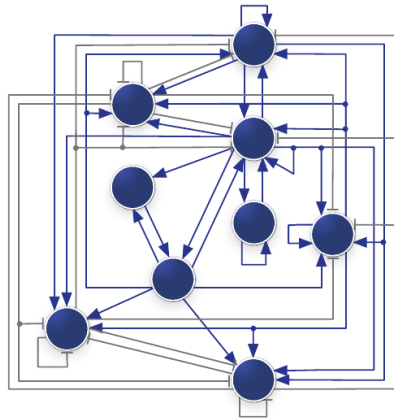


Section Microtechnique

Master Thesis



Scalable Reverse Engineering of Nonlinear Gene Networks

by Thomas Schaffter

Professor Dario Floreano, EPFL
Advisors Daniel Marbach, EPFL
 Claudio Mattiussi, EPFL
Expert Sven Bergmann, UNIL

Laboratory of Intelligent Systems

Lausanne, January 18th, 2008

PROJET DE MASTER

Titre: Scalable Reverse Engineering of Nonlinear Gene Networks

Candidat(s): Thomas Schaffter (MT)

Professeur: Dario Floreano

Assistant 1: Daniel Marbach

Assistant 2: Claudio Mattiussi

Donnée & travail demandé:

The effective reverse engineering of gene regulatory networks is one of the great challenges of systems biology and is expected to have substantial impact on the pharmaceutical and biotech industries in the next decades. A gene network is formed by regulatory genes, which code for proteins that enhance or inhibit the expression of other regulatory and/or non-regulatory genes, thereby forming a complex web of interactions. The goal of reverse engineering is to automatically identify such a network from experimental data.

In this project, we explore an evolutionary reverse engineering approach for gene networks of medium size. As benchmark we use a dataset of a fifty-gene in silico network, which has been published as an international reverse engineering competition for the second DREAM conference (Dialogue for Reverse Engineering Assessments and Methods). Our goal is to provide an accurate reconstruction of the gene network using biologically plausible dynamical models. With this kind of nonlinear models, traditional methods of system identification are generally not applicable. Hence, the reverse engineering algorithm must rely on stochastic global search methods and a "divide-and-conquer" strategy to achieve scalability.

Remarques:

Un plan de travail (Gantt chart) sera établi et présenté à l'assistant responsable avant la fin de la deuxième semaine.

Une présentation intermédiaire (10 minutes de présentation et 10 minutes de discussion) de votre travail aura lieu le 24 octobre 2007 à partir de 13 heures. Elle a pour objectifs de donner un rapide résumé du travail déjà effectué, de proposer un plan précis pour la suite du projet et d'en discuter les options principales.

Un rapport, comprenant en son début l'énoncé du travail (présent document), suivi d'un résumé d'une page, devra être rédigé. La page de résumé (A4, seulement recto) doit comporter, en plus de la description du projet et de 1 ou 2 figures représentatives, la date, le nom du laboratoire, le titre du projet, son type (projet de master), vos noms et prénoms ainsi que ceux du professeur responsable et des assistants. Dans le rapport, l'accent sera mis sur la description des expériences et la présentation des résultats obtenus. Une version préliminaire du rapport sera remise à l'assistant au plus tard 10 jours avant la date de rendu final. La version finale sera remise au secrétariat de votre section le 18 janvier 2008 avant 12 heures en 3 exemplaires signés et datés.

Une défense de 30 minutes (environ 20 minutes de présentation et démonstration, plus 10 minutes de réponses aux questions) aura lieu dans la période du 4 au 15 février 2008. Vous serez jugé sur les résultats de votre projet tels qu'ils seront présentés dans votre rapport et lors de votre défense.

Tous les documents en version informatique, y compris le rapport (en version source et en version PDF), la page de résumé au format PDF et le document de la présentation orale, ainsi que les sources des différents programmes doivent être gravés sur un CD-ROM et remis à l'assistant au plus tard lors de la défense finale.

Le professeur responsable:

L'assistant responsable:

Signature:

Dario Floreano

Signature:

Daniel Marbach

Lausanne, le 19 septembre 2007

Scalable Reverse Engineering of Nonlinear Gene Networks

Thomas Schaffter, Section Microtechnique

Advisors: Daniel Marbach, Claudio Mattiussi

Professor: Dario Floreano

Introduction

Current advanced molecular biology techniques provide gene expression levels (mRNA levels) of selected, if not all, genes of an organism. In such aggregate, expression levels of one gene are mediated by the presence of specific proteins and metabolites produced by other genes. Gene Regulatory Networks include such structures, where a node (gene) is linked by a few number of interactions with others genes (Fig. 1). Knowledge of gene-gene relationships in networks, and so having possibility to act and control them is important in biotech and pharmaceutical industries.

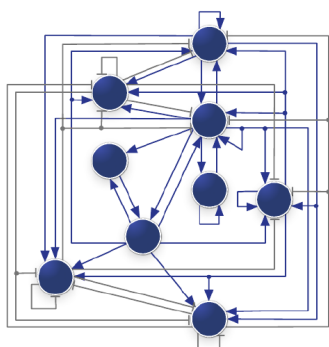


Figure 1 Gene Regulatory network.

Methodology

From mRNA measurements methods based on Bayesian Networks, statistics and information theory-based or linear dynamical systems offer the possibility to reverse-engineer underlying GRNs, i.e. to find the topology or wiring of such networks. The method described here is based on single-gene optimization, which allows to process large scale non-linear GRNs (> 50 genes).

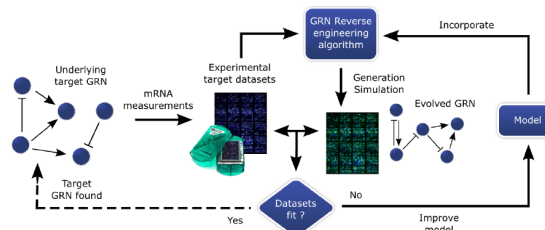


Figure 2 Framework of single-gene optimization-based reverse engineering algorithm.

From experimental datasets, the algorithm generates several different in silico GRNs and simulates them to produce datasets comparable to given ones. This loop is repeated many times until the algorithm finds an in silico network that fits the experimental data (Fig. 2). CMA-ES is used as Evolutionary Algorithm, which is faster than classical Genetic Algorithms.

Results

Efficiency of the algorithm is assessed on an in silico 5 gene network with 6 gene-gene interactions. The present process recovers successfully all connections with correct type and detects only one false positive. Concerning computation time, one say which one needs 6-8 hours to infer this entire network, where present one needs only about one minute in the best cases to do this.

Future work

The next stage is to evaluate this algorithm on a gold standard: the 50 gene network used as challenge for the second DREAM conference. Using Precision-Recall and ROC curves, obtained results can be directly compared with published ones of participating teams.



Contents

1	Introduction	1
2	Scalable Reverse Engineering of Nonlinear Gene Networks	5
2.1	Preface	5
2.2	Definitions and Problem to solve	7
2.3	Reverse Engineering Algorithm	8
2.3.1	Overview	8
2.3.2	Gene Model	9
2.3.3	Algorithm	10
2.4	Theory of CMA-ES	13
2.4.1	Generalities	13
2.4.2	The Multivariate Normal Distribution	13
2.4.3	Initialization	14
2.4.4	Sampling individuals	14
2.4.5	Selection and Recombination	15
2.4.6	Adaptation the Covariance Matrix	15
2.4.7	Limited memory version of CMA-ES, L-CMA-ES	15
2.5	Apply CMA-ES to Reverse Engineering Algorithm	16
2.5.1	Implementation of CMA-ES used	16
2.5.2	Individuals	16
2.5.3	Bound Evolvable Parameters	16
2.5.4	Evaluation of Evolved Genes	17
2.5.5	Fitness and Prediction Error	18
3	Efficiency of the Algorithm on a Five Gene Network	21
3.1	Goal	21
3.2	In Silico Target Network	21
3.3	In Silico Target Datasets	22
3.4	Calibration of CMA-ES parameters	23
3.4.1	Procedure	23
3.4.2	Evaluation of Population size	23
3.4.3	Evaluation of Initial step-length	25
3.4.4	Evaluation of Recombination strategies	28
3.4.5	Summary of best CMA-ES parameters found	29

3.5	Results	30
3.5.1	Raw data	30
3.5.2	Weights threshold	35
3.5.3	Single-gene optimization of G0	36
3.5.4	Single-gene optimization of G1	36
3.5.5	Single-gene optimization of G2	38
3.5.6	Single-gene optimizations of G3 and G4	39
4	Conclusions	41
5	Acknowledgements	43
A	Wyrd Handbook	49
A.1	Installation	49
A.1.1	Requirements	49
A.1.2	Compilation from sources	49
A.2	Introduction	50
A.3	Quickstart	50
A.3.1	User Interface	50
A.3.2	Project Id and filenames convention	51
A.4	Generation of Gene Regulatory Networks	52
A.4.1	Preface	52
A.4.2	Broken repressilator and Grntest	52
A.4.3	Random networks	53
A.4.4	XML structure for Wyrd GRNs files	55
A.4.5	Initial mRNA concentrations for Time Series experiments	55
A.5	Simulation of GRNs	56
A.5.1	Introduction	56
A.5.2	Time Series datasets	57
A.5.3	Steady-States datasets	60
A.6	Reverse Engineering of GRNs	64
A.6.1	Introduction	64
A.6.2	Prepare inputs	64
A.6.3	Initialization of parameters	65
A.6.4	Run GRNs Reverse Engineering process	67
A.6.5	Management of CMA-ES	67
A.6.6	Results and output files	70
A.7	Visualization of Wyrd results	71
A.8	Doxygen Documentation	71
B	WyrdReport Handbook	73
B.1	Introduction	73
B.2	Quickstart	73
B.2.1	WyrdReport GUI - wrgui	73
B.2.2	HTML reports	76
B.2.3	Advices	78
B.3	Using Matlab tools in command lines	79
B.3.1	Instance of wtInitCore	79

B.3.2	WyrdReport	80
B.3.3	Others WyrdReport scripts	80
B.4	Precision-Recall and ROC Matlab scripts	80
C	Wyrd Settings file	85
D	In silico Target Datasets of Grntest	89
D.1	grntest_ss_wt.csv	89
D.2	grntest_ss_ko.csv	89
D.3	grntest_ss_hz.csv	89
D.4	grntest_ts.csv	89
E	Wyrd Class Dependencies	93
F	Matlab Scripts Documentation	95
F.1	bestRun	97
F.2	computePR	98
F.3	confidence	99
F.4	get	101
F.5	printDistances	102
F.6	printGrnTopology	103
F.7	printSquareError	104
F.8	printTimeSeries	105
F.9	resizeProgressBar	106
F.10	set	107
F.11	wrInitCore	108
F.12	writeGrnPart	109
F.13	writeIndex	110
F.14	writeMenu	111
F.15	writePredictions	112
F.16	writeSsPart	114
F.17	writeTsPart	115
F.18	wyrdReport	116

1

Introduction

Current advanced molecular biology techniques provide gene expression levels (mRNA levels) of selected, if not all, genes of an organism. In such aggregate, expression levels of one gene are mediated by the presence of specific proteins and metabolites produced by other genes [1]. Therefore, mRNA levels of most of the genes are directly or indirectly affected by some other genes which are in turn regulated by some other ones. Biological systems of such structure are gene regulatory networks where a node (gene) is linked by a few number of connections (excitatory or inhibitory gene-gene interaction) to others genes (Figure 1.1). Knowledge of gene-gene relationships in networks, and so having possibility to act and control them is important in biotech and pharmaceutical industries.

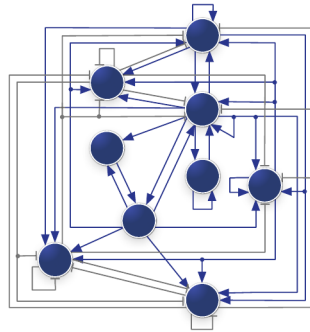


Figure 1.1 In gene regulatory networks, nodes represent genes and directed connections the excitatory or inhibitory gene-gene interactions.

From mRNA measurements, reverse engineering algorithms offer the possibility to infer the underlying gene regulatory networks, i.e. to found the topology or wiring of such networks. Algorithm may differ by several ways as the required experimental data or still the framework of the process used. One of the first methods to solve the present problem is the use of Bayesian Networks (BN). The key features of BN remain in a graphical compact and intuitive representation of a probability distribution, captures of causal relationships between genes, deals with noisy data and integration of prior knowledge as sparseness of biological networks. [6]

Statistics and information theory-based is second kind of algorithms widely used and represents an other ways to study gene regulatory networks. Features of networks are selected in order to compute correlation between them and then to classify genes with similar expression pattern in different classes. However, measurements of mRNA levels are often noisy and contain thousands of features with many of them not related to classes defined. Therefore, results are very dependent on the features selection. [32]

A third type of reverse engineering algorithms use linear dynamical systems to describe the problem. Very popular, this method associates models to genes that describes mRNA production rates. Systems of coupled differential equations governing entire networks are built and solved using systems identification theory [17], e.g. linear regression. Limit is that the network must be linear and thus differential equations approximated by first order models. Identification algorithms solve deterministic and quickly such linearised systems, so methods based on linear dynamical systems allow to infer large scale gene regulatory networks. [1], [5], [6], [7], [8].

However, it is known that mRNA production rates of genes are strongly non-linear and thus applications of linear dynamical systems algorithms are only approximations of behaviours of regulatory network. And in place to linearise systems of coupled differential equations to be able to solve it quickly, Evolution Algorithms as Genetic Algorithms (GA) can be implemented to generate candidate solutions for our reverse engineering problem. Still yet rarely applied, this approach is the one studied in [19] where a novel biomimetic representation called Analog Genetic Encoding (AGE) is employed, which can be used with non-linear gene models where analytical approaches or local (gradient-based) optimization methods are not appropriate.

Present research is based on this last class of algorithms. This leads to a new state-of-the-art reverse engineering algorithm which offers the possibility to infer large scale non-linear gene regulatory networks.

Chapter 2 introduces the reader to this new state-of-the-art reverse engineering algorithm. Some definitions are given as the description of the goal to reached. The complete frameworks of processes used are detailed step by step, from simulation of in silico networks to produce gene expression levels data to the reverse engineering process itself. In Chapter 3, present algorithm is applied to an in silico five gene network and results are discussed. Finally, Chapter 4 presents the conclusions based on results found in Chapter 3. As the five gene network used to evaluate efficiency of the developed algorithm is not a gold standard, a future application on the fifty gene network challenge made available for the second DREAM conference (Dialogue for Reverse Engineering Assessments and Methods) is introduced.

All results are obtained with a C++ implementation of the reverse engineering algorithm presented in this research. The code name of this program is Wyrð¹. Annexe A contains an handbook to guide the user through all operation offered by Wyrð as creation, simulation or reverse engineering of regulatory networks. To quickly analyse results generated by Wyrð, a set of Matlab scripts are centralized in one easy to use. Output is an HTML report with network topologies, presentation of results in tables, time series courses, etc. These Matlab tools are introduced to the user in Annexe B.

¹In Norse mythology, Wyrð means the destiny which governs every creature or object of the nine worlds. The destiny is an infinite web where each wire is a creature or an object of the nine worlds. Wyrð is chosen by analogy to wiring of gene regulatory network.

2

Scalable Reverse Engineering of Nonlinear Gene Networks

2.1 Preface

The goal of reverse engineering is to establish and quantify causal relationships between all genes of a biological regulatory network from experimental measurements. The challenge is to recover the topology, or wiring, of such biological network by a reverse engineering approach. Figure 2.1 illustrates the process used.

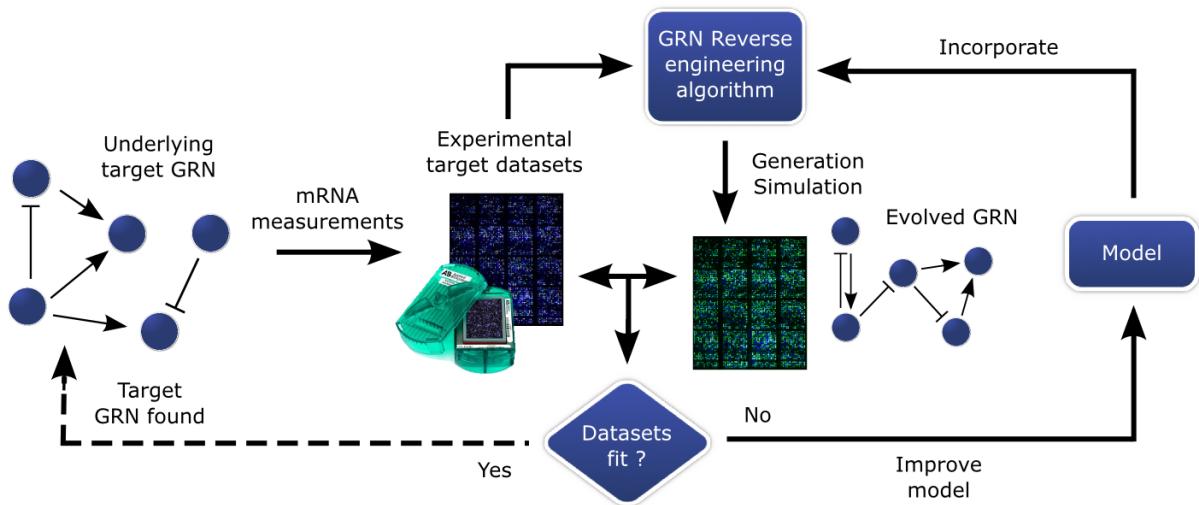


Figure 2.1 Schema of the reverse engineering used. From experimental target datasets obtained after several gene perturbations experiments and measurements of mRNA expression levels of all genes of a biological network, the reverse engineering algorithm is applied. It generates in silico evolved networks, simulates them to produce datasets and compares in silico and target datasets. If they do not fit, the algorithm changes the gene model's parameters and gene-gene interactions to obtain new evolved networks. An Evolutionary Algorithm is used to handle this, where the fitness is the distance between evolved and target datasets. After several generations, an evolved network that generates similar data as the experimental ones is found. The dotted line highlights the fact that the network found is not necessary the target one searched if the problem is under-determined.

Current techniques allow to measure mRNA concentration levels of thousands biomolecules of an organism at the same time. From a set of genes and by applying perturbations on their production rates (gene perturbations), a dataset of mRNA concentration levels can be built from measurements of the expression levels of all genes of this set. Steady-states and times series are two types of widely used datasets. The first one consists to modify the production rate of one or several genes in the network and to measure expression levels of all genes once the system reaches steady-state. Knockout or null-mutant perturbations set to zero transcription rate of a single gene, when heterozygous perturbations divide the rate by two. Time series experiments are also used to study a wide range of biological systems. This type of experiments consists to measure variations of production rates of all genes during a given time. The measures are taken at predefined time points and the most of the time, the number of points is small (eight time points or fewer in most cases). Evolution of the variations of the production rates depend largely on the initial conditions (initial mRNA expression levels of all genes).

From experimental datasets of a network of several genes, the goal is to recover the underlying relationships between them. These target datasets are given as input for the Gene Regulatory Network (GRN) reverse engineering algorithm. In this algorithm, the genes are characterised by a model which is defined by few parameters. An example of non-linear model is the standard sigmoid model with two parameters per gene as widely used in this work. All gene-gene interactions are directed and defined by weights. To prepare reverse engineering process, the algorithm begins with generating a first initial in silico network, most of the time with random values for the gene model's parameters and weights.

Then, this initial network is simulated to produce the same kind of datasets as the experimental target datasets associated to the real biological network given as input for the reverse engineering algorithm. So steady-state and/or time series are computed for the in silico evolved network and are compared to target datasets. If the two evolved and target datasets are very different (normally the case if the initial evolved network is a random one), evolved network is not yet the searched one. Parameters of this evolved network are changed through the use of an Evolutionary Algorithm. With modifying the gene model's and weight's values, a new fully specified evolved network is defined and simulated to produce datasets which are again compared to the experimental target datasets.

After several pathways of the right loop of Figure 2.1, the reverse engineering algorithm generates a gene regulatory network which matches the experimental target datasets. The algorithm ends and considers that the network with the best fit found is the underlying one searched. However, if the problem is under-determined (not enough information about the target network), solutions are not unique and several different gene regulatory networks that generate the same data as the target datasets exist. In this case, additional criteria should be introduced, for example the notion of sparseness in networks in order to discriminate some topologies found by the reverse engineering algorithm.

2.2 Definitions and Problem to solve

A gene regulatory network is composed of several genes and multiple gene-gene interactions. In biological organisms, each gene is characterised by a mRNA concentration level or gene expression level. This level of mRNA concentration can be measured in thousand of genes at the same time and represents the state of a gene network. For one gene, the variation of mRNA concentration level is given by:

$$\frac{dx_i}{dt} = F_i(\mathbf{x}) - \delta_i x_i \quad (2.1)$$

where $F_i()$ is the gene production rate defines as the product of the maximum transcription rate $v_{max,i}$ with the gene model used for gene i . \mathbf{x} the state vector containing all gene expression levels, δ_i a decay constant and x_i the mRNA expression level of gene i .

For an entire network of size N , the system is composed of N coupled differential equations:

$$\frac{dx_1}{dt} = F_1(\mathbf{x}, t) - \delta_1 x_1 \quad (2.2)$$

$$\frac{dx_2}{dt} = F_2(\mathbf{x}, t) - \delta_2 x_2 \quad (2.3)$$

$$\vdots \quad (2.4)$$

$$\frac{dx_N}{dt} = F_N(\mathbf{x}, t) - \delta_N x_N \quad (2.5)$$

In vectorial notation and supposing that all genes have the same model, the system of N coupled differential equations that characterizes the production of gene mRNA concentration levels of the entire gene regulatory network is:

$$\frac{d\mathbf{x}}{dt} = F(\mathbf{x}, t) - \boldsymbol{\delta}\mathbf{x} \quad (2.6)$$

with

$$\mathbf{x} = [x_1, x_2, \dots, x_N]^T \quad (2.7)$$

$$\boldsymbol{\delta} = [\delta_1, \delta_2, \dots, \delta_N]^T \quad (2.8)$$

$\boldsymbol{\delta}$ is constant and \mathbf{x} is the state vector of Equation 2.6.

The vast majority of reverse engineering algorithms use Identification tools with an approximation of the first order of the system of differential equations given in Equation 2.6. This deterministic way to solve the problem allows to reverse engineer large gene networks. However, it is known that biological behaviours of genes are strongly non-linear, whereas such algorithms are restricted to use linear gene models.

The new algorithm presented here is actually one of the rare that allows to reverse engineer non-linear and large gene regulatory networks.

2.3 Reverse Engineering Algorithm

2.3.1 Overview

The idea of this research is to use Evolutionary Algorithms (EAs) to generate candidate solutions (fully specified genes) for the present reverse engineering problem. EAs are population-based optimization algorithms and use some mechanisms inspired by biological evolution as the reproduction, mutation, recombination and selection. Several EAs have been developed as Genetic Algorithms, Genetic programming, Evolution strategy or still Learning classifier system. As classic Genetic Algorithms take generally lot of time to solve systems of higher dimensions due to the fact that a great number of solutions are tested, the choice of the Evolutionary Algorithm used in this work fell on Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [13], [12], [11], [10]. We use a variation of CMA-ES implemented by Knight and Lunacek [16].

In [19], Marbach applies also EAs and uses the same key steps as introduced in Figure 2.1. In his work, a given network is evolved as a whole. To do this, all the parameters of the network are included in the phenotype. Thus, the dimension of the system to solve is:

$$N^2 + N \cdot \text{number of gene model's parameters} \quad (2.9)$$

where the N^2 first parameters are the elements of the weights matrix \mathbf{w} which represent the topology and the interaction strengths of the gene network. In addition, there are still N times the number of parameters of the gene model used. So, the total dimension of the problem for a five gene network using the standard sigmoid model (two parameters, α and β) is 35.

In this current research, the main part of the idea is to break down the system of N coupled equations 2.6. This is equivalent to optimize gene by gene the regulatory network. System 2.6 is divided into N equations of the following form:

$$\frac{dx_i}{dt} = F_i(\hat{\mathbf{x}}, t) - \delta_i \hat{\mathbf{x}} \quad (2.10)$$

where $\hat{\mathbf{x}}$ contains the measurements of all gene expression levels. With the two parameters of the sigmoid model, the dimension of each gene optimization is:

$$N + \text{number of gene model's parameters} \quad (2.11)$$

It is much more easier and faster to solve an equation of dimension proportional to N , N times than to solve once a system of dimension proportional to N^2 .

2.3.2 Gene Model

As it is known that the behaviours of genes are strongly non-linear and thanks to the fact that the present reverse engineering algorithm handles non-linear gene regulatory networks, a non-linear gene model is used. The model chosen and widely used throughout this research and other ones is the standard sigmoid model:

$$f_i^\sigma(z) = \sigma(\alpha_i \cdot \sum_{j \in R_i} w_{ij} x_j + \beta_i), \quad \text{with} \quad \sigma(y) = \frac{1}{1 + e^{-y}} \quad (2.12)$$

$j \in R_i$ is the set of all genes, w_{ij} are elements of the square weights matrix \mathbf{w} which describes the topology or wiring of the entire network. w_{ij} defines the weight or strength of the interaction from gene j to gene i . If $w_{ij} > 0$, the interaction is excitatory, else if $w_{ij} < 0$, inhibitory and otherwise $w_{ij} = 0$ if there is no interaction between gene i and j . α is a multiplicative constant for the regulatory activity (steepness of sigmoid). β determines basal transcription (where the sigmoid crosses the y-axis). Influences of α and β on the gene transcription rate are shown in Figure 2.2. For (a) and (b), x-axis is proportional to the sum $\sum_{j \in R_i} w_{ij} x_j$ or the influence of all genes that have an outgoing connection towards gene i . y-axis represents the mRNA normalized transcription rate which is always positive or zero.

Output values of the sigmoid model are included in interval $[0, 1]$. As the mRNA production rate for the gene i is defined by

$$\text{production rate } F_i(z) = v_{\max} \cdot f_i^\sigma(z) \quad (2.13)$$

so, production rates are always comprised between $[0, v_{\max}]$.

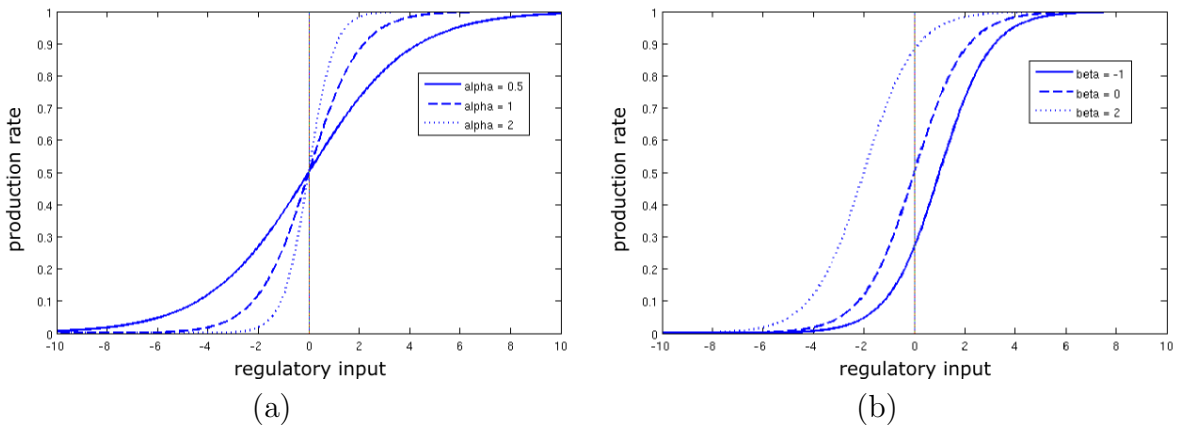


Figure 2.2 Influence of (a) α and (b) β on the gene transcription rates in the standard sigmoid model. α is a multiplicative constant for the regulatory activity (steepness of sigmoid) and β determines basal transcription (where the sigmoid crosses the y-axis).

2.3.3 Algorithm

Initialization

The basic idea of this reverse engineering algorithm is to optimize one-by-one the genes of the regulatory network. Before optimizing the first gene, the reverse engineering algorithm generates a random network of the same size as the target one. Random networks have all gene parameters and the matrix \mathbf{w} set randomly. As sparseness is a property of many biological networks [29], elements of \mathbf{w} can be set to 0 in place of random values. So the network starts with no gene-gene interaction.

At this point, a fully specified in silico random network is available as initial point for the reverse engineering algorithm.

Simulation of Evolved Genes

During a single-gene optimization, only the parameters of one gene and its incoming interaction's weights are estimated. This notion is introduced later in Section 2.5.2. So only these parameters change during the evolution process handled by CMA-ES algorithm. To evaluate new evolved solutions generated by CMA, the network must be simulated in order to produce in silico datasets similar to experimental target dataset given as input. As only the parameters of a single-gene change, *only the gene that is being evolved is simulated* rather than the entire network. This saves computation time.

Three types of steady-states experiments that are widely used in the literature and by the present reverse engineering algorithm: wild type (no perturbation), knockout or null-mutant (single-gene perturbation, $v_{max,i} = 0$) and heterozygous steady-states (single-gene perturbation, $v_{max,i}$ is halved). i is the index of the gene perturbed. Produce steady-state dataset for the gene i consists to solve Equation 2.10 with $\frac{dx}{dt} = 0$, i.e. to find its mRNA concentration levels x_i when the steady-states are reached. For example, the following wild type target dataset of a five gene network can be given as input for the reverse engineering algorithm.

	G0	G1	G2	G3	G4
wild type	\hat{x}_0^{wt}	\hat{x}_1^{wt}	\hat{x}_2^{wt}	\hat{x}_3^{wt}	\hat{x}_4^{wt}

Table 2.1 Experimental target wild type steady-state dataset of a five gene network. Elements are measured mRNA concentration levels of the underlying target gene regulatory network when it reaches a steady-state without external gene perturbation.

To generate steady-state dataset associated to gene i , Equation 2.14 must be solved. The solution x_i is the searched value x_i^{wt} in wild type experiment. In implementation of the reverse engineering algorithm, the Multidimensional Root-Finding of GSL¹ is used to solve Equation 2.14.

$$\frac{dx_i}{dt} = 0 = F_i(\hat{\mathbf{x}}) - \delta_i x_i \quad (2.14)$$

¹The GNU Scientific Library

With the hypothesis that gene regulatory networks do not have self-interaction, i.e. $w_{ii} = 0$ in \mathbf{w} (used to simplify time series integration as presented later), $F_i(\hat{\mathbf{x}})$ is independent of x_i due to the fact that $w_{ii}x_i = 0$ in the sum of Equation 2.12, and thus x_i of the term $\delta_i x_i$ can be easily retrieved.

The second type of datasets used are the time series experiments. From initial state of gene regulatory network, the expression level is measured (mRNA concentration level) at fixed points in time. In example of the five gene network, a time series experiment with 7 time points is given in Figure 2.3 (a).

	G0	G1	G2	G3	G4
t0	$x_0^{ts,t0}$	$\hat{x}_1^{ts,t0}$	$\hat{x}_2^{ts,t0}$	$\hat{x}_3^{ts,t0}$	$\hat{x}_4^{ts,t0}$
t1	$x_0^{ts,t1}$	$\hat{x}_1^{ts,t1}$	$\hat{x}_2^{ts,t1}$	$\hat{x}_3^{ts,t1}$	$\hat{x}_4^{ts,t1}$
t2	$x_0^{ts,t2}$	$\hat{x}_1^{ts,t2}$	$\hat{x}_2^{ts,t2}$	$\hat{x}_3^{ts,t2}$	$\hat{x}_4^{ts,t2}$
t3	$x_0^{ts,t3}$	$\hat{x}_1^{ts,t3}$	$\hat{x}_2^{ts,t3}$	$\hat{x}_3^{ts,t3}$	$\hat{x}_4^{ts,t3}$
t4	$x_0^{ts,t4}$	$\hat{x}_1^{ts,t4}$	$\hat{x}_2^{ts,t4}$	$\hat{x}_3^{ts,t4}$	$\hat{x}_4^{ts,t4}$
t5	$x_0^{ts,t5}$	$\hat{x}_1^{ts,t5}$	$\hat{x}_2^{ts,t5}$	$\hat{x}_3^{ts,t5}$	$\hat{x}_4^{ts,t5}$
t6	$x_0^{ts,t6}$	$\hat{x}_1^{ts,t6}$	$\hat{x}_2^{ts,t6}$	$\hat{x}_3^{ts,t6}$	$\hat{x}_4^{ts,t6}$

Figure 2.3 Example of time series data where the gene G0 is the current evolved one. Element $x_0^{ts,t0}$ means the gene expression level of gene G0 in time series experiment at time point t0. As only the parameters of this gene are modified and so only this gene is simulated to generate time series data, all other time series gene expression levels are set with their values in the target time series dataset.

Figure 2.3 helps to explain how one gene is simulated to produce time series data. Let gene G0 be the current gene that is being optimized. As the parameters of other genes are not modified during optimization of G0, only G0 is simulated to generate its time series data, i.e. the column under the label “G0” in Figure 2.3. To generate time series data, Equation 2.15 must be solved to found expression level x_i of evolved gene i .

$$\frac{dx_i}{dt} = F_i(\hat{\mathbf{x}}) - \delta_i x_i \quad (2.15)$$

For each time point, we must compute the expression level of gene G0 by integrating Equation 2.15. First, initial mRNA concentrations of all genes are set with their value in the target time series dataset given as input for the reverse engineering algorithm. If we will integrate gene expression level of G0 from $t0$ to $t1$ to obtain mRNA level of G0 at time point $t1$, initial conditions are set with values of the target dataset as:

$$\hat{\mathbf{x}} = [\hat{x}_0^{ts,t0}, \hat{x}_1^{ts,t0}, \hat{x}_2^{ts,t0}, \hat{x}_3^{ts,t0}, \hat{x}_4^{ts,t0}]^T \quad (2.16)$$

Equation 2.1 is only depending on x_0 which is the gene expression level searched of G0 at different time points. However, the time step used for the integration of G0 expression levels between $t0$ and $t1$ is not $(t1 - t0)$ but is smaller. A time step used can be for example $\frac{t1-t0}{5}$. To compute expression levels of all not evolved genes that are used to set initial concentration of Equation 2.1, a linear interpolation between two points is made, e.g. between $\hat{x}_1^{ts,t0}$ and $\hat{x}_1^{ts,t1}$ to produce mRNA levels of gene G1 at time points multiples of

$\frac{t_1-t_0}{5}$ between t_0 and t_1 . Figure 2.4 illustrates the principle of the linear interpolation to produce more time points and expression levels to guide integration computation in order to have a smaller integration step than simply $(t_1 - t_0)$.

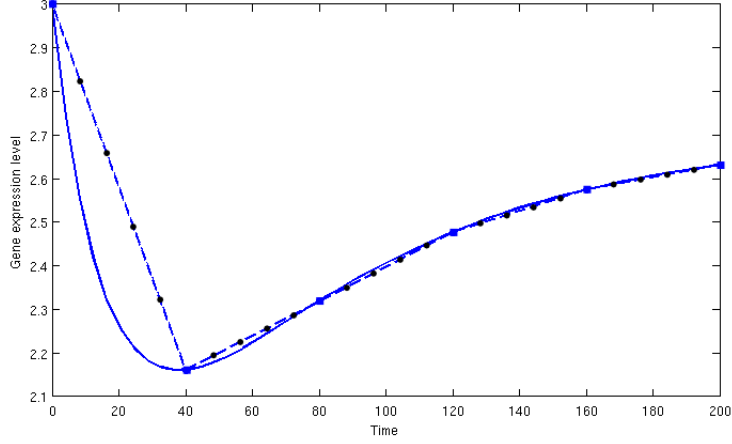


Figure 2.4 Integration curve in plain line is obtained when the entire target network is simulated. Dotted lines are first order interpolations between two given points (blue ■) of the plain line curve as given in the target time series data. Vertexes of the approximated dotted curve are mRNA levels at different time points of the target time series dataset. Between two given points, first order interpolation is used to produce intermediary points (●) useful for integration process.

The mRNA levels obtained by linear interpolation between two given target point to process integration can lead to some errors due to this approximation. Ideally, these intermediary points (●) must be located on the plain line curve in Figure 2.4. A solution may be to interpolate with an order larger than one between all given points (blue ■).

As time series as well as steady-states datasets are generated several thousands of time in one gene optimization, the following hypothesis is defined in order to accelerate time series generation. Supposing there are not gene self-interaction, in others words any activator or inhibitor interaction from a gene to itself. In this case, all elements in the diagonal of the weights matrix \mathbf{w} are set to 0. If the current gene evolved is the gene i , the product $w_{ij}x_j$ with $i = j$ in the sum belonging to Equation 2.12 is 0. So the gene model given by this equation is independent of the variable x_i , and the integration of 2.10 is more easier, and generally more faster.

After integration from t_0 to t_6 , we have the following set of values which represents the time series data when the gene G0 is simulated independently of others.

$$[x_0^{ts,t_0}, x_0^{ts,t_1}, x_0^{ts,t_2}, x_0^{ts,t_3}, x_0^{ts,t_4}, x_0^{ts,t_5}, x_0^{ts,t_6}]^T \quad (2.17)$$

Optimization with CMA-ES

Before introduce the way that CMA-ES is used to process single-gene optimization, a brief introduction to this Evolution Algorithm is presented in Section 2.4. Application of CMA to solve current problem is introduced in Section 2.5.

2.4 Theory of CMA-ES

2.4.1 Generalities

CMA-ES for Covariance Matrix Adaptation Evolution Strategy is a stochastic, population-based, iterative optimization method belonging to the class of Evolutionary Algorithms (EA). CMA allows to adapt the covariance matrix C of the multivariate normal mutation distribution in the EA [13]. The domain defined by the mutation distribution is used to produce new candidate solutions. Where some algorithms estimate a first order model of the objective function (gradient), CMA means learning a second order model which is an approximation of the inverse Hessian matrix H^{-1} , and thus use more the topology information of the objective function to accelerate convergence towards the hoped global optimum. [10]

This section aims to introduce the reader to the key parameters for understanding and putting into practice a CMA-ES. The key steps of the algorithm are highlighted following the nomenclature and the theory presented by Hansen in [11]. More generally, [11] contains all the developments and is an excellent tutorial for anyone wanting to fully understand the functioning of a CMA.

2.4.2 The Multivariate Normal Distribution

A multivariate normal distribution $\mathcal{N}(m, C)$ is centred around its mean $m \in \mathbb{R}^n$ and defined by its symmetric, positive definite covariance matrix $C \in \mathbb{R}^{n \times n}$. The covariance matrix can be uniquely identified with the (hyper-)ellipsoid $\{x \in \mathbb{R}^n | x^T C^{-1} x = 1\}$ as shown in Figure 2.5 (b). [11]



Figure 2.5 (a) Isotropic normal distribution. No direction of the search space is promoted as in classic Genetic Algorithm. It is also the case of CMA at generation $g = 0$. $\sigma \in \mathbb{R}_+$ (b) C is a definite full covariance matrix. CMA uses the topology information of the objective function contains in the covariance matrix to accelerate convergence towards the hoped global optimum [10]. Black point is the mean m of the mutation distribution. Objective function is represented by a fields of lines of equal density.

Principal axis of the ellipsoid are defined by the eigenvectors of C . Square modules, or

square lengths, of these axis are the associated eigenvalues of C . Figure 2.6 illustrates all key steps of CMA algorithm. Following paragraphs described all parameters used and their usefulness.

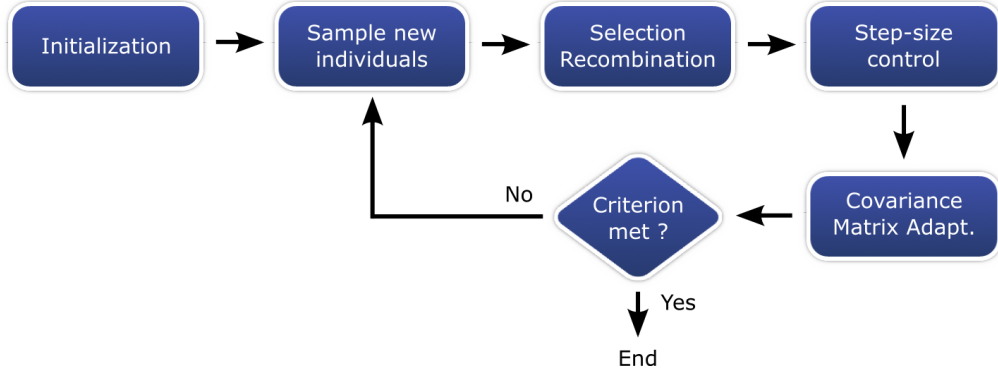


Figure 2.6 CMA-ES is divided into several key steps. Initialization of all parameters, sample of new candidate solutions in the domain bound by the normal distribution $\mathcal{N}(m, C)$, selection of the best μ individuals (according to fitness) as parents for the reproduction, update of the step-size σ and adaptation of the covariance matrix C . If the stop criterion is met, as convergence or best fitness found, EA ends.

2.4.3 Initialization

Initially, the normal distribution associated to the covariance matrix C is isotropic and defined by $\mathcal{N}(m, \sigma I)$ where σ is the step-size, $\sigma \in \mathbb{R}_+$ and $I \in \mathbb{R}^{n \times n}$ the identity matrix of dimension n . Current geometrical interpretation of C is identical to Figure 2.5 (a). The initial mean m for generation $g = 0$ can be set to desired initial value or set randomly with a value $\in \mathbb{R}^n$.

2.4.4 Sampling individuals

As in all population-based Evolution Strategies, a population of individuals is sampled. Individuals are candidate solutions to the problem to solve. In CMA, new individuals are restricted at each generation in the space \mathbb{R}^n defined by the mutation distribution, the normal distribution $\mathcal{N}(m, C)$ associated to the covariance matrix. Individuals are taken randomly following the distribution $\mathcal{N}(m, C)$. So if the population size is λ with $k = 1, \dots, \lambda$, individuals are at generation g : [11]

$$z_k^{(g+1)} \sim N(0, I) \quad (2.18)$$

$$y_k^{(g+1)} = B^{(g)} D^{(g)} z_k^{(g+1)} \sim N(0, C^{(g)}) \quad (2.19)$$

$$x_k^{(g+1)} = m^{(g)} + \sigma^{(g)} y_k^{(g+1)} \sim N(m^{(g)}, \sigma^{2, (g)} C^{(g)}) \quad (2.20)$$

where $B^{(g)} \in \mathbb{R}^{n \times n}$ is an orthogonal matrix. Columns of $B^{(g)}$ are eigenvectors of $C^{(g)}$ with normalized lengths set to unit. $D^{(g)} \in \mathbb{R}^{n \times n}$ is a diagonal matrix with elements equal to the eigenvalues of $C^{(g)}$. These two matrix are the product of decomposition of $C^{(g)}$.

2.4.5 Selection and Recombination

Selection and recombination consists to take the $\mu \leq \lambda$ best individuals between all $x_k^{(g+1)}$, $k = 1, \dots, \lambda$ as parents that will be used for reproduction. Individual are ranked according to fitness as $f(x_{1:\lambda}^{(g+1)}) \leq f(x_{2:\lambda}^{(g+1)}) \leq \dots \leq f(x_{\lambda:\lambda}^{(g+1)})$ where $f()$ is the objective function to minimize. With these μ parents, λ offsprings are obtains by reproduction around the mean of the mutation distribution. To update the mean m of the mutation distribution, Equation 2.21 is used.

$$m^{(g+1)} = \sum_{i=1}^{\mu} w_i x_{i:\lambda}^{(g+1)} \quad (2.21)$$

$$\sum_{i=1}^{\mu} w_i = 1, \quad w_1 \geq w_2 \geq \dots \geq w_{\mu} > 0 \quad (2.22)$$

where $w_{i=1\dots\mu} \in \mathbb{R}^+$ are positive weight coefficients for recombination. Other reproduction strategies exist with the use of different weights w_i . If all $w_i = \frac{1}{\mu}$, Equation 2.21 calculates the mean value of the μ parents. [11]

2.4.6 Adaptation the Covariance Matrix

It is here that the step-size $\sigma^{(g+1)}$ and the covariance matrix $C^{(g+1)}$ are updated. Rank-one-update of C uses a single new individual to update covariance matrix. Thus, C is re-estimate for all new points, that needs a lot of computation. To accelerate convergence, a rank- μ -update was developed. This method re-estimates not the entire covariance matrix as in the rank-one-update, but used information of C of previous generations to update it. This makes the algorithm faster but less robust and less oriented global search. [11]

2.4.7 Limited memory version of CMA-ES, L-CMA-ES

Knight and Lunacek have developed an interesting variant of the CMA-ES in order to improve its space and time complexity [16]. Called L-CMA-ES for Limited memory CMA-ES, the idea is that instead of re-estimate covariance matrix for all new individuals, update only the principal directions of the search space \mathbb{R}^n are sufficient. To make that, $m \ll n$ principal eigenvectors associated to the m greatest eigenvalues are updated. This is equivalent to only update a sub-space of dimension m of the entire search space of dimension n . Results show that while the total number of evaluations of the objective function increases for $m < n$, the increase in computational efficiency leads to a lower overall run time. [16]

2.5 Apply CMA-ES to Reverse Engineering Algorithm

2.5.1 Implementation of CMA-ES used

The implementation of CMA-ES used is that of Knight and Lunacek as introduced in [16]. Some improvement have been made to increase robustness to NaN errors (Annexe A.6.5). The original original implementation of CMA and L-CMA-ES used has several bug of stability, that will be corrected in a next version (J.N. Knight, personel communication).

2.5.2 Individuals

For a single-gene optimization, dimension of the search space is given by

$$N + \text{number of gene model's parameters} \quad (2.23)$$

where N is the number of possible incoming interactions towards the gene evolved. Using the sigmoid model, the set of parameters to be inferred is given by

$$\Theta_i = [v_{max,i}, \alpha_i, \beta_i, \delta_i, w_{i1}, w_{i2}, \dots, w_{iN}] \quad (2.24)$$

2.5.3 Bound Evolvable Parameters

To limit the search space of candidate solutions, all individual's parameters are bounded. As CMA-ES does not implement it, the following strategy is used. For each individual generated by the Evolution Algorithm, each value of parameters is compared to its lower and upper bounds. If one parameter p does not belong to the interval $[\text{lowerBoundOfP}, \text{upperBoundOfP}]$, a penalty is associated for violating the complementary space of this interval. For one parameter, this penalty is defined as the square error between the value of the parameter p and the violated bound. Figure 2.7 displays the evolution of the penalty for one parameter. For an entire individual, the total penalty is the sum of all parameter's penalties.

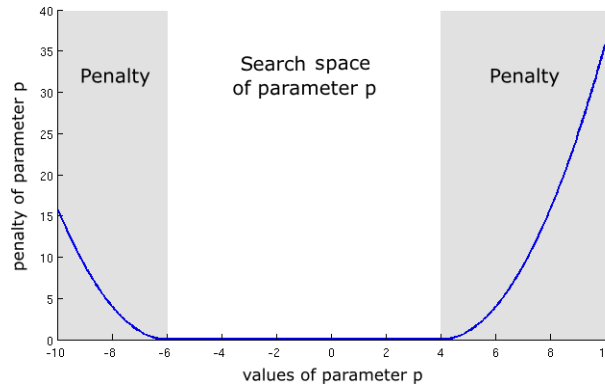


Figure 2.7 Null penalty is associated to the parameter p inside its search space defined by the bounds $[-6, 4]$. Outside, the penalty is defined as the square error between the value of the parameter p and its violated bound.

2.5.4 Evaluation of Evolved Genes

To differentiate good evolved genes from bad ones, the definition of the fitness is an important key step in almost all Evolution Algorithms. The fitness is a real scalar $\in \mathbb{R}$ which is the evaluation of the objective function with a given set of parameters Θ_i . In reverse engineering algorithm, the goal is to found a gene regulatory network which produces identical datasets than the experimental ones when it is simulated. Simulation of single-gene is introduced previously in Section 2.3.3. So, the fitness chosen is the distance between evolved in silico and target datasets (Figure 2.1). As two types of datasets are used, some precisions for steady-states and time series distances calculation are given.

An example of structure for wild type and knockout steady-state datasets as obtained when an five gene evolved network is simulated by the reverse engineering algorithm is given in Table 2.2.

	G0	G1	G2	G3	G4
wild type	x_0^{wt}	x_1^{wt}	x_2^{wt}	x_3^{wt}	x_4^{wt}
knockout gene 0	0	$x_1^{ko,0}$	$x_2^{ko,0}$	$x_3^{ko,0}$	$x_4^{ko,0}$
knockout gene 1	$x_0^{ko,1}$	0	$x_2^{ko,1}$	$x_3^{ko,1}$	$x_4^{ko,1}$
knockout gene 2	$x_0^{ko,2}$	$x_1^{ko,2}$	0	$x_3^{ko,2}$	$x_4^{ko,2}$
knockout gene 3	$x_0^{ko,3}$	$x_1^{ko,3}$	$x_2^{ko,3}$	0	$x_4^{ko,3}$
knockout gene 4	$x_0^{ko,4}$	$x_1^{ko,4}$	$x_2^{ko,4}$	$x_3^{ko,4}$	0

Table 2.2 mRNA concentration levels for wild type and knockout steady-states datasets as obtained when an in silico five gene network is simulated. Experiments are presented line by line.

$x_i^{ko,j}$ is the mRNA concentration level of gene i in a knockout steady-state evolved experiment where production rate of gene $j = i$ is forced to zero as shown in the diagonal of the knockout experiments in Table 2.2.

Difference or distance between evolved and target datasets is calculated and used as fitness in EA. As genes are optimised one-by-one independently, only mRNA levels of current gene evolved are taken into account in the calculation of the distance. Distance used as fitness refers to the mean square error between target and evolved dataset as defined by Equation 2.25.

$$f_{ss,i} = \frac{\sum_j^{E(wt)} (x_i^{wt} - \hat{x}_i^{wt})^2 + \sum_j^{E(ko)} (x_i^{ko,j} - \hat{x}_i^{ko,j})^2 + \sum_j^{E(hz)} (x_i^{hz,j} - \hat{x}_i^{hz,j})^2}{E(wt) + E(ko) + E(hz)} \quad (2.25)$$

where i is the index of the gene evolved, $f_{ss,i}$ its fitness and $E(wt)$, $E(ko)$ and $E(hz)$ respectively the number of experiments of the same type available (often $E(wt) = 1$, $E(ko) = E(hz) = N$). For the distance between two evolved and target time series experiments datasets, it is defined as the square error over all time points of all time series for one gene.

$$f_{Ts,i} = \frac{\sum_{e=1}^E \sum_{pt=1}^{PT(e)} [x_i^{e,pt} - \hat{x}_i^{e,pt}]^2}{\sum_{e=1}^E PT(e)} \quad (2.26)$$

E is the total number of time series experiments and $PT(e)$ the number of time points in experiment e . $x_i^{e,pt}$ and $\hat{x}_i^{e,pt}$ represent respectively mRNA levels of gene i in time series experiment e at time point pt in evolved and target datasets.

If a reverse engineering algorithm uses both steady-states and time series datasets, the total fitness f_{tot} of a given individual is the sum of both steady-states and time series distances.

$$f_{tot} = f_{ss} + f_{Ts} \quad (2.27)$$

2.5.5 Fitness and Prediction Error

Fitness and Prediction error are used as key words and may mislead the reader on their meanings. Figure 2.8 illustrates the difference between these two concepts.

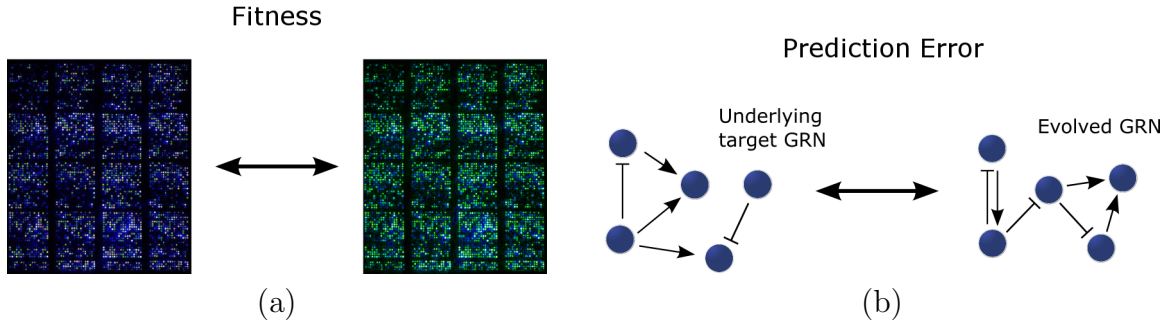


Figure 2.8 Difference between fitness and prediction error that may mislead the reader because they are both square errors. (a) Fitness evaluates the difference between the corresponding mRNA concentration levels of two datasets as achieved by measures on the underlying biological network or by simulating in silico evolved networks generated by the reverse engineering algorithm. (b) Prediction error refers to the square error between all parameters of two solutions (individuals) in one single-gene optimization. In in silico experiments, square error is generally computed between an evolved solution and the target solution (if target network is available.)

The definition of fitness was given in previous section and characterises the mean square error between an experimental target dataset and datasets as obtained by simulating an evolved single-gene.

In application of the reverse engineering algorithm on real biological problem, there is no way of knowing if a gene regulatory network found is the real target GRN, even if datasets of the found and target networks are identical. As briefly mentioned in Section 2.1, several different networks can be found and generate the same datasets. This is due to the fact that the problem to solve is under-determined.

To evaluate the reverse engineering algorithm, *in silico* test cases are used. This allows to record the prediction error between evolved and target network and to know if the evolution goes towards the target network searched and known. As reverse engineering algorithm optimizes gene-by-gene, the prediction error is defined as the sum of the square errors between all parameters of two genes. For the individual given by Equation 2.24, the *square error* is given by:

$$\text{Prediction error}_i = [v_{max,i} - v_{max,i}^{tar}]^2 + [\alpha_i - \alpha_i^{tar}]^2 + \dots + [w_{0i} - w_{0i}^{tar}]^2 + \dots \quad (2.28)$$

3

Efficiency of the Algorithm on a Five Gene Network

3.1 Goal

To evaluate the efficiency of the reverse engineering algorithm introduced in Chapter 2, an *in silico* five gene regulatory network is built and used as target network. This network is simulated to produce steady-states and time series experiments, that will given as input for the reverse engineering algorithm. From these data, it will try to recover the wiring of the target *in silico* gene network.

Experiments are described step-by-step and results obtained are presented and discussed at every stage. This is used to define correct hypothesis on which next steps of experiments are based.

As the *in silico* target network is available which is not the case in biological experiments, it is possible to compare found and target solutions to define the efficiency of the reverse engineering algorithm. Results show that the process recover successfully the six gene-gene interactions of the five gene test case with only one false positive.

3.2 In Silico Target Network

Rather than generate a random network, the five gene network used as test case is built by hand. Indeed, random networks have often bad dynamics. For this reason, we use a network built manually. The *in silico* target network used in the current evaluation of the reverse engineering algorithm is defined in Figure 3.1.

As a property of many gene regulatory networks is the sparseness, few gene-gene interactions are set in target network. In total, three excitatory and same number of inhibitory connections are defined. All genes have two incoming or outgoing interactions, except genes G0 and G1 that have three.

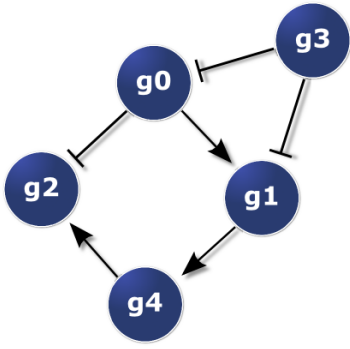
In Silico Target Network (Grntest)			
			
	v_{max}	δ	\mathbf{w}^1
g0	0.10	0.01	$\begin{pmatrix} 0 & 0 & 0 & -0.5 & 0 \\ 1 & 0 & 0 & -2 & 0 \\ -2 & 0 & 0 & 0 & 4.5 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 \end{pmatrix}$
g1	0.15	0.05	
g2	0.20	0.10	
g3	0.25	0.15	
g4	0.34	0.20	

Table 3.1 Grntest is used as target network throughout this chapter to illustrate the efficiency of the reverse engineering algorithm introduced in Chapter 2. All genes follow the standard sigmoid model (Section 2.3.2). They are described by four parameters: the maximum transcription rate v_{max} , a multiplicative constant for the regulatory activity (steepness of sigmoid) α , the basal transcription (where the sigmoid crosses the y-axis) β and a decay constant δ . All α and β are set respectively to 1 and 0 in this experiment.

3.3 In Silico Target Datasets

In biological case, only experimental mRNA concentration levels are available in the form of datasets to reverse engineer underlying networks. So the first thing to do is to simulate the in silico target network to produce different type of datasets. For this experiment, the following noiseless datasets have been generated from Grntest.

- 1 wild type steady-state experiment
- 5 knockout steady-states experiments
- 5 heterozygous steady-states experiments
- 4 time series experiments with different initial conditions, each one with 21 time points

¹Element w_{ij} in the i^{th} line, j^{th} column of the weights matrix \mathbf{w} represents the excitatory ($w_{ij} > 0$) or inhibitory ($w_{ij} < 0$) interaction from gene j to gene i .

The theory used to generate these datasets is the one presented in Section 2.3.3. To generate them from Grntest using Wyrdd², section “Simulation of GRNs” of the Wyrdd Handbook can help the user in this process (Annexe A.5). Target datasets generated by Wyrdd and used to reverse engineer Grntest are available in Annexe D.

3.4 Calibration of CMA-ES parameters

3.4.1 Procedure

Before trying to recover any target gene regulatory networks, parameters of the Evolution Algorithm used to process single-gene optimization must be calibrated in order to obtain the best performances. To evaluate them, the gene G0 of the Grntest network is optimized from target datasets described in previous section. For each different parameters of CMA-ES, G0 is optimized 50 times. The calibration procedure starts with the following values of key parameters of CMA to evaluate.

1. Total number of evaluations (50000)
2. Population size λ (10)
3. Initial σ (1)
4. Recombination strategy (superlinear)

For all estimations of these parameters, the total number of evaluations made by CMA is fixed to 50000. The value of the population size λ is the first estimated. For the best λ found, second parameter σ_0 is in turn evaluated, etc. For each value of one parameter, evolution of the fitness and prediction errors through generations of the Evolution Algorithm are displayed.

3.4.2 Evaluation of Population size

In CMA-ES, the total number of solutions evaluated during optimization is defined by the parameter of total number of evaluations. So, the total number of generations done by CMA-ES depends on the population size λ as given by:

$$\text{Number of generations} \sim \frac{\text{Total number of evaluations}}{\text{Population size } \lambda} \quad (3.1)$$

At each generation, both median fitness and prediction errors of the λ individuals forming the population of candidate solutions are recorded and plotted in Figure 3.1. Table 3.2 highlights some interesting results.

²Wyrdd is the name project of the C++ reverse engineering program implemented for the purposes of the present Master thesis.

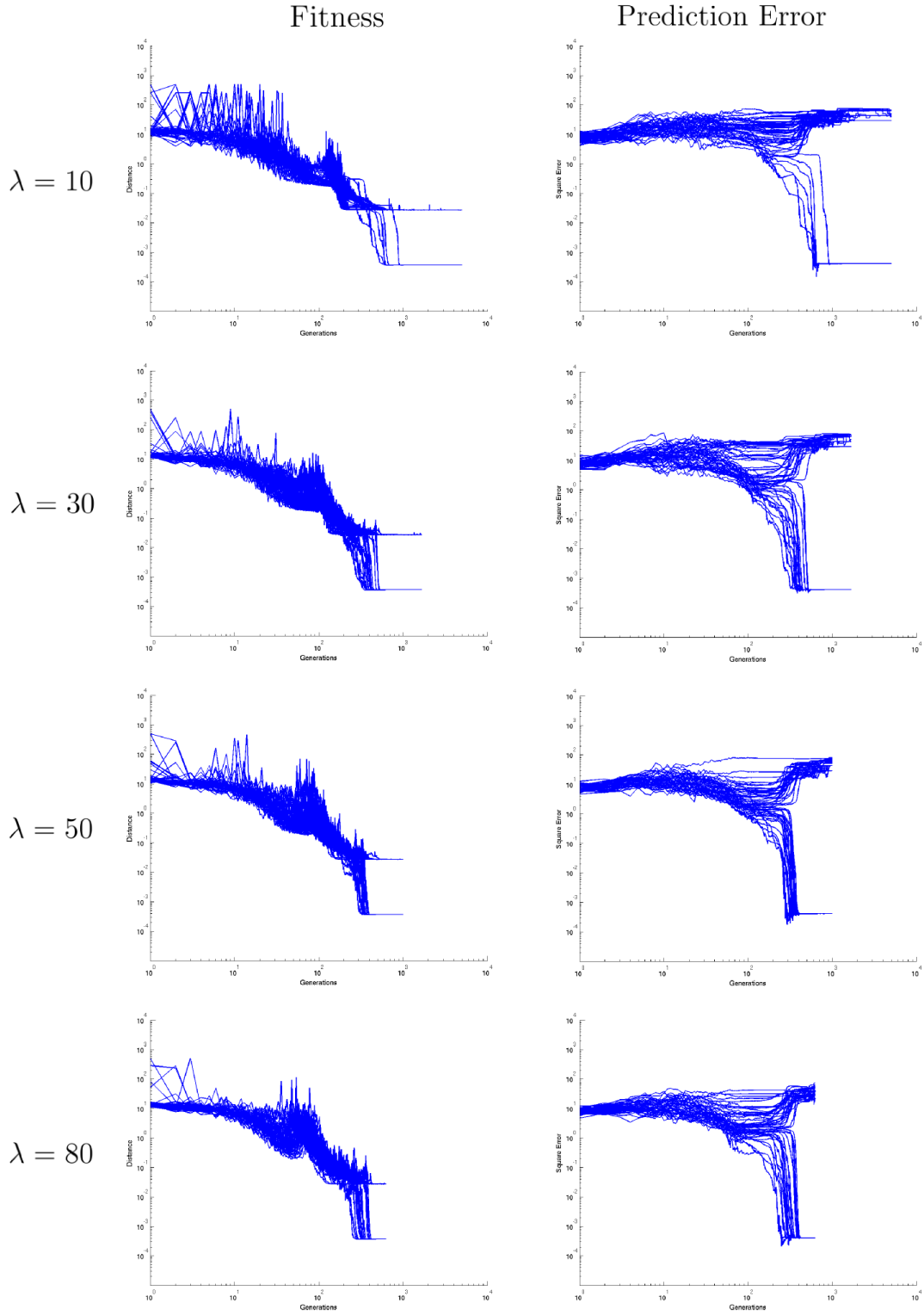


Figure 3.1 Number of evaluations = 50000, $\sigma_0 = 1$, recombination strategy: super-linear. For each population size, evolution of (a) the population median fitness and (b) median prediction error through generations are shown.

λ	Num. successful runs	Generations	Num. evaluations
10	6	612	6120
30	17	383	11490
50	20	327	16350
80	19	312	24960

Table 3.2 Influence of population size λ on results of a batch of optimizations. Second column counts the number of runs among 50 that found a fitness under $1e-3$, third the mean generation where the value of the best fitness is found and fourth the total number of evaluations necessary to met the best fitness found (λ -value of third column).

Looking at plots in Figure 3.1, all 50 optimizations converge towards two distinct solutions with fitness equal to 0.027 and $3.6 \cdot 10^{-4}$ for the best one. Note that single-gene optimizations are minimizations due to the fact that fitness is the distance between target and evolved datasets which must be minimized (Section 2.5.4).

From now, analysis focus on runs that have found the best fitness. Table 3.2 highlights the best runs. For each population size λ , the count of runs that converge towards the best fitness found is recorded. We can see that this number increases when λ increases, with no great difference between $\lambda = 50$ and $\lambda = 80$. This is because at each generation, optimization with $\lambda = 50$ samples more solutions than when $\lambda = 10$ or 30, and so have a better sample of the search space. Small populations require more generations. They are more “blind” than evolutions with large λ . As the logic is, it is better to make few generations with a good sample from the beginning than process “blind” operations during a great number of generations. This can be supported by the difference of total number of evaluations between different values of λ . Last column of Table 3.2 shows that fewer number of evaluations are necessary to reach the best fitness found in case where λ is small. This performance is only useful if the optimization goes towards this optimum from the beginning (from generation 0).

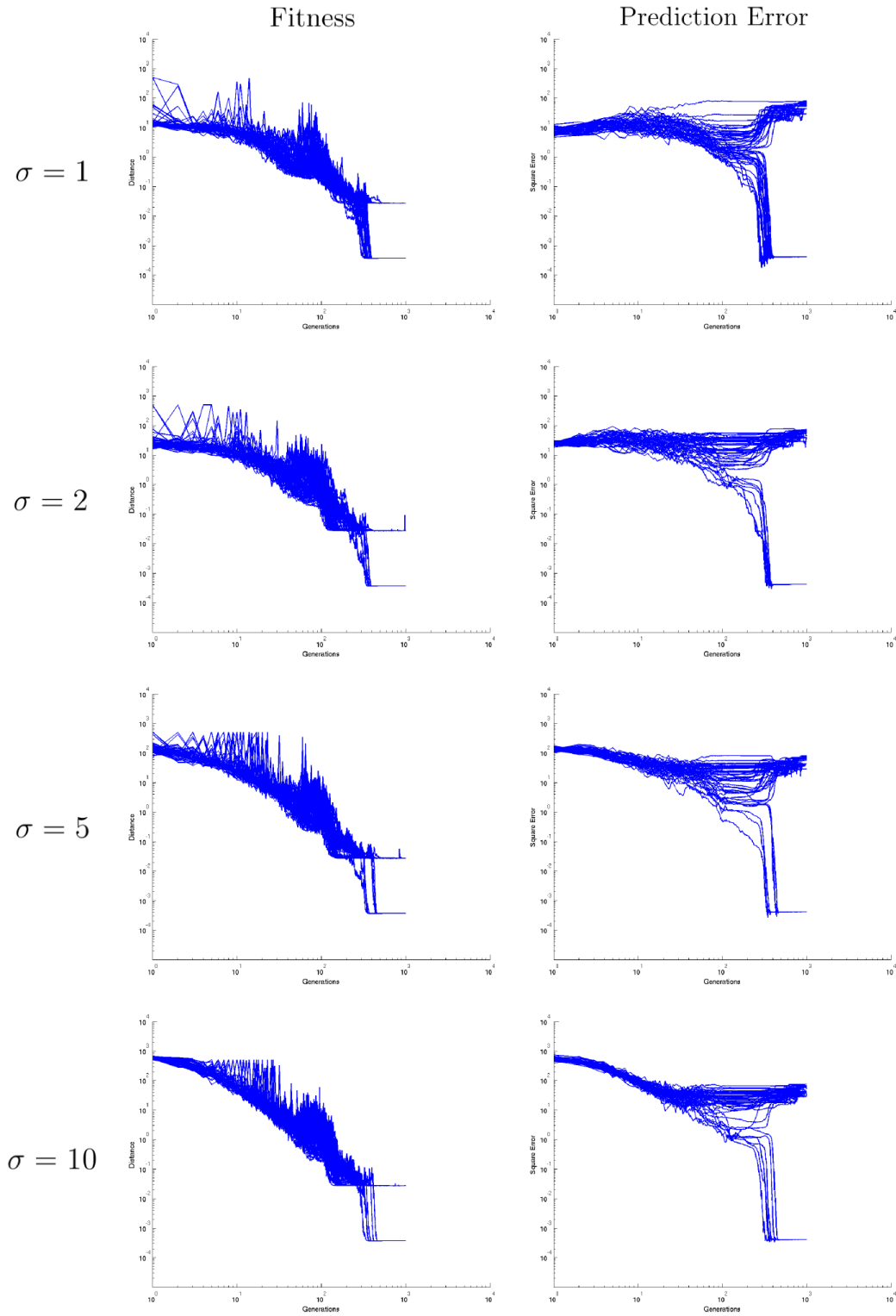
Figure 3.1 displays also the evolution of median prediction error between target and evolved solutions. In all cases, we can see than the 50 runs are divided into two categories: one part converges towards a bad solution with prediction error equal to 200, and one part that found a prediction error equal to $4.1 \cdot 10^{-4}$. This small prediction error means that this last category found a solution very close of the target network. The runs that found the best fitness are the ones that converge towards the smallest prediction error.

From now, population size will be set to 50 because it has the highest number of runs with fitness $< 10^{-3}$.

3.4.3 Evaluation of Initial step-length

Initial value of the step-length σ_0 is used to set initial mutation distribution given by the normal distribution $\mathcal{N}(m, \sigma_0 I)$. Mutation distribution bounds a domain of the search space to sample of size proportional to σ at each generation.

Figure 3.2 displays the evolution of fitness and prediction errors respectively for $\sigma_0=1, 2, 5, 10$ and 20 . Table 3.3 highlights some results.



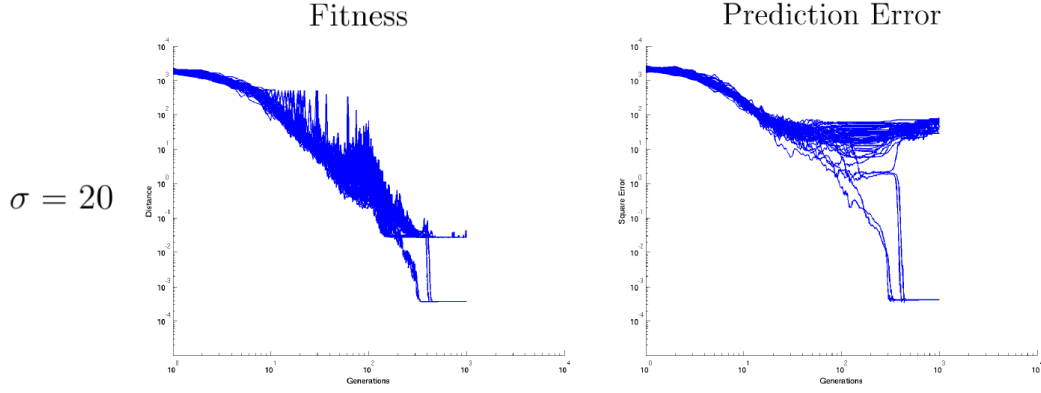


Figure 3.2 Number of evaluations = 50000, population size $\lambda = 50$, recombination strategy: superlinear. For each population size, evolution of (a) the population median fitness and (b) median prediction error through generations are shown.

σ_0	Num. successful runs
1	20
2	7
5	6
10	6
20	4

Table 3.3 Influence on the efficiency of the CMA-ES for different values of σ_0 . Second column counts the number of runs that found a fitness smaller than $1e-3$. Use of $\sigma_0=1$ leads to best results.

Table 3.3 shows that the number of successful runs increases when σ_0 decreases. This is due to the fact that if σ_0 is large, the search space bounded by the mutation distribution is large and so Evolution Algorithm has difficulty to converge, mainly towards the optimum found by the successful runs. Value of σ_0 depends on the intervals of the parameters evolved. In this application, the largest lower or upper bound in absolute value is 5 (weights).

Normally, a large mutation in Evolution Algorithms allows to cover better the search space of solutions and to access to valley with optimum at the bottom that are not achievable for an optimization which starts with a given point and a small mutation distribution. However in WyrD, all initial points of the 50 optimizations are different and are sampled randomly and uniformly in the space defined by the bounds of evolvable parameters (one between several initial points selection strategies available). This allows to compensate the weak of small σ (weak coverage of the entire search space of solutions) with its strength (good convergence).

Again, optimizations that lead to the best fitness found are associated to the smallest final prediction error. For future experiments, σ_0 is fixed to 1 according to these results.

3.4.4 Evaluation of Recombination strategies

Two recombination strategies are implemented in CMA-ES by Knight and Lunack and differ in the ways to set weights $w_{i;\mu}$ which influences updates of mean m of the mutation distribution at each generation (Section 2.4.5). Having ordered the μ parents according to their fitness, weights for the best one ($i = 1$) to the worst one ($i = \mu$) can be set following these two different strategies:

- **Linear**
 $w_i = \mu + 1 - i$
- **Superlinear**
 $w_i = \ln(\mu + 1) - \ln(i)$

Figure 3.3 displays the evolution of both median fitness and prediction error respectively for linear and superlinear recombination strategies. Table 3.4 highlights some useful results.

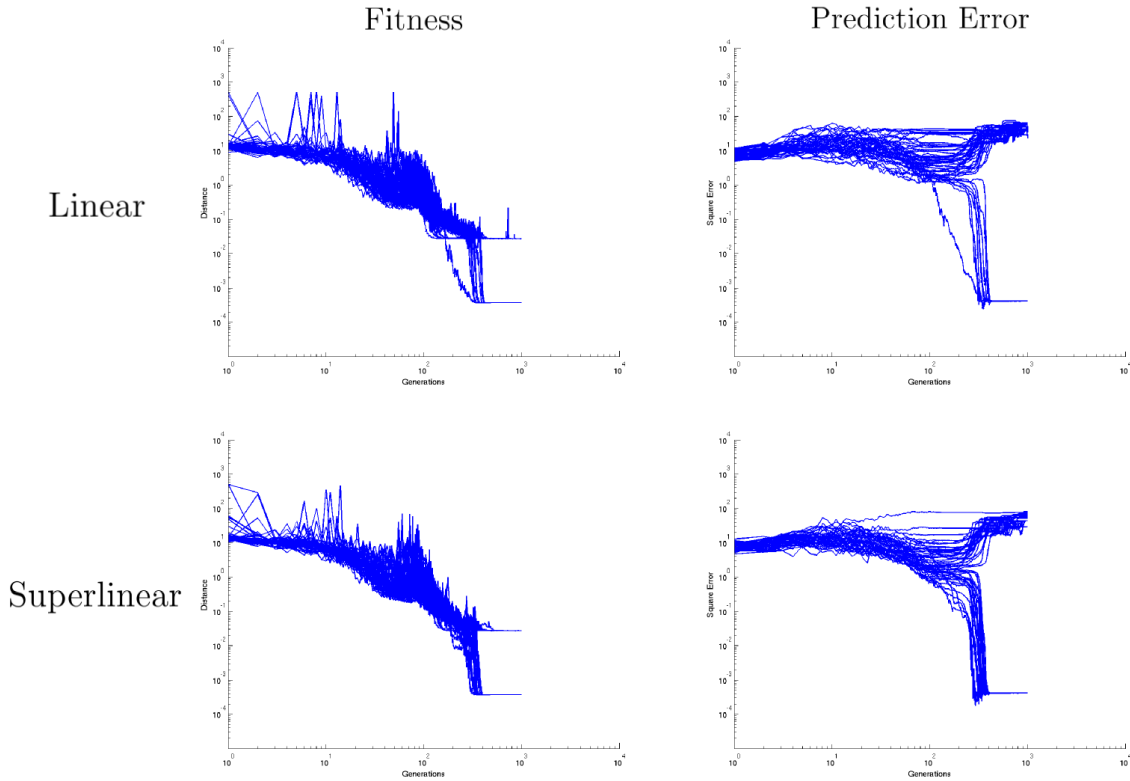


Figure 3.3 Number of evaluations = 50000, population size $\lambda = 50$, $\sigma_0 = 1$. For each population size, evolution of (a) the population median fitness and (b) median prediction error through generations are shown.

For a community of $\mu = 10$ parents in descending order according to their fitness with $i = 1$ is the index of the best parent, Figure 3.4 displays the difference between weights set by the linear and superlinear recombination strategies.

Reproduction Strategy	Num. successful runs
Linear	9
Superlinear	20

Table 3.4 Influence on the efficiency of the CMA-ES for two different recombination strategies. Second column counts the number of runs that found a fitness smaller than $1e-3$. Use of superlinear recombination strategy leads to best results.

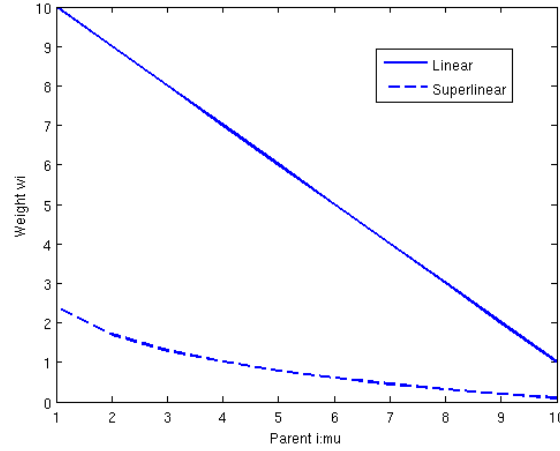


Figure 3.4 Difference of weights $w_{i;\mu}$ between linear and superlinear recombination strategies. Weights are used in CMA-ES to update mean of the mutation distribution.

As introduced in Section 2.4.5, mean of the mutation distribution is given by:

$$m^{(g+1)} = \sum_{i=1}^{\mu} w_i x_{i;\lambda}^{(g+1)} \quad (3.2)$$

where $x_{i;\lambda}^{(g+1)}$ are all solutions contained in the population of size λ at generation $(g+1)$. In superlinear strategy, all parents have almost the same weights, so m is very close to the real mean of all solutions $x_{i;\lambda}$. There is less equity between parent's influence on the update of m in linear strategy, and mean of the mutation distribution can be strongly biased in some cases. Even if no great differences in fitness or prediction errors are observable, superlinear strategy is preferred following the results of Table 3.4.

3.4.5 Summary of best CMA-ES parameters found

From previous experiments, the following key parameters of CMA-ES are chosen to reverse engineer the five gene network Grntest.

1. Number of optimization runs for each gene (150)
2. Total number of evaluations (50000)
3. Population size λ (50)
4. Initial σ (1)

5. Recombination strategy (superlinear)

For this configuration, one optimization run for one gene takes about 14-16 seconds and thus the network can be inferred with only one run per gene in about one minute. Here, optimization of all five genes each with 150 runs takes between 2h40' and 3h20'.

3.5 Results

3.5.1 Raw data

Gene parameters

In silico target datasets generated at Section 3.3 are given as input for the reverse engineering algorithm to try to recover the wiring of the in silico target network presented in Section 3.2. This last one is also given to record the evolution of prediction errors between found and target solutions through generations.

To optimize the first gene belonging to the five gene network Grntest, the following solution vector or set of parameters Θ_i is given to the CMA algorithm:

$$\Theta_i = [v_{max,0}, \delta_0, w_{01}, w_{02}, w_{03}, w_{04}] \quad (3.3)$$

At the end of the Section 2.3.3, the hypothesis that genes of regulatory networks do not have self-interaction is set and used in the implementation of the program to simplify the integration process necessary to compute time series experiments. So weights w_{ij} with $i = j$ are always equal to zero and are not added into Θ_i . Finally in the current test case, dimension of the problem to solve to reverse engineer one single-gene is six.

All parameters of the set Θ_i are bounded to limit the search space of candidate solutions. As the target network is known, bounds are set so that all parameters of the target network are contained in the bounded intervals. Table 3.5 presents the selected lower and upper bounds for all evolved gene parameters.

	Lower bounds	Upper bounds
Weights	-5	5
$v_{max,i}$	0	2
δ_i	0.01	0.3

Table 3.5 Lower and upper bounds values for all evolved gene parameters.

After processing all genes of the network Grntest, the evolved genes presented in Table 3.6 are the best ones found. These results will be discussed more in detail in the following sections.

	f	se	δ_i	$v_{max,i}$	w_{i0}	w_{i1}	w_{i2}	w_{i3}	w_{i4}
g0	3.640e-4	4.112e-4	0.010	0.096	0	0.001	0.014	-0.486	0.003
g1	1.011e-3	4.171e-1	0.042	0.123	0.989	0	-0.140	-1.370	-0.022
g2	3.130e-3	21.667	0.132	0.263	-2.063	0.209	0	3.969	2.079
g3	6.824e-9	1.288e-7	0.150	0.250	2.959e-5	4.352e-5	1e-4	0	2e-4
g4	7.173e-4	1.140e-1	0.300	0.450	0.033	3.228	0.058	-0.155	0

Table 3.6 Raw data given as output after the reverse engineering of Grntest. Columns f refers to the fitness of the best run. Se is the prediction error. Next columns present the values found for all gene's parameters.

Fitness and Prediction errors plots

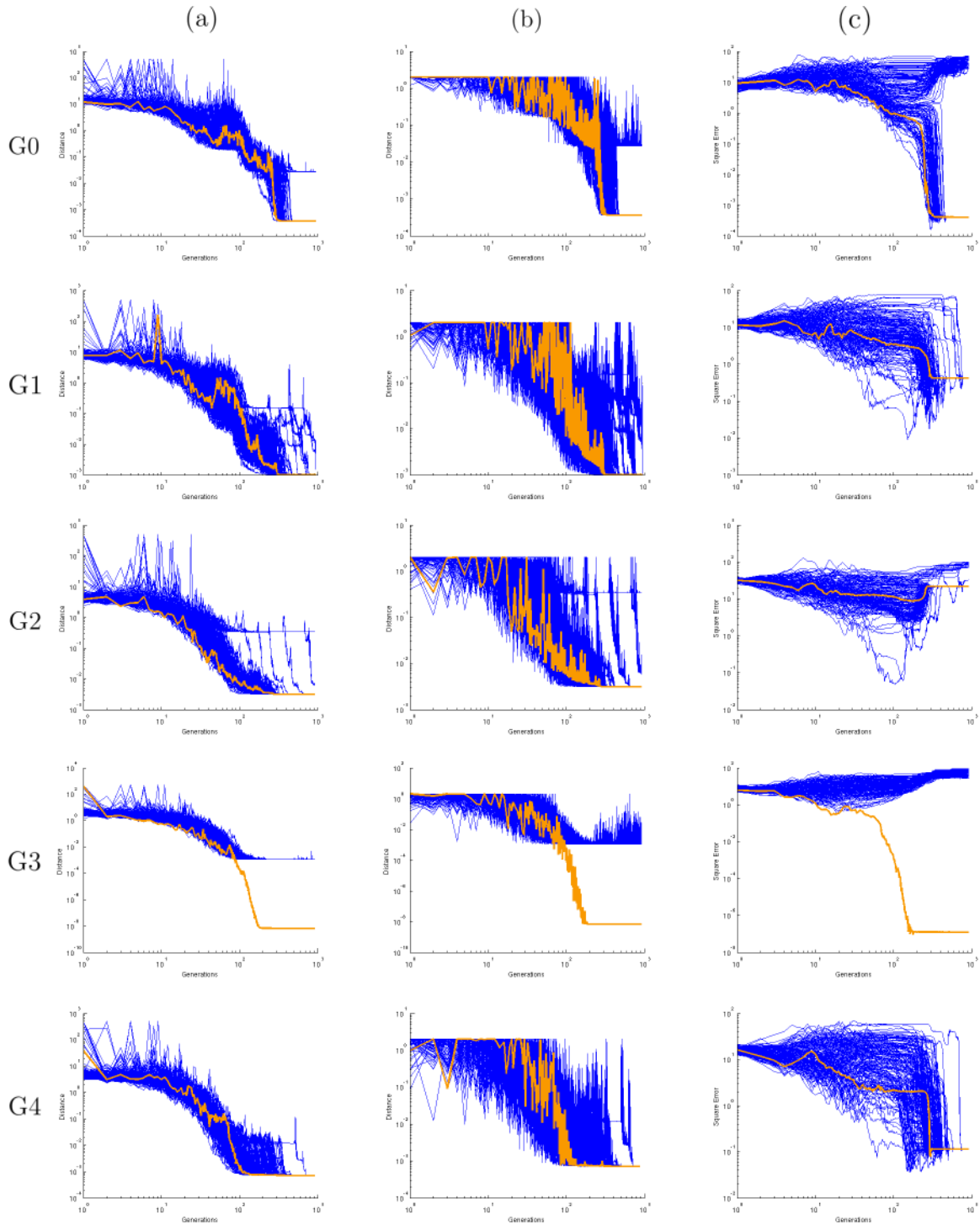
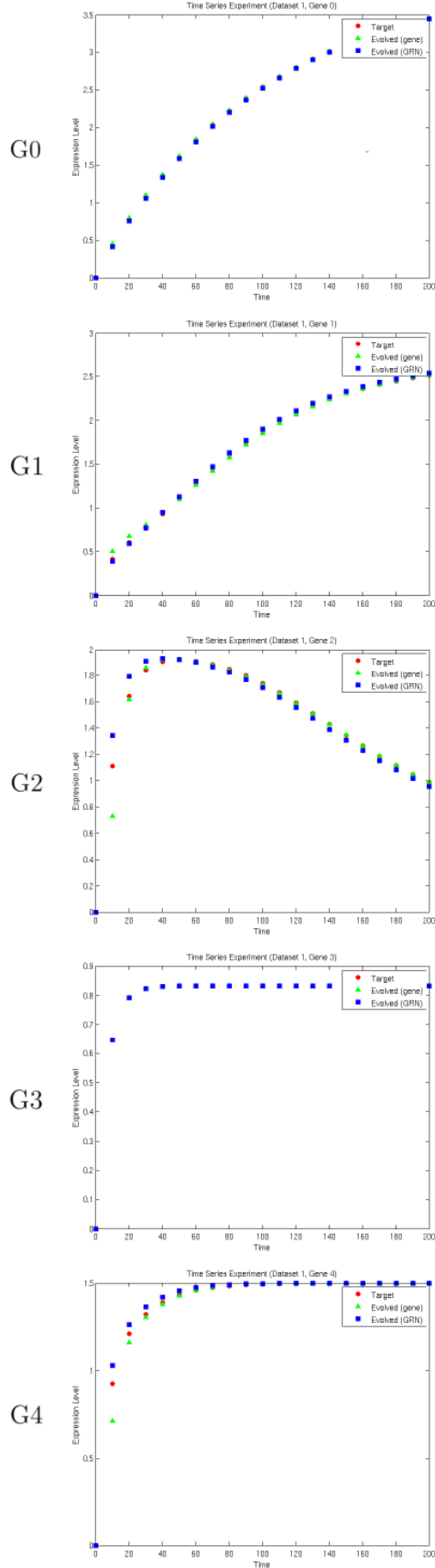


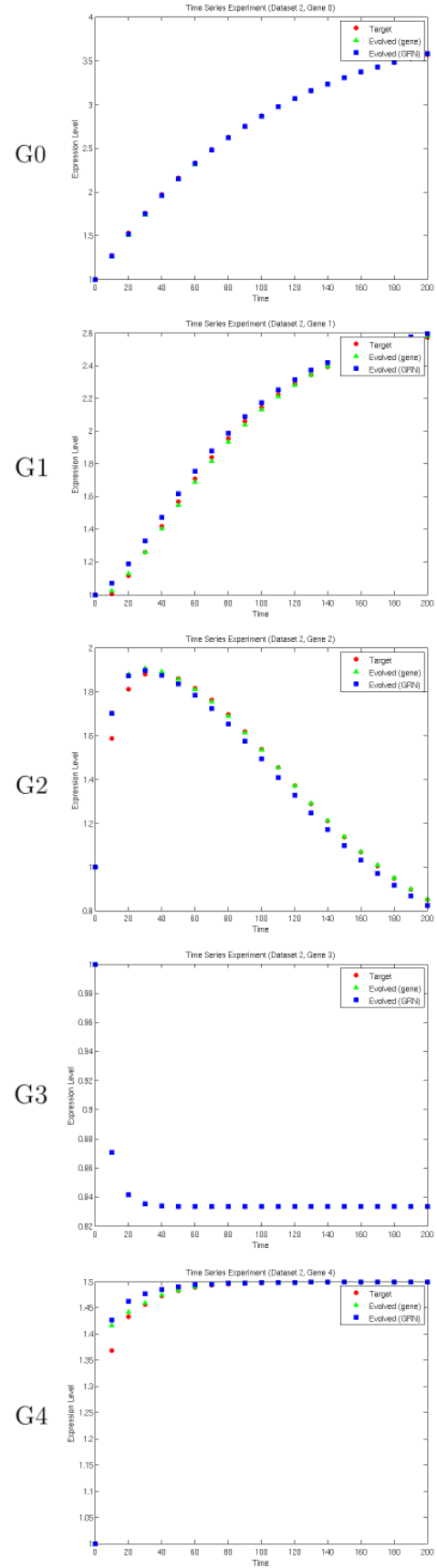
Figure 3.5 Each line corresponds to a single-gene optimization, there are 150 runs. (a) Median and (b) minimum fitness, and (c) median prediction error of the population of size $\lambda = 50$. For each gene, the best run is highlighted.

Time Series Experiments plots

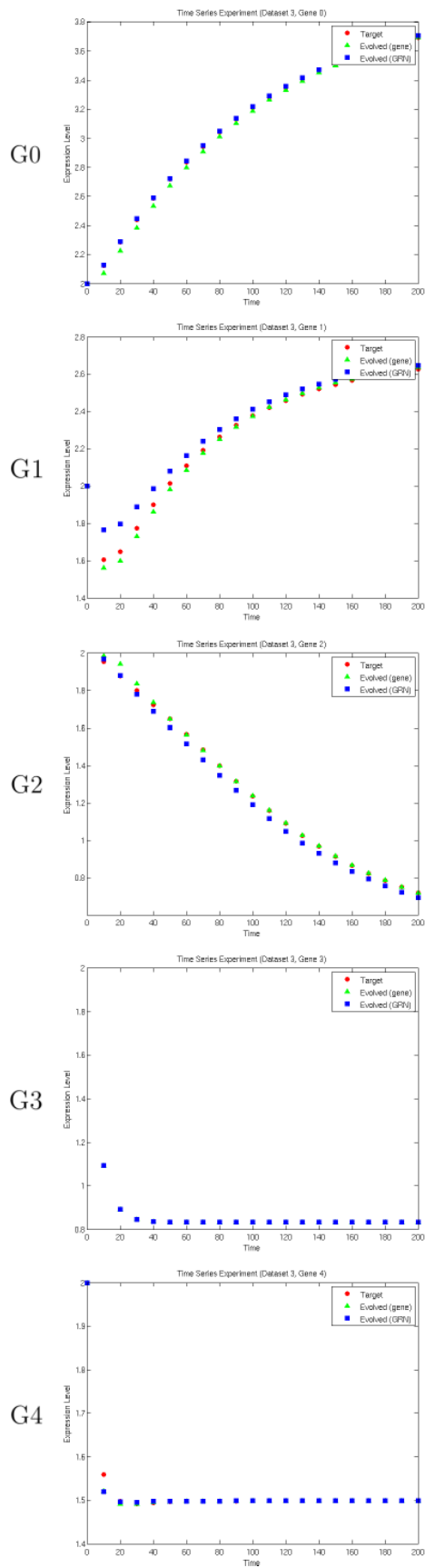
Experiment 1



Experiment 2



Experiment 3



Experiment 4

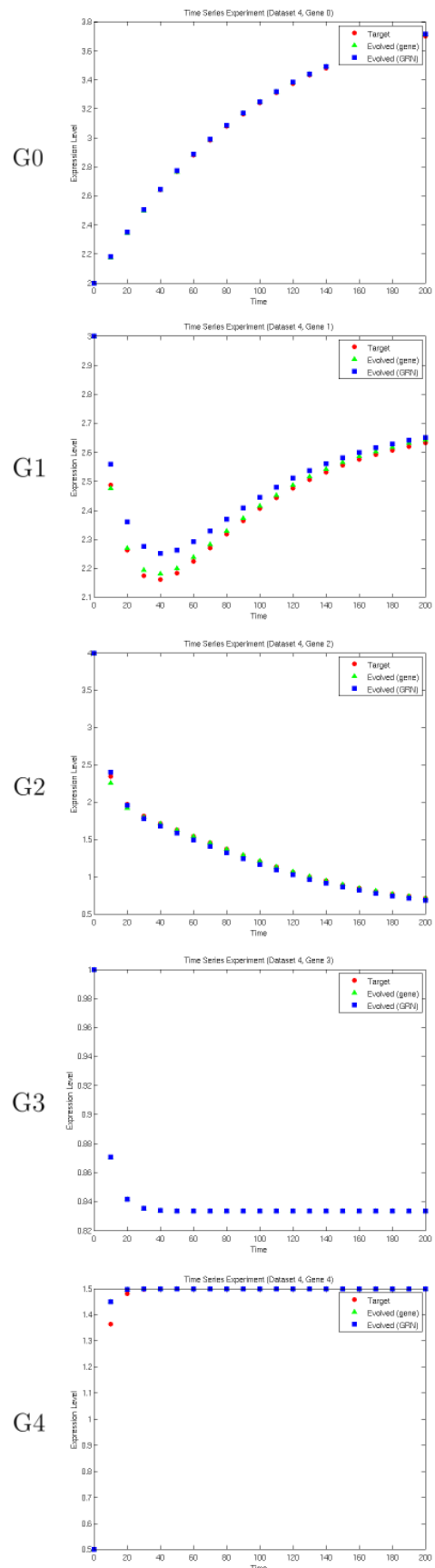


Figure 3.6 Four time series experiments used to reverse engineer the five gene network Grntest. For each experiment, integration curves \bullet are displayed from the in silico target time series dataset. Integration curves \blacksquare are the result of the simulation of the entire found network. This last one is built from best the solutions found in each single-gene optimization. \blacktriangle points are obtained by simulating independently all genes one-by-one as it is calculates during single-gene optimizations. In all time series experiments, single-gene simulated are the best one found. For example, all \blacktriangle points in all G0's graphs are obtained by only simulating the best gene found for G0. All these time series plots are automatically generated by a Matlab tool developed to quickly render HTML reports after reverse engineering entire gene regulatory networks with Wyrd (Annexe B).

3.5.2 Weights threshold

The primary goal is to recover the topology of the underlying gene regulatory network. Due to the numerical process, all interaction's weights found are non-zero. To remove weak interactions, a threshold must be defined. Choose threshold is not obvious. The values of all weights found as presented in Table 3.6 are entered into an histogram as shown in Figure 3.7. As a cluster of weak weights locates between $w = 0$ and $w = 0.3$, threshold $w_{thres} = 0.3$ is selected.

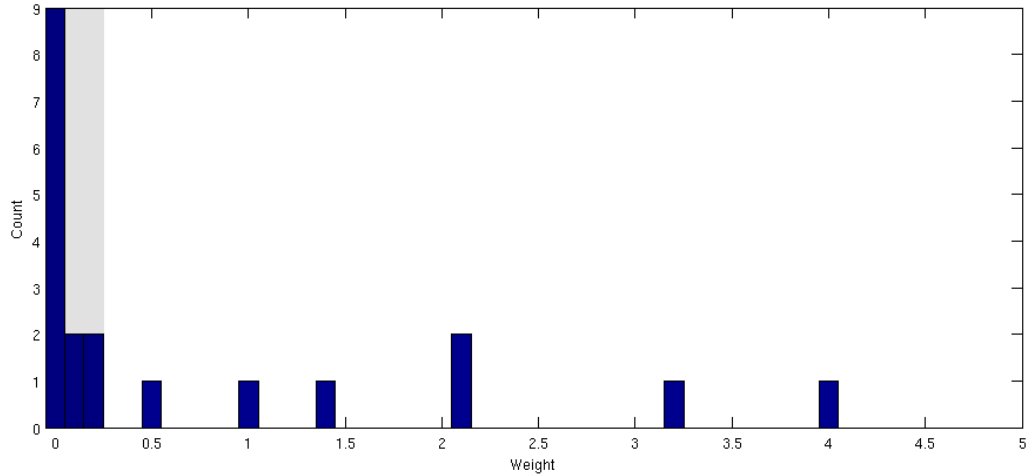


Figure 3.7 Histogram of all weight's values. Weak weights (highlighted within a grey rectangle) are clustered between $w = 0$ and $w = 0.3$. Thus threshold $w_{thres} = 0.3$ is selected to eliminate this cluster of weak weights.

3.5.3 Single-gene optimization of G0

From raw data of Table 3.6 and by applying the weight threshold $w_{thres} = 0.3$, results optimization of G0 are given in Table 3.7.

	δ_i	$v_{max,i}$	w_{i0}	w_{i1}	w_{i2}	w_{i3}	w_{i4}	f	se
Target	0.01	0.1	0	0	0	-0.5	0	3.640e-4	4.112e-4
Found	0.010	0.096	0	0	0	-0.486	0		

Table 3.7 Numerical results of the single-gene optimization G0. $w_{0j} < w_{thres}$ are set to zero. The unique incoming inhibitory interaction is successfully recovered.

After filtered raw weights by setting all weights $w_{0j} < w_{thres}$ to zero, the only gene-gene interaction remaining is an inhibitory connection from gene G3 successfully recovered. The fitness of this optimization is good ($< 10^{-3}$), but it is especially the small value of the final prediction error that confirms the success of the reverse engineering on this gene.

By looking at Figures 3.5 (a), (b) and (c) associated with G0, we see that the 150 optimizations of this gene go towards two distinct solutions. CMA converges 85 times to the local optimum characterized by the final fitness and mean prediction errors (0.027, 47.042). The optimizations that converge towards the better fitness also correspond to the solution with the smallest final prediction error. Time series plots can be a good way to see the correctness of the solution found. In Section 3.5.1, the found gene G0 is simulated independently of the others genes to produce integration curves with marker \blacktriangle . We can see that they fit almost perfectly the target time series data (\bullet). Errors of fitting occurs mainly near $t = 0$. It is here that the variation of mRNA concentration levels $\frac{dx_0}{dt}$ is the highest.

3.5.4 Single-gene optimization of G1

As in previous section, a table with target and found solution is built.

	δ_i	$v_{max,i}$	w_{i0}	w_{i1}	w_{i2}	w_{i3}	w_{i4}	f	se
Target	0.05	0.15	1	0	0	-2	0	1.011e-3	0.417
Found	0.010	0.096	0.989	0	0	-1.370	0		

Table 3.8 Numerical results of the single-gene optimization G1. $w_{1j} < w_{thres}$ are set to zero. Both incoming excitatory and inhibitory interactions are successfully recovered.

The reverse engineering algorithm recovers again successfully both incoming interactions of gene G1. By looking at Figure 3.5 (a), (b) and (c) for G1, we can see that all 150 optimizations converge towards a unique solution. Although the process is a success, the prediction error of the solution found is not excellent and target weight $w_{13}^{tar} = -2$ is only approximated by -1.370 . By returning to Figure 3.5 (c), run 62 approach a final prediction error equal about to 0.02 around generation 190. Figures 3.5 (a) and (c), that represent the evolution of median fitness and prediction error of the population, are reproduced in Figure 3.8 highlighting run 62 in black. Just before generation 190, the median fitness of the population increases when the median prediction error of the population decreases. In both a peaking appears.

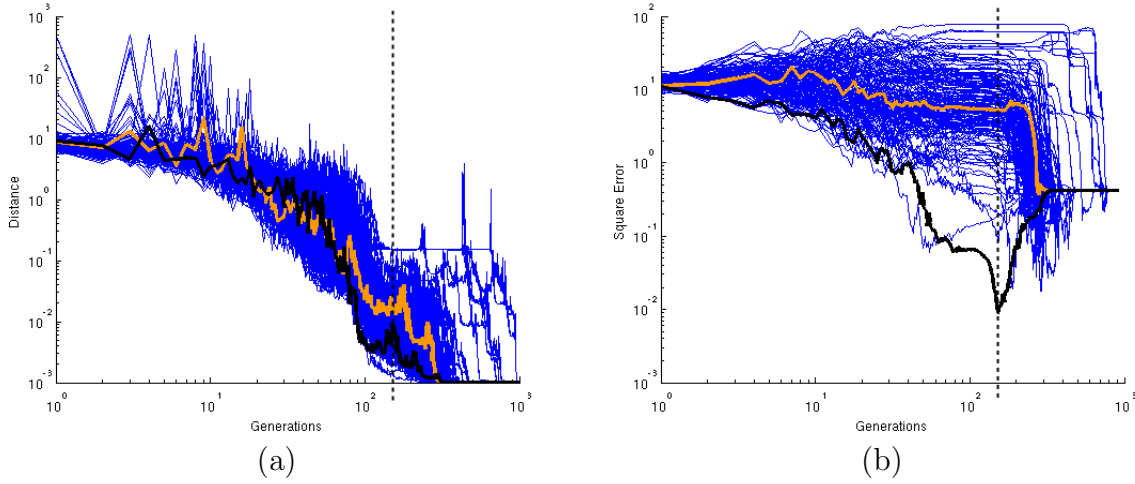


Figure 3.8 Figures 3.5 (a) and (c) are taken again with highlighting run 62 in black. Orange line represent (a) the median fitness and (b) the median prediction error of the population.

This illustrates the fact that the target solution is not necessary the best one for the reverse engineering algorithm where genes are processed independently from others. Here and in all in silico cases, target steady-states and time series datasets are generated by simulating the entire target network as a whole. In the single-gene optimization process, all candidate genes are simulated independently from other ones to produce the same kind of datasets than the given target ones. To do that and in time series experiments, initial conditions for integration are set with initial mRNA concentration levels of the target time series datasets and with points interpolated following a linear interpolation between two given target points, as explained before in Section 2.3.3. So this approximation can lead to some errors.

This effect can be decreased, first by given target time series datasets with more time points (often not available) or use interpolations of degree greater than 1 in algorithm.

3.5.5 Single-gene optimization of G2

	δ_i	$v_{max,i}$	w_{i0}	w_{i1}	w_{i2}	w_{i3}	w_{i4}	f	se
Target	0.1	0.2	-2	0	0	0	4.5	3.13e-3	21.667
Found	0.132	0.263	-2.063	0	0	3.969	2.079		

Table 3.9 Numerical results of the single-gene optimization G2. $w_{2j} < w_{thres}$ are set to zero. Both incoming excitatory and inhibitory interactions w_{20} and w_{24} are successfully recovered. However a false positive interaction is found from gene G3 to G2.

Both incoming interactions from G0 and G4 are successfully recovered. However, a false positive interaction from gene G3 is found this time and contributes significantly to the high final prediction error. Here, all 150 runs converge towards this unique solution as looking in Figures 3.5 (a) and (c). This error can be due to the reason about integration approximation in time series computation, discussed above. It is probably not the fault of

CMA-ES and some local optimum because with 150 runs we can hope that convergence towards the same local optimum does not occurs.

3.5.6 Single-gene optimizations of G3 and G4

	δ_i	$v_{max,i}$	w_{i0}	w_{i1}	w_{i2}	w_{i3}	w_{i4}	f	se
Target	0.15	0.25	0	0	0	0	0	6.824e-9	1.288e-7
Found	0.150	0.250	0	0	0	0	0		

Table 3.10 Numerical results of the single-gene optimization G3. $w_{3j} < w_{thres}$ are set to zero. No incoming gene-gene interaction are found by the reverse engineering algorithm as it is the case in reality. Final fitness and prediction error are both excellent.

	δ_i	$v_{max,i}$	w_{i0}	w_{i1}	w_{i2}	w_{i3}	w_{i4}	f	se
Target	0.2	0.3	0	3	0	0	0	7.173e-4	1.140e-1
Found	0.300	0.450	0	3.22	0	0	0		

Table 3.11 Numerical results of the single-gene optimization G4. $w_{4j} < w_{thres}$ are set to zero. The unique incoming excitatory interaction from G1 is well recovered.

In optimization of gene G3, algorithm converges towards two distincts solutions. The best one shows that with its fitness equal to $6.8 \cdot 10^{-9}$ and especially its prediction error equal to $1.2 \cdot 10^{-7}$, the global optimum is reached. Results of reverse engineering of gene G4 is more humble, but the unique interaction found is correct.

Conclusions

Present reverse engineering algorithm applied on the five gene regulatory network Grn-test gives very good results. With only one false positive interaction detected, the algorithm recovers all real gene-gene interactions with their correct excitatory or inhibitory type. Exact values of weights are not always reached partly due to the first order approximation made at time series level, to help integration in time series data generation in case where only few target time series points are available, as concluded in Section 3.5.4 where the effect is met the first time.

Time computation of the single-gene reverse engineering algorithm gives excellent results. Even if the time depends on the size of the dataset given as input and handled during the entire process, a simple computation time comparison is made between the present algorithm and the biomimetic algorithm based on Analog Genetic Encoding (AGE) developed in [19]. Results show that applications on a five gene regulatory network take respectively 6-8 hours¹ for the biomimetic algorithm versus about one minute for Wyrld to infer the target network. We hope that this speedup will allow us to reverse engineer large networks as the 50 gene network given as challenge in the second DREAM (Dialogue for Reverse Engineering Assessments and Methods) conference, which will be inferred in a close future.

Even if results of the five gene in silico experiment are good, some precisions shall be added. First, all genes of the in silico target network built by hand use the standard sigmoid model to generate target datasets. So as this model is the only one currently implemented in Wyrld, reverse engineering algorithm generates evolved networks with genes following the sigmoid model, too. This bias is obviously an advantage to the process. In biological cases, biomolecules measured do not follow well known and defined gene models. As sigmoid model used are an approximation of the biological behaviours of gene [2], this difference between approximation and reality, added to the fact that used target datasets are noiseless can lead to some supplementary difficulties to infer gene regulatory networks.

¹Both algorithms runned on Intel®Pentium®4 2.8GHz, 1GB RAM

More, all evolved parameters are bounded in the single-gene optimization (Section 2.5.3). As these last ones are known and have not been changed from the generation of the *in silico* target network to reverse engineering of it, no problem occurs related to this. However if the target network is unknown as it is the case in biological networks, parameters of *in silico* gene model shall be evaluated. If lower and upper bounds are far one from other, search space of parameters are consecutively large. If bounds are close to avoid large search space, there is a risk that the optimal parameter is out of range. In a more general way, all experiment parameters as CMA-ES ones need to be studied carefully to found optimal values to set.

As the five gene network used to evaluate the efficiency of the present algorithm is not a gold standard, a future application on the fifty gene network challenge made available for the second DREAM conference will be done soon. To infer this *in vivo* network, wild type and knockout (null-mutant) steady-states experiments are available, as 22 time series experiments with each 26 time points. Due to the fact that the DREAM2 challenges are over for a while, many supports are available as results of participating teams, topology of the target fifty gene network and information about the way that the results are calculated.

Criteria used in DREAM2 challenges to assess reverse engineering algorithms results are Areas Under Curve of the Precision-Recall (PR) and Receiver Operating Characteristics (ROC) curves. These two widely used measures are very important to evaluate search engine such as Information Retrieval and statistical classification and other binary decision problems in machine learning [4]. David shows that a deep connection links PR and ROC curves and proves that PR gives more information than ROC. To build such plots in the present case, two distinct predictions lists must be written: one for each excitatory and inhibitory gene-gene interactions. For all possible connections in one regulatory network, a confidence level $\in [0, 1]$ is defined from reverse engineering results. For each list, a PR curve can be drawn, which shows how the number of correctly classified positive samples varies with the number of incorrectly classified negative samples². So the efficiency of the present reverse engineering algorithm on the fifty gene network challenge may be compared to published results of other algorithms used by challenging teams.

²Matlab tools has been implemented to write prediction lists, draw PR and/or ROC curves and computes AUC from Wyrd results. Please refer to Annexe B.4

5

Acknowledgements

I would like to special thank my responsible assistant Daniel Marbach for the support he provided to me weeks after weeks in the realization of this research, as well as his constant available and for the rereading of the present thesis report.

My gratitude goes also to Dario Floreano, Sven Bergmann and Claudio Mattiussi for their participation in this Master Thesis.

To James N. Knight, thanks for his efficient implementation of CMA and L-CMA-ES and for discussion about this Evolutionary Algorithm.

Finally, thanks to Peter Dürri for his help in the use and strategies about CMA-ES, and to all the LIS collaborators.

Lausanne, January 18th, 2008

Thomas Schaffter

CD's most played during writing report: Armand Amar - Songs from a world apart, Anouar Brahem - Le pas du chat noir, The Lord of the Rings - The Fellowship of the Ring, The Lord of the Rings - The Return of the King, Loreena McKennitt - An Ancient Muse.

Bibliography

- [1] P Brazhnik. Inferring gene networks from steady-state response to single-gene perturbations. *Journal of Theoretical Biology*, 237:427–440, 2005.
- [2] Kuang-Chi Chen, Tse-Yi Wang, Huei-Hun Tseng, Chi-Ying F. Huang, and Cheng-Yan Kao. A stochastic differential equation model for quantifying transcriptional regulatory network in *saccharomyces cerevisiae*. *Bioinformatics*, 21(12):2883–2890, 2005.
- [3] C. Corne, D. and Pridgeon. Investigating issues in the reconstructability of genetic regulatory networks. In *Evolutionary Computation, 2004. CEC2004. Congress on*, volume 1, pages 582–589, June 2004.
- [4] Jesse Davis and Mark Goadrich. The relationship between precision-recall and roc curves. In *ICML '06: Proceedings of the 23rd international conference on Machine learning*, pages 233–240, New York, NY, USA, 2006. ACM.
- [5] P. D’haeseleer, X. Wen, S. Fuhrman, and R. Somogyi. Linear modeling of mrna expression levels during cns development and injury, 1999.
- [6] Nir Friedman, Iftach Nachman, and Dana Peér. Learning bayesian network structure from massive datasets: The ”sparse candidate” algorithm. pages 206–215, 1999.
- [7] Timothy S. Gardner, Diego di Bernardo, David Lorenz, and James J. Collins. Inferring genetic networks and identifying compound mode of action via expression profiling. *American Association for the Advancement of Science (AAAS)*, 301:102–105, July 2003.
- [8] Timothy S. Gardner and Jeremiah J. Faith. Reverse-engineering transcription control networks. *Physics of Life Reviews*, 2:65–88, January 2005.
- [9] Nabil Guelzim, Samuele Bottani, Paul Bourguine, and Franois Kps. Topological and causal structure of the yeast transcriptional regulatory network. *Nature Genetics*, 31:60–63, April 2002.
- [10] Nikolaus Hansen. Tutorial: Covariance matrix adaptation (cma) evolution strategy. Slides, September 2006.
- [11] Nikolaus Hansen. The cma evolution strategy: A tutorial, August 2007.
- [12] Nikolaus Hansen and Stefan Kern. *Parallel Problem Solving from Nature - PPSN VIII*, volume 3242/2004, chapter Evaluating the CMA Evolution Strategy on Multimodal Test Functions, pages 282–291. Springer Berlin / Heidelberg, 2004.

- [13] Nikolaus Hansen, Sibylle D. Mller, and Petros Koumoutsakos. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es). *Evolutionary Computation*, 11(1):1–18, 2003.
- [14] Christian Igel and Nikolaus Hansen. Covariance matrix adaptation for multi-objective optimization. *Evolutionary Computation*, 15(1):1–28, 2007.
- [15] Johannes F. Knabe, Chrystopher L. Nehaniv, and Maria J. Schilstra. Evolutionary robustness of differentiation in genetic regulatory networks. In Stefan Artman and Peter Dittrich, editors, *Proceedings of the 7th German Workshop on Artificial Life (GWAL-7)*, pages 75–84, Jena, 2006. Akademische Verlagsgesellschaft Aka, Berlin.
- [16] James N. Knight and Monte Lunacek. Reducing the space-time complexity of the cma-es. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 658–665, New York, NY, USA, 2007. ACM.
- [17] Ljung Lennart. System identification, May 1995.
- [18] Hong-Wu Ma, Bharani Kumar, Uta Ditges, Florian Gunzer, Jan Buer, and An-Ping Zeng. An extended transcriptional regulatory network of escherichia coli and analysis of its hierarchical structure and network motifs. *Nucleic Acids Research*, 32(22):6643–6649, December 2004.
- [19] Daniel Marbach, Claudio Mattiussi, and Dario Floreano. Bio-mimetic Evolutionary Reverse Engineering of Genetic Regulatory Networks. In *5th European Conference on Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics (EvoBIO 2007)*, pages 155–165, 2007. Editors: E. Marchiori J. H. Moore J. C. Rajapakse (Eds.) Publisher: Springer-Verlag Berlin Heidelberg.
- [20] Claudio Mattiussi. *Evolutionary Synthesis of Analog Networks*. PhD thesis, Ecole Polytechnique Fdrale de Lausanne, 2005.
- [21] P. Mendes. Numerical biology: uses of in silico biochemical networks.
- [22] P. Mendes. Developing strategies for systems biology. In *Virginia Bioinformatics Institute 2005 Scientific Annual Report*, 2005.
- [23] P. Mendes, W. Sha, and Ye K. Artificial gene networks for objective comparison of analysis algorithms. In *Bioinformatics*, volume 19, pages 122–129(8). Oxford University Press, September 2003.
- [24] Nasimul Noman and Hitoshi Iba. Inference of gene regulatory networks using s-system and differential evolution. In *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 439–446, New York, NY, USA, 2005. ACM.
- [25] John Reinitz and David H. Sharp. Mechanism of eve stripe formation. *Mechanisms of Development*, 49:133–158, 1995.
- [26] David M. Rocke and Blythe Durbin. A model for measurement error for gene expression arrays. *Journal of Computational Biology*, 8(6):557–569, 2001.

- [27] D. Thu and Z. S. Qui. Structural comparison of metabolic networks in selected single cell organisms. *BMC Bioinformatics*, 6(8), 2005.
- [28] T. Van den Bulcke, K. Van Leemput, B. Naudts, P. Van Remortel, H. Ma, A. Verschoren, B. De Moor, and K. Marchal. Syntren: a generator of synthetic gene expression data for design and analysis of structure learning algorithms. *BMC Bioinformatics*, 7(43), 2006.
- [29] E. P. van Someren, L.F.A. Wessels, M.J.T. Reinders, and E. Backer. Searching for limited connectivity in genetic network models, 2002.
- [30] M. Wahde, J. Hertz, and M.L. Andersson. Reverse engineering of sparsely connected genetic regulatory networks. In *Proc. Fifth Annual Inter. Conf. on Computational Molecular Biology (RECOMB)*, 2001.
- [31] M. K. S. Yeung, J. Tegnér, and J. J. Collins. Reverse engineering gene networks using singular value decomposition and robust regression. *Proceedings of the National Academy of Sciences of the United States of America*, 99(9):6163–6168, Mars 2002.
- [32] Yun Zheng and Chee Keong Kwoh. Identifying simple discriminatory gene vectors with an information theory approach. In IEEE, editor, *Computational Systems Bioinformatics Conference*, volume 1 of 8-11, pages 13–24, 2005.

A

Wyrd Handbook

A.1 Installation

A.1.1 Requirements

Wyrd is implemented in order to make maximum use of standard libraries already installed on a complete Linux distribution like openSUSE, Mandriva or Ubuntu that are opensource operating systems. However to only run Wyrd (from binary, in opposition to compile it from sources), you will be required to have at least the following additional components installed on your system.

- **libboost_program_options ($\geq 1.33.1$)**
Already installed as part of Boost C++ Libraries on many Linux distributions¹.
- **libxml++ (≥ 2.6)**
C++ Interface for XML Files

A.1.2 Compilation from sources

If you wish to build Wyrd from sources, the following dependencies must be first installed.

- **Bindings Library of Boost Sandbox**
It is not an official part of Boost, but is available from the Boost official web page. There is other stuff in the sandbox, but only the bindings library is used.
- **BLAS and LAPACK**
Probably already installed on most of Linux distributions.

These components are needed to compile the CMA-ES library used in Wyrd and originally implemented by J.N. Knight and M. Lunacek [16]. Several changes have been made to ensure the stability of the algorithm and its interaction with Wyrd.

¹After installation, it is possible to have to create a symbolic link from `libboost_program_options.so` to `libboost_program_options.so.x.xx.x`. where `x.xx.x` is the actual version, for example `1.33.1`.

If KDevelop is present on your system, you can directly open the project file `wyrd.kdevelop` with it and build Wyrd project through **Build** → **Build Project**.

If you do not have KDevelop installed, Wyrd compilation is also possible in command lines. In the root source files directory of Wyrd, *aclocal* without argument generates `aclocal.m4` which contains macro necessary for further actions. Then, *autoconf* without argument will build the configure script. *autoheader* without argument and *automake -a -c* create respectively the `config.h.in` file necessary to the `config.h` file and build Makefiles. Finally, *./configure* and *make* build the Wyrd binary. If there is no error, Wyrd is ready for use.

A.2 Introduction

Wyrd is designed to study non-linear Gene Regulatory Network (GRN), where the challenge is to find topologies - the gene-gene interactions - and gene's parameters of real, biological GRNs from measurements of mRNA levels of the genes that compose them. For complete theory of the reverse engineering process implemented in Wyrd, please refer to Chapter 2 of this report.

Wyrd implements the three following key features:

- **Generation of GRNs**
Generates fully determined GRNs and saves the into XML files.
- **Simulation of GRNs**
Takes fully determined GRNs in XML format as input, simulates them and so generates steady-states or time series datasets of gene's mRNA levels.
- **Reverse engineering of underlying GRNs**
Takes one or several datasets and try to recover the associated underlying GRN, which generated these datasets.

A.3 Quickstart

A.3.1 User Interface

The user can interact with Wyrd through two ways: the command line options and the configuration file. An example of configuration file is given at the end of this manual. The `-config-file` or `-c` flag allows to specify to Wyrd which file it must use as configuration file. This flag is the only one that must be always given to the program.

The configuration file contains all parameters, for example the integration and calculation or the evolution parameters. All this different settings must be filled with a valid value, otherwise there is an error message.

Some parameters of the configuration file can be directly given to Wyrd through command line options. In this case, the value of parameter given through this way overwrites the corresponding value in the configuration file.

```
$ ./wyrd -c configuration.cfg
```

Figure A.1 The specification of the configuration file and the shortest way to run Wyrd.

A.3.2 Project Id and filenames convention

A project Id, also called process Id, is necessary to build all input and output filenames used by Wyrd. First, this allows to differentiate the results of multiple runs of the program, and second, taking advantage of this filenames standardisation to run Wyrd in non-interactive mode using `-interactiveMode 0` or `-i 0` flags. I.e. run the program without any user's additional action. The project Id can be defined in the configuration file or using the `-pId` or `-p` flags. If no project Id is specified in later after launching the program, it will be one of the first data asked to the user. Supposing that the process Id is "key", the following filenames will be asked as input or created in order to save output data.

- **key.xml and key_evo.xml**

Both refer to a fully specified GRN file in XML format. `key.xml` is the output GRN in GRNs generation mode. In simulation mode, `key.xml` is the input GRN from which steady-states and time series datasets are generated. In reverse engineering mode, this file can be given as the target GRN (optional). In this last mode, `key_evo.xml` refers to the evolved GRN topology found after optimization processes.

- **key_ss_wt.csv, key_ss_wt_evo.csv and key_ss_wt_evo2.csv**

In simulation mode, `key_ss_wt.csv` is the wild type steady-states dataset generated from a fully specified GRN, always in XML format. In reverse engineering mode, `key_ss_wt.csv` can be one of the input datasets if wild type steady-states measurements are available. In this last mode, `key_ss_wt_evo.csv` is obtained when the entire found GRN is simulated, while `key_ss_wt_evo2.csv` is the result of the independent gene's simulations.

- **key_ss_ko.csv, key_ss_ko_evo.csv and key_ss_ko_evo2.csv**

Idem that the previous item for knockout steady-states datasets.

- **key_ss_hz.csv, key_ss_hz_evo.csv and key_ss_hz_evo2.csv**

Idem that the previous item for heterozygous steady-states datasets.

- **key_ts.csv, key_ts_evo.csv and key_ts_evo2.csv**

Idem that previous item for time series datasets.

- **key_ts_init.csv**

Contains initial mRNA concentrations for time series experiments. This file is used only in simulation mode.

- **key_log_dist_g*.stat, id_log_se_g*.stat**

In reverse engineering mode, the fitness of all individuals are saved in `key_log_dist_g*.stat`.

In the same mode and if target GRN is available, square errors between all individuals (solutions) and the target solution are saved in `id_log_se_g*.stat`. `*` refers to the index of the current gene evolved, $* \in [0, numGenes - 1]$.

- **key_log_bOptim.stat**

In this file, each line contains information about the best run, regardless of fitness, of a gene's optimization. Elements saved in this file are among others the index of the best run, the best fitness, the associated square error (if GRN target available) or still the found solution (gene's parameters).

A.4 Generation of Gene Regulatory Networks

A.4.1 Preface

If someone without Wyrd's specific resource wants to experiment Wyrd, the first step is to create a fully specified Gene Regulatory Network (GRN). In generation mode, this is possible through two ways.

- **Interactive mode, `-i 1`**

Select the first main menu "Generate a fully specified GRN from a selected test case or random network." (choice 0) and follow the indications.

- **Non-interactive mode, `-i 0`**

Here, even if `-i 0` is enable, some actions from the user are required if the output GRN is a random one. In this case, Wyrd needs specific data to generate random networks, as presented in Section A.4.3. The command line option `--TestCase choice` or `-t choice` or the parameter `testCase` set to `choice` in the config file allow to pre-select what type of networks Wyrd should generate. `choice` can take the following values:

- 0 Broken Repressilator (Section A.4.2)
- 1 Grntest (Section A.4.2)
- 2 Random networks (Section A.4.3)

A project identifiant is needed and used to build the name of the output XML file in which the GRN topology will be saved.

A.4.2 Broken repressilator and Grntest

The two fully specified networks are represented in Table A.1. The broken repressilator is composed of five genes and four inhibitor interactions. Originally, this topology had an extra connection between gene "g0" and "g4", that was deleted in order to remove the oscillatory state involved and so to allow the possible presence of steady-states. The Grntest network has five genes, too, and has six gene-gene interactions of the two excitatory and inhibitory types. All genes of both networks follow the sigmod model introduced in Section 2.3.2, which is actually the only gene's model implemented in Wyrd.

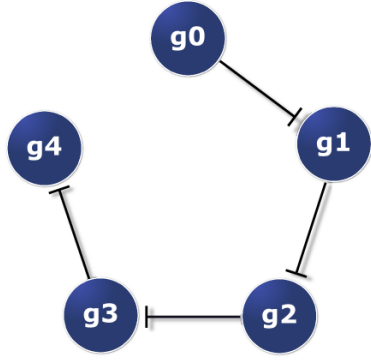
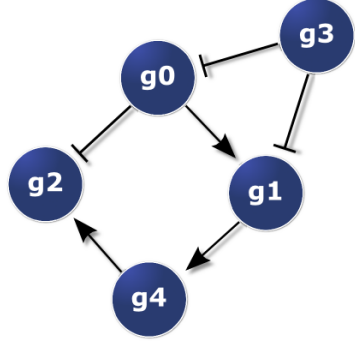
Broken Repressilator					Grntest			
								
	v_{max}	α	β	δ	v_{max}	α	β	δ
g0	0.90	1.00	3.20	0.15	0.10	1.00	0.00	0.01
g1	1.10	1.00	2.20	0.20	0.15	1.00	0.00	0.05
g2	0.60	1.00	5.00	0.09	0.20	1.00	0.00	0.10
g3	1.10	1.00	3.70	0.12	0.25	1.00	0.00	0.15
g4	1.00	1.00	4.00	0.10	0.30	1.00	0.00	0.20
$\mathbf{w}^2 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ -4.1 & 0 & 0 & 0 & 0 \\ 0 & -4.3 & 0 & 0 & 0 \\ 0 & 0 & -3.9 & 0 & 0 \\ 0 & 0 & 0 & -3.3 & 0 \end{pmatrix}$					$\mathbf{w} = \begin{pmatrix} 0 & 0 & 0 & -0.5 & 0 \\ 1 & 0 & 0 & -2 & 0 \\ -2 & 0 & 0 & 0 & 4.5 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 \end{pmatrix}$			

Table A.1 The two types of fully determined GRNs available in the current version of Wyrd. Both contain five genes and all gene's behaviors follow the sigmoid model. This model uses four parameters: the maximum transcription rate v_{max} , a multiplicative constant for the regulatory activity (stepness of sigmoid) α , the basal transcription (where the sigmoid crosses the y-axis) β and a decay constant δ . Wyrd can also generate random networks.

A.4.3 Random networks

Wyrd is able to generate random network. If the user wants to build one, Wyrd needs the following parameters.

- **A network size, numGenes**
- **The two parameters of a normal distribution, μ and σ**

The strategy of networks building is the next one. First, *numGenes* genes are created with gene's parameters filled with random values taken in uniform real distributions bounded by the corresponding parameters (v_{max} , δ , etc.) define in the settings file. To

¹Element w_{ij} in the i^{th} line, j^{th} column of the weights matrix \mathbf{w} represents the excitatory ($w_{ij} > 0$) or inhibitory ($w_{ij} < 0$) interaction from gene j to gene i .

set the topology, different numbers of incoming interactions *numInteractions* are defined for each gene using the normal distribution, centred in μ with standard deviation σ . These two parameters may be floating. Random floating numbers taken by the random numbers generator in this distribution are rounded to integral value, regardless of rounding direction. Then, the following sub-strategy is repeated *numInteractions* times: an integral random number is taken in the interval $[1, numGenes]$, using uniform distribution, which defines the index of the gene responsible of the incoming connection. Finally, a floating random number is taken in a real uniform distribution $[-maxWeight, maxWeight]$ (*maxWeight* is defined in the config file) to give a strength to this incoming interaction. If *selfLoop* is set to 0, no gene auto-loop will be placed. Figure A.2 displays an example of random networks generated by WyrD.

```
The process ID is not yet defined. Process
ID is:
```

```
>> eighteen
```

```
Test Case Generation
```

```
=====
```

```
What do you want to process?
```

```
(0) Generate broken Repressilator
```

```
(1) Generate GrnTest
```

```
(2) Generate a random network
```

```
>> 2
```

```
Please specify the network size:
```

```
>> 18
```

```
Number of incoming gene interactions fol-
lows a normal distribution. Please provide
the following information.
```

```
Mean (mu):
```

```
>> 2
```

```
Standard deviation (sigma):
```

```
>> 1
```

```
XmlGRN::save: grn file eighteen.xml is cor-
rectly saved!
```

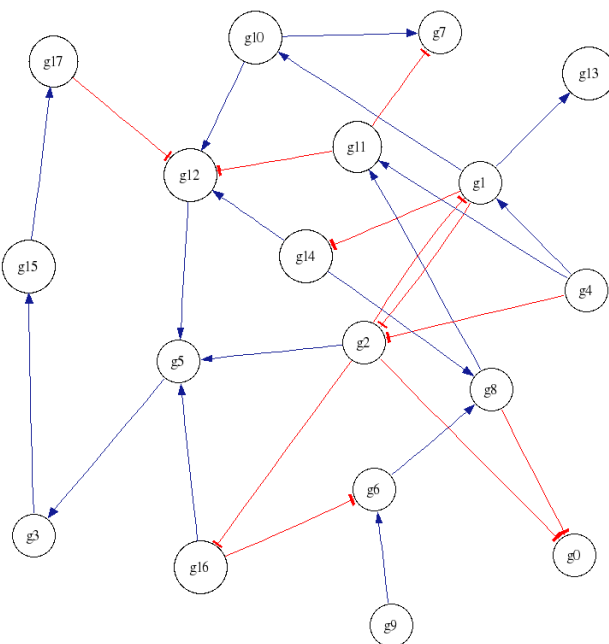


Figure A.2 Example of generation of random networks in WyrD. The given parameters are 18 for the network size and the couple of values (2, 1) to define the normal distribution used to set the number of gene's incoming interactions, different from a gene to another. The parameter *rngSeed* is used, so for the same three previous parameters (*numGenes*, μ , σ) and the same seed, the random network generated is always the same. The seed used in this example is 9.

It is important to highlight the fact that such random networks do not reflect the properties of biological ones. In domain of in silico networks generation, two main types of topologies have been studied and are used in practice to generate networks that display some biological networks properties. The first one is known as "small-world" due to the average distance between any vertices in the graph which is small. It has been shown that the metabolic network E.coli displays this property. The second kind of topologies

is named "scale-free" following the work of Barabasi and Albert (1999) because the majority of genes have only few interactions with others, during a minority of them have a very large number of connections. A more detailed overview is presented by Mendes in [23].

A.4.4 XML structure for Wyrd GRNs files

In Wyrd, all the handled networks are saved in XML format. This format was chosen because desired data are structured. This allows to find at a glance any data in the XML file if this last one is open with a text editor. To structure data, the following tags are used.

- **<GRN id="grnId">**
One per file, this tag includes all the parameters of the network. A GRN identifier can be defined by the attribute *id*.
 - **<numGenes>value</numGenes>**
Contains the number of genes in the network.
 - **<gene model="sigmoid" id="g0">**
This tag includes all gene's parameters as well as incoming interactions. There is *numGenes* **<gene>** blocs one after another. The attribute *model* gives the gene's model. Here it is the sigmoid model which is used. The attribute *id* allows to give a name to all genes of the network.
 - * **<vmax>value</vmax>**
Here are defined the parameters of each gene as the maximum transcription rate v_{max} . For example, sigmoid model still contains α , β , δ . A parameter *c0* may be present, too. It may be used to save a particular value of mRNA concentration (e.g. gene's steady-state).
 - * **<interactions>**
Contains the **<weight>** tags. There is *numGenes* tags **<weight>** include in this tag.
 - **<weight>value</weight>**
This group of tags represent the strength of the incoming interactions. If an interaction is inexistant, the corresponding value is set to 0.
 - * **</interactions>**
 - **</gene>**
- **</GRN>**

A.4.5 Initial mRNA concentrations for Time Series experiments

As discussed later, the generation of time series experiments needs initial points from which the integrations of the $\frac{dx}{dt}$, where x represents mRNA concentrations, start. After generating a full specified GRN in generation mode, Wyrd may still generate a time series file with initial mRNA concentrations for each gene and each experiment, which can directly be used as it stands in time series simulation mode or modified thereafter

by the user. This file is created if the parameter *ts0* is set to 1 in the configuration file. In this case, the number of initial points, whose dimension is equal to the number of genes in the network, is the number of time series experiments defined by the parameters *numTimeSeries*. The initial mRNA concentrations of all genes are taken randomly in a uniform real interval $[0, ts0MaxC0]$ where the parameters *ts0MaxC0* is also defined in the settings file.

The name of this file is built from the process Id as *key_ts_init.csv*. The complete description of this file is given in Section A.5.2.

A.5 Simulation of GRNs

A.5.1 Introduction

Simulation of networks consists to generate gene's expression levels or mRNA concentrations datasets. Wyrd allows to generate both types of the most commonly used data: the wild type, knockout and heterozygous steady-states and time series datasets. To generate datasets, the user needs a fully specified GRN file in XML format as input, with the same structure as presented in Section A.4.4. More generally, Section A.4 introduces the user to the generation of such networks. In Wyrd, the simulation mode can be called through two ways.

- **Interactive mode, -i 1**

Select the second item in the main menu, "Simulate a fully specified Genes Regulatory Network (GRN) and generate datasets." (choice 1) and follow the indications.

- **Non-interactive mode, -i 0**

It is also possible to run the program without that any additional action are required from the user. So the following information should be specified through the command line options or via the settings file before run Wyrd.

- *runMode* = 1 in settings file or flag *-r 1* in command line.
Run the program in simulation mode.
- *simulationMode* = *choice* in settings file or flag *-s choice* in command line. In simulation mode, this parameter defines what to do. *choice* takes the following values.
 - * 0 Generation of time series datasets and possibility to save them into one CSV file. ³
 - * 1 Generation of wild type, knockout and heterozygous steady-states datasets and possibility to save them into three distinct CSV files.
 - * 2 Computation of the wild type steady-state and display it.
- *-p key*
A project Id is necessary to build filenames of all input and output files.

³The user can set Wyrd to display more information during processes as current state of datasets. To enable this functionality, set *displayDatasets* to 1 in the configuration file.

So, the command line shown in Figure A.3 runs Wyrd in non-interactive (*-i 0*) and simulation mode (*-r 1*), the program computes all steady-states datasets from the network *eighteen.xml* (*-s 1, -p eighteen*) and saves them into three different files: *eighteen_ss_wt.csv*, *eighteen_ss_ko.csv* and *eighteen_ss_hz.csv* without other action from the user than to run Wyrd.

```
$ ./wyrd -c configuration.cfg -i 0 -r 1 -s 1 -p eighteen
```

Figure A.3 Example of the use of the simulation mode in Wyrd. This command runs Wyrd in non-interactive (*-i 0*) and simulation mode (*-r 1*), the program computes all steady-states datasets from the network *eighteen.xml* (*-s 1, -p eighteen*) and saves them into three different files: *eighteen_ss_wt.csv*, *eighteen_ss_ko.csv* and *eighteen_ss_hz.csv* without other action from the user than to run Wyrd.

A.5.2 Time Series datasets

Introduction

A time series experiment is defined as the evolution of all gene's mRNA concentrations from an initial point (expression levels of all genes at $t = t_0$) until the establishment of a steady-state, if it exists. For more theory aspect of time series data, please refer to Section 2.3.3.

Setup

To generate time series datasets from a fully specified GRN in XML format, the procedure is the following one.

1. **Set numTimeSeries** (settings file)
Set the number of time series experiments that Wyrd should generate.
2. **Set numTimePoints** (settings file)
The number of points taken between t_0 and t_{max} . The distance between two points represents the integration step. More *numTimePoints* is large, more integration curves contain information and are accurate, and more the generated time series dataset will help to reverse engineer the gene regulatory network. This property is shown in Figure A.4.
3. **Create an initial mRNA concentrations file**
The establishment of a steady-state in the network after a period of time depends, among others, on the initial conditions, like the initial expression levels of all genes of the network. So, to generate different time series experiments, Wyrd needs a file containing all the *numTimeSeries* initial point, with dimension equal to the gene network's size, in order to compute *numTimeSeries* different experiments. The structure of this file is presented in the next section.

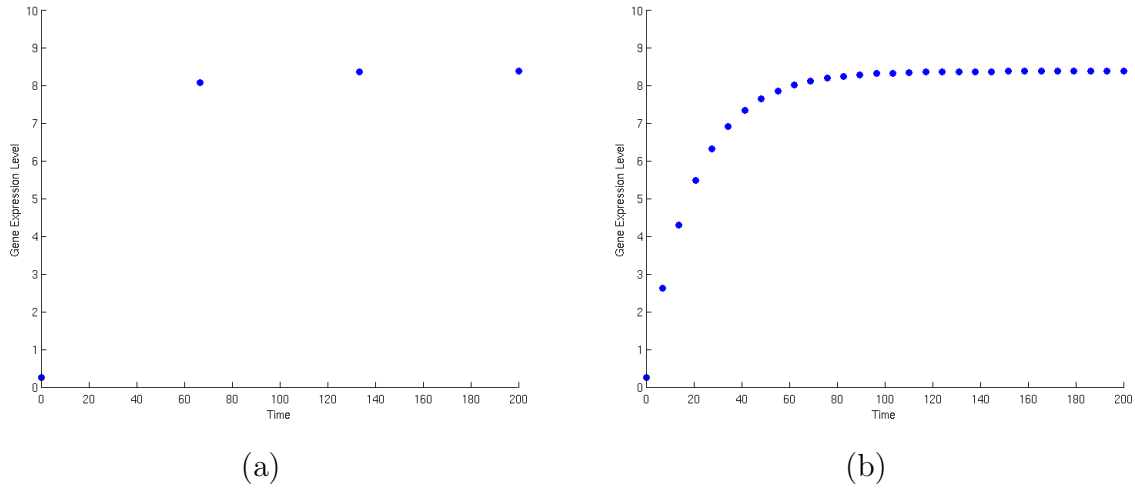


Figure A.4 The integration curve of $\frac{dx}{dt}$ for one gene between $t_0 = 0$ and $t_{max} = 200$ is displayed respectively for **a)** $numTimePoints = 4$ and **b)** $numTimePoints = 30$. The second one contains clearly more information about the gene network.

Note: If the network to simulate has been generated by Wyrld and if during this process the parameter $ts0$ was set to 1 in config file, so Wyrld has also generates an initial random mRNA concentrations file with $numTimeSeries$ points (Section A.4.5).

Initial mRNA concentrations file

To generate time series experiments in simulation mode, Wyrld needs initial expression levels for each gene of the network and for each experiment as input. This filename is defined as `key_ts_init.csv` if *key* is the project Id. The file contains $numTimeSeries$ lines with each containing $numGenes$ real values separated by a tab. For a given gene regulatory network of size five, an example of a content of `key_ts_init.csv` is given Table A.2.

0.1037	3.645	5.019	4.991	4.958
0.07543	1.338	3.162	1.421	0.1332
2.186	8.772	4.185	1.375	2.481

Table A.2 Example of a content of `key_ts_init.csv` for the five gene regulatory network Grntest. The number of experiments $numTimeSeries$ is 3 and the dimension of all initial points is $numGenes$ equal to 5 in the GRN Grntest.

Generation of Time Series datasets

When setup is clear, Wyrld can generate the time series dataset `key_ts.csv` from the network `key.xml` and the initial concentrations file `key_ts_init.csv`. Here again, two ways are possible.

- **Interactive mode, $-i\ 1$**

Select the second main menu, "Generate a fully specified GRN from a selected test case or random network." (choice 1), follow the indications, enter the project Id if it is not already specified, and select the first menu of the simulation mode, "Generate Time Series datasets from a fully specified GRN" (choice 0). A duration time t_{max} for integration process will still be asked.

- **Non-interactive mode, $-i\ 0$**

The command line option $-r\ 1$ starts Wyrd in simulation mode. $-s\ 0$ allows to generate time series datasets under simulation mode. A duration time will however be asked to the user after launching Wyrd.

Wyrd will generate *numTimeSeries* experiments. Following the content of *key_ts_init.csv*, two special cases can appear.

- **Number of lines in *key_ts_init.csv* < numTimeSeries**

If *key_ts_init.csv* contains less initial concentration points than the number of experiments, extra experiments will begin from the starting point built with gene's expression levels defined in the GRN XML file *key.xml* stored in tag $\langle c0 \rangle$ (Section A.4.4). So if the value of *numTimeSeries* minus the number of lines in *key_ts_init.csv* is greater than one, all extra experiments will have the same initial point, and the time series experiments will be the same ones providing no more additional information about the network. So it is advisable to have the parameters *numTimeSeries* equal to the number of given initial points in *key_ts_init.csv*, or greater of one to take advantage of the initial solution compiled from the tags $\langle c0 \rangle$ of the GRN file *key.xml*.

- **Number of lines in *key_ts_init.csv* > numTimeSeries**

Only the first *numTimeSeries* initial points of *key_ts_init.csv* will be taken into account.

The command line of Figure A.5 generates a time series dataset from the five gene network *Grntest*. Figure A.6 illustrates the execution of this process.

```
$ ./wyrd -c configuration.cfg -i 0 -r 1 -s 0 -p grntest
```

Figure A.5 Example of simulation of time series dataset from a fully specified GRN file, *grntest.xml*. The time series dataset is saved in the CSV file, *grntest_ts.csv*.

```

Simulation mode
=====
XmlGRN: Grn file grntest.xml is correctly open!
Simulation time:
>> 200
[TS] Loading Initial Concentrations file grntest_ts_init.csv ...
[TS] There are initial concentrations for 3 experiment(s)!
[TS] WARNING: Number of initial concentrations in file grntest_ts_init.csv (3) is not equal
to parameter 'numTimeSeries' in settings file (4, the real number of experiments)!
[TS] Experiment(s) with no given initial concentrations C0 will start with default Gene
Network C0!
[TS] Datasets: grntest_ts.csv is correctly saved!

```

Figure A.6 Illustration of the command displayed in Figure A.5. Even if the non-interactive mode is enable (*-i 0*), a duration time t_{max} for time series integrations is asked to the user.

Structure of time series datasets

The content of the file `grntest_ts.csv` created by the example given in Figure A.5, is displayed in Figure A.3 for the two first time series experiments. All elements on the same line are separated by tab and no empty lines are present.

The first line is the header of the dataset. Its first element, "Time" is used to verify that a given file is well a time series dataset. The other elements in the header are the names of the genes.

The second bloc contains the first time series experiment. The left column contains the time scale, from 0 to the duration time given by the user. Here $t_{max} = 200$. The interval $[0, t_{max}]$ is divided into *numTimePoints*, here equals to 13. mRNA concentrations on the line $t = 0$ correspond to the initial concentrations data of the file `key_ts_init.csv`, in bold style in Figure A.3.

A.5.3 Steady-States datasets

Introduction

The network reached a steady-state when the variation of mRNA concentration $\frac{dx_i}{dt}$ approach zero for all genes. The generation of steady-states datasets is to compute the steady-state of the network with some variants in order to obtain a maximum of information about the network through its responses from gene's perturbations. Calculate a steady-state is to solve a system of *numGenes* coupled differential equations. For more information about steady-states theory, please refer to Section 2.3.3.

Wyrd handles the three different types of steady-steates:

- **Wild Type Steady-State**
- **Knockout Steady-States ($v_{max,i} = 0$)**

"Time"	"g0"	"g1"	"g2"	"g3"	"g4"
0	0.1037	3.645	5.019	4.991	4.958
16.67	0.4742	1.666	2.569	1.174	1.618
33.33	0.9892	1.151	2.097	0.8612	1.467
50	1.445	1.17	1.988	0.8354	1.454
66.67	1.833	1.358	1.933	0.8332	1.469
83.33	2.162	1.588	1.87	0.8333	1.485
100	2.44	1.807	1.784	0.8333	1.492
116.7	2.675	1.994	1.672	0.8333	1.496
133.3	2.875	2.147	1.541	0.8333	1.497
150	3.043	2.268	1.401	0.8333	1.498
166.7	3.186	2.362	1.264	0.8333	1.499
183.3	3.307	2.435	1.137	0.8333	1.499
200	3.409	2.492	1.026	0.8333	1.499

0	0.07543	1.338	3.162	1.421	0.1332
16.67	0.6361	0.8709	2.185	0.8815	1.367
33.33	1.145	0.919	2.013	0.8372	1.401
50	1.579	1.133	1.956	0.8331	1.441
66.67	1.947	1.394	1.909	0.8333	1.471
83.33	2.258	1.646	1.843	0.8333	1.487
100	2.521	1.865	1.75	0.8333	1.493
116.7	2.744	2.045	1.63	0.8333	1.496
133.3	2.933	2.188	1.495	0.8333	1.498
150	3.092	2.3	1.355	0.8333	1.498
166.7	3.228	2.388	1.221	0.8333	1.499
183.3	3.342	2.455	1.099	0.8333	1.499
200	3.439	2.508	0.9931	0.8333	1.499

Table A.3 Content of grntest_ts.csv for the five gene regulatory network Grntest. Only the two first time series experiments are displayed. Each experiment contains $numTimePoints \times numGenes$, here 13×5 gene's expression levels. The first left column is the time points scale $[0, t_{max}]$. Gene's expression levels on line $t = 0$ correspond to the initial concentrations data taken from the file grntest_ts_init.csv. These mRNA concentrations are highlight in bold style.

• **Heterozygous Steady-States**($\mathbf{v}_{max,i} \rightarrow \frac{1}{2}\mathbf{v}_{max,i}$)

Where the steady-state i represents a change in the maximal transcription rate v_{max} of the gene i .

Setup

Several parameters must be fixed:

1. **Set SS_abs_precision and SS_rel_precision**

These two parameters are used to evaluate the convergence of the root finding of

the system of *numGenes* coupled equations. The convergence is tested following Equation A.1.

$$|dX_i| < \text{SS_abs_precision} + \text{SS_rel_precision} \cdot |X_i| \quad (\text{A.1})$$

where X_i is the current solution at iteration i and dX_i the difference between $(X_i - X_{i-1})$. More `SS_abs_precision` and `SS_rel_precision` are small, more the number of iterations to solve the system will be large due to the high precision required. In some cases, the required time by a resolution with this two parameters set with very small values may explode for a tiny improvement of the solution X . So a trade-off between these two aspects must be found. A couple of values advised for this two parameters is (0.001, 0.001).

2. Set `maxIterGslSs`

The maximum number of iterations to solve the system of *numGenes* equations. 2000 iterations has been a good choice in many cases.

In order to resolve the system of equations, the choice of initial point is important, so two different initial solutions may very well go toward two different steady-states. But as only a full specified network is taken as input, any indication to choice this initial point is available. In the steady-states simulation mode of Wyrld, the initial point $X_0 = [11111...]^T$ is always taken.

Generation of Steady-States datasets

This section introduces the user how to generate wild type, knockout and heterozygous steady-states from a fully specified GRN in XML format. This three types of steady-states are saved respectively in `key_ss_wt.csv`, `key_ss_ko.csv` and `key_ss_hz.csv` if the input network is contained in the `key.xml` file. As usually, two ways are possible to process a desired operation on Wyrld.

- **Interactive mode, `-i 1`**

Select the second main menu, "Simulate a fully specified Genes Regulatory Network (GRN) and generate datasets." (choice 1), follow the indications, enter the project Id if it is not already specified, and select the second menu of the simulation mode, "Generate Steady-States datasets (wild type, knockout and heterozygous) from a fully specified GRN" (choice 1).

- **Non-interactive mode, `-i 0`**

The command line option `-r 1` starts Wyrld in simulation mode. `-s 1` allows to generate in the same time all three types of steady-states datasets.

To illustrate this use, Figure A.7 and A.8 are displayed.

```
$ ./wyrd -c configuration.cfg -i 0 -r 1 -s 1 -p grntest
```

Figure A.7 This command will runs Wyrd in simulation mode (*-r 1*), generates the wild type, knockout and heterozygous datasets associated to the GRN file *grntest.xml* (*-p grntest*). The three output steady-states datasets are saved into three different files: *key_ss_wt.csv*, *key_ss_ko.csv* and *key_ss_hz.csv*.

```
Simulation mode
=====
XmlGRN: Grn file grntest.xml is correctly open!

[SS] Wild Type SS Experiment:          grntest_ss_wt.csv is correctly saved!
[SS] Knockout SS Experiment:           grntest_ss_ko.csv is correctly saved!
[SS] Heterozygous SS Experiment:       grntest_ss_hz.csv is correctly saved!
```

Figure A.8 Illustration of the command shown in Figure A.7.

Structure of Steady-States datasets

Files that contain wild type steady-states displayed the structure shown in Figure A.4. All elements on the same line are separated by tab and no empty lines should be present. Top-left element "Ss" is used as tag to test validity of all wild type steady-state files.

"Ss"	"G0"	"G1"	"G2"	"G3"	"G4"
"wt"	3.973	2.728	0.4636	0.8333	1.5

Table A.4 Content of *grntest_ss_wt.csv* generated by the command in Figure A.7 for the five gene regulatory network *Grntest*.

Knockout and heterozygous steady-states files display a similar structure. The top-left elements are also used as validity tags and are respectively "SsKo" and "SsHz". Steady-states are presented line by line.

"SsKo"	"G0"	"G1"	"G2"	"G3"	"G4"
"G0"	0	0.4766	1.991	0.8333	1.21
"G1"	3.973	0	0.02048	0.8333	0.75
"G2"	3.973	2.728	0	0.8333	1.5
"G3"	5	2.98	0.07462	0	1.5
"G4"	3.973	2.728	0.0007077	0.8333	0

Table A.5 Content of *grntest_ss_ko.csv* generated by the command in Figure A.7 for the five gene regulatory network *Grntest*. Knockout steady-states are presented line by line.

"SsHz"	"G0"	"G1"	"G2"	"G3"	"G4"
"G0"	1.987	1.738	1.879	0.8333	1.492
"G1"	3.973	1.364	0.4259	0.8333	1.475
"G2"	3.973	2.728	0.2318	0.8333	1.5
"G3"	4.481	2.924	0.1972	0.4167	1.5
"G4"	3.973	2.728	0.02046	0.8333	0.7498

Table A.6 Content of `grntest_ss_hz.csv` generated by the command in Figure A.7 for the five gene regulatory network `Grntest`. Heterozygous steady-states are presented line by line.

Wyrd used three separated files because it is possible that one or more types of steady-states datasets are not available to reverse engineer a given gene regulatory network.

A.6 Reverse Engineering of GRNs

A.6.1 Introduction

The first goal of Wyrd is to reverse engineer networks from steady-states and/or time series experiments, in order to find the underlying gene regulatory networks that have generated them. Wyrd optimizes gene after gene the whole network following the idea introduces in Section 2.3.3. Instead of solving a system of *numGenes* coupled differential equations, the system is divided into *numGenes* simple equations which are much faster to solve. Thus, Wyrd is one of the few existing algorithms that are able to reverse engineer large and non-linear gene regulatory network.

A.6.2 Prepare inputs

Process Identifiant

The first thing to define is a process identifiant. Once chosen, ensure that all input filenames (steady-states and time series datasets, eventually target GRN in XML file if it is available) follow the filenames standard introduces in Section A.3.2.

Time Series datasets

If available, verify that the content of this file respects the description given in Section A.5.2. The name of this file must be `key_ts.csv` where *key* is the process Id. Time series datasets are not mandatory if one or more steady-states datasets are available.

Steady-States datasets

If available, verify that the content of these files respect the description given in Section A.5.3. The names of these files must be `key_ss_wt.csv`, `key_ss_ko.csv` and `key_ss_hz.csv` where *key* is the process Id for respectively datasets of wild type, knockout and heterozygous steady-states. Steady-states datasets are not mandatory if a file containing one or more time series experiments is available.

Target network

The target network refers to the network that has generated the available steady-states and/or time series datasets. If it is given to Wyrd as input for the reverse engineering, it is possible to record the evolution of the mean square error distance between the found evolved solution (the evolvable gene's parameters) and the target solution through the generations of evolution.

A.6.3 Initialization of parameters

The following parameters are taken into account during the reverse engineering process. All these variables are defined in the settings file.

- **displayDatasets**
Display intermediate results. Like a verbose+ mode.
- **SS_abs_precision and SS_rel_precision**
When an evolved gene is simulate independently in order to generate evolved datasets to compare to the target datasets, these two parameters are used as convergence test during the resolutions of differential equations. The relation A.1 stops the multidimensional root-finding when it is satisfied.
- **maxIterGslSs**
Maximum number of iterations to solve systems of equations (use GSL⁴).
- **selfLoop**⁵ Specify to Wyrd if gene's self interaction are possible.
- **maxWeight (evolvable param)**
The range [-maxWeight, maxWeight] defines the space search for the interactions weights.
- **minMax and maxMax (sigmoid model)(evolvable param)**
The range [minMax, maxMax] defines the space search for the maximum transcription rate v_{max} .
- **minAlpha and maxAlpha (sigmoid model)(evolvable param)**
The range [minAlpha, maxAlpha] defines the search space for the multiplicative constant for the regulatory activity α (steepness of sigmoid).
- **minBeta and maxBeta (sigmoid model)(evolvable param)**
The range [minBeta, maxBeta] defines the space search for the basal transcription β (where the sigmoid crosses the y-axis).
- **minDelta and maxDelta (sigmoid model)(evolvable param)**
The range [minDelta, maxDelta] defines the space search for the decay constant δ .

⁴GNU Scientific Library

⁵Assuming that there is no gene's self interaction allows to simplify the problem when steady-states are searched, and thus save time during process. **Must be always set to 0 (i.e. no gene's self interaction present) in the current version of Wyrd.**

- **ssWt, ssKo, ssHz, tse and target**
Define respectively if wild-type steady-state, knockout steady-states, heterozygous steady-states, time series experiments and target network are available for GRN reverse engineering.
- **zeroRandomInitilization (evolution)**
The starting point of all gene's optimizations is a random network. All initial weights of this random network are set to zero if *zeroRandomInitilization* equal to 1, else if equal to 0, weights are set to random values taken in range $[-\text{maxWeight}, \text{maxWeight}]$.
- **rngSeed (evolution)**
Used to generate initial random solution of gene's optimizations.
- **evolutionAlgorithm (evolution)**
Defines if the algorithm of optimization is CMA-ES (0) or L-CMA-ES (1).
- **numGeneOptim (evolution)**
The number of optimization runs for each gene's optimization.
- **restartPrevSolution (evolution)**
If set to 1, the initial solution of a run of one gene's optimization is the solution found at the end of the previous run.
- **gene2optimize (evolution)**
The index of gene to optimize, from 0 to *numGenes*-1. If *gene2optimize* is set to -1, Wyrd will optimized all genes of the network one after another.
- **initSigma (CMA-ES)**
Initial value of sigma.
- **maxEvals (CMA-ES)**
The number of maximum solutions evaluated. The number of generations for an evolution is given by $\frac{\text{maxEvals}}{\text{popSize}}$ and rounded.
- **popSize (CMA-ES)**
Population size.
- **tolX (CMA-ES)**
Stop optimization if the change in the population mean between two generations is less than *tolX*.
- **tolFitness (CMA-ES)**
Stop optimization if the change in the minimum fitness between two generations is less than *tolFitness*.
- **stopFitness (CMA-ES)**
Stop current optimization if the fitness if less than *stopFitness*.
- **mDim (L-CMA-ES)**
The subspace dimension *m*, in order to have $m \ll n$.

Evolvable parameters

In reverse engineering process and for sigmoid model, a solution for a particular gene is composed by its values of v_{max} , α , β , δ and its incoming interaction's weight. In Wyrd, all of these parameters that have their min and max bounds identical are not considered as evolvable and are not inserted in the phenotype.

A.6.4 Run GRNs Reverse Engineering process

To launch the reverse engineering process of an underlying gene regulatory network, the user can as usually choice between interactive and non-interactive mode.

- **Interactive mode, $-i\ 1$**

Select the third main menu, "Find underlying GRN from given datasets." (choice 2), enter the project Id if it is not already specified and follow the indications.

- **Non-interactive mode, $-i\ 0$**

The command line option $-r\ 2$ runs Wyrd in reverse engineering mode. Specification of wild type, knockout, heterozygous steady-states, time series datasets and target GRN availability can be specified with respectively $-w\ 1$, $-k\ 1$, $-h\ 1$, $-ts\ 1$ and $-target\ 1$.

If all types of datasets are available for the process, Figure A.9 shows how run Wyrd in non-interactive mode. Regardless of the way chosen to run Wyrd, one of the first things that it will do is to display a listening of all given inputs and a confirmation that they have been correctly open. Figure A.10 illustrates this.

```
$ ./wyrd -c configuration.cfg -i 0 -w 1 -k 1 -h 1 -ts 1 -target 1 -p grntest
```

Figure A.9 This command runs Wyrd with all types of datasets as input ($-w\ 1$, $-k\ 1$, $-h\ 1$, $-ts\ 1$). The target GRN is also given ($-target\ 1$), thus distance between target and evolved solutions can be recorded. Remember that all these parameters are in the configuration file, and that command line options simply redefine them.

A.6.5 Management of CMA-ES

Wyrd allows to optimize only one gene, or all genes of the network, one after another. This can be controlled with the parameter *gene2optimize*. If it is set with a value of the integral interval $[0, numGenes-1]$, only the selected gene will be process. If set to -1, all genes will be process, from gene's index 0 to gene *numGenes-1*. Each gene's optimization is composed of *numGeneOptim* runs, launched one after another in order to evolved *numGeneOptim* times the same gene and saved the results of the best run.

In order to optimize genes, two evolutionary algorithms are available: CMA-ES (Covariance Matrix Adaptation Evolutionary Strategy) and L-CMA-ES. L-CMA-ES can be

```

GRN Reverse Engineering
=====
Some Steady-States Experiment(s) are available!
XmlGRN: Grn file grntest.xml is correctly open!
[SS] The dataset grntest_ss_wt.csv is correctly loaded!
[SS] The dataset grntest_ss_ko.csv is correctly loaded!
[SS] The dataset grntest_ss_hz.csv is correctly loaded!
Some Time Series Experiment(s) are available!
[TS] Loading dataset(s) file grntest_ts.csv ...
Number of genes in this dataset: 5
Size of dataset: 13 x 5
Size of dataset: 25 x 5
Size of dataset: 30 x 5
The dataset grntest_ts.csv is correctly loaded!

```

Figure A.10 Listening of input files given to WyrD in reverse engineering mode. Here, there are three time series experiments with 13, 25 and 30 time points.

useful when the dimension of the problem is large, so when the network size is large (dimension is equal to a little number of gene's parameters and *numGenes* or *numGenes-1* (if *selfLoop* parameter is set to 0) interaction's weights). L-CMA-ES is a variant of CMA-ES that allows to reduce the dimension space of the problem n to m as $m \ll n$. In this case, a subspace is built with the m principal dimensions of the original space. The importance of the multiple dimensions is characterized by their eigenvalue in the covariance matrix. If L-CMA-ES is used, the parameter *mDim* defined the parameter m used in the L-CMA-ES theory. For complete description of CMA and L-CMA-ES algorithms, please refer to [10] and [16].

In interactive mode, the user can choose which CMA or L-CMA-ES algorithms he wishes to use. Four choices are offered: use CMA-ES for only the next gene's optimization, use CMA-ES for all further genes's optimization, and the two same possibilities with L-CMA-ES. This allows, if the user wishes it, to change the algorithm of optimization from one gene to another. These four choices are shown in Figure A.11. In non-interactive mode, all gene's optimization will be processed with the algorithm strategy defined by the parameter *evolutionAlgorithm* in the configuration file.

At this stage, WyrD begins the *numGeneOptim* runs of the first or the selected index of gene. The initial solution is a random one, generated with the seed *rngSeed*. Soon the second run, an initial points strategy defined by the user is applied: initial point of run i is the best solution found at run $i - 1$ (*restartPrevSolution* = 1), or initial points of the second run and others are all new random points with values taken in real uniform intervals defined by the gene's parameters bounds (*restartPrevSolution* = 0).

A progress bar indicates the state of the current run of one gene's optimization. At the end of each run, the number of evaluations, the best fitness found and its associated solution are highlighted. If the fitness of a run is the best ever one, all the results of this run are saved. Figure A.12 shows the result of the run $0/\text{numGeneOptim}$ of one gene's

```

=====
| Optimization of Gene 0
=====
What Evolutionary Algorithm would you like to use?
(0) CMA-ES, only for this gene's optimization.
(1) CMA-ES, always (for all the following genes's optimizations).
(2) L-CMA-ES, only for this gene's optimization.
(3) L-CMA-ES, always.

```

Figure A.11 In interactive mode, the user can choice between four optimization strategies.

optimization.

```

Run 0
=====
CMA Initial Point:
[6](0.115694,0.0207483,0,0,0,0)
CMA Progress || ===== || End
CMA Maximum number of evaluations reached ... stoping optimization.
CMA Number of Evaluations: 50050
CMA Best Fitness: 0.0037049
CMA Best Point: [6](0.0115139,0.104788,0.0120821,-0.0568978,-0.386043,0.0531433)
CMA Actually, run 0 is the best one (fitness=0.0037049)

```

Figure A.12 Example of Wyrd display for each run of each gene optimized.

CMA algorithms stop evolution run when one of these three stopping criteria is met.

- **Maximum number of evaluations done**

Define by *maxEvals*.

- **Best specified fitness found**

Define by *stopFitness*.

- **NaN error occure**

In the current implementation of CMA by Knight and Lunacek, some NaN error can occure in different places. The majority of them have been managed in the implementation of Wyrd and CMA code in order to avoid program crash and be able to continue optimization. However, a NaN error occurs always when *maxEvals* is very large ($>5e6$) and thus very little numbers handle by CMA algorithm, which leads to NaN errors. In this case, the CMA stops and return the best fitness and associated results. In all cases, this error is not critical. More, a good use is to set *maxEvals* to a large number in order to make the maximum number of evaluations versus a large computation time.

A.6.6 Results and output files

Wyrdr saves all results in several files, with filenames defined in accordance with the standard introduced in Section A.3.2.

- **key_evo.xml**
The best found GRN topology. Contains the best gene's parameters and interaction's weights found for all genes at the end of the gene's optimizations.
- **key_ss_wt_evo.csv, key_ss_ko_evo.csv and key_ss_hz_evo.csv**
After found the whole evolved gene regulatory network, Wyrdr simulates it to generate steady-state datasets. A type of evolved steady-states datasets are computed and saved only if the associated target datasets have been available for the GRN reverse engineering process.
- **key_ts_evo.csv**
If target time series experiments are given to Wyrdr, the entire evolved gene regulatory network is simulated to obtain the evolved time series dataset.
- **key_ss*_evo2.csv and key_ts_evo2.csv**
Similar to the previous steady-states and time series datasets, these ones contain datasets generated gene by gene independently from key.xml.
- **key_log_dist_g[0]_r[1].stat**
One file per run of gene's optimization, it contains the evolution of the fitness through generations. This file is in CSV format with the elements on the same line separated by a tab (so it is more a TSV file, but CSV denomination is kept). Each line contains the fitness of all the individuals of the current population. Thus, there are *popSize* elements in each line, with an additional one: the first element of each line is the index of the generations. [0] refers to the gene's index [0, *numGenes*-1] and [1] to the gene's run index [0, *numGeneOptim*].
- **key_log_se_g[0]_r[1].stat**
Similar as key_log_dist_g[0]_r[1].stat, these files save not the fitness but the square error between evolved individual and target solution. So, the content of these files are computable only if the target GRN is available.
- **key_log_bOptim.stat**
One line per gene optimized. Each line contains the index of the best run, the best fitness, the associated square error and best genotype.

A.7 Visualization of Wyrd results

Parallel to the development of Wyrd, a Matlab tool with GUI was implemented in order to be able to quickly analyse the results after a Wyrd process. It is composed of several Matlab scripts that can be used separately to realize specific tasks.

The files generated during a reverse engineering of a gene regulatory network are taken as inputs, and a HTML report are built including GRN representations (using Graphviz), fitness and square error graphs, presentation of GRN's parameters in tables, time series graphs, etc. Please refer to the WyrdReport Handbook for more information about the use of this tool.

A.8 Doxygen Documentation

A complete Doxygen documentation is available to help developers or end-user of Wyrd. HTML and L^AT_EX formats are located in ./src directory includes in the main folder of Wyrd project.

B

WyrddReport Handbook

B.1 Introduction

Wyrdd creates several data files during and at the end of all reverse engineering processes. In order to quickly and easily visualize these data, a Matlab tool has been implemented. It takes in input all the files generated by Wyrdd and produces a structured HTML report as output. In current version, all reports can be divided into two main parts. In the first one, general informations on the found and target (if available) Gene Regulatory Networks (GRN) are displayed, as general experiment informations, GRN topologies, gene's parameters or still fitness or square errors graphics between evolved and target GRN parameters through evolutions.

The second part is about the time series experiments. From time series data files, integration curves of genes expression levels are represented for all genes of the GRN, with three types of points: simulation of a single gene, as computed during individual gene's optimizations, simulation of the entire evolved GRN found at the end of the reverse engineering process and composed of all genes independently optimized, and target time series experiments given as input to the GRN reverse engineering process.

Matlab was selected as development support because in addition to being multiplatform, it already has a large number of statistical tools and graphical representations available. Under the denomination of WyrddReport, this tool is composed of several little scripts that can be used for the most independently of the others, e.g. draw a GRN topology from a Wyrdd GRN XML file, graphs fitness evolution, time series integration, etc.

B.2 Quickstart

B.2.1 WyrddReport GUI - `wrgui`

A Graphic User Interface (GUI) was designed to allow the user to change quickly the main parameters of the HTML reports. In the Matlab scripts directory, the command `wrgui` runs WyrddReport and displays a window identical to the one shown in Figure

B.1.

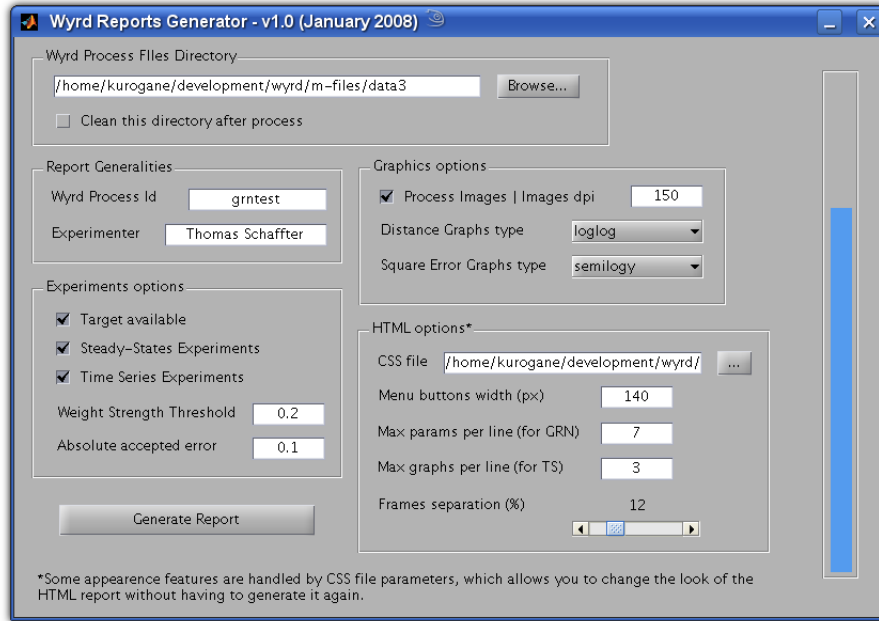


Figure B.1 Screenshot of the WyrddReport Graphical User Interface.

A brief description of each components grouped by blocks is given:

- **Wyrdd Process Files Directory**

It is here that the path of the folder containing all the output files generated by a Wyrdd during the reverse engineering process must be given. If the checkbox “Clean this directory after process” is checked, all the content of this folder will be moved into a new folder called “data” in the main new HTML report directory.

- **Report Generalities**

The first editbox must be filled with the process Id given to Wyrdd in order to generate all input and output filenames as described in Section A.3.2. The name of the main folder of the HTML report will be the same as the project Id. It will be created in the current work directory of Matlab. The second field contains the name of the experimenter.

- **Experiments options**

The report is intended to be divided into three parts: GRN Topology (general information), Steady-States and Time Series parts. Actually, steady-states part is not used but its management is still available in order to integrate future interesting information on steady-states experiments. If one of this three checkboxes is checked, the correspond part will be process and include in the HTML report.

Graphical representations of GRN’s topologies are available. In evolved GRN’s topologies, even the non-existing interactions determined by Wyrdd have generally always non-zero weights. So a weight threshold is necessary to discriminate weak interactions and to not represent them in the Graphviz representation.

The editbox “Absolute accepted error” is used to display in red evolved gene’s parameters that do not respect the following relation. So, this feature is only used if target GRN is available.

$$|\text{evolved parameter} - \text{target parameter}| > |\text{accepted error}| \quad (\text{B.1})$$

- **Graphics options**

HTML reports contain many images as GRN’s topologies, fitness and square errors through evolutions and time series experiments. If the checkbox “Process Images” is checked, images will be processed. In this case, the dpi given in the editbox on the same line is used. Process images takes the most part of the whole computation time necessary to produce a Wyrd report. A trick is that if a report has already been generated once time and that the user changes data as the name of the experimenter or the “Absolute accepted error”, there is no need to generate a second time the images. Then the editbox “Process Images” can be unchecked and so the update of the report will be almost instantaneous.

The next two features allow to display fitness and square errors in four types of plot in Matlab: plot, semilogx, semilogy and loglog.

- **HTML options**

All reports are in HTML format, and contain a left menu and a main page. The menu allows to quickly navigate between the different sections of the main page. The HTML page’s design are defined by a CSS file (style sheet) which defines properties as text fonts, images sizes, borders, page’s colors, etc. It is in this block of options that the CSS file to use must be specified. Note that this method has been chosen because modification in the CSS file does not require modification in the source code of the HTML page. So all changes in this file are directly visible after reload the HTML page in the internet browser.

The next three editboxes define respectively the width of the menu’s buttons, the maximum number of gene’s parameters in a single line (GRN Topology part) and the maximum number of time series graphics in a single line (Time Series part).

It is possible to resize the width of the menu using the unique slider of the GUI. The position of the slider defines the width of the menu in percentage of the total width of the internet browser window. If the value of the slider is 0, no menu will be included in the HTML report, useful to print it.

- **Progress bar**

This feature informs the user on the progress of the generation of the HTML report. If the checkbox “Clean this directory after process” is not checked, a little bit of the total size of progress bar will remain at the end of the process to recall that the folder containing the input data had not been cleaned.

Once all fields are filled, the Wyrd HTML report can be generated by pressing the button “Generate Report”.

B.2.2 HTML reports

During report generation, a folder with the same name that the project Id is created. In this last one, two others folders are created, “data” and “images”, and three HTML files: index.html, menu.html and key.html where *key* is the project Id. To visualize the report, open the page index.html with an internet browser as Opera, Firefox or Safari among many others. The report should look like Figure B.2.

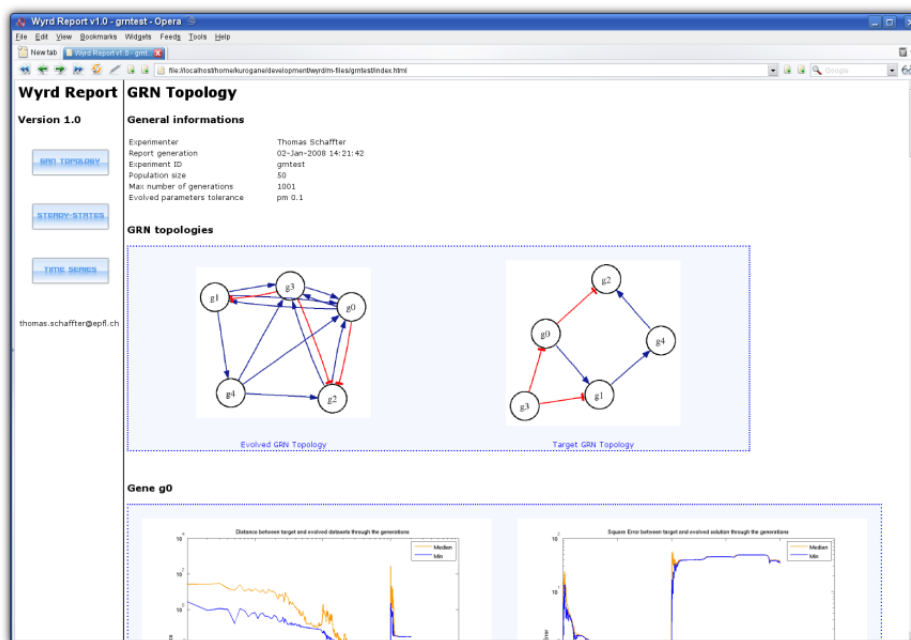


Figure B.2 The HTML page index.html must be open with an internet browser as Opera, Firefox or Safari among many others.

In section “General Informations”, some data as the experimenter’s name, the date of the HTML report generation or the process Id are displayed. Then, evolved GRN topology is built using Neato (Graphviz) from the XML file. If target network are available, the associated target topology is also built and displayed to the right. As all images in the report, it is possible to click on them and so open the image in full screen.

After GRN topologies, a short report about all gene’s optimizations is included. The first two graphs represent respectively the evolution of the fitness and the evolution of the square errors. The fitness is defined by the mean square distance between the evolved and target dataset as steady-states and/or time series datasets. The goal is to minimize this fitness in order to find an evolved network that generates the same dataset which them generated by the target network. The square error is the mean square distance between all gene’s parameters of the evolved and target network. More square error is small, more the gene’s parameters of the evolved and target network are similar. So, square errors graphics can only be drawn if the target network is available and given to Wyrld during the GRN reverse engineering process.

The first next table displays the numerical values of the best fitness found and the associated square error. The second table contains all the evolved gene’s parameters. If

target network is present, target gene's parameters are displayed. In this case, the couple of gene's parameters that do not respect the relation B.1 are display in red. The report of evolution about gene "g0" is displayed in Figure B.3.

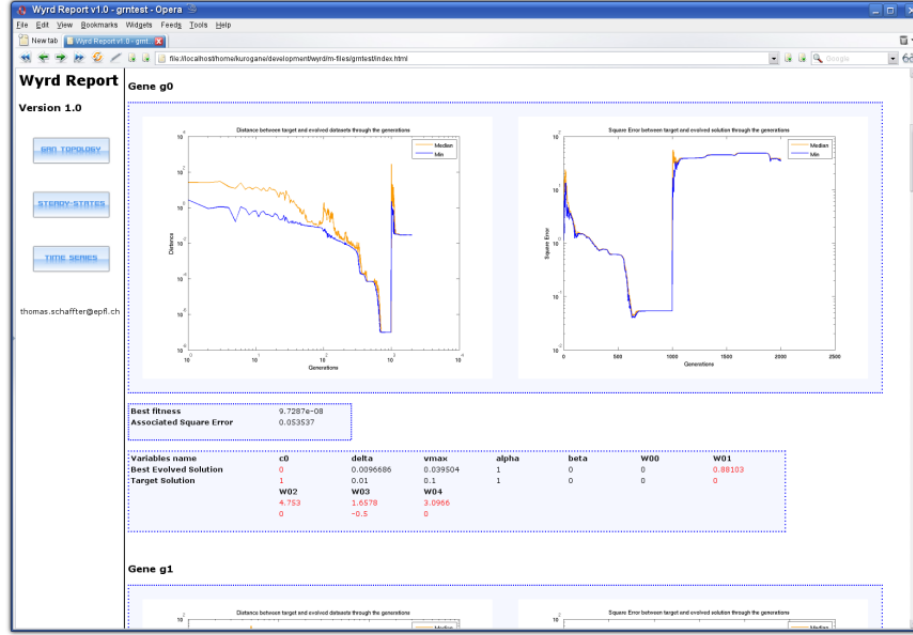


Figure B.3 All reports about gene's optimizations contains at least a graphic of the evolution of the fitness and numerical values of the best fitness and of the gene's parameters found. If target network is available, a graphic of the evolution of the square errors and numerical values of the square errors associated to the best evolved solution and target gene's parameters are included, too.

In the time series section, all experiments are represented. Each experiment is mainly composed of *numGenes* genes integration graphs associated to the *numGenes* of the gene regulatory network. In one experiment, a short table presents the number of time point which is the same for all gene's integrations in a same experiment. Then, the total normalized distance between target time series and evolved datasets are displayed. Note that the evolved dataset is here the final time series dataset obtained when the entire evolved network is simulated. This total normalized distance is given by:

$$\frac{\sum_{g=1}^G \sum_{e=1}^E \sum_{pt=1}^{PT(e)} |mRNA_{evo}(g, e, pt)^2 - mRNA_{target}(g, e, pt)^2|^{\frac{1}{2}}}{numGenes * \sum_{e=1}^E PT(e)} \quad (B.2)$$

where $G = numGenes$, E the total number of time series experiments, $PT(e)$ the number of time points in experiment e and $mRNA(g, e, pt)$ the mRNA expression level of gene g in experiment e at time point pt .

In each integration curve, three types of points are drawn:

- **Target dataset**

- Evolved dataset (gene)
- Evolved dataset (GRN)

where the points in green are taken from the file `key_ts_evo2.csv` and the points in blue from `key_ts_evo.csv`. `key_ts_evo.csv` contains mRNA levels when the entire evolved GRN is simulated, and `key_ts_evo2.csv` contains the mRNA levels of the independent gene's simulations as computed during gene's optimizations. Figure B.4 shows integration curves of experiment 1 for the five genes of the Grntest network. The value under each graph is the mean square distance between red and blue points.



Figure B.4 Integration curves of experiment 1 for the five genes network Grntest. Red points represent target mRNA concentrations, green points independent optimized genes mRNA and blue points mRNA concentrations when the entire found network is simulated at the same time.

B.2.3 Advices

All user's parameters and basic parameters are handled by an instance of class *wrInitCore* called *wrInit*. One of the first things that make *wrgui* when it is launched for the first time is to allocate this object and initialize all fields of the GUI with default values saved into *wrInitCore.m*. When the user changes a parameter as the name of the experimenter or the dpi for images, data are saved into the current instance *wrInit*. So if the user close *wrgui* window and open it a second time, the *wrgui* parameters will be the same that when the window have been closed. The only way to restore default values is to clear the instance *wrInit*.

Uncheck "Clean this directory after process" may be useful for intermediary generation of the HTML report. For example, if the user changes the value of the weight's

threshold used to remove weak interactions, it can directly generate the report a second time without changing the folder which contains the input data files.

Like process images takes a big part of the whole computation time, it is interesting to disable images processing by unchecking “Process Images” if the user changes for example only the experimenter’s name. But in the case where the user changes the weight’s threshold, GRN topologies images must be processed again (connections drawn depend of the threshold) but not the time series images. These last ones can be numerous if there is a lot of experiments and genes. Unchecking “Time Series Experiments” avoids to process all integration curves of time series experiments. Like now time series section is no longer part of the HTML report, a quick new generation of the report with checking “Time Series Experiments” and unchecking “Process Images” will add again time series section as part of the HTML report.

Before printing reports, it is advisable to remove the left menu of the report. This is possible with setting the unique slider of the GUI to 0. The menu frame will not only be resized to width 0, but is really removed.

B.3 Using Matlab tools in command lines

B.3.1 Instance of `wrInitCore`

Like *wrgui*, many scripts of WyrdReport need an instance of the class *wrInitCore* called *wrInit*. To instantiate this object manually:

```
>> wrInit = wrInitCore();
```

Figure B.5 Creation of *wrInit*, instance of the class *wrInitCore*, which is used by several WyrdReport scripts.

To change or access *wrInit* parameters, the user must use specific methods *set* and *get*. For example, the commands shown in Figure B.6 set the dpi parameter to 200.

To avoid sometimes errors due to mismanagement of global instances, the best and fastest way is to change directly the values of parameters contained in *wrInitCore.m*.

```
>> wrInit = set(wrInit, 'dpi', 200);
>> get(wrInit, 'dpi')
ans =
200
```

Figure B.6 *Set* and *get* methods must be used to change or access to class instance’s parameters.

B.3.2 WyrdReport

Once *wrInit* is instantiated, the next command generate an HTML report:

```
>> wyrdReport(wrInit)
```

Figure B.7 *WyrdReport* takes as parameter the instance *wrInit*. If an error occurs, verify that the path of the input folder and that the process Id are correct.

B.3.3 Others WyrdReport scripts

Some Matlab scripts are implemented to allow direct use of them in ordre to realize specific tasks.

If we want to build a GRN representation using Graphviz in PNG format from a fully specified GRN in XML format following the syntax used in Wyrd (Section A.3.2), the script *printGrnTopology* may be used as display in Figure B.8. Here, *printGrnTopology* takes the XML file *grntest_evo.xml* as input which locates in the work directory of Matlab (empty path " used) and create the image *grntest_evo.png* in this same directory.

```
>> printGrnTopology("", 'grntest_evo');
```

Figure B.8 From a fully specified GRN in XML format with the syntax used in Wyrd (Section A.3.2), it is easy to create a PNG representation using Graphviz.

Note that this same script can extract the weight matrix of the network or the gene's identifiants as shown in Figure B.9.

The command *help* in Matlab offer good support for all scripts. Annexe F contains the hierarchy of Matlab scripts and their use given by the command *help*.

B.4 Precision-Recall and ROC Matlab scripts

Precision-Recall and ROC are two widely used measures for evaluating the quality of results in domains such as Information Retrieval and statistical classification. In Wyrd, reverse engineering produce among others as output interaction's weights between gene, that can be excitatory ($w_{ij} > 0$) or inhibitory ($w_{ij} < 0$). So two lists of predictions can be built, one for positive and one for negative weights. From these lists of predictions and the target solution, it is possible to build Precision-Recall or ROC curves to estimate the quality of the Information Retrieval algorithms.

A set of Matlab scripts has been implemented so as to easily calculate the area under Precision-Recall and ROC curves, and curves themselves. The hierarchy of this scripts is shown in Figure B.10.


```
>> [a,a,W] = printGrnTopology(", 'grntest_evo');
>> W
w =
0          0.8810    4.7530    1.6578    3.0966
1.0194     0        -0.0296   -1.4153   -0.1525
-2.0119   -0.0156     0        -0.4714    4.8268
4.9490     3.9609    4.9999     0        4.9999
0.0329     3.0585    0.1142   -0.1550     0
```

Figure B.9 One use of *printGrnTopology*. From the same XML file that used previously, we extract the weights matrix from tags `<weight>`.

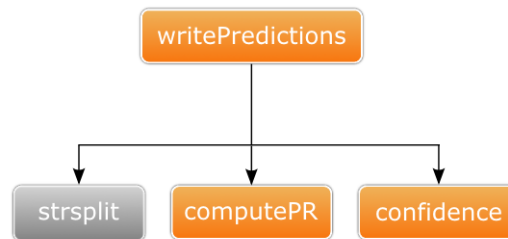


Figure B.10 A set of three main scripts is available. *confidence* takes a CSV file with evolved weights elements and a Matlab matrix of the target weights as input and create Matlab matrix with source and target gene's indexes and confidence level for the associated connections. *computePR* takes a vector of sorted prediction in descending order of size *numGenesXnumGenes*, and a solution vector of the same size with element equal to 1 (interaction present between two specific genes) or 0 (no interaction). Following the Precision-Recall or ROC choice, *computePR* returns the area under the curve and the x-axis and y-axis of this curve in order to plot it. *writePredictions* regroups these two scripts. *strsplit* is used to split strings. This script has been implemented by Gie Spaepen. Scripts in grey are the work of an another person.

The script *confidence* needs as input a file `data.stat` which contains multiple weights matrix of evolved networks of the same reverse engineering problem. All elements on the same line must me separated by a tab. If there is no self-interaction as it is handled in Wyrd, it is possible to remove the gene's self-interactions, i.e. all w_{ii} elements. Figure B.1 (a) displays an example of content of `data.stat`.

In Matlab syntax, this input file is built as a column vector of matrix $[W1; W2; \dots; WN]$ where each W is a weight matrix. If target solution is available, a Matlab matrix of size *numGenesXnumGenes* must be created with target weights. This matrix can contain real values of weights, but 0, 1 and -1 elements are sufficient to defined if there is positive interaction between two specific genes (1), negative interaction (-1) or no interaction (0). See *help confidence* for more information about file syntax.

From the input file `data.stat` and a specified threshold, *confidence* computes a Matlab

matrix of predictions for gene-gene interactions. The threshold is used to set to zero weak weights. Note that prediction lists are not sorted here in descending order of confidence levels. If the target solution is available, two additional columns are added, one for each future lists, in the predictions matrix which contain elements 1 (there is an interaction between to specific genes) and 0 (no interaction). The beginning of a such output Matlab matrix is displayed for the five genes Grntest network in Table B.1 (b).

*	*	...	*	w_{00}	w_{01}	w_{02}	w_{03}	w_{04}	0	0	0	0	0	0
*	*	...	*	w_{10}	w_{11}	w_{12}	w_{13}	w_{14}	1	0	0	0	0	0
*	*	...	*	w_{20}	w_{21}	w_{22}	w_{23}	w_{24}	2	0	0	0	0	0
*	*	...	*	w_{30}	w_{31}	w_{32}	w_{33}	w_{34}	3	0	0	0.98	0	1
*	*	...	*	w_{40}	w_{41}	w_{42}	w_{43}	w_{44}	4	0	0.15	0.22	0	0
*	*	...	*	w_{00}	w_{01}	w_{02}	w_{03}	w_{04}	0	0	0.86	0	1	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	1	0	0	0	0	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Table B.1 (a) Structure of data.stat for a five genes network. Elements * are any parameters and can be of any number on each line. On the right is the evolved matrix of weights found. Blue elements must be deleted in data.stat if parameter *selfLoop* of script *confidence* is set to 0. The 6th line is the begin of a new evolved W. key_log.bOptim.stat files produced by Wyrld follow this syntax. (b) Output matrix of *confidence* script. Each line represents a gene-gene interaction. Columns are in the order: source gene index, target gene index, confidence level for positive weights list, confidence level for negative weights list. If target **w** is available, there are two more columns: real presence of positive interactions for positive list and real presence of negative interactions for negative list. For example, the 4th line shows that Wyrld found an inhibitory interaction with confidence level 0.98 in the 4th column from gene 3 to gene 0, and that there is really an inhibitory interaction at this place as defined by the target network (1 in the 6th column).

After sorting the matrix of Table B.1 by descending order the first column for positive predictions or by descending order the fourth column for negative predictions, the script *computePR* can be used. It takes as input a column vector of confidence level of size *numGenes* \times *numGenes* sorted in descending order, e.g. the 3th or 4th column of the Figure B.1 (b), and the associated solution column vector, respectively the 5th or the 6th column of the same figure. *computePR* return the area under Precision-Recall or ROC curve and x-axis and y-axis of this same curve. *writePredictions* automates this process. The following commands process:

```
>> [area,x,y]=writePredictions(5,0,'data.stat',0.2,'pr',1,'xml','grntest.xml');
>> plot(x,y)
>> [area,x,y]=writePredictions(5,0,'data.stat',0.2,'roc',1,'xml','grntest.xml');
>> plot(x,y)
```

Figure B.11 *writePredictions* regroups all *confidence* and *computePR* features, and more.

- Open the file data.stat which contains one or more evolved GRN weights matrix, in CSV format.
- 'xml','grntest.xml' opens the target GRN in XML format, grntest.xml. It is also possible to give directly the target weights matrix as a Matlab matrix with 'mat', 'targetMatrix' where targetMatrix is the Matlab target matrix.
- Sort in descending order the output matrix given by *confidence* regardless of prediction levels of positive weights (1), (0) for the negative weights.
- Write in a file dataEXCITATORY.list the excitatory predictions list sorted in descending order.
- Give to *computePR* the sorted column of confidence levels. *computePR* returns area and Precision-Recall curve ('pr') and x-axis and y-axis. For ROC features, replace 'pr' by 'roc'.
- Return this area and the two axis.
- Make the same process for the ROC curve and plot it.

Figure B.12 displays the two PR and ROC plots for excitatory gene-gene interactions.

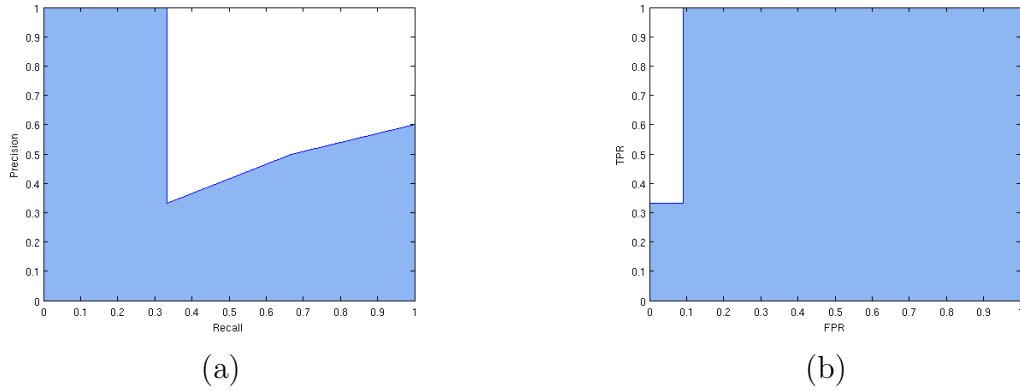


Figure B.12 For excitatory interactions, Precision-Recall (a) and ROC (b) curves are drawn. Area under curve are highlighted. In both curves, more the predictions are correct, more the areas under curves are large. FPR=“False Positive Rate”, TPR=“True Positive Rate”. The algorithm used is based on the paper of J. Davis [4].

C

WyrD Settings file

```
##### WyrD PARAMETERS #####
# Configuration file for WyrD 1.0
#
# Thomas Schaffter, Thomas.Schaffter@epfl.ch
# Master Project, "Scalable Reverse Engineering of Nonlinear Gene
# Networks", September 2007 - February 2008
# Laboratory of Intelligent Systems (LIS)
# Ecole Polytechnique Fdrale de Lausanne (EPFL), CH-Suisse
#####
# =====
# GENERAL PARAMETERS

# Interactive mode (0=No, 1=Yes)
interactiveMode = 0

# Run modei
# 0 = Create a Test Case (i.e generate a XML file from of a fully
# specified GRN)
# 1 = Simulation of a fully specified GRN
# 2 = Find an underlying Genes Regulatory Network
runMode    = 2

# Select the operation to process in simulation mode
# 0 = Generate Time Series data
# 1 = Generate Steady-States data (wild type, knockout and heterozygous)
# 2 = Compute Wild Type Steady-States of a f.s. (fully specified) GRN
simulationMode = 0

# Test Cases
# 0 = BreakeRepressilator
# 1 = Grntest
# 2 = Random GRN
testCase    = 2
```

```
# Display generated datasets during processes
displayDatasets = 0

# Process id
#pId = grntest

# =====
# INTEGRATION AND STEADY STATE CALCULATION

# Absolute precision in integration / steady state
SS_abs_precision = 0.001
# Relative precision in integration / steady state
SS_rel_precision = 0.001
# Max number of iterations for GSL Multidimensional root finding
maxIterGslSs      = 2000
# Time step of integrator (strongly affects integration time, though it
# should not affect accuracy)
dt      = 5

# =====
# UPPER AND LOWER BOUNDS FOR THE GRN PARAMETER VALUES

# Presence of self-loop in GRN totopoly (0 for no)
# Actually, set to 1 is an error (computation of x wrong!)
selfLoop      = 0
# Interaction strength (-max ... max)
maxWeight     = 5
# maximum transcription rate
minMax        = 0
maxMax        = 2
# Sigmoid parameter alpha (minAlpha was 1)
minAlpha      = 1
maxAlpha      = 1
# Sigmoid parameter beta (corresponds to non-saturated part of sigmoid)
minBeta       = 0
maxBeta       = 0
# Decay rate
minDelta      = 0.01
maxDelta      = 0.3
# Bounds for initial conditions c0 (do not change)
minC0         = 0
maxC0         = 0

# =====
# EXPERIMENTS

# Load Time Series initial points (No=0, otherwise Yes)
```

```

# SIMULATION Use pId_ts_init.csv with initial concentrations to generate
# Time Series datasets. If no (=0), initial point will be those specify
# in XML GRN file (pId.xml).
# GENERATION In generation mode, if ts0=1, WyrD generates a file with
# random initial expression levels. Set for numTimeSeries experiments,
# initial CO don't exceed ts0MaxCO
ts0      = 1
# Max CO mRNA levels, used if ts0=1
ts0MaxCO = 10

# REVERSE ENGINEERING
# Wild Type Steady-States dataset availability
ssWt     = 1
# Knockout Steady_States dataset availability
ssKo     = 1
# Heterozygous Steady_States dataset availability
ssHz     = 1
# Time Series datasets availability
tse      = 1
# Target GRN availability (if available, solution square error can be
# computed)
target   = 1

# SIMULATION Number of time series experiments
numTimeSeries      = 3
# SIMULATION Number of time points in each time series (dt is used as
# time step)
numTimePoints      = 13
# Set true to normalize target expression values between 0 and 1 (this
# option is not handled)
normalize          = 0
# =====
# EVOLUTION

# Random networks are initialized with all weights set to:
# 0: random weights
# 1: weights set to zero
zeroRandomInitilization = 1
# Seed for the Random Number Generator (only to generate the initial
# random GRN before start evolution)
rngSeed              = 9
# Evolution algorithm
# 0: CMA-ES, 1: L-CMA-ES
evolutionAlgorithm   = 0
# For each gene, number of independent run's optimizations
numGeneOptim         = 1
# During a batch of single gene's run optimization

```

```
# 1 = initial point of the new CMA run is the final previous one (allow
# to continue previous optimizations)
# 0 = new initial points in intervals defined in section "UPPER AND LOWER
# BOUNDS FOR THE GRN PARAMETER VALUES"
restartPrevSolution      = 0
# Index of gene to optimize (from 0 to numGenes-1, -1 to optimize all genes
# one after one)
gene2optimize            = -1

# =====
# CMA++

# Initial population variance
initSigma    = 1
# Stop if the fitness function has been evaluated more than maxFunEvals times
maxEvals     = 500000
# Population size
popSize      = 50
# Stop optimization if the change in the population mean between two
# generations is less than tolX
tolX         = 1e-6
# Stop if the change in the minimum fitness between two generations is less
# than tolFun
tolFitness   = 1e-15
# Stop if f_min < stopFitness
stopFitness  = 1e-15
# For L-CMA-ES, subspace dimension m
mDim         = 10
```


D

In silico Target Datasets of Grntest

D.1 grntest_ss_wt.csv

"Ss"	"G0"	"G1"	"G2"	"G3"	"G4"
"wt"	3.973	2.728	0.4636	0.8333	1.5

D.2 grntest_ss_ko.csv

"SsKo"	"G0"	"G1"	"G2"	"G3"	"G4"
"G0"	0	0.4766	1.991	0.8333	1.21
"G1"	3.973	0	0.02048	0.8333	0.75
"G2"	3.973	2.728	0	0.8333	1.5
"G3"	5	2.98	0.07462	0	1.5
"G4"	3.973	2.728	0.0007077	0.8333	0

D.3 grntest_ss_hz.csv

"SsHz"	"G0"	"G1"	"G2"	"G3"	"G4"
"G0"	1.987	1.738	1.879	0.8333	1.492
"G1"	3.973	1.364	0.4259	0.8333	1.475
"G2"	3.973	2.728	0.2318	0.8333	1.5
"G3"	4.481	2.924	0.1972	0.4167	1.5
"G4"	3.973	2.728	0.02046	0.8333	0.7498

D.4 grntest_ts.csv

"Time"	"g0"	"g1"	"g2"	"g3"	"g4"
0	0	0	0	0	0
10	0.4277	0.4163	1.11	0.6476	0.9252
20	0.776	0.6022	1.644	0.7921	1.209
30	1.083	0.7622	1.843	0.8244	1.322

40	1.358	0.9295	1.91	0.8314	1.387
50	1.607	1.104	1.924	0.8329	1.43
60	1.832	1.279	1.914	0.8332	1.458
70	2.036	1.447	1.888	0.8333	1.474
80	2.22	1.604	1.851	0.8333	1.484
90	2.387	1.747	1.801	0.8333	1.49
100	2.538	1.875	1.741	0.8333	1.493
110	2.675	1.988	1.672	0.8333	1.495
120	2.798	2.086	1.595	0.8333	1.497
130	2.91	2.171	1.513	0.8333	1.497
140	3.011	2.244	1.429	0.8333	1.498
150	3.103	2.307	1.345	0.8333	1.498
160	3.186	2.361	1.264	0.8333	1.499
170	3.261	2.407	1.186	0.8333	1.499
180	3.328	2.447	1.114	0.8333	1.499
190	3.39	2.482	1.047	0.8333	1.499
200	3.445	2.511	0.9861	0.8333	1.499
0	1	1	1	1	1
10	1.273	1.004	1.59	0.8705	1.369
20	1.528	1.115	1.812	0.8416	1.433
30	1.76	1.261	1.881	0.8351	1.457
40	1.971	1.416	1.885	0.8337	1.473
50	2.161	1.568	1.861	0.8334	1.483
60	2.334	1.71	1.818	0.8334	1.489
70	2.49	1.839	1.762	0.8333	1.493
80	2.631	1.955	1.696	0.8333	1.495
90	2.759	2.057	1.621	0.8333	1.496
100	2.874	2.145	1.54	0.8333	1.497
110	2.979	2.222	1.457	0.8333	1.498
120	3.073	2.288	1.373	0.8333	1.498
130	3.159	2.344	1.29	0.8333	1.499
140	3.236	2.393	1.212	0.8333	1.499
150	3.307	2.435	1.137	0.8333	1.499
160	3.37	2.471	1.069	0.8333	1.499
170	3.427	2.502	1.006	0.8333	1.499
180	3.479	2.529	0.9486	0.8333	1.499
190	3.526	2.552	0.897	0.8333	1.499
200	3.569	2.573	0.8508	0.8333	1.499
0	2	2	2	2	2
10	2.123	1.606	1.957	1.093	1.559
20	2.284	1.649	1.877	0.8911	1.498
30	2.441	1.775	1.8	0.8458	1.492
40	2.586	1.901	1.725	0.836	1.494
50	2.718	2.013	1.648	0.8339	1.496
60	2.837	2.11	1.568	0.8335	1.497
70	2.946	2.193	1.485	0.8334	1.498
80	3.043	2.264	1.4	0.8333	1.498

90	3.132	2.325	1.317	0.8333	1.498
100	3.212	2.377	1.237	0.8333	1.499
110	3.284	2.421	1.161	0.8333	1.499
120	3.35	2.459	1.091	0.8333	1.499
130	3.409	2.492	1.026	0.8333	1.499
140	3.463	2.52	0.9667	0.8333	1.499
150	3.511	2.545	0.9134	0.8333	1.499
160	3.555	2.566	0.8654	0.8333	1.499
170	3.595	2.585	0.8225	0.8333	1.499
180	3.631	2.601	0.7842	0.8333	1.499
190	3.664	2.615	0.7501	0.8333	1.499
200	3.693	2.627	0.7197	0.8333	1.499
0	2	3	4	1	0.5
10	2.178	2.488	2.346	0.8705	1.364
20	2.347	2.262	1.973	0.8416	1.481
30	2.501	2.175	1.816	0.8351	1.496
40	2.641	2.161	1.714	0.8337	1.498
50	2.768	2.184	1.625	0.8334	1.498
60	2.882	2.224	1.539	0.8334	1.498
70	2.986	2.271	1.453	0.8333	1.498
80	3.08	2.319	1.368	0.8333	1.498
90	3.165	2.364	1.285	0.8333	1.499
100	3.242	2.406	1.206	0.8333	1.499
110	3.312	2.444	1.132	0.8333	1.499
120	3.375	2.477	1.064	0.8333	1.499
130	3.432	2.506	1.001	0.8333	1.499
140	3.483	2.532	0.9445	0.8333	1.499
150	3.53	2.555	0.8934	0.8333	1.499
160	3.572	2.575	0.8475	0.8333	1.499
170	3.61	2.592	0.8065	0.8333	1.499
180	3.645	2.607	0.7699	0.8333	1.499
190	3.676	2.62	0.7373	0.8333	1.499
200	3.704	2.632	0.7084	0.8333	1.499

E

Wyrđ Class Dependencies

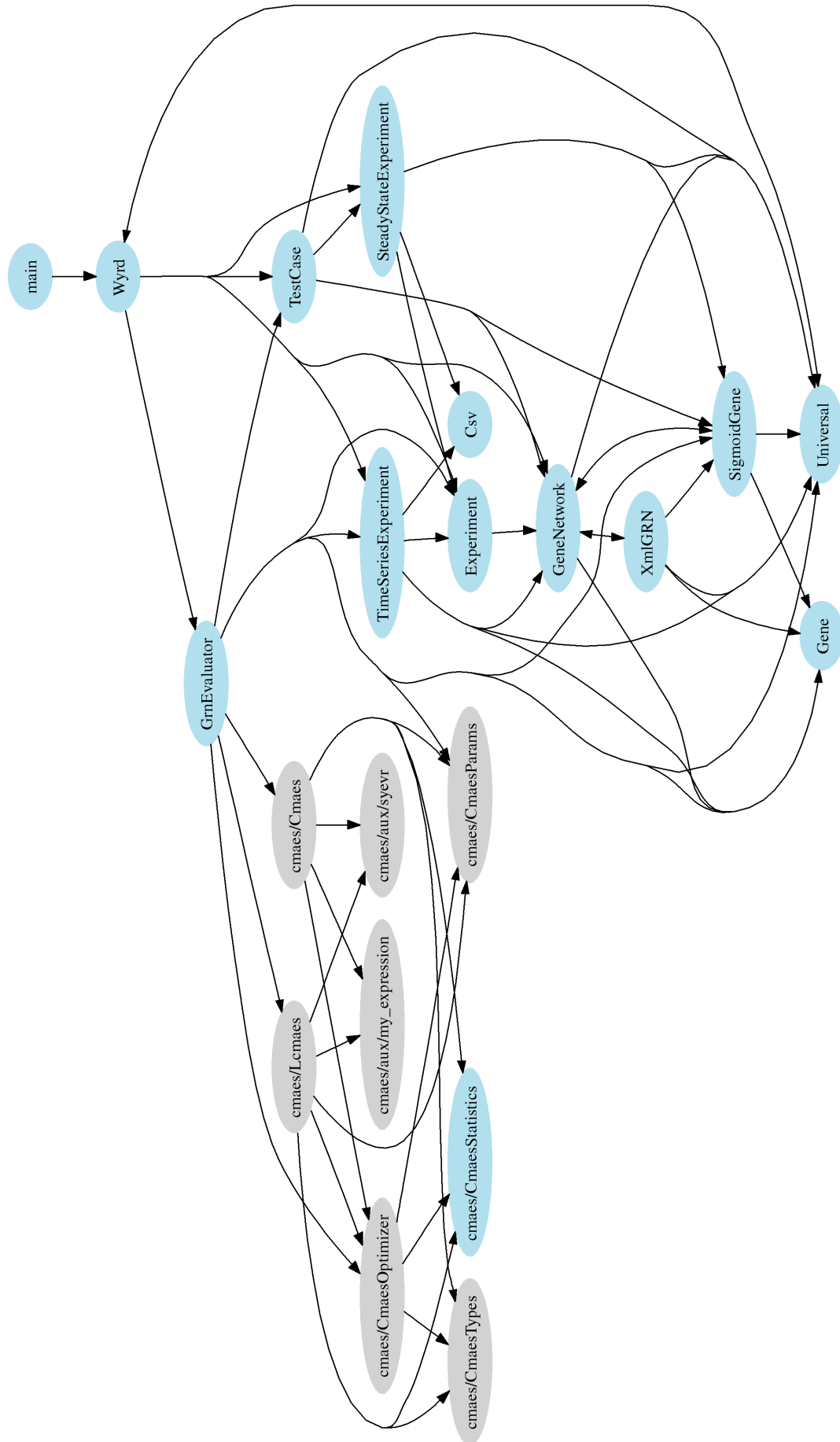


Figure E.1 Class dependencies of all C++ class of Wyrdr. CMA-ES implementation is the one introduced by Knight and Lunacek in [16]. Files highlighted in greylight are the work of others.

F

Matlab Scripts Documentation

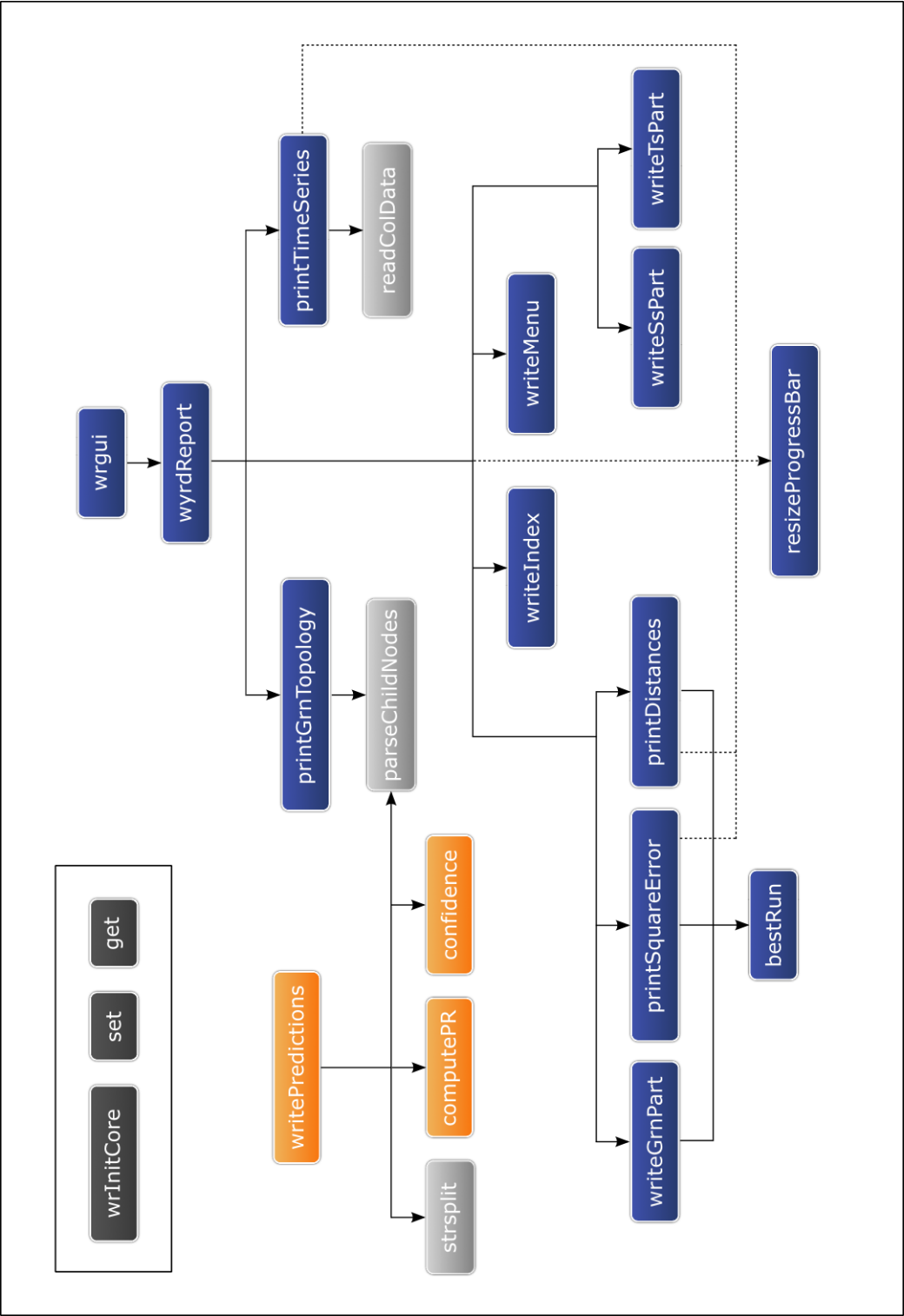


Figure F.1 Dependency of Matlab scripts that composed the tool WyrdrReport. HTML reports can be generated with *wyrdrReport* script. A Graphical User Interface is developed with GUIDE and can be called by *wrgui*. Scripts have been implemented in order to be run independently of others. The majority of scripts need an instance of *wrInitCore* called *wrInit*. This object handles all user's and general's parameters. Independent of WyrdrReport, the script *writePredictions* allows to write predictions lists into files and drawn Precision-Recall and ROC curves as to calculate the area under these curves. Dotted lines mean that the dependence to the target script is not mandatory.

F.1 bestRun

PURPOSE

BESTRUN - Gives informations on the best run of a gene's optimization.

SYNOPSIS

```
bRun = bestRun(indexOptimization)
```

DESCRIPTION

Input:

indexOptimization : index of the gene optimized

Output:

bRun.indexGeneOptim : is equal to indexOptimization
 bRun.indexBRun : index of the best run
 bRun.fitness : best fitness of this run
 bRun.se : associated square error (-1 if not available)
 bRun.solution : vector that contains the solution (phenotype)

IMPORTANT: Need a Wyrd Initializing Core called wrInit! Can be created before calling the present script with wrInit = wrInitCore(); instruction. If it does not already exist, this object will be created.

Author:¹




Thomas Schaffter, Thomas.Schaffter@epfl.ch
 Master Project, "Scalable Reverse Engineering of Nonlinear Gene Networks", September 2007 - February 2008
 Laboratory of Intelligent Systems (LIS)
 Ecole Polytechnique Fdrale de Lausanne, CH-Suisse

CROSS-REFERENCE INFORMATION²

This function calls:

 get
 wrInitCore

This function is called by:

 printDistances
 printSquareError
 writeGrnPart

¹In the following pages, only the scripts of this author are documented.

²Matlab script M2HTML by Guillaume Flandin is used to build cross-reference information.

F.2 computePR

PURPOSE

COMPUTEPR - Compute Precision-Recall or ROC curves and area under these curves.

SYNOPSIS

```
[area, x, y] = computePR(confidence, correct, curve)
```

DESCRIPTION

From a confidence predictions vector sorted in descending order and a vector that contains if yes or no a connection exists in reality, COMPUTEPR computes Precision-Recall (PR) curve (following DREAM2 pseudo-code and J. Davis paper[1].) and area under the curve. COMPUTEPR can also compute Receiver Operator Characteristic (ROC). If PR strategy is selected, results are the Area Under Curve of PR, x is Recall and y is Precision. For ROC, x is False Positive Rate (FPR) and y True Positive Rate (TPR).

[1] Jesse Davis and Mark Goadrich, 23rd International Conference on Machine Learning (ICML), Pittsburgh, PA, USA, 26th - 28th June, 2006

Input:

confidence : column vector of length NxN where N is the number of genes in the network. Elements are confidence (in interval [0,1]) predictions and are sorted in descending order.

correct : column vector of same length as confidence vector. Elements could take value 0 or 1. Element k in 'correct' vector is associated to element k in 'confidence' vector. If element k are set to 1 in 'correct' so an interaction exists in reality, otherwise 0.

curve : PR or ROC selection
{ 'pr' | 'roc' }

Output:

area : Area Under Curve

x : x-axis. Recall if curve='pr', FPR if curve='roc'

y : y-axis. Precision if curve='pr', TPR if curve='roc'

CROSS-REFERENCE INFORMATION

This function calls:

 confidence

This function is called by:

 writePredictions

F.3 confidence

PURPOSE

CONFIDENCE - Compute confidence predictions list.

SYNOPSIS

```
mat = confidence(numGenes, selfLoop, repos, dataFilename, ext, threshold)
mat = confidence(numGenes, selfLoop, repos, dataFilename, ext, threshold, targetW)
```

DESCRIPTION

CONFIDENCE extracts from a Wyrd batch file of optimizations, the weights matrix found, and after filtered these raw data with a threshold in order to set to 0 weak interactions, two confidence lists predictions are built, one for positive samples (interactions) and one for negatives ones. If weights matrix target is available, returns matrix contains to supplementary columns, the first is the 'correct' column vector for positive weights, the second one the 'correct' vector for negatives ones. (See COMPUTEPR help for more informations). Note that the implementation allow to use directly a batch file of optimizations generated by Wyrd, which lines have the following structure: [gene'index, bestRun, fit, squErr, [gene's params], wi0, wi1, ..., winumGenes]. If 'selfLoop' parameter is set to 0 in Wyrd, it must be also the case here, and in this case, each line of the batch file (which is a single gene's optimization) contains numGenes-1 in place of numGenes (all gene's self-interaction are equal to 0 and are not included in the evolved phenotype).

Also, CONFIDENCE is not responsive to the number of elements before weights elements in each line of batch optimizations file because CONFIDENCE extracts weights elements from the last element of the line.

Input:

numGenes.....: number of genes in the Gene Regulatory Network
 repos : directory where is the batch file of data
 selfLoop : gene's self-interactions, equal to 0 if there are not possible.
 dataFilename : name of the batch file, without extension
 ext : extension of dataFilename file, e.g. '.stat'
 threshold : all abs(weight) < threshold in batch data will be set to 0
 targetW : the numGenesXnumGenes, target W (Matlab matrix)
 targetW(i,j) := weight gj -> gi.

Output:

mat: a matrix of size (numGenes X numGenes) X (4 or 6)
 column 1 : source interaction genes index
 column 2 : inferred (target) genes index
 column 3 : confidence levels for positive weights
 column 4 : confidence levels for negative weights

If targetW is given:

column 5 : element = 1 if an excitatory interaction exists in reality, otherwise 0.

column 6 : idem for inhibitory interactions

CROSS-REFERENCE INFORMATION

This function calls:

-

This function is called by:



computePR



writePredictions

F.4 get

PURPOSE

GET - Gathers all get methods of wrInitCore instance, in order to access to wrInitCore's properties.

SYNOPSIS

```
val = get(obj, propName)
```

DESCRIPTION

Input:

obj : instance of wrInitCore
propName : the name of the property that we will access

Output:

val : desired value's property of wrInitCore's instance

CROSS-REFERENCE INFORMATION

This function calls:

-

This function is called by:

-  bestRun
-  printDistances
-  printGrnTopology
-  printSquareError
-  printTimeSeries
-  resizeProgressBar
-  wrgui
-  writeGrnPart
-  writeIndex
-  writeMenu
-  writeTsPart
-  wyrdReport

F.5 printDistances

PURPOSE

PRINTDISTANCES - Reads distances logs files generated during Wyrd processes and builds graphics representations.

SYNOPSIS

```
[nbGenerations, popSize] = printDistances(numGenes, repos, genericFileName, scale)
[nbGenerations, popSize] = printDistances(numGenes, repos, genericFileName, scale,
proBar)
```

DESCRIPTION

Input:

numGenes : number of genes in the Genes Regulatory Network
repos : path of the repository (no '/' at end) which contains
distances logs files
genericFileName : generic filename of distances log files (CSV format)
scale : type of graph representation
{'plot' | 'semilogx' | 'semilogy' | 'loglog'}
proBar : reference to a progress bar, if available

FORMAT genericFileName : #Generation D(I1) D(I2) ... D(In) where D=
Distance, I=Individual, n=popSize

Output:

nbGenerations : number of generations in Wyrd for this optimization
popSize : size of population (Evolutionary Algorithm)

IMPORTANT: Need a Wyrd Initializing Core called wrInit! Can be created
before calling the present script with wrInit = wrInitCore(); instruction.
If it does not already exist, this object will be created.

CROSS-REFERENCE INFORMATION

This function calls:

 get
 wrInitCore
 bestRun
 (resizeProgressBar)

This function is called by:

 wyrdReport

F.6 printGrnTopology

PURPOSE

PRINTGRNTOPOLOGY - Reads a Wyrd XML file which describes a Gene Regulatory Network (see Wyrd Handbook for the format) and builds a graphic representation using neato of Graphviz.

SYNOPSIS

```
[numGenes,genesName,W,paramName,paramData] = printGrnTopology(repos, xml-File)
```

DESCRIPTION

Input:

repos : path of the repository (no '/' at end) which contains xmlFile
xmlFile : GRN XML file




Output:

numGenes ... : number of genes in the GRN
genesName : list of all gene's names
W : weights matrix
paramName : list of all parameter's names of all genes
paramData : vector of all parameter's value of all genes


IMPORTANT: Need a Wyrd Initializing Core called wrInit! Can be created before calling the present script with wrInit = wrInitCore(); instruction. If it does not already exist, this object will be created.

CROSS-REFERENCE INFORMATION

This function calls:

 get
 wrInitCore
 parseChildNodes

This function is called by:

 wyrReport

F.7 printSquareError

PURPOSE

PRINTSQUAREERROR - Reads square error logs files generated during Wyrd processes and builds graphics representations.

SYNOPSIS

```
[nbGenerations, popSize] = printSquareError(numGenes, repos, genericFileName,
scale)
[nbGenerations, popSize] = printSquareError(numGenes, repos, genericFileName,
scale, proBar)
```

DESCRIPTION

Input:

numGenes.....: number of genes in Genes Regulatory Network
 repos : path of the repository (no '/' at end) which contains
 distances logs files (CSV format)
 genericFileName: generic filename of distances log files
 scale : type of graph representation
 {'plot' | 'semilogx' | 'semilogy' | 'loglog'}
 proBar : reference to a progress bar, if available

FORMAT genericFileName : #Generation D(I1) D(I2) ... D(In) where D=
 Distance, I=Individual, n=popSize


Output:

nbGenerations : number of generations in Wyrd for this optimization
 popSize : size of population (Evolutionary Algorithm)

IMPORTANT: Need a Wyrd Initializing Core called wrInit! Can be created
 before calling the present script with wrInit = wrInitCore(); instruction.
 If it does not already exist, this object will be created.

CROSS-REFERENCE INFORMATION

This function calls:

 get
 wrInitCore
 bestRun
 (resizeProgressBar)

This function is called by:

 wyrReport

F.8 printTimeSeries

PURPOSE

PRINTTIMESERIES - Reads Time Series experiments datasets generated during Wyrd process (CSV format) and builds graphic representations.

SYNOPSIS

```
numTimeSeriesExp = printTimeSeries(numGenes, repos, inputFile)
numTimeSeriesExp = printTimeSeries(numGenes, repos, inputFile, proBar)
```

DESCRIPTION

Input:

numGenes : number of genes in the Genes Regulatory Network
 repos : path of the repository (no '/' at end) which contains
 distances logs files
 inputFile : generic filename of Time Series Experiments (CSV format)
 proBar : reference to a progress bar, if available

FORMAT inputFile : "Time" "G0" "G1" ... "GN"
 t00 G0mRNA G1mRNA ... | Time
 t01 ... | Series
 : | Experiment
 t0n ... | 0
 t1n
 : ..
 tmn ...





Output:

numTimeSeriesExp : the number of Time Series Experiments

IMPORTANT: Need a Wyrd Initializing Core called wrInit! Can be created before calling the present script with wrInit = wrInitCore(); instruction. If it does not already exist, this object will be created.

CROSS-REFERENCE INFORMATION

This function calls:

 get
 wrInitCore
 readColData
 (resizeProgressBar)

This function is called by:

 wyrReport

F.9 resizeProgressBar

PURPOSE

RESIZEPROGRESSBAR - Handles the size of an improvised progress bar for Matlab GUI.

SYNOPSIS

```
resizeProgressBar(bar, orient, usableSize)
```

DESCRIPTION

This progress bar is based on the use of a 'text' (style) uicontrol, defined by no 'String' and color background different of the interface background color.

Input:

bar : reference to the uicontrol 'text' (style of the uicontrol)
orient : the orientation of progress bar
{0 := horizontal | 1 := vertical}
usableSize : the subtracted size

Output:

-





IMPORTANT: Need a Wyrld Initializing Core called wrInit! Can be created before calling the present script with wrInit = wrInitCore(); instruction. If it does not already exist, this object will be created.

CROSS-REFERENCE INFORMATION

This function calls:

 get
 set

This function is called by:

 printDistances
 printSquareError
 printTimeSeries
 wyrldReport

F.10 set

PURPOSE

SET - Gathers all get methods of wrInitCore instance, in order to set wrInitCore's properties.

SYNOPSIS

```
obj = set(obj, propName, value)
```

DESCRIPTION

Input:

obj : instance of wrInitCore
propName : the name of the property that we will modify
value : set propName with value

Output:


obj : modified instance of wrInitCore

CROSS-REFERENCE INFORMATION

This function calls:

-

This function is called by:

 resizeProgressBar
 wrgui
 wyrdReport

F.11 wrInitCore

PURPOSE

WRINITCORE - This class includes all properties necessary to the right process of Wyrld Reports Generator. The values hard coded in this file can be used as default settings for Wyrld Report processes.

SYNOPSIS

```
wrInit = wrInitCore()
```

DESCRIPTION

Input:

-

Output:

wrInit : new instance of wrInitCore

CROSS-REFERENCE INFORMATION

This function calls:

-

This function is called by:

-  bestRun
-  printDistances
-  printGrnTopology
-  printSquareError
-  printTimeSeries
-  wrgui
-  writeGrnPart
-  writeIndex
-  writeMenu
-  writeTsPart
-  wyrldReport

F.12 writeGrnPart

PURPOSE

WRITEGRNPART - Writes the GRN part of Wyrd reports.

SYNOPSIS

```
writeGrnPart(htmlFile, globVar_)
```

DESCRIPTION

WRITEGRNPART writes the GRN part of Wyrd reports, containing experiment's generalities, GRN topologies representations (uses Graphviz), gene's optimizations with distance (fitness, between target and evolved datasets) and square error (mean square errors between target (if available) and evolved gene's parameters).

Input:

htmlFile : flow to the main page HTML file
 globVar_ : specific structure that contains several parameters extracted during previous operations (numGenes, genesName, etc.) in wyrdReport script.



Output:

-

IMPORTANT: Need a Wyrd Initializing Core called wrInit! Can be created before calling the present script with wrInit = wrInitCore(); instruction. If it does not already exist, this object will be created.

CROSS-REFERENCE INFORMATION

This function calls:

 get
 wrInitCore
 bestRun

This function is called by:

 wyrdReport

F.13 writeIndex

PURPOSE

WRITEINDEX - Create the HTML index file index.html of Wyrd reports.

SYNOPSIS

```
writeIndex()
```

DESCRIPTION

Input:

-

Output:

-

IMPORTANT: Need a Wyrd Initializing Core called wrInit! Can be created before calling the present script with `wrInit = wrInitCore();` instruction. If it does not already exist, this object will be created.

CROSS-REFERENCE INFORMATION

This function calls:

 `get`
 `wrInitCore`

This function is called by:

 `wyrdReport`

F.14 writeMenu

PURPOSE

WRITEMENU - Create the HTML menu file menu.html of Wyrd reports.

SYNOPSIS

```
writeMenu()
```

DESCRIPTION

Input:

-

Output:

-


IMPORTANT: Need a Wyrd Initializing Core called wrInit! Can be created before calling the present script with `wrInit = wrInitCore();` instruction. If it does not already exist, this object will be created.

CROSS-REFERENCE INFORMATION

This function calls:

 `get`
 `wrInitCore`

This function is called by:

 `wyrdReport`

F.15 writePredictions

PURPOSE

WRITEPREDICTIONS - Writes prediction confidences lists and processes PR or ROC analysis.

SYNOPSIS

```
[area, x, y] = writePredictions(numGenes, loop, statfile, thres, curve, sign, targetType, targetW)
```

DESCRIPTION

Input:

Necessary to write prediction confidences list:

numGenes : number of genes in the Gene Regulatory Network
loop : if auto gene connections permitted, loop=1, else 0.
statfile : batch file of Wyrld optimizations. More generally, this file must have the following structure (illustrated with Matlab syntax): [Wi=1;Wi=2;...] where Wi are the weights matrix found at ith run of Wyrld or by another method. So, each line contain at least (numGenes-1) elements if loop=0 and file contains numGenes x numGenes lines. Each elements must be separated by a tab. For more informations, see CONFIDENCE help.
thres : threshold, see CONFIDENCE help.

Necessary to process PR or ROC analysis:

curve : type of analysis {'pr' | 'roc'}
sign : type of weights
{1=positive weights | 0=negative weights}
targetType : specify the form of the given target GRN topology
{'xml'=Wyrld XML GRN file | 'mat'=Matlab matrix}
targetW : target GRN topology (weights matrix). If targetType='xml', targetW is a file, if targetType='mat', targetW is a Matlab matrix of size numGenes x numGenes





Output:

area : Area Under Curve
x : x-axis. Recall if curve='pr', FPR if curve='roc'
y : y-axis. Precision if curve='pr', TPR if curve='roc'

IMPORTANT: Need a Wyrld Initializing Core called wrInit! Can be created before calling the present script with wrInit = wrInitCore(); instruction. If it does not already exist, this object will be created.

CROSS-REFERENCE INFORMATION

This function calls:

-  `computePR`
-  `confidence`
-  `parseChildNodes`
-  `strsplit`

This function is called by:

-

F.16 writeSsPart

PURPOSE

WRITESSPART - Writes Steady-States results.

SYNOPSIS

```
writeSsPart(htmlFile, globVar_)
```

DESCRIPTION

WRITEGRNPART writes the GRN part of Wyrd reports, containing experiment's generalities, GRN topologies representations (uses Graphviz), gene's optimizations with distance (fitness; between target and evolved datasets) and square error (mean square errors between target (if available) and evolved gene's parameters).

Input:

htmlFile : flow to the main page HTML file
globVar_ : specific structure that contains several parameters extracted during previous operations (numGenes, genesName, etc.) in wyrdReport script.

Output:

-


IMPORTANT: Need a Wyrd Initializing Core called wrInit! Can be created before calling the present script with `wrInit = wrInitCore();` instruction. If it does not already exist, this object will be created.

CROSS-REFERENCE INFORMATION

This function calls:

-

This function is called by:

 wyrdReport

F.17 writeTsPart

PURPOSE

WRITETSPART - Writes Time Series result, mainly Time Series integration curves, one by gene and by experiment.

SYNOPSIS

```
writeTsPart(htmlFile, globVar_)
```

DESCRIPTION

WRITEGRNPART writes the GRN part of WyrD reports, containing experiment's generalities, GRN topologies representations (uses Graphviz), gene's optimizations with distance (fitness; between target and evolved datasets) and square error (mean square errors between target (if available) and evolved gene's parameters).

Input:

htmlFile : flow to the main page HTML file
 globVar_ : specific structure that contains several parameters extracted during previous operations (numGenes, genesName, etc.) in wyrDReport script.

Output:

-

IMPORTANT: Need a WyrD Initializing Core called wrInit! Can be created before calling the present script with wrInit = wrInitCore(); instruction. If it does not already exist, this object will be created.

CROSS-REFERENCE INFORMATION

This function calls:

 get
 wrInitCore

This function is called by:

 wyrDReport

F.18 **wyrdReport**

PURPOSE

WYRDREPORT - Generates a HTML report from Wyrd log files.

SYNOPSIS

```
wyrdReport()           : Default use
wyrdReport(wrInit)     : Uses a given wrInitCore instance previously
                        and called wrInit instantiated.
wyrdReport(wrInit, guiHandles) : If called by a Graphical Interface
```

DESCRIPTION

WYRDREPORT generates a HTML report from Wyrd log files generated during a wyrd process. The report is divided into three parts:

- Generalities: target (if available) and evolved GRN topologies, and for each optimization, distance (fitness) and square error representation. The best solution for each optimization are also displayed.
- Steady-States Experiments informations
- Time Series Experiments: for each experiment and for each gene, the integration curves of target and evolved GRN are displayed on the same graph. The mean square distance on all points for each integrations are also displayed.

To illustrate report, several pictures are processed. The script tools used for this are the following:

- Grn topologies : printGrnTopology.m (uses Graphviz)
- Distance graphs : printDistances.m
- Square Error graphs : printSquareError.m
- Time Series Integrations : printTimeSeries.m

HTML pages are written by many scripts:

- index.html : writeIndex.m
- menu.html : writeMenu.m
- 'reportId'.html -> Grn part : writeGrnPart.m
- 'reportId'.html -> SS part : writeSsPart.m
- 'reportId'.html -> TS part : writeTsPart.m

Input:

- wrInit : wrInitCore instance. See the following IMPORTANT note.
- guiHandles : structure of all graphical control of a GUI.

Output:

-

IMPORTANT: Need a Wyrd Initializing Core called wrInit! Can be created before calling the present script with `wrInit = wrInitCore();` instruction. If it does not already exist, this object will be created.

CROSS-REFERENCE INFORMATION

This function calls:

-  `get`
-  `set`
-  `wrInitCore`
-  `printDistances`
-  `printGrnTopology`
-  `printSquareError`
-  `printTimeSeries`
-  `(resizeProgressBar)`
-  `writeGrnPart`
-  `writeIndex`
-  `writeMenu`
-  `writeSsPart`
-  `WriteTsPart`

This function is called by:

-  `wrgui`