectively θ_v and θ_e. Furthermore every vertex starts with an (known) initial value $g_v[0]$ and additionally, an updating function $\mathbf{g_v}[t] = f_{update}(\mathbf{g_v}[t-1], \mathbf{exp}; \boldsymbol{\theta_e}, \boldsymbol{\theta_v})$ maps the vector of values of the vertices $\mathbf{g_v}[t-1]$ to their subsequent values $\mathbf{g_v}[t]$ for the next time step t given a certain biological experiment **exp**. By repeatedly applying the update function, we thus obtain a time series of values for every individual vertex.

1.2.3 Updating function

In the previous section we discussed at a high level that there is some function mapping the values of vertices to their values at the next time step. We want to learn a function f_{exp} that maps an experiment **exp** to a time series of node states or: $f_{exp} : \mathbf{exp} \rightarrow \mathbb{R}^{N \times T}$ where N is the amount of nodes in the network and T the amount of discrete timepoints for which the values have to be estimated and **exp** is a tuple corresponding to a biological experiment. To this purpose we first learn a function f_{update_gene} for each node i in the network that estimates the value of the next time step given the values of the current time. From previous studies[10] there is already information available about the structural aspects of the GRN for the diauxic shift. Based on this we defined the regression task as following:

$$g_v[t+1] = f_{update\_gene}(Pa(g_v[t]) \cup g_v[t]) \tag{1.1}$$

where g_v are genes, t is time, $Pa(\cdot)$ indicates the parents of a gene. We assume that the level of a gene at a certain time is dependent on the the influential genes from [10] and itself at the previous time. We also discretize time for this purpose and assume gene level updates are done in a synchronous manner.

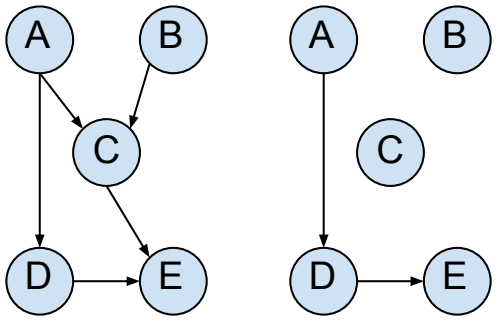
1.2.4 Modelling interventional experiments

Up till now, we only defined how to map gene levels from one time step to gene levels of the subsequent step. However we are interested in predicting the output for biological knockout experiments. To accomplish this these biological experiments are modelled as operations that can occur on vertices of the networks in the simulator. The first operation, also shown in figure 1.1, performs a knockout of a vertex. In a graph based context this corresponds to removing all edges, both incoming and outgoing, from the vertex that was knocked out[18] and this new configuration is then kept throughout the simulation. It is important to note that in a biological context this is a simplification of what is actually occurring as it cannot be guaranteed that the gene will not influence other genes (albeit with a small effect). An additional operation is adjusting the initial values based on the values for the input. In contrast with the previous operation, these values are not kept stationary in subsequent timesteps of a simulation, but they will be updated as specified in section 1.2.3.

1.2.5 Initial values for the network

For the initial values of the vertices, we took two different approaches. For the first approach we took two datasets[3, 4] that investigate gene expressions before, during and after the diauxic shift and used the normalized starting values as the initial values. The second approach consisted of a probabilistic/statistical approach. We extracted the gene expression levels for yeast under many different environments from [9]. Based on these values we estimated the marginal distributions of the genes by fitting them to different continuous probability distributions and selecting the fit that has the lowest Bayesian information criterion[1] value. Based on previous studies regarding the continuous distributions

[1] $BIC = -2 \ln L + k \ln(n)$ where L is the maximized value of the likelihood, n is the number of data points and k is the number of free parameters.



(a) Toy gene regulatory network for wild type i.e. without any knockouts

(b) Same toy gene regulatory network, but where gene C is knocked out

Figure 1.1: As an example of a gene knockout experiment, consider the toy gene regulatory network in 1.1a. When we knock out vertex C, it corresponds to cutting all incident and outgoing edges of C.

of genes [citation needed] only the Weibull, Gaussian and Gamma distributions were considered.

1.3 Metabolic Network

1.3.1 Introduction

A metabolic network consists of all cellular biochemical reactions catalyzed by enzymes as together they form interconnected metabolic pathways and as such a network. The vertices are called metabolites, which comprises all intermediates or products that in the metabolism of an organism.

1.3.2 Dynamic flux balance analysis

"In flux balance analysis, one constrains the metabolic network by the balance of metabolic fluxes around metabolites. When the metabolic network is operating in a steady state, the mass balance is described by a set of linear equations." [15] When the batch time is divided into several time intervals and the optimization problem is solved at the beginning of each time interval, one can get a dynamic solution for the metabolic fluxes. The results are then integrated over the intervals.

1.4 Interaction between GRN and Metabolic Network

Thus far the GRN and metabolic network have only been discussed separately from eachother, but it's possible to combine them. The GRN contains several nodes that are actually metabolites and therefore values obtained by the flux

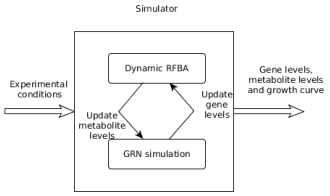


Figure 1.2: Illustration of the inputs, outputs, networks and dynamics of the simulator.

balance analysis can be fed to the GRN. Additionally, the metabolic network contains information about what genes are influencing what chemical reactions. Dependent on the value of those genes, the optimization problem can be redefined by adapting the relevant maximal fluxes. To map the values of the network from the ranges $[-1, 1]$ to $[0, 1]$, we feed the gene values to a parametrized logistic function. Figure 1.2 shows at a high level how the simulator is composed and what the inputs and outputs of the systems are. Additionally, pseudocode of the simulator is available at 0.

function SIMULATION (experiment, initialVertexValues, modelParameters)
Require: experiment, initialVertexValues, modelParameters
metabolicModel, geneRegulatoryModel ← initialize_networks(modelParameters)
metaboliteValues[0] ← initializeVertices(metabolicModel, geneRegulatoryModel, initialVertexValues)
adaptWithExperiment(metabolicModel, geneRegulatoryModel, experiment)
for time in 0:experiment.totalTime **do**
geneValues[time+1] ← grnSimulation(geneRegulatoryModel, metaboliteValues[time])
metaboliteValues[time + 1], growth[time + 1] ← metabolicSmulation(metabolicModel, geneValues[time+1])
end for
return growth, geneValues, metaboliteValues
end function

1.5 Probabilistic Estimates

As typically, there are different sources contributing to uncertainty. The simulator was extended to allow to express different aspects of this uncertainty: initial levels of vertices, model parameters, experimental values. Currently we only allow specification by means of a (continuous) uniform, Gaussian or Beta distribution, however we can easily extend this to other, both continuous as discrete, distributions. Additionally, a combination of deterministic and probabilistic information is still possible: for instance if we are certain that knocking out a gene completely disables the corresponding vertex, the vertex can be completely cut out. However if there could be a small non-zero value associated with that vertex, it might be more suitable to model it by means of a probability density function.

1.6 Learners

A predictor for equation 1.1 was obtained by training support vector regressor models with different kernels.

N is the amount of training data $x_k \in \mathbb{R}^m$ and output values $y_k \in \mathbb{R}$. ϵ is a hyperparameter related to the required accuracy and ξ_k and ξ_k^* are additional slack variables for the unfeasible case. C is a regularization paremeter controlling the trade-off between achieving a low error on the training data and minimizing the norm of the weights.

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \frac{1}{2}||w||^2 + C\sum_{k=1}^{N}(\xi_k + \xi_k^*) \\ \text{subject to} \quad & y_k - w^T\phi(x_k) - b \leq \epsilon + \xi_k \\ & w^T\phi(x_k) + b - y_k \leq \epsilon + \xi_k^* \\ & \xi_k, \xi_k^* \geq 0 \end{aligned}$$

http://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html γ is a free hyperparameter related to how far the influence of a single training example reaches.

$$K_{RBF}(x, x') = exp(-\gamma||x - x'||^2)$$

[16]

```
To learn the function described by equation \ref{eq:update}, we used support vector regres
C : float, optional (default=1.0)
    Penalty parameter C of the error term. The penalty is a squared l2 penalty. The bigger
epsilon : float, optional (default=0.1)

    Epsilon parameter in the epsilon-insensitive loss function. Note that the value of thi
The free parameters in the model are C and epsilon.

The implementation is based on libsvm.
    \cite{scikit-learn}
kernel : string, optional (default='rbf')

    Specifies the kernel type to be used in the algorithm. It must be one of 'linear', 'po
gamma : float, optional (default='auto')

    Kernel coefficient for 'rbf', 'poly' and 'sigmoid'. If gamma is 'auto' then 1/n_featur
"gamma" : [ 2**k for k in range(-10, 15) ]l\footnote{with hyperparameter values $C \in \{
$\epsilon = \{0,0.01,0.05,0.1,0.15,0.2\}$ and L1 loss function}
 hyperparameters were determined by 10-fold crossvalidation
```

1.7 Results

1.7.1 Results of estimates for gene regulatory network

Data

Available datasets can typically be divided in two types: time-course experiments to observe changes in expression levels over time and perturbation experiments to observe effects of a change or treatment on a cell. Since our interest with regard to the gene regulatory network lies on the temporal behaviour of gene regulation i.e. the dynamical behaviour, we only selected timeseries datasets from SGD[2]. We did not limit ourselves to datasets solely concerning diauxic shift as we assume that the genetic influences can be averaged over the different conditions. This resulted in the following subset: [4, 17, 7, 8, 13, 5]. Prior to regression tasks, the datasets were normalized (independently from eachother) such that all expression values are contained in the interval $[-1, 1]$.We obtain appropriate datasets by choosing a $t_{defined}$ and only selecting those datapoints for which we have data of $\Delta t = t_{defined}$. Datasets with a narrower time discretization are also used for the rougher discretizations, for instance when a dataset contains gene values every 30 minutes, we also subsample from this dataset to acquire values that are 60 minutes apart. The reverse however was not done: datasets with for instance a discretization of 60 minutes are not interpolated to acquire discretizations of 20 minutes. As the datasets are very diverse in their timescales, this results less available data per reaction. These time differences are expected to have a significant effect on the output. This resulted in datasets for which the summary can be found in table 1.1.

Δt(min)	Min	Max	Mean
10	37	94	82.14
15	58	111	101.91
20	7	53	44.42
30	86	170	153.04
60	70	136	121.32
120	22	52	46.68
180	11	31	27.95

Table 1.1: Summary of amount of datapoints available per reaction for different time differences. Reported are the minimum amount of data for a reaction, the maximum amount and the average amount.

1.8 Evaluation measures

To estimate the performance of our model we calculated different measures. We will now briefly discuss these. In the following equations, y denotes the true target output, $\hat{y}$ the output estimated by the model, $n_{samples}$ the amount of examples in the test set and $Var(\cdot)$ the variance.

[2]http://www.yeastgenome.org/download-data/expression

- Explained variance score(Expl. var.):

$$explained\_variance(y, \hat{y}) = 1 - \frac{Var\{y - \hat{y}\}}{Var\{y\}}$$

- Mean Absolute Error(MAE):

$$MAE(y, \hat{y}) = \frac{1}{n_{samples}} \sum_{i=0}^{n_{samples}-1} |y_i - \hat{y}_i|$$

- Mean Squared Error(MSE):

$$MSE(y, \hat{y}) = \frac{1}{n_{samples}} \sum_{i=0}^{n_{samples}-1} (y_i - \hat{y}_i)^2$$

- Median Absolute Error(MedAE):

$$MedAE(y, \hat{y}) = median(|y1 - \hat{y1}|, ..., |y_n - \hat{y}_n|)$$

- R^2 score, the coefficient of determination:

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=0}^{n_{samples}-1}(y_i - \hat{y}_i)^2}{\sum_{i=0}^{n_{samples}-1}(y_i - \hat{y}_i)^2}$$

In our case y corresponds to a gene level at a specific time point and

1.8.1 Results of estimates for gene regulatory network and discussion

We trained both a linear support vector regressor and one with an RBF kernel as specified in section 1.2.3. The results are summarized in table 1.2, which shows different qualitative measures for the different regressors and for the different time discretizations, but for more clarity only the best time discretizations are shown. After the hyperparameters were determined by 10-fold crossvalidation, the regressor was retrained with the whole training set. Reported results are on a separate testset, divided in two 80%-20% disjoint sets (respectively training and test set). The linear version is better than the RBF SVR, suggesting that a linear approximation is not necessarily a bad approximation. As a starting network for further optimization, we will therefore use the parameters obtained from the linear support vector regressor with the discretization of 30 minutes. Subsequent gene levels are calculated by the following formula:

$$g_v[t+1] = \sum_{j:g_j \in Pa(g_v) \cup \{g_i\}} \theta_{e,(i,j)} g_j[t] + \theta_{v,i} \tag{1.2}$$

Learner	Δt	Type	Min	Max	Mean	Stdev	Median
SVR_{Linear}	20 min	MAE	0.0159	0.2879	0.0867	0.0016	0.0779
		MSE	0.0003	0.1612	0.0158	0.0003	0.0099
		MedAE	0.0139	0.2435	0.0687	0.0014	0.061
		R^2	-20.5604	0.9725	0.2555	2.8958	0.6092
	30 min	MAE	0.0378	0.4082	0.0879	0.0012	0.0823
		MSE	0.0025	0.2411	0.0161	0.0003	0.0121
		MedAE	0.0217	0.3196	0.0683	0.0009	0.0627
		R^2	-1.6544	0.8871	0.4311	0.1062	0.4944
	60 min	MAE	0.027	0.2427	0.0971	0.0012	0.091
		MSE	0.0018	0.1156	0.0198	0.0003	0.0154
		MedAE	0.0217	0.1986	0.0715	0.0009	0.0656
		R^2	-1.1329	0.8619	0.3672	0.0997	0.4185
SVR_{RBF}	20 min	MAE	0.0157	0.241	0.0895	0.0014	0.0827
		MSE	0.0004	0.0907	0.0147	0.0002	0.0098
		MedAE	0.0058	0.2366	0.0761	0.0012	0.0716
		R^2	-129334.1088	0.9561	-397.5627	51468962.4212	0.6484
	30 min	MAE	0.04	0.2983	0.0851	0.001	0.0786
		MSE	0.003	0.1682	0.0145	0.0002	0.0106
		MedAE	0.0235	0.2722	0.0675	0.0008	0.0632
		R^2	-1.1632	0.8885	0.4661	0.0769	0.5155
	60 min	MAE	0.0459	0.262	0.0949	0.0009	0.0897
		MSE	0.0033	0.1309	0.0178	0.0002	0.0143
		MedAE	0.0272	0.2283	0.0737	0.0008	0.0687
		R^2	-3.792	0.8471	0.3839	0.1352	0.4388

Table 1.2: Summary of the results for the learned regressors for the updating function as described in section 1.2.3

1.8.2 Results of combined model

Chapter 2

Simulation-based optimization using gradient methods

2.1 Introduction

In chapter 1, we discussed what the simulator is composed of and how it predicts growth curves for gene knockout experiments. This chapter will handle how the simulator is used to improve the growth curve predictions of the initial model. In order to test the optimization algorithm, we identified different scenarios which we will discuss in the next sections.

2.2 Different scenarios

Recall from chapter 1 that the simulator actually consists of a gene regulatory network and a metabolic network. From the point of view of our application, we are primarily interested in learning the gene regulatory network. The different scenarios are related to what the actual output is and thus what can actually be used in the optimization problem. In every case the simulator takes as input a knockout experiment, which we will call **exp**. Additionally we will denote the number of vertices with N and the number of timesteps T.

2.2.1 Scenario 1: all gene values at every time step

In this case the output of the simulator exists of the values of every vertex in the graph at every timestep. This means the simulator is characterized by, $sim_1 : \mathbf{exp} \rightarrow \mathbb{R}^{N \times T}$. This corrresponds to the case where the GRN (only) contains genes and where an experiment is performed using microarrays.

2.2.2 Scenario 2: the values of a subset of gene at every time step

Compared to the first scenario, the only difference lies in the fact that instead of using all the vertices in the graph, we can only see a subset of them. This could correspond to learning a gene regulatory network from microarray experiments for which we cannot see all the concentrations, for instance because there are non-gene vertices. $sim_2 : \mathbf{exp} \to \mathbb{R}^{N_{sub} \times T}$, with $N_{sub} \subset N$.

2.2.3 Scenario 3: one value per time step obtained by applying some function on gene values

Instead of the values of the vertices being available at every time step, the values are now aggregated in a function that maps the values to a single value at every time step, $f : \mathbb{R}^N \to \mathbb{R}$. By doing so, the simulator creates a single time series instead of one per vertex in the network. The simulator is the transformed into: $sim_3 : \mathbf{exp} \to \mathbb{R}^T$. These functions, that operate as proxies for the FBA, were chosen from typical benchmark problems in optimization theory. In the following descriptions of the different functions, we will denote the output as $out[t]$, similar to before, g_i denotes the value of the i-th vertex and N the amount of vertices in the graph.

Summation

$$y[t] = \sum_{i=1}^{N} g_i[t] \tag{2.1}$$

Summations of squared values

$$y[t] = \sum_{i=1}^{N} g_i^2[t] \tag{2.2}$$

Rosenbrock function

$$y[t] = \sum_{i=1}^{N-1} [100(g_{i+1}[t] - g_i^2[t])^2 + (g_i[t] - 1)^2] \tag{2.3}$$

Styblinski–Tang function

$$y[t] = \sum_{i=1}^{n} g_i^4[t] - 16g_i^2[t] + 5g_i[t] \tag{2.4}$$

2.2.4 Scenario 4: one value per time step, obtained with FBA

The results of the metabolic genes are fed to the metabolic network by adjusting the corresponding flux rates and the resulting metabolite fluxes are passed to the metabolic vertices in the gene regulatory network as described in section 1.4.

2.3 Parameters

There are two different sets of parameters that can be optimized, those related to the GRN and those related to the metabolic network. In scenarios 1 to 3, only the GRN parameters are to be optimized, in scenario 4 both sets. A short description of these is provided in the next sections.

2.3.1 GRN parameters

Previously, we already discussed that the simulator's gene regulatory network has parameters associated with their vertices and edges. Given equation 1.2, we can define a matrix of parameters related to the adjacency matrix of the underlying network:

$$\mathbf{W} = \begin{bmatrix} \theta_{e,(1,1)} & \cdots & \theta_{e,(1,N)} \\ \vdots & \ddots & \vdots \\ \theta_{e(N,1} & \cdots & \theta_{e,(N,N)} \end{bmatrix} \tag{2.5}$$

and

$$\boldsymbol{\theta_v} = \begin{bmatrix} \theta_{v,1} \\ \vdots \\ \theta_{v,N} \end{bmatrix} \tag{2.6}$$

Where $\theta_{e,(i,j)}$ is non-zero if there is a directed edge from vertex i to j and the value being the influence that vertex i has on j and $\theta_{v,i}$ is a constant term indicating the behaviour on the level of a vertex i when there is no interaction (i.e. the incoming nodes have level 0) cfr. equation 1.2. This gives us the following vector of parameters:

$$\theta = \begin{bmatrix} \theta_{e,(1,1)} \\ \vdots \\ \theta_{e,(N,N)} \\ \theta_{v,1} \\ \vdots \\ \theta_{v,N} \end{bmatrix} \tag{2.7}$$

2.3.2 Metabolic network parameters

Additionally we parametrized how the gene regulatory network interacts with the metabolic network similar by adjusting the fluxes by incorporating gene expression levels as in [6]. As the gene regulatory network generates continuous values, we apply a logistic function to map these to the interval $[0, 1]$. In the equation of the logistic curve

$$f(x) = \frac{L}{1 + e^{-k(x-x0)}} \tag{2.8}$$

To allow for the correct range, we keep L fixed to 1. This however leaves other parameters that need to be set: k and $x0$, so for every gene that influences the

metabolic network there are 2 additional parameters. The metabolic network contains a total of 904 genes[1], whereas the gene regulatory network for diauxic shift contains 333 genes. Taking the intersection of those two sets yields a total of 146 genes The metabolism of yeast is well understood, at least better than the gene regulatory network[citation needed]. For this reason we primarily parametrize the gene regulatory network and how it interacts with the metabolic network and leave the metabolic network as is.

2.4 Optimization problem

2.4.1 Introduction

In this section we will discuss several aspects of the optimization problem. This includes the optimization algorithm, adjustments of the standard algorithms, tunable parameters and objective functions.

2.4.2 Gradient descent

Gradient descent is an optimization method that tries to optimize an objective function by iteratively updating the parameters of its model. In gradient descent the parameter vector is updated iteratively by calculating the gradient at a certain point and updating the weights in the direction of the negative gradient as this is the direction in which the function is declining. In regular gradient descent the calculation of the gradient is done on the whole training set. Given a multi-variable function and parameter vector $\mathbf{x}$ $F(\mathbf{x})$ and if it is defined and differentiable in a certain point $\mathbf{a}$, and given a certain learning rate γ the new point $\mathbf{b}$ is calculated as follows.

$$\mathbf{b} = \mathbf{a} - \gamma \nabla F(a) \tag{2.9}$$

2.4.3 Stochastic Gradient Descent

Stochastic Gradient Descent(SGD) is a simplification of the Gradient Descent(GD) algorithm. Whereas GD updates the parameter vector by calculating the gradient for all training examples, SGD estimates the gradient based on a single randomly picked example, resulting in a noisier approximation of the gradient.

2.4.4 Objective Function

Recall that equation 2.9 requires us to specify some function $F(\mathbf{x})$ whose value will be minimized. For this work, we employed the Mean Squared Error. As the outputs for the different scenerios are not the same, there are some differences in the equations which we will quickly discuss.

Scenario 1

$$F_1(\mathbf{x}) = \frac{1}{N \cdot T} \sum_{i=1}^{N} \sum_{t=1}^{T} (\hat{y}_i[t] - y_i[t])^2 \tag{2.10}$$

[1] In the metabolic model iMM904

where N is the total amount of values, which is the amount of genes multiplied with the amount of time steps, i iterates over all these values, $\hat{y}_i$ is the value estimated by the model and y_i is the actual value.

Scenario 2

$$F_2(\mathbf{x}) = \frac{1}{N_{sub} \cdot T} \sum_{i=1}^{N_{sub}} \sum_{t=1}^{T} (\hat{y}_i[t] - y_i[t])^2 \tag{2.11}$$

In this case only a subset $N_{sub} \subset N$ of the vertices is used .

Scenarios 3 & 4

$$F_{3,4}(\mathbf{x}) = \frac{1}{T} \sum_{t=1}^{T} (\hat{y}[t] - y[t])^2 \tag{2.12}$$

with $y[t]$ and $\hat{y}[t]$ denote respectively the actual and estimated output of the aggregate function or of the growth curve obtained from FBA at time t.

2.4.5 Gradient of objective function

In fact, it's not the objective function itself which is needed for the gradient descent algorithm, but rather the gradient of the objective function. If analytical solutions of the derivates are available, it is better to apply them directly. In some cases however it is not possible to obtain these analytical solutions. In our application this would involve obtaining a closed-form solution to the optimization problem defined by flux balance analysis, which at the very least is not straightforward. The simplest way to numerically approximate the gradient is the technique of finite differences. The approximation consists of obtaining the values for the objective function where a *small* number is added and subtracted for every parameter separately and then dividing the difference of them by two times the *small* number. This approximation is however highly numerically inefficient as this requires an amount of function evaluations equal to two times the size of the parameter vector. On the other hand, in simultaneous perturbation stochastic approximation(SPSA)[11] the gradient is estimated by stochastically choosing a direction in which to perturb the operating point to calculate a finite difference gradient.

$$\nabla \hat{f}(c)[d] = \frac{f(x_k + \delta_k \Delta_k[d] e_d) - f(x_k - \delta_k \Delta_k[d] e_d)}{2\delta_k \Delta_k[d]} \tag{2.13}$$

"where e_d is a D-component vector with 1 for the dth component and 0 for all other components, δ_k is a scalar that generally shrinks as the number of iterations k increases, and Δ_k is a random D-component perturbation vector." This approach only requires 2 evaluations of f to calculate a gradient.

2.4.6 Learning rates

It is possible to keep the learning rate γ from equation 2.9 fixed throughout the different iterations, however previous studies have shown that improved convergence can be obtained by allowing γ to be altered in subsequent iterations [12]. These techniques are however still sensitive to the initial choice of the

Algorithm 1 Computing ADADELTA update at time t

Require: Decay rate ρ, Constant ϵ
Require: Initial parameter x_1
Initialize accumulation variables $E[g^2]_0 = 0$, $E[\Delta x^2]_0 = 0$
for $t = 1 : T$ **do** %% Loop over # of updates
  Compute Gradient: g_t
  Accumulate Gradient: $E[g^2]_t = \rho E[g^2]_{t-1} + (1-\rho)g_t^2$
  Compute Update: $\Delta x_t = -\frac{\text{RMS}[\Delta x]_{t-1}}{\text{RMS}[g]_t} g_t$
  Accumulate Updates: $E[\Delta x^2]_t = \rho E[\Delta x^2]_{t-1} + (1-\rho)\Delta x_t^2$
  Apply Update: $x_{t+1} = x_t + \Delta x_t$
end for

Figure 2.1: An overview of the AdaDelta algorithm.

learning rate. A (relatively new) technique called AdaDelta[19] avoids choosing and having to manually tune a learning rate altogether and good results for this adaptive schemes have been reported. An overview of the revised algorithm is shown in figure 2.1.

2.4.7 Stopping mechanism

To avoid overfitting an early stopping mechanism is applied on a whithheld validation set[2]. If the performance does not improve for a preset amount of iterations, the algorithm is terminated and returns the best parameter vector before those 50 iterations. Apart from that strategy, some other methods were also implemented in order to save on computation time:

1. The maximum amount of epochs is parametrized. When the optimizer reaches this amount, it stops and returns the currently optimal value.

2. We also calculate the errors between epochs. When the average absolute error is lower than a preset[3] tolerance level in the last preset amount of iterations, the optimizer terminates as well and returns the last optimal value.

2.5 Experiments

2.6 Introduction

This section will cover some aspects about the experiments that we perform with the optimization algorithm such as how the the datasets were generated or obtained, how we evaluate the results.

2.6.1 Datasets

Toy datasets

We considered two types of networks, complete digraphs[4] and scalefree networks [5]. We have opted for this kind of networks as a lot of real-life networks can be

[2] Training set, validation set and test set are divided in respectively 60%-20%-60%.

[3] 0.00001

[4] "A complete digraph is a directed graph in which every pair of distinct vertices is connected by a pair of unique edges (one in each direction)."

[5] "A network whose degree distribution follows a power law, at least asymptotically. That is, the fraction P(k) of nodes in the network having k connections to other nodes goes for

characterized as being scalefree. Examples of such graphs include biological networks such as gene regulatory networks and protein interaction networks, social networks and more [1] The parameters related to the vertices and edges are randomly sampled from the normal distribution with mean 0 and standard deviation 1. Given this network we then generated a list of all possible knockout experiments(input to the system), and calculated the outputs with this network and the simulator, adding different levels of noise. These toy datasets were then used for scenarios 1, 2 and 3. Remark: There has been some discussion about the usage of scalefree networks [14] for biological networks.

Eve Growth curves

Currently not available.. Needed for scenario 4.

2.6.2 Evaluation

To evaluate the output of the parameter estimation algorithm we calculate the mean squared error on a separate testset (see Section 2.4.4) and the Euclidian distance of the optimal parameter vector to the real parameter vector (see Section 2.6.2). When using the output of the optimization algorithm as samples we use the kolmogorov-smirnov statistic rather than the Euclidian distance. Every set of algorithmic parameters is repeated 10 times with different randomized networks. The length of the time series is kept at 10.

2.6.3 Implementation

The implementation of the optimization algorithms can be found at `AdaLab/src/optimization` of the evaluation code in `adalab/src/executables`. Additionally, some parts of the implementations are exposed to R. These exposed classes and methods are found in the file `AdaLab/src/R_interface_optimization.cpp`.

2.6.4 Discussion

The results showed non-sparse networks and thresholding the learned values did not show a correlation with the true edges. As this is highly unlikely, we changed the approach in order to learn more sparse structures and obtain a distribution over networks.

large values of k as $P(k) \sim k^{-\gamma}$ where γ is a parameter whose value is typically in the range $2 < \gamma < 3$ [2], although occasionally it may lie outside these bounds"

Chapter 3

Simulation-based optimization using sampling methods

3.1 Introduction

In order to obtain sparse network structures and a distribution over possible networks, we changed the approach to a sampling based optimization algorithm as this allows for straightforward specification of priors.

3.2 Algorithm

Implementation is located at `AdaLab/MetabolicModel/R/find_grn`. The algorithm is outlined at a high level in Algorithm **??**. (1) in the algorithm is implemented as a random walk in the structure space and (2) as a random walk in parameter space.

3.3 Structure prior

Currently a structure prior based on the Zimmer network is used. Edges that are present in Zimmer are interpreted as being more likely to be present in the actual graph ($p = 0.9$), whereas edges not present in Zimmer are deemed unlikely to be present in the actual graph ($p = 0.9$)

3.4 Parameter priors

As not much is known about the values of the priors, we assume that they should be (relatively) small non-zero values and hence we assign them a prior gaussian with mean 0 and standard deviation 1.

Algorithm 1 High-level outline of the algorithm to find the grn

function FIND_GRN(maxStructIterations, maxParamIterations, exampleDatabase)
Require: A posterior distribution for the network $\pi_{network}$
Require: A posterior distribution for the parameters π_{params}
Require: A proposal distribution for the structure q_{struct}
Require: A proposal distribution for the parameters q_{params}
1)Initialize network structure θ^0_{struct} and parameters θ^0_{param}
2)structIteration $\leftarrow 0$
3)Calculate the network posterior $\pi_{network}(\theta^{[structIteration]}_{struct}, \theta^{[structIteration]}_{param})$
4)Generate a candidate for the structure (1)$\theta^*_{struct} \leftarrow q_{struct}(\theta^{[structIteration]}_{struct})$
while s **do**tructIteration $<$ maxStructIterations
5)paramIteration $\leftarrow 0$
while p **do**aramIteration $<$ maxParamIterations
6)Calculate the parameter posterior $\pi_{param}(\theta^{[paramIteration]}_{param}$ for θ^*_{struct}
7)Draw a candidate for the parameters
(2)$\theta^*_{param} \leftarrow q_{param}(\theta^{[paramIteration]}_{param})$
8)Calculate the posterior of the candidate π^*_{param}
9)$\theta^{[paramIteration+1]}_{param} \leftarrow \theta^*_{param}$
with probability $min(a(\theta^*_{param}, \theta^{[paramIteration]}_{param}), 1$
$\theta^{[paramIteration+1]}_{param} \leftarrow \theta^{[paramIteration]}_{param}$
with probability $1 - min(a(\theta^*_{param}, \theta^{[paramIteration]}_{param}), 1$
10) paramIteration++
end while

$\theta^{opt}_{param} \leftarrow \theta^{maxParamIterations}_{param}$
11)Calculate the candidate network posterior
$\pi_{network}(\theta^{[structIteration]}_{struct}, \theta^{opt}_{param})$
12)$\theta^{[structIteration+1]}_{struct} \leftarrow \theta^*_{struct}$
with probability $min(a(\theta_{param}struct^*, \theta^{[structIteration]}_{struct}), 1$
$\theta^{[structIteration+1]}_{struct} \leftarrow \theta^{[structIteration]}_{struct}$
with probability $1 - min(a(\theta^*_{struct}, \theta^{[structIteration]}_{struct}), 1$
13)structIteration++
end while
return $\theta^{maxStructIterations}_{struct}$ and θ^{opt}_{param}
end function

3.5 Implementation

The implementation of the algorithm can be found in `AdaLab/src/optimization/`, `AdaLab/MetabolicModel/R/find_grn`, `AdaLab/MetabolicModel/R/optim.R` and in `AdaLab/src/R_interface_optimization.cpp`.

3.6 Synthetic data creation

Creating a random network structure

The random creation of a network structure is achieved by the function get_random_network at `AdaLab/MetabolicModel/R/simulation.R`. It generates random scalefree networks[1] that contain no parallel edges and self loops.

Generating random parameters for the network structure

This is achieved by the function random_parameters at `AdaLab/MetabolicModel/R/simulation.R`.

Random inputs

Achieved by generate_examples in `AdaLab/MetabolicModel/R/simulation.R`

3.7 Results

Results are omitted as there is still a bug present in the metabolic part.

[1] By default with $\gamma = 2.5$

Bibliography

[1] Albert-László Barabási. Scale-free networks: a decade and beyond. *science*, 325(5939):412–413, 2009.

[2] Albert-Laszlo Barabasi and Zoltan N Oltvai. Network biology: understanding the cell's functional organization. *Nature reviews genetics*, 5(2):101–113, 2004.

[3] Matthew J Brauer, Curtis Huttenhower, Edoardo M Airoldi, Rachel Rosenstein, John C Matese, David Gresham, Viktor M Boer, Olga G Troyanskaya, and David Botstein. Coordination of growth rate, cell cycle, stress response, and metabolic activity in yeast. *Molecular biology of the cell*, 19(1):352–367, 2008.

[4] Matthew J Brauer, Alok J Saldanha, Kara Dolinski, and David Botstein. Homeostatic adjustment and metabolic remodeling in glucose-limited yeast cultures. *Molecular biology of the cell*, 16(5):2503–2517, 2005.

[5] Orna Carmel-Harel, Robert Stearman, Audrey P Gasch, David Botstein, Patrick O Brown, and Gisela Storz. Role of thioredoxin reductase in the yap1p-dependent response to oxidative stress in saccharomyces cerevisiae. *Molecular microbiology*, 39(3):595–605, 2001.

[6] Sriram Chandrasekaran and Nathan D Price. *Proceedings of the National Academy of Sciences*, 107(41):17845–17850, 2010.

[7] Raymond J Cho, Michael J Campbell, Elizabeth A Winzeler, Lars Steinmetz, Andrew Conway, Lisa Wodicka, Tyra G Wolfsberg, Andrei E Gabrielian, David Landsman, David J Lockhart, et al. A genome-wide transcriptional analysis of the mitotic cell cycle. *Molecular cell*, 2(1):65–73, 1998.

[8] Joseph L DeRisi, Vishwanath R Iyer, and Patrick O Brown. Exploring the metabolic and genetic control of gene expression on a genomic scale. *Science*, 278(5338):680–686, 1997.

[9] Jeremiah J Faith, Michael E Driscoll, Vincent A Fusaro, Elissa J Cosgrove, Boris Hayete, Frank S Juhn, Stephen J Schneider, and Timothy S Gardner. Many microbe microarrays database: uniformly normalized affymetrix compendia with structured experimental metadata. *Nucleic acids research*, 36(suppl 1):D866–D870, 2008.

[10] Ludwig Geistlinger, Gergely Csaba, Simon Dirmeier, Robert Küffner, and Ralf Zimmer. A comprehensive gene regulatory network for the diauxic shift in saccharomyces cerevisiae. *Nucleic acids research*, 41(18):8452–8463, 2013.

[11] Megan Hazen and Maya R Gupta. Gradient estimation in global optimization algorithms. In *2009 IEEE Congress on Evolutionary Computation*, pages 1841–1848. IEEE, 2009.

[12] Robert A Jacobs. Increased rates of convergence through learning rate adaptation. *Neural networks*, 1(4):295–307, 1988.

[13] SE Lee, A Pellicioli, J Demeter, MP Vaze, AP Gasch, A Malkova, PO Brown, D Botstein, T Stearns, M Foiani, et al. Arrest, adaptation, and recovery following a chromosome double-strand break in saccharomyces cerevisiae. In *Cold Spring Harbor symposia on quantitative biology*, volume 65, pages 303–314. Cold Spring Harbor Laboratory Press, 2000.

[14] Gipsi Lima-Mendez and Jacques van Helden. The powerful law of the power law and other myths in network biology. *Mol. BioSyst.*, 5:1482–1493, 2009.

[15] Radhakrishnan Mahadevan, Jeremy S Edwards, and Francis J Doyle. Dynamic flux balance analysis of diauxic growth in escherichia coli. *Biophysical journal*, 83(3):1331–1340, 2002.

[16] Alex J. Smola and Bernhard Schölkopf. A tutorial on support vector regression. *Statistics and Computing*, 14(3):199–222, 2004.

[17] Paul T Spellman, Gavin Sherlock, Michael Q Zhang, Vishwanath R Iyer, Kirk Anders, Michael B Eisen, Patrick O Brown, David Botstein, and Bruce Futcher. Comprehensive identification of cell cycle–regulated genes of the yeast saccharomyces cerevisiae by microarray hybridization. *Molecular biology of the cell*, 9(12):3273–3297, 1998.

[18] SM Minhaz Ud-Dean and Rudiyanto Gunawan. Optimal design of gene knock-out experiments for gene regulatory network inference. *Bioinformatics*, 32(6):875–83, March 15 2015.

[19] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.