OPTIMISATION OF DEFINITION STRUCTURES & PARAMETER VALUES IN PROCESS ALGEBRA MODELS USING EVOLUTIONARY COMPUTATION

DAVID RALIX OAKEN

Doctor of Philosophy

Computing Science and Mathematics, School of Natural Sciences
University of Stirling

March 2014

David Ralix Oaken: *Optimisation of Definition Structures & Parameter Values in Process Algebra Models Using Evolutionary Computation,* Doctor of Philosophy, © March 2014

DECLARATION

I declare that all work presented in this document is my own unless expressly stated otherwise, whereupon acknowledged work is fully referenced. Furthermore, I certify that all work presented is true and in accordance with academic rules and ethical conduct.

Stirling, March 2014

David Ralix Oaken

ABSTRACT

Process Algebras are a Formal Modelling methodology which are an effective tool for defining models of complex systems, particularly those involving multiple interacting processes. However, describing such a model using Process Algebras requires expertise from both the modeller and the domain expert. Finding the correct model to describe a system can be difficult. Furthermore, even with the correct model, parameter tuning to allow model outputs to match experimental data can also be both difficult and time consuming.

Evolutionary Algorithms provide effective methods for finding solutions to optimisation problems with large and noisy search spaces. Evolutionary Algorithms have been proven to be well suited to investigating parameter fitting problems in order to match known data or desired behaviour.

It is proposed that Process Algebras and Evolutionary Algorithms have complementary strengths for developing models of complex systems. Evolutionary Algorithms require a precise and accurate fitness function to score and rank solutions. Process Algebras can be incorporated into the fitness function to provide this mathematical score.

Presented in this work is the Evolving Process Algebra (EPA) framework, designed for the application of Evolutionary Algorithms (specifically Genetic Algorithms and Genetic Programming optimisation techniques) to models described in Process Algebra (specifically PEPA and Bio-PEPA) with the aim of evolving fitter models.

The EPA framework is demonstrated using multiple complex systems. For PEPA this includes the dining philosophers resource allocation problem, the repressilator genetic circuit, the G-protein cellular signal regulators and two epidemiological problems: HIV and the measles virus. For Bio-PEPA the problems include a biochemical reactant-product system, a generic genetic network, a variant of the G-protein system and three epidemiological problems derived from the measles virus.

Also presented is the *EPA Utility Assistant* program; a lightweight graphical user interface. This is designed to open the full functionality and parallelisation of the EPA framework to beginner or naive users. In addition, the assistant program aids in collating and graphing after experiments are completed.

ACKNOWLEDGMENTS

I would like to thank my wonderful parents Lynne and Raymond Marco for encouraging me and supporting me through life and academia; my beloved husband Kaspian Oaken for keeping me sane; my primary supervisor Carron Shankland for pushing me to network and publish; and my secondary supervisor David Cairns for putting me on the path to studying evolutionary computation.

Another big thank you to SICSA (*Scottish Informatics and Computer Science Alliance*) for the prize studentship award which funded my studies.

A thanks to all family, friends, colleagues and strangers that have influenced how I got to where I am now.

LIST OF PUBLICATIONS

- D. Marco, C. Shankland, and D. Cairns, "Optimisation of Process Algebra Using Evolutionary Computation," in *IEEE Congress on Evolutionary Computation*, pp. 1296–1301, IEEE, 2011.
- D. Marco, E. Scott, D. Cairns, A. Graham, J. Allen, S. Mahajan, and C. Shankland, "Investigating co-infection dynamics through evolution of Bio-PEPA model parameters," in *The 10th Conference on Computational Methods in Systems Biology*, vol. 7605, pp. 227–246, LNCS, Springer-Verlag, 2012.
- D. Marco, C. Shankland, and D. Cairns, "Evolving Bio-PEPA Process Algebra Models Using Genetic Programming," in *GECCO Conference Proceedings*, pp. 177–183, ACM, 2012.

These papers were published under the author's pre-married name. David Marco changed his name to David Oaken in January 2013. Supporting documentation is available upon request.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

ABC	Approximate Bayesian Computation
ACO	Ant Colony Optimisation
Bio-PEPA	Biologically-Inspired PEPA
CTMC	Continuous Time Markov Chain
DE	Differential Evolution
EA	Evolutionary Algorithm
EC	Evolutionary Computation
ECJ	Evolutionary Computation Java
EDA	Estimation of Distribution Algorithm
EM	Expectation-Maximisation
EPA	Evolving Process Algebra
FM	Formal Modelling
GA	Genetic Algorithm
GP	Genetic Programming
HODE	Higher-order Ordinary Differential Equation
IDE	Integrated Design Environment
NFL	No Free Lunch
ODE	Ordinary Differential Equation
PA	Process Algebra
PC	Process Calculi
PEPA	Performance Evaluation Process Algebra
PSO	Particle Swarm Optimisation
SBGN	Systems Biology Graphical Notation
SBML	Systems Biology Markup Language
SBSI	Systems Biology Software Infrastructure
SBW	Systems Biology Workbench
SEIR	Susceptible-Exposed-Infected-Recovered
SEI	Susceptible-Exposed-Infected
SIR	Susceptible-Infected-Recovered
SI	Swarm Intelligence
WSCCS	Weighted Synchronous Calculus of Communicating Systems

1

INTRODUCTION

Proposed here is the application of Evolutionary Computation (EC) techniques for the optimisation of models to more accurately describe systems defined in Process Algebras (PAs) in order to match existing time series data. The hybrid approach will utilise the complementary strengths of EC techniques and PA modelling while negating individual weaknesses for the description of complex bioscience systems.

Process Algebra is one of many established formal methods adopted by computational biology for constructing models of complex systems. It is especially good for capturing interactions between individual elements [4]: for instance, describing a biological system which may be viewed as large networks of interacting components, where they themselves have individual complex stochastic behaviour. This field of research and the contributions of this work hold potential for assisting users in creating models to describe complex systems: domain experts most likely will not be well-versed in Formal Modelling (FM) techniques. Therefore a individual focused approach may be more familiar when producing early-stage models. Once created, these early models can be optimised and refined into accurate descriptions of the systems.

Evolutionary Computation (also and interchangeably referred to as Evolutionary Algorithm (EA)) techniques are powerful and flexible search algorithms, capable of finding robust solutions in a large or deceptive search space [2, 3]. More traditional, systematic or exhaustive search techniques frequently become too computationally costly to apply to these sorts of problems.

Independently, the domains of Formal Modelling and Evolutionary Computation are established and well explored. Research into combining these disciplines is still in its infancy. As mentioned above, these two approaches have complementary strengths which can negate the weaknesses of using either individually. For example, a valid and well-defined model along with relevant performance criteria are all vital for an Evolutionary Algorithm to successfully find robust solutions. The use of a formal approach such as a Process Algebra can provide this model.

However, a general difficulty in using Process Algebras is fine-tuning models to accurately match known data. This uncertainty as to the correct model structure provides an appropriate search space ideally suited to the application of an Evolutionary Algorithm. This is especially true as the size and complexity of the PA model increases: the computational power required for the application of systematic search techniques becomes infeasible.

The Process Algebras used are Performance Evaluation Process Algebra (PEPA) [40] and the Biologically-Inspired PEPA (Bio-PEPA) [20], with the majority of work done using the latter. The Evolutionary Computation techniques used here are a Genetic Algorithm (GA) for parameter matching work and a tree-based Genetic Programming (GP) technique for model structure optimisation. These components are combined into the developed Evolving Process Algebra (EPA) framework. The research focuses mostly on the combination of the two domains as well as reproducibility, therefore more complex EA techniques are not utilised.

1.1 THESIS STATEMENT

The aim of this research is to develop a bridging framework by combining Evolutionary Computation techniques with Process Algebra models. The purpose of such a framework will be in assisting domain experts. The Evolving Process Algebra (EPA) framework will be used for parameter matching optimisation and direct optimisation of the Process Algebra models, written in PEPA or Bio-PEPA notation.

The focus is on biological and epidemiological models with available existing data. Knowing this focus, the EPA framework can incorporate prebuilt constructs to improve experimental results.

Domain experts most likely will not be well versed in Formal Modelling techniques. Process Algebras offer a lower level, individual based approach to modelling. This may be more familiar when producing early-stage models which can be optimised using the developed EPA framework. The framework itself is a command-line entity with no user interface. The *EPA Utility Assistant*, a complimentary program, provides the user interface for the domain expert to use the EPA framework.

1.2 THESIS STRUCTURE

The structure of this document is as follows. Chapter 2 provides background information and elaborates on the current state of the art with descriptions of the Evolutionary Computation and Process Algebra fields bridged by the EPA framework. Related work currently being done in the field follows in Chapter 3. This includes examples of immediate neighbours and alternatives to this field, such as direct optimisation of Ordinary Differential Equation (ODE) structures, simulated annealing and Bayesian inference. The technical specifics of the EPA framework are detailed in Chapter 4 which provides information on the framework's implementation and functions. The choice of default values and justifications for these values are also presented in this chapter.

The following four chapters detail the experiments performed for incrementally developing and testing the EPA framework. Chapter 5 details the PEPA models utilised by the EPA

framework during initial Genetic Algorithm parameter matching experiments. Chapter 6 moves the focus onto Genetic Programming optimisation of systems described in Bio-PEPA; specifically the species definition segments of those models. Following the species definition optimisation, the kinetics definition experiments are detailed in Chapter 7. Chapter 8 details the final experiments which combine the work of the three previous chapters to perform Genetic Algorithm and Genetic Programming optimisation simultaneously.

The conclusions and final discussion are presented in Chapter 9 which summarises the work done in this thesis and discusses the overarching impacts of the EPA framework. Future uses for the framework and possible enhancements are also presented in this chapter.

Following the final chapter and finishing this document is Appendix A, *EPA Utility Assistant*. This section presents the assistant program designed to aid with using the EPA framework with the *HTCondor* high throughput computing toolkit (Chapter 4 provides details of HTCondor and its implementation). The assistant program also aids in post-experiment result collation and graphing.

2

STATE OF THE ART

The hybrid approach of this work bridges two fields of research: Evolutionary Computation (EC) and Formal Modelling (FM). This chapter provides an overview of both overarching fields plus specifics on Genetic Algorithms (GAs), Genetic Programming (GP), Performance Evaluation Process Algebra (PEPA) and Biologically-Inspired PEPA (Bio-PEPA).

The Evolving Process Algebra (EPA) framework is discussed fully in Chapter 4. In short, the framework is applied to partial Process Algebra models which describe complex systems to improve their accuracy. This chapter will also discuss complex systems and the related emergent behaviour to provide background on what is being modelled by the Process Algebras and why they are useful.

2.1 COMPLEX SYSTEMS

A complex system is a system comprised of individual parts which are interconnected, can communicate and exhibit one or more specific behaviours [44]. Examples of complex systems can be found in many problem domains; for example:

SYSTEMS BIOLOGY: epidemiology models, metabolic networks, cell signalling networks

ECONOMICS: artificial stock market, currency exchange predictors

GEOPHYSICS: climate modelling, rock formation patterns

SOCIAL SCIENCE: Schelling's Tipping Point, Artificial Anasazi

In any domain, the complexity of a system can be one of two types: *disorganised* or *organised* complexity [68]. Disorganised complexity is caused by a huge number of individuals existing simultaneously in the system. Organised complexity can exist in systems of any scale and is system specific; it is the behaviours, connectedness and interdependencies of parts inside the system that dictate the system's complexity.

The Greek philosopher Aristotle famously quoted: *"The whole is greater than the sum of its parts"* which particularly applies to emergent behaviour. By its nature, the overall behaviour of a system which exhibits emergent properties cannot be predicted by considering the parts in isolation. Only when the complex system is observed operating as a whole can we see the emergent properties or *emergent behaviour*.

For an example of emergent behaviour, consider the social interaction models posited by Pruitt & Gahagan [59]. Pruitt was influential in the development of three important social

models: the *aggressor-defender* model, the *conflict spiral* model and the *structural change* model. The most signature of Pruitt's social models is the aggressor-defender model.

The rules of the aggressor-defender interaction are as follows: each individual in the system will arbitrarily target two other individuals when the system starts; the two targets are then labelled as *aggressor*, *defender* or *victim*. The behaviour associated with the system depends on the configuration of these labels. Figures 2.1 and 2.2 illustrate these two configurations and related behaviours as simulated in Java by *Icosystem* [24]. The colour of the individuals is added to make tracking movements easier; it is not related to the labels.

In configuration one (the *aggressor* behaviour) the individual is threatened by one target which it labels an aggressor and must protect itself using the second target which it labels a defender. Every individual in the system follows the same configuration and each individual can be targeted multiple times.

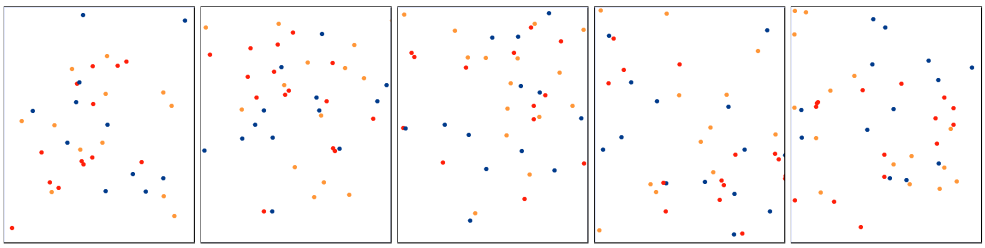


Figure 2.1: Configuration one (aggressor behaviour) captured at two second intervals

At all times, every individual attempts to keep its defender between itself and its aggressor. Figure 2.1 displays the behaviour of the system under these rules: individuals start distributed around the space and seem to drift around randomly to maintain the positioning of their aggressor and defender targets. Individuals will drift in this manner indefinitely.

In configuration two (the *defender* behaviour) the individual labels its first target as a victim which is under threat from its second target which is labelled an aggressor. As in configuration one, each individual in the system follows the same configuration and each individual can be targeted multiple times.

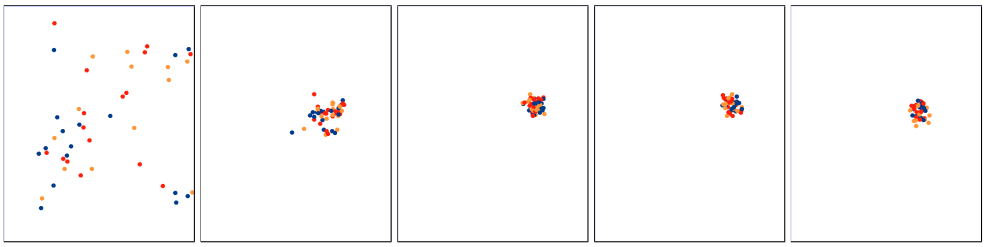


Figure 2.2: Configuration two (defender behaviour) captured at two second intervals

The rules of this configuration dictate the individual must interpose itself between its victim and its aggressor. Figure 2.2 illustrates the behaviour of the system with these rules: as in configuration one, individuals start distributed around the space. However, all individuals in

the system quickly converge and cluster together. Individuals will remain clustered indefinitely but may move together around the space.

In neither the *aggressor* or *defender* configuration is it possible to predict the behaviour of the system by examining the properties of nodes with the *aggressor*, *defender* or *victim* labels independently. Frequently, only by simulating the system as a whole does the emergent behaviour become apparent.

2.2 PROCESS ALGEBRAS

Process Algebras (PAs), also known as Process Calculi (PC), are a Formal Modelling (FM) technique with a strict mathematical semantic structure. This formality allows rigorous testing and analysis of the described systems. The mathematical structure and small collection of operators allows large complex systems to be described using a small collection of semantic building blocks.

Originally, Process Algebras were developed in the early 1970s to model distributed computation systems. For a full historical overview, see Baeten [4]. By nature, computer networks are highly distributed, concurrent, involve communication and also co-operation. This behaviour of interactions, communications and synchronisations can be seen in biological and epidemiological systems. Interest in using Process Algebras for the modelling of bioscience systems has been increasing over the last decade [22, 23, 58].

Process Algebras describe complex systems by the composition of multiple parallel agents, called processes. Each process carries out actions, which may require communication with other processes to complete. Choices between actions can be made in a deterministic or stochastic way.

One of the main advantages of Process Algebras is modelling at an individual level. By tracking individuals as they navigate a system, local changes can become tipping points that lead to changes in the overall system, thus resulting in *emergent behaviour*, as discussed previously in Section 2.1.

The constrained nature of Process Algebra syntax makes them suitable for use with Genetic Programming optimisation techniques. They are mathematically based and allow rigorous analysis, meaning it is possible to effectively evaluate the fitness of candidate solutions in several ways during the selection process, see Section 2.3.1.2.

2.2.1 *PEPA*

Performance Evaluation Process Algebra (PEPA) was developed at the University of Edinburgh by Hillston [40]. PEPA is described as an enhanced Process Algebra originally designed for performance modelling. It has a broad application area including biological [15] and

epidemiological systems [10]. Furthermore, it is not so restricted to those fields as to be unusable in other problem domains; for instance, priority queue modelling [19].

Consider Figure 2.3, a simple Susceptible-Exposed-Infected-Recovered (SEIR) epidemiological model written in PEPA. The PEPA syntax consists of a few elements which are combined to describe complex systems.

$$rContact = 0.5 \qquad rInfect = 0.1 \qquad rRecover = 0.4$$

$$\begin{aligned} S &\stackrel{def}{=} (contact, \top).E \\ E &\stackrel{def}{=} (infect, rInfect).I \\ I &\stackrel{def}{=} (contact, \top).I + (recover, rRecover).R \\ R &\stackrel{def}{=} (idle, \top).R \end{aligned}$$

$$\begin{aligned} Transmitter &\stackrel{def}{=} (contact, rContact).Transmitter + (recover, \top).Dormant \\ Dormant &\stackrel{def}{=} (infect, \perp).Transmitter \end{aligned}$$

$$(S[990] \;||\; I[10]) \underset{contact,infect,recover}{\bowtie} (Transmitter[990] \;||\; Dormant[10])$$

Figure 2.3: SEIR epidemiology model written in PEPA

Components (S, E, I, R, *Transmitter* and *Dormant*) describe any individual inside the system and the states in which they can exist. In epidemiology, for example, an individual could be a human which can move from a susceptible state (S), to being exposed to some pathogen (E), infected by that pathogen (I) before eventually entering a recovered state (R). Consider malaria transmitted via mosquito bite as an example; a susceptible human can enter the mosquito's territory and become exposed, once bitten they will enter an infected state until eventually being cured and entering a recovered state. The roles of additional components, such as *Transmitter* and *Dormant*, are discussed later in this section.

Actions (*contact*, *infect*, *recover*, *idle*) are performed and will lead to transitions from one state into another state. For instance, a susceptible individual (S) can perform a *contact* action and become exposed (E). An infected individual performing the same action will remain in the infected state. PEPA can be viewed as a Markov Chain; due to the structure of this underlying Markov Chain, the state still transitions by following an edge back into the infected state. If one action appears in many component definitions (such as *contact* appearing in S, I and *Transmitter*), it is handled by the system equation, explained below.

Rates (*rContact*, *rInfect*, *rRecover*) determine when an action can occur; the larger the rate value, the more readily all associated actions becomes available to be performed. Rates are acceptable in numeric form or scientific e notation, i.e. 0.001 or $1.0e-3$. The special rate *infinity* ($\top$) denotes an action which is always available, such a rate will be restricted by the rates of cooperating actions; discussed below with communication. Similarly, the special rate $\perp$ denotes an action which is rarely available. Consider the recovered state (R) in Figure 2.3; this

species requires a definition but performs no meaningful action, $\perp$ is useful for representing the rate for such an action.

Communication between components and the initial population sizes are defined in the final line of the PEPA syntax: the system equation. Components on the left can communicate with those on the right over actions which appear under the PEPA cooperation symbol ($\bowtie_{actionSet}$). In the SEIR example in Figure 2.3, the *contact*, *infect* and *recover* actions connect the S and I components to the *Transmitter* and *Dormant* components.

Choice (denoted by a +) allows different actions to occur. The path chosen is determined by which action and cooperating components are ready first. Readiness is determined by all communicating components for a particular action being present when a rate becomes stochastically available.

One restriction of the PEPA syntax is that components cannot be created or destroyed. Instead creation is modelled by recruiting from idle pools (*Transmitter* and *Dormant*) of waiting components and destruction returns individuals to those pools. This syntactical structure is also required for the modelling of one-to-one communication paths instead of the default mass communication. For instance, to allow one infected to encounter one susceptible in a *contact* action; instead of all infected individuals encountering all susceptible individuals simultaneously.

As previously mentioned, Process Algebras allow for rigorous algebraic testing and analysis. A plug-in, also developed at Edinburgh University, allows PEPA models to be interpreted in many ways [29], including as Continuous Time Markov Chains (CTMCs) and Ordinary Differential Equations (ODEs).

This research uses the simulation option to produce the time series data results. ODEs view the system from the population level and may miss emergent behaviours which may be of interest to modellers. Simulation will track individual scale interactions and changes in population level to capture the emergent behaviours. This is the default method used by the EPA framework, however ODE evaluation is possible if the domain expert prefers.

2.2.2 *Bio-PEPA*

Biologically-Inspired PEPA (Bio-PEPA) is a descendant of PEPA aimed at problems in the biological domain, developed by Ciocchetta with PEPA's creator Hillston [20]. Bio-PEPA models utilise a different view of complex systems than PEPA. With PEPA, agents are considered along with activities in which they may participate and the paths they may follow during execution in a *reagent-centric* view. However, with Bio-PEPA, the focus is on changing values or levels of those agents in a *pathway-centric* view.

Consider the model in Figure 2.4 which represents the biochemical reaction $2X + Y \rightarrow 3Z$. A standard PEPA view of this system is that molecule X can change state to behave like molecule

Z. Similarly, molecule Y can also change state to behave like molecule Z. This does not make sense in the context of biochemistry, therefore the Bio-PEPA view of the system is that levels of species X and Y decrease as number of species Z increases.

$$
\begin{aligned}
& r = 0.001 \\
& kineticLawOf\ alpha : fMA\ (r) \\
\\
& X \stackrel{def}{=} (alpha, 2) \ll X \\
& Y \stackrel{def}{=} alpha \ll Y \\
& Z \stackrel{def}{=} (alpha, 3) \gg Z \\
\\
& X[200]\ <alpha>\ Y[100]\ <*>\ Z[0]
\end{aligned}
$$

Figure 2.4: Biochemical Reactant-Product model written in Bio-PEPA

The simple model in Figure 2.4 demonstrates the core components of the Bio-PEPA syntax. A system modelled in Bio-PEPA can have rates (r). Just as in PEPA, these can be numeric or use scientific e notation.

Species (X, Y and Z) are defined in a similar manner to PEPA syntax. Note that the expression $Y \stackrel{def}{=} (alpha, 1) \ll Y$ is equivalent to $Y \stackrel{def}{=} alpha \ll Y$. When the stoichiometry coefficient is 1, it and the brackets may be removed. Additionally the expression $Y \stackrel{def}{=} alpha \ll Y$ is equivalent to $Y \stackrel{def}{=} alpha \ll$. The final species identifier is a throwback to PEPA semantics where this indicates a species transition. No such species transition occurs in Bio-PEPA so this identifier can be omitted in any species definition. These are both examples of syntactic shorthand which do not change the model's function.

The operators $\gg$ and $\ll$ (shortened from the ASCII >> and << which the EPA utilises) indicate that the species either increases in the interaction (a product), or decreases in the interaction (a reactant), respectively. When an action linked to these operators is executed, the level of all species involved is either raised or lowered by their stoichiometry coefficient (or by 1 if no coefficient is listed).

Additional operators available include the $(+)$ and $(-)$ to indicate an activator (or enzyme: a biological catalyst) or an inhibitor respectively. Finally, there is a generic modifier $(.)$ which is neither an activator nor an inhibitor and indicates the species is involved in the reaction without altering the behaviour of that reaction. If a species uses one of these operators in a reaction then its quantity remains unchanged: these operators only modify the behaviour of other species involved in the reaction. For example, this may be used to describe a molecule which must be present to participate in a reaction, but which is not depleted or increased by that reaction.

Just as in PEPA, Bio-PEPA syntax supports species definitions which can contain multiple statements separated by the choice operator (a plus, +). Which path is chosen is dictated by the rates of each action involved: actions with faster rates are more likely to occur, assuming

the activities in which they take part are all enabled. For instance, the *alpha* kinetic action in Figure 2.4 can only be performed if the X, Y and Z species are all available: i.e. involved species are not performing another action and have sufficient levels to satisfy their stoichiometry coefficients.

There are inbuilt kinetic laws which define the rates of activity (*alpha*) based on rates and stoichiometry coefficients. This example uses the kinetic law of mass action (*fMA*) and the information that two molecules of X are needed for every one molecule of Y to create three molecules of Z. Kinetic rates may also be defined by mathematical expressions which can involve species levels, arithmetic functions and trigonometric functions.

For a model to be considered well-formed, only species which cooperate on an action can be used in the definition of that action. Therefore, a species definition should include an action in order for that species to appear in the action definition.

Similarly to PEPA syntax, the final line is the system equation that defines communication paths and initial population sizes. In this example, X and Y can communicate only over action *alpha* while Z can communicate over any action (*alpha* is the only action, but this illustrates the syntax). The populations start at size 200, 100 and 0 respectively; if no population size is stated for a species, it is assumed to be one.

A further syntactical element exists; the *compartment* allows a notion of separate physical space where species can exist. An epidemiology example could include one species of pathogen carrying individuals which could move between two or more physical locations while retaining the same properties of that individual. The compartment is partially supported by the EPA framework: GA parameter matching can be performed on a model that contains compartments, however GP optimisation techniques of species definitions to include compartments are not supported.

The concept of separate physical spaces and how individuals can move between such spaces requires understanding of the represented system. Genetic Programming techniques do not understand the systems to which they are applied, as such, they would be unable to incorporate this level of understanding during evolution. An additional reason compartments were not included was the capability to represent the same notions of physical space using naming conventions; i.e. by using differently named species definitions which each model the same behaviours.

A sibling plug-in to the PEPA plug-in was also developed at the University of Edinburgh for use with Bio-PEPA [21]. The plug-in is capable of evaluating models in many ways, such as simulation, CTMCs or ODE analysis. The EPA framework utilises this plug-in to simulate Bio-PEPA models; just as the work with PEPA models, simulation is used to capture the emergent properties of the models. As with PEPA work described in the previous section, this is the default approach used by the EPA framework, however ODE evaluation is possible at the domain expert's preference.

2.2.3 *PEPA or Bio-PEPA*

During the transition from GA parameter matching work to GP structure optimisation techniques, the choice between PEPA and Bio-PEPA was a decision to be finalised before commencing with GP experiments. PEPA is a more succinct language, thus is capable of expressing complex behaviour with relatively few syntactic building blocks.

The drawback is this smaller number of available building blocks can lead to models with complicated structures. Consider the need for PEPA to contain mirrored components to treat as recruitment pools for individuals, as detailed in Section 2.2.1. Such structures are time consuming to achieve using an evolutionary approach.

Bio-PEPA has a more complex grammar than PEPA, giving it a larger number of semantic building blocks. Additionally, a measure of the model complexity is moved into the semantics of the kinetic laws. Consequently, models described in Bio-PEPA tend to be more elegant and compact, although the components themselves are more complex. While the elegance of a Process Algebra model is a modeller's concern, the increased number of higher level operators allows the Evolutionary Computation techniques to skip evolving such constructs from scratch.

A model describing crowd dynamics in PEPA and Bio-PEPA syntax was presented by Scott [64]. The nature of mirror components and counting mechanisms in the PEPA syntax were complex and unwieldy. The Bio-PEPA version of this model is far smaller and less complex while still accurately describing the system. The G-protein system provides an example used by the EPA framework; Section 5.5 describes the system in PEPA and Section 6.4 describes the same system more succinctly in Bio-PEPA.

When selecting a Process Algebra for application of GP optimisation, Bio-PEPA became the preferred choice. This is because Bio-PEPA has the larger bank of operators and does not require mirrored components which generally means a smaller model size.

The species identifier being the same as the species being defined makes converting Bio-PEPA into GP tree structures simpler, see Section 4.1.1.2.

$$
\begin{aligned}
I_{\ PEPA} &\stackrel{def}{=} (recover, rRecover).R \\
R_{\ PEPA} &\stackrel{def}{=} (idle, \perp).R \\
I_{\ Bio\cdot PEPA} &\stackrel{def}{=} recover \ll I \\
R_{\ Bio\cdot PEPA} &\stackrel{def}{=} recover \gg R
\end{aligned}
$$

Notice the state change in the definition of I using PEPA notation; when the *recover* action is taken, I transitions to R. In Bio-PEPA notation, this is viewed as a reduction in the level of I and an increase in the level of R. Bio-PEPA syntax guarantees the species identifier (the final term in a definition) is the same as the species identifier (the first term).

Additionally, the simulation time of Bio-PEPA models is significantly shorter than equivalent models written in PEPA. Average simulation time is measured in minutes for PEPA models

but in seconds for Bio-PEPA. This is because the PEPA simulator uses a semi-complex apparent rate calculation which increases the simulation time due to how PEPA selects which action to perform. This is especially true as the number of choices and complexity of a model increases: this is *organised* complexity (Section 2.1). Model specific complexity is discussed with the presentation of complex systems utilised by the EPA framework with the experimental results, see Chapters 5, 6, 7 and 8.

PEPA was used exclusively for Genetic Algorithm parameter matching functionality. Bio-PEPA became the focus as the EPA framework was extended to incorporate Genetic Programming optimisation techniques. The EPA framework supports GA parameter matching for PEPA models. Additionally, the framework is capable of GA parameter matching and GP direct model optimisation for Bio-PEPA: this can be done simultaneously or independently.

2.3 EVOLUTIONARY COMPUTATION

Evolutionary Algorithms (EAs) are a subset of optimisation techniques. EAs are a set of generic metaheuristic optimisation techniques capable of searching a vast search space for an optimum solution. Study in this area started as early as the 1950s with the work of Nils Aall Barricelli [7], though his publication was not widely noticed. In the late 1950s Alex Fraser published work regarding *artificial selection* [34].

The popularity of these computing techniques increased in the 1960s; developments during that time are still present in modern EC techniques. Important works include Fraser and Burnell [35] for their explanations of general genetic theory and algebraic evolutionary methods; Crosby [26] created his own language (*Mendol*) which he uses to explain the methodology and programming of genetic simulations; and Bremermann [17] who studies the mathematics of biological optimisation. Additional pioneers in the field have had their early papers reprinted by Fogel [32].

Artificial selection is the process of artificially selecting desirable traits to solve a given problem; this is inspired by *natural selection*. Charles Darwin presented the concept of natural selection in his signature text *The Origin of Species* [27]. Natural selection is the fundamental mechanism of evolution which allows nature to fill narrow biological niches. Evolutionary Computation optimisation techniques seek to emulate natural selection and the processes of biological evolution with the goal of evolving strong and robust solutions: where a solution will solve a specified problem.

2.3.1 *Evolutionary Computation Concepts*

In essence evolutionary techniques are guided trial-and-error approaches. The initial generation is populated with random solutions, though some prior knowledge can be seeded

into the first generation. These solutions are evaluated against a desired behaviour; high scoring individuals are more likely to pass their attributes onto future solutions whereas poorer individuals die out.

2.3.1.1 *Fitness*

Selection occurs in choosing which individuals will breed and pass on their traits to the next generation. It is not necessarily the strongest or smartest individual which is selected. Instead, by choosing the best suited and most adapted individuals the species as a whole will prosper. The measure of how suited an individual is to its environment (or solution is to its problem) is its *fitness*.

As a simplified example, consider Figure 2.5, a pack of wolves living in a snowy environment. Hunting will be easier with natural camouflage: i.e. wolves with darker fur must expend more effort to survive because prey will notice them more easily. This additional expenditure of time and energy will lead to less food, poorer health and less chance at breeding. Therefore, over the generations, dark-furred wolves will die out. Thus, the better suited light-furred wolves will be selected for reproduction and pass on their favourable attributes, i.e. their light-furred pelts.

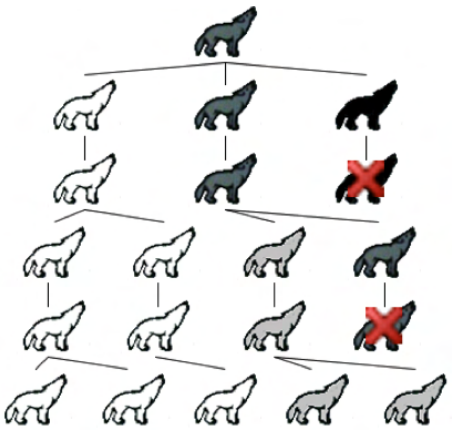


Figure 2.5: Selection scenario in wolves of different fur colours

Each candidate solution in a generation will be scored by some predefined fitness function. In Evolutionary Computation (EC), by convention the fitness score is generally minimising: the score spans zero (best) and positive infinity (worst). The default fitness function for this research scores deviation from a target using Euclidean distance; description of this measure and reasons for its use are detailed in Section 4.3.4.

2.3.1.2 *Selection*

Once a fitness has been assigned, selection criteria must be chosen. Selection techniques can be *exploitative* or *explorative*. An exploitation-heavy approach will optimise rapidly but is

prone to premature convergence: finding and becoming stuck in a local optimum. In contrast, exploration-heavy techniques are near random: jumping around the search space with little guidance. A balance of exploitation and exploration must be maintained for the most fit solutions to be located.

Two popular selection techniques are *Roulette-Wheel* selection and *Tournament* selection.

Roulette-Wheel selection normalises the fitness scores of all candidate solutions in a population to a total of 1; fitter individuals will occupy a larger portion of the hypothetical wheel. Selection for breeding is done by generating a random number between 0 and 1 and selecting the corresponding individual from the wheel. This technique favours the fittest individuals for selection. Roulette-Wheel is more exploitative so premature convergence at a local optimum must be avoided.

Tournament selection arbitrarily selects a group of candidate solutions, this is called the tournament pool. The most fit individual of a pool is selected for breeding. The size of the tournament pool influences the balance of exploitation and exploration. A larger pool encompasses more of the population, therefore the most fit individual of the generation is more likely to be found, leading towards exploitation. Conversely, a smaller pool will allow weaker individuals to be chosen, leading towards exploration. A benefit of tournament selection is the ease of adjusting this balance just by altering the pool size.

Another factor influencing selection is *elitism*. A specified number of highest fitness individuals, *elites*, automatically bypass the selection process. They gain a free entry into the following solution set. This ensures the best solutions cannot be lost to overly explorative selection criteria. Elitism can cause a negative impact on optimisation if too many elites are permitted into subsequent generations; the elites would dominate the population and cause a premature convergence.

2.3.1.3 *Replacement*

After selection, breeding occurs to create new candidate solutions for the next generation. Newly created offspring individuals can be added to parent solutions, replace them, or some combination.

The technique chosen also affects the balance of exploration versus exploitation; keeping more of the previous generation leans towards exploration by allowing more time to investigate poorer solutions, whereas more replacement tends towards exploitation.

The *steady-state* replacement technique is one form of replacement. This technique breeds a small number of offspring individuals: this can be as low as one. The same number of individuals (plus any elites) are then removed from the existing population and are discarded. This removal can be done arbitrarily, or by a reverse-selection technique; i.e., the poorest fitness individuals are most likely to be selected for removal. The new individuals are then entered into the population to replace those removed. This approaches preserves the majority of the

parent population each generation, increasing explorative influences. Steady-state can be more stable, but also slow to change with new individuals being suppressed by the population of high scoring individuals.

The *generational* replacement technique is another form of replacement. This technique breeds an entirely new generation of individuals from selected parents. Once breeding is complete, the parent generation is discarded entirely. The new population comprises entirely of elites and offspring individuals. This approach is highly exploitative because the whole generation is built upon favourable traits preserved during selection and breeding. As such, this requires balancing with explorative influences

2.3.2 *Evolutionary Computation Representations*

Under the heading of EC there are many implementations: including Ant Colony Optimisation (ACO) and Particle Swarm Optimisation (PSO), two examples of Swarm Intelligence (SI), as well as Estimation of Distribution Algorithms (EDAs). Examples of each will be presented through the following subsections.

It should be noted that there will be similarities between all Evolutionary Computation techniques. The variations will appear in the representation of individuals and implementation of the algorithms.

2.3.2.1 *Ant Colony Optimisation*

Ant Colony Optimisation solves optimisation problems by modelling ants and their pheromone trails [28]. In nature, ants explore the area surrounding their nest for food. Initially, their paths will be random. Once a food supply is located, the ant returns to the nest and produces a pheromone trail. As ants journey to that food supply, the pheromone trail becomes stronger. Unused trails will evaporate quickly. Larger collections of food will have more ants visiting it, which in turn leads to an even stronger pheromone trail. Smaller food supplies will have weaker pheromone trails.

1. Build population p of 'ant trails' (an ACO candidate solution)
2. **do**
3. **for**: Each ant trail at_i in p
4. **if** *Best* = *null* or Fitness(at_i) > Fitness(*Best*)
5. *Best* = at_i
6. Update p with strong traits based on high scoring trails
7. **while** *Best* < ideal and time != expired and generations < max
8. **return** *Best*

Figure 2.6: Pseudo-code representation of Ant Colony Optimisation

The pseudo-code for the ACO algorithm is presented in Figure 2.6. If we consider the pheromone trails to be equivalent to optima, the largest (strongest) will be the global optimum.

The implementation contains no concept of the ants themselves, but focuses on the pheromone trails instead. Ant Colony Optimisation attempts to identify the strongest pheromone trail (*ant trail*).

Originally, Ant Colony Optimisation was called the Ant System and was designed to solve the travelling salesman problem. A major advantage of ACO over other Evolutionary Computation techniques is its ability to dynamically adjust at runtime to a changing search space.

2.3.2.2 *Particle Swarm Optimisation*

Particle Swarm Optimisation solves optimisation problems by modelling the flocking and swarming behaviour of fish and birds [45]. Individuals, called *particles*, traverse a search space and move based on the fitness gradient around their position in attempts to locate the global optimum.

```
1.  Build swarm s of 'particles' (a PSO population and candidate solution)
2.  do
3.      for: Each particle p_i in s
4.          if Best = null or Fitness(p_i) > Fitness(Best)
5.              Best = p_i
6.      for: Each particle p_i
7.          Adjust velocity v_i (see description)
8.  while Best < ideal and time != expired and generations < max
9.  return Best
```

Figure 2.7: Pseudo-code representation of Particle Swarm Optimisation

Figure 2.7 illustrates the pseudo-code for Particle Swarm Optimisation. Unlike other optimisation techniques, PSO uses a static population and updates the swarm for each optimisation iteration without any form of selection or replacement.

The adjusted velocity (Figure 2.7, Line 7) of a particle is based upon a portion of: the particle's previous velocity, the particle's individual global best and the overall population's best. Additionally, a jump size variable allows particles to traverse areas of poor fitness to approach new areas of the search space, thus helping to avoid premature convergence or stagnation. An advantage of Particle Swarm Optimisation is the ability to optimise on particularly irregular or noisy search spaces.

2.3.2.3 *Estimation of Distribution Algorithm*

Estimation of Distribution Algorithms (EDAs) are a form of EC designed to break away from the traditional one-parent (asexual) or two-parent (sexual) reproduction systems found in nature. As inspired by nature, many EC techniques perform selection twice for each breeding occurrence: this mimics two parent breeding in nature. The biological restriction of one- or two-parent reproduction need not apply to Evolutionary Algorithms, this multi-parent avenue

has led to Estimation of Distribution Algorithms (EDAs) which sample many parents to form offspring [49].

```
1.  Build population pop
2.  do
3.      for: Each individual ind_i in pop
4.          if Best = null or Fitness(ind_i) > Fitness(Best)
5.              Best = ind_i
6.      for: pop_size times
7.          for: num_parents
8.              Parent p_i = Selection(pop)
9.          Offspring o = BreedViaDistribution(selectedParents)
10.     Update pop with new offspring individuals
11. while Best < ideal and time != expired and generations < max
12. return Best
```

Figure 2.8: Pseudo-code representation of Estimation of Distribution Algorithms

The pseudo-code for EDAs is illustrated in Figure 2.8. Unlike traditional Genetic Algorithm techniques (discussed next), three or more parents are selected for breeding. Similar to tournament selection (Section 2.3.1.2), the number of parents influences the balance of exploration and exploitation. At each position in the solution structure, that same point is examined on all parents and a distribution is formed. Consider a binary example, with four parents:

P_1	0	1	1	1	0	1	0	0
P_2	1	0	1	0	0	1	0	1
P_3	0	1	1	0	0	0	1	1
P_4	0	0	1	1	0	1	1	1
O	0.25	0.5	1	0.5	0	0.75	0.5	0.75

The offspring individual is made up of a probability set based on the distribution of the selected parents. These probabilities will become a variable set appropriate to the problem; in this example, a set of binary values.

By incorporating desirable traits from multiple parents, the resultant offspring aims to be fitter than in one- or two-parent algorithms. Interest in applying multi-parent techniques to other EA techniques has been growing [1].

2.3.2.4 *Genetic Algorithms & Genetic Programming*

Further EA implementations include Genetic Algorithms (GAs) and Genetic Programming (GP): the two EC techniques utilised for this research. The research focuses mostly on the combination of the two domains as well as easy reproducibility, therefore more complex Evolutionary Algorithm techniques are not utilised. The pseudo-code for both GA and GP optimisation techniques is illustrated in Figure 2.9. This example is more detailed as it is implemented in the EPA framework.

As noted, Genetic Algorithm and Genetic Programming optimisation techniques utilise the same pseudo-code for optimisation. The differences are in the representations of the solution

```
1.  Population pop = null
2.  for: pop_size times
3.      pop = pop + new individual
4.  Best = null
5.  do
6.      for: Each individual ind_i in pop
7.          if Best = null or Fitness(ind_i) > Fitness(Best)
8.              Best = ind_i
9.      Population tmp = null
10.     for: pop_size times
11.         Parent p_1 = Selection(pop)
12.         Parent p_2 = Selection(pop)
13.         Offspring o = Breed(p_1, p_2)
14.         tmp = tmp + o
15.     pop = tmp
16. while Best < ideal and time != expired and generations < max
17. return Best
```

Figure 2.9: Detailed pseudo-code representation of GA and GP optimisation techniques

individual. Each will be discussed in finer detail in Sections 2.3.4 and 2.3.5 respectively. First, Section 2.3.3 details how variation is added into the candidate solutions with each generation (Figure 2.9, Line 13).

2.3.3 *Variation*

Stagnation in the course of optimisation would occur quickly if selected individuals (Figure 2.9, Lines 11 & 12) for breeding were cloned (i.e. copied unchanged) into the offspring generation (Figure 2.9, Line 14). Selected parents go through a breeding process to introduce variation into the created offspring. In Genetic Algorithms and Genetic Programming optimisation techniques, breeding is represented through the use of recombination (or crossover) and mutation operators.

The recombination operator, commonly called crossover, mixes the solution structure of two selected parents. This attempts to combine the fit attributes of two selected parents into an even fitter offspring.

The mutation operator has a chance of resetting the value at a random location on the solution structure. A higher mutation rate reduces the chance of becoming caught in a local optimum, thus reducing the chance of premature convergence or stagnation. However, if the rate becomes too high the EA technique will have difficulty locating an optimised solution because solutions will bounce near-randomly around the search space.

As previously mentioned, the key differences between Evolutionary Computation techniques are in the representation used. The representation is both EC specific and problem specific, meaning a solution structure cannot be transplanted from one problem domain into

an alternative problem domain. How variation is applied to GA and GP representations is discussed separately over the following sections.

2.3.4 *Genetic Algorithms*

Genetic Algorithms (GAs) are one track which developed from Evolutionary Computation work, made popular in the 1970s by the work of John Holland [42]. As a biologically-themed approach, biological terms are used interchangeably with computational terms. For example, individual and candidate solution, population and a group of solutions, or generation and a search iteration.

Like other Evolutionary Computation techniques, GAs use methods inspired by natural selection to optimise solutions, these include selection, reproduction, recombination and mutation. The concept of reproduction to produce offspring solutions is achieved through recombination and mutation of the selected parent individuals. In-depth details can be found in Goldberg [38].

Genetic Algorithms take inspiration from nature for their structure. Chromosomes are strings of *DNA* which exists in all cells of all living organisms. A single chromosome consists of blocks of DNA, otherwise known as *genes*. These genes encode traits about the organism and any possible settings for these genes are called *alleles*. For example, the trait of hair colour would have alleles of blond, brown, black or red. The position of a gene on a chromosome is called its *locus*.

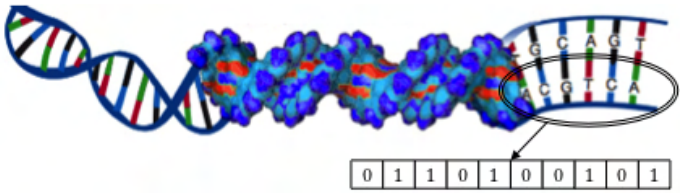


Figure 2.10: Translating a DNA strand to a binary string

The traditional representation of solutions in Genetic Algorithms is as a string of binary digits. This mimics the nucleic acid molecules: adenine, cytosine, guanine and thymine (A, C, G and T), which sprout from the unfurled DNA strand, as seen in Figure 2.10. While a binary string representation is traditional, many others now exist. For simplicity, the remainder of this section will use binary string representation examples.

With the linear structure of GA solutions, the implementation of crossover operators is simplified. There are three forms of crossover operator; *one-point*, *two-point* and *uniform* crossover. One-point crossover is presented to the left of Figure 2.11. The chromosome structure is split at an arbitrary point; in this case, between locus 5 and 6. The offspring solutions comprise of the left section of one parent and the right of the other. One-point

crossover allows links between alleles of neighbouring loci to be preserved while adding variation.

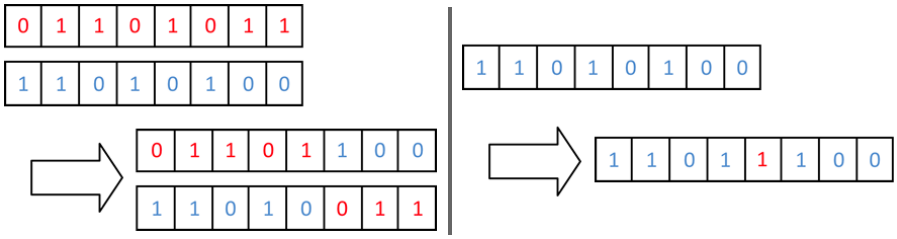


Figure 2.11: Crossover (left) and Mutation (right) operators on binary string solutions

Two-point crossover operates similarly, except two arbitrary points are selected along the chromosome length. The offspring solution comprises of one parent between those two points with the remainder on each end from the other parent. Two-point crossover maintains links between alleles of neighbouring loci, particularly those representative of circular DNA, such as *E. coli*. Uniform crossover moves along the length of the chromosome; for each locus, the offspring's locus is randomly chosen from either of its parents. This effectively randomises each locus and breaks any dependencies that may exist; as such, this is more explorative than the other crossover operators.

The mutation operator has a chance of resetting the value at zero or more random locations on the chromosome. To the right of Figure 2.11, the fifth locus has been selected and the value is reset from 0 to 1.

With GAs, the crossover operator is heavily exploitative. Conversely, the mutator operator is explorative as changes can move the solution into new areas of the search space.

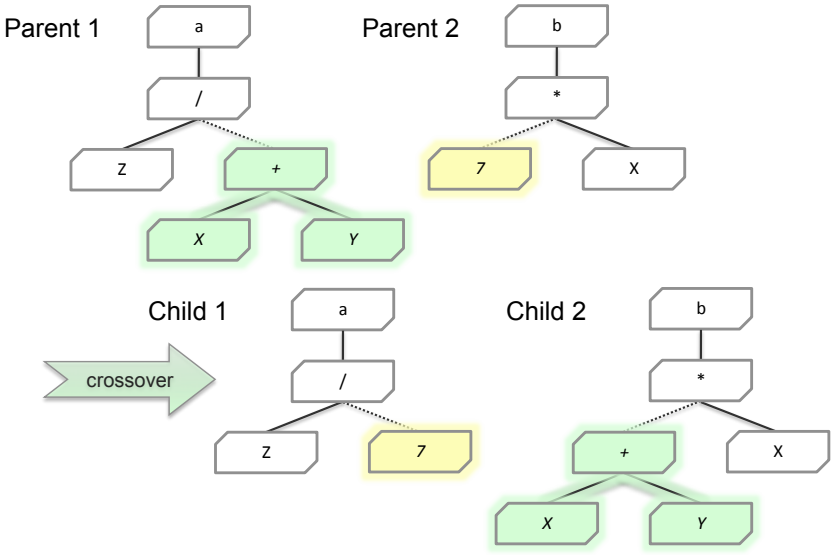
2.3.5 *Genetic Programming*

Another track which developed from Evolutionary Computation work was Genetic Programming (GP). In 1964, *On the Organization of Intellect* by Fogel [33] became the basis for further work in the GP field. Later, in 1985, the first description of the modern tree-based GP techniques was given by Cramer [25]. This field was expanded greatly by the work of Koza, a leading pioneer in Genetic Programming development. A list of Koza's work is available at [48].

Similar to GAs, and other evolutionary approaches, GPs use techniques inspired by natural selection and use biological terminology. In-depth details can be found in Koza [47] and subsequent texts.

The GP crossover operator is not multimodal (i.e. one-point, two-point or uniform) in the same way as the GA crossover operator. The GP crossover operator functions by choosing a connecting edge between two nodes on both parent individuals; due to the large variety of tree dynamics, it is unlikely but permissible for the same edge on both parents to be

chosen. Figure 2.12 illustrates the GP crossover operator on two arbitrary definition trees. Once selected, all nodes below the connecting edge (referred to as the subtree) are detached and swapped, thus resulting in two new offspring individuals.

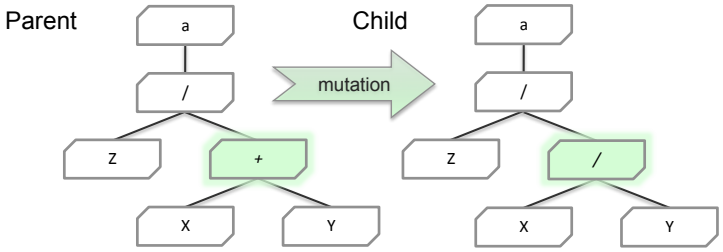


$$a \stackrel{def}{=} Z \,/\, (X+Y) \quad \Rightarrow \quad a \stackrel{def}{=} Z \,/\, 7$$
$$b \stackrel{def}{=} 7 * X \quad \Rightarrow \quad b \stackrel{def}{=} (X+Y) * X$$

Figure 2.12: Crossover operator on a generic GP tree

The standard GP mutation operator has two functions available to it: the *point* mutation and the *subtree* mutation. These two modes can be applied at different ratios to change the balance of exploration and exploitation, as discussed in Section 2.3.1.2.

The point mutation operator is illustrated in Figure 2.13. One node is stochastically chosen and that node has its contents reset to any permissible value; numbers will still be numbers, etc. To maintain the tree dynamics (i.e. tree depth, width, structure), the resultant node will have the same number of child nodes, no other part of the GP tree is altered. The point mutator is heavily exploitative of the parent structure.

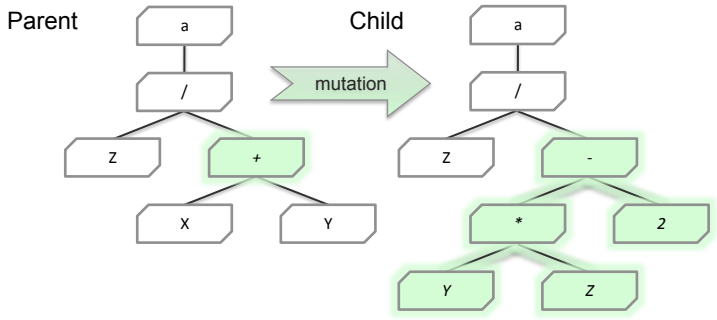


$$a \stackrel{def}{=} Z \,/\, (X+Y) \quad \Rightarrow \quad a \stackrel{def}{=} Z \,/\, (X \,/\, Y)$$

Figure 2.13: Mutation (point) operator on a generic GP tree

The GP subtree mutation operator can cause a more drastic change to the parent structure than the point mutation operator. Figure 2.14 illustrates the subtree mutator with the same

initial parent and selected node as the point mutator above. The key difference is what is mutated. Any existing child nodes (and their children) of the selected node are entirely removed. The chosen node is then rebuilt as a new subtree which can be a terminal or include child nodes, child-of-child nodes, etc..



$$a \stackrel{def}{=} Z \,/\, (X+Y) \quad \Rightarrow \quad a \stackrel{def}{=} Z \,/\, ((Y*Z)-2)$$

Figure 2.14: Mutation (subtree) operator on a generic GP tree

The point mutation operator can guarantee the tree size and structure will not change, whereas the subtree mutation has a low probability of retaining the original tree dynamics. Losing the original tree dynamics is not necessarily bad; it allows new areas of the search space to be explored. The subtree mutator is more exploitative than the crossover operator, while being more explorative than the point mutation operator.

When using Genetic Programming optimisation techniques, the roles of the crossover and mutator operators are reversed. The crossover operator is the more destructive influence on breeding due to the large changes that can be realised by swapping large subtrees. GP crossover operators are most successful when they are heavily constrained so they will crossover like with like definitions. This is biologically pertinent; for instance, the genetic material of a heart will not be recombined with material of an ear since the result would likely be biologically invalid.

In contrast, the mutator operator allows for more subtle changes to the structure of the GP definition trees. This is less true for the subtree mutator operator which can have similar effects to a crossover from a non-existent parent, depending on the depth at which the operator is applied.

2.4 STATE OF THE ART SUMMARY

This chapter has presented the background and concepts of the two major domains bridged by the EPA framework: Evolutionary Computation (EC) techniques and Process Algebra (PA) Formal Modelling (FM) languages. The choice of moving from PEPA to Bio-PEPA was made due to the concise nature of the Bio-PEPA syntax and the higher level syntactic building blocks

available to it. After this discussion, the EC techniques which would be applied to those Process Algebras were detailed.

The concepts of EC optimisation have been discussed with some pseudo-code examples of Ant Colony Optimisation (ACO), Particle Swarm Optimisation (PSO) and Estimation of Distribution Algorithms (EDAs) with detailed pseudo-code for Genetic Algorithm (GA) and Genetic Programming (GP) optimisation techniques. The generic concepts for these have been detailed in this chapter. The Evolving Process Algebra (EPA) framework is discussed in Chapter 4 with full implementation specifics of these generic concepts, along with exact variable values and accompanying justifications.

3

RELATED WORK

This work proposes that Evolutionary Computation (EC) techniques and Process Algebras (PAs) have complementary strengths for the modelling of complex bioscience systems. Other researchers and groups are performing related work which will be discussed in this chapter. The discussion is broken down into sections based on what is being optimised: the parameters or the model structure itself. This chapter concludes with a discussion of neighbouring fields.

3.1 PARAMETER OPTIMISATION

This research uses Genetic Algorithms (GAs) to navigate a search space to find fitter parameter values for Process Algebra models. The research done in this area is not novel, it acted as a stepping stone towards the overarching goal of direct PA model optimisation. The individual work of Gesweiller [36] and Prandi [57] use other optimisation techniques to optimise parameter fitting for Process Algebra models.

Geisweiller has developed *EM-PEPA* which finds rate values inside a PEPA model through the application of two Expectation-Maximisation (EM) algorithms [36]. This is an iterative process which attempts to fit a set of parameter values to expected observations to locate a local maxima in the search space. This is akin to the hill-climbing exploitative half of a GA technique with minimal exploration influence.

Prandi utilises Particle Swarm Optimisation (PSO) instead of GA optimisation techniques. PSO is an Evolutionary Algorithm optimisation technique which works by simulating the flocking behaviour in birds and fish as detailed previously in Section 2.3.2. Prandi implemented his work inside the *R* system with *BlenX* (a stochastic Process Algebra) to optimise parameter fitting [57]. This was demonstrated using a system of an idealised ecological food web example and matching to a synthetic oscillatory behaviour. The system was able to provide sets of parameter values to match the original model.

The Systems Biology Workbench (SBW) was developed in attempts to unify the increasing number of different representations available for complex bioscience systems [62]. The workbench allowed for assistance in modelling, analysis, and data manipulation of bioscience systems. The primary strength of the SBW is the component view of different compatible packages; this allows for users to employ the strengths of multiple modelling techniques simultaneously. Furthermore, the SBW introduced the Systems Biology Markup Language (SBML) as a new standard for systems biology representation.

Specific to the parameter fitting of Bio-PEPA models is the Systems Biology Software Infrastructure (SBSI) developed by *SynthSys* at the University of Edinburgh [65]. The SBSI tool accepts models written in SBML, as mentioned above. Bio-PEPA models can be automatically translated into SBML using the plug-in tool, see Chapter 4. Once in SBML notation, a model can be translated back into Bio-PEPA form via a prototype translator developed by Loewe, Moodie & Hillston [52]. This is achieved by first translating the SBML into Systems Biology Graphical Notation (SBGN) and then into Bio-PEPA notation.

The SBSI tool can perform simultaneous Genetic Algorithm calculations to find fitter parameter values. This is equivalent to the EPA parameter matching work performed on PEPA models.

3.2 MODEL STRUCTURE OPTIMISATION

Ross and Imada have laid down the foundations for this work by demonstrating the feasibility of applying Evolutionary Computation techniques to Process Algebra models. They apply an EA to a subset of stochastic π-calculus to evolve their models [61]. The focus of their work is on determining the best way to measure fitness for candidate solutions through a suite of statistical tests. These tests include raw means, raw standard deviations, distribution skew, serial correlation, chaos and periodicity.

Their approach has been applied to a genetic circuit, a chemical reaction model and a version of the repressilator (used with the EPA framework, see Section 5.4). Their experiments evolved both the parameters and the model itself.

A major difference between their work and this research is the focus. Their work concentrates on the statistical suite and manually determining which test(s) are most appropriate for scoring fitness of solutions. This research uses simpler tests so the focus can remain on the optimisation process itself. Though the focuses are different, the two works may be combined cooperatively for a stronger optimisation approach.

Their research and this work have had success with simple models. Future work for both their and this research is in handling stochastic noise and scaling up the complexity of the models that can be successfully evolved. This interdisciplinary field of research is in its infancy. Work to this point is far from exhaustive and still holds potential.

3.3 NEIGHBOURING FIELDS

This section discusses optimisation techniques which are closely neighbouring to the application of Evolutionary Algorithm techniques to Process Algebra models as used in this research. Three specific fields have been highlighted, including *simulated annealing* as an approach similar to this work and *inference* as a non-empirical alternative approach. Discussion commences

with the growing field of utilising Genetic Programming techniques on Ordinary Differential Equation (ODE) structures directly.

3.3.1 *Applying GP to ODEs*

An Ordinary Differential Equation is a form of equation commonly found in the fields of mathematics and science; ODEs are comprised of an independent variable and any derivatives of that variable. Representing and solving ODEs is an intricate process. ODEs are solved by hand or, increasingly, by computer and provide approximate solutions which are equivalent to large numbers of averaged simulations at a population level.

The Bio-PEPA plug-in allows for Bio-PEPA models to be translated and interpreted as Ordinary Differential Equations. However, the EPA framework is incapable of optimising the structure of ODEs directly. This could be a logical extension of the EPA framework (discussed in Section 9.3), but is beyond the scope of this work. Other research groups are making progress in this area, discussed below.

In 1999, Cao et al. used a combination of Genetic Algorithm and Genetic Programming optimisation techniques to construct ODE models [18]. The GP optimisation was utilised for the structure and the GA techniques for the parameters. The ODEs created may include arithmetic operators in addition to trigonometric functions. Furthermore, the ODE definitions could potentially include Higher-order Ordinary Differential Equations (HODEs). Most ODEs describe change over time so that the values of t are dependent on the values at t-1. HODEs go further and may describe change over time to include t-1, t-2 to t-n. Cao et al. score the fitness of each candidate via a normalised difference between the observed and target solutions; this is akin to the Euclidean distance measure as described in Section 4.3.4.

Tun's thesis focused on this field of work; ODEs are optimised by using a combination of Evolutionary Computation techniques [67]. A Genetic Programming technique was used for the structure of the Ordinary Differential Equation. Various EA methods were utilised for the parameter estimation; a small portion is GA optimisation with the majority being Differential Evolution (DE): an abstract form of Evolutionary Algorithm similar to GAs. Tun measures fitness of candidate solutions by a normalised difference function, similar to Cao et al. and the Euclidean distance measure of this research. In addition, the fitness can be adjusted based on the complexity of the constructed ODE. Finally, the ODE model produced is analysed with a statistical analysis tool to check the performance of the EC; for example, gene diversity & distribution and average model size.

As a final example of applying GP techniques to generate ODEs; Iba & Sakamoto did work which uses a combination of Genetic Programming techniques and *least squares* means to produce ODE models [43]. Least square means works by attempting to minimise a squared difference between an observed value and its target, commonly used in data fitting problems.

For simplicity, the ODEs constructed in Iba & Sakamoto's work omits trigonometric functions. They offer a quote which highlights the nature of work involving GP techniques: "*Our aim is to obtain the candidates scattered in the huge search space and to propose to users the possible causal relationship among the observable components.*" This field of work and neighbouring fields can offer multiple strong candidate solutions to a given problem. In turn, a singular best solution is unlikely to be found, but valuable insight into the nature of the described system can be gained and returned to the user.

3.3.2 *Simulated Annealing*

Annealing is a metallurgical heat treatment performed on metals to increase its ductility: the ability for a metal to deform under high stress thus becoming more workable. This is done by heating the metal above its critical temperature; this is metal specific but is generally to the point of glowing red or even white. Once heated, the metal is allowed to cool gradually in resting air where bonds broken during the heating stage are allowed to recrystallise, resulting in a higher quality metal.

Simulated annealing is a generic probabilistic metaheuristic which optimises by imitating the metallurgical annealing process. Simulated annealing is commonly used in discrete search spaces. Similar to Evolutionary Computation techniques, simulated annealing can offer a strong solution more efficiently than an exhaustive search but may not find the singular best solution in a noisy search space.

```
1.  Temperature t = some initially high value
2.  Best = Solution s = some randomly chosen candidate solution
3.  do
4.          Solution v = Vary(s)
5.          if Quality(v) > Quality(s) or Random(0, 1) < P(t, v, s)
6.                  s = v
7.          Decrement t
8.          if Quality(s) > Quality(Best)
9.                  Best = s
10. while Best < ideal and time != expired and generations < max
11. return Best
```

Figure 3.1: Pseudo-code representation of Simulated Annealing

Figure 3.1 illustrates the pseudo-code for the simulated annealing algorithm [54]. The implementation includes a *temperature* variable which influences the behaviour of the optimisation process. This is controlled by the $P(t, v, s)$ term (Figure 3.1, Line 5) which is defined as:.

$$P(t, v, s) = e^{\frac{\text{Quality}(v) - \text{Quality}(s)}{t}}$$

where e is a mathematical constant (sometimes referred to as *Euler's number*). For clarity, e is raised to the power. This equation approaches 1 in two ways: if Quality(v) is close to

Quality(s), secondly if t is a high value. The higher this temperature, the more likely a poorer fitness solution will be accepted as an alternative to the current solution. Therefore, as the temperature cools, the strength of the current solution will tend towards a strong solution.

From an Evolutionary Algorithm perspective, the simulated annealing algorithm utilises a steady state replacement technique (Section 2.3.1.3) with a population size of one. The parent solution may be replaced by its child with a probability based on its comparative fitness and the current temperature.

While the temperature is high, the offspring solution is more likely to replace the parent, resulting in a high degree of exploration. As the temperature cools, the child must be a stronger solution than the parent for this replacement which is akin to an exploitative hill-climbing approach. The heating and cooling process can be performed iteratively in an attempt to find a stronger solution.

3.3.3 *Bayesian Inference*

This research and the previous sections of related and neighbouring fields have discussed empirical approaches for optimisation. Therefore, any solution discovered and returned by a discussed optimisation technique has been observed and is known to be highly fit. *Inference* is an alternative approach which draws logical conclusions based on premises assumed or known to be true. In statistics, inference can be used to draw conclusions from datasets afflicted by variation: such as noisy data.

The English mathematician Thomas Bayes (1701 - 1761) posited a solution to an inverse probability problem in an essay which was read to the Royal Society in 1763 after Bayes' death. Due to the success in solving probability problems in the early eighteenth century, this became known as *Bayes' theorem* (alternatively *Bayes' rule* or *Bayes' law*). Bayes' theorem in the theory of probability has been likened to Pythagoras' theorem in geometry due to their impacts in their respective fields.

In mathematics, Bayes' theorem explains the relationship between probabilities. Given statements A and B and their probabilities of being true P(A) and P(B), the conditional probabilities of A given B and vice versa are P$(A \mid B)$ and P$(B \mid A)$. Using this notation, Bayes' theorem can be simply stated as:

$$\mathrm{P}(A \mid B) = \frac{\mathrm{P}(B \mid A)\ \ \mathrm{P}(A)}{\mathrm{P}(B)}$$

where observations A and B need not be singular entities. Frequently A and/or B are collections of observations; where each such observation will have its own associated probability of trueness.

In the context of parameter matching and structure optimisation of mathematical models, the Approximate Bayesian Computation (ABC) method (also known as likelihood-free method)

has been gaining popularity in the last decade [55]. As an example of inference related to this research, below is a representation of the repressilator system; this was chosen because the PEPA version is used with the EPA framework. The PEPA centric view and related Process Algebra model of the repressilator system are presented in Section 5.4. The inference version of the repressilator is taken from the work of Barnes et al. [6].

$$\left\{ \begin{array}{l} \frac{dx_i(t)}{dt} = -x_i(t) + \frac{\alpha}{1+y_{-i}^n(t)} + \alpha_0 \\ \frac{dy_i(t)}{dt} = -\beta(y_i(t) - x_i(t)) \end{array} \right\}$$

where there are three genes. Protein 1 (y_1) inhibits expression of gene 2 (x_2) which would express protein 2 (y_2), this in turn inhibits gene 3 (x_3) that would express protein 3 (y_3) which completes the cyclic behaviour by inhibiting gene 1 (x_1). The observations of known gene expression levels (x_i) and protein expression levels (y_i) are used to infer the values of alpha (α) and beta (β). This example only infers the parameter values; the model structure is not included.

One implementation of ABC is the *rejection algorithm*; inference proceeds by an iterative cycle of sampling points from the known observations to improve upon or match those observations. The known observations ($P(A)$) are sampled a user-specified number of times; each time a new set of observations is generated. The discrepancy between the known and generated observations is then measured, possibly by Euclidean distance (Section 4.3.4). At a user-specified tolerance, the discrepancy will cause the generated observations to be accepted to further improve upon in the next inference iteration.

The inference view of the repressilator system was included to highlight the differences in representation as compared to the PEPA view of Section 5.4. As a simpler example of Bayes' theorem, consider the following scenario.

An ornithologist is categorising a bird which may be one of two subspecies; one common and one rare. The rarer subspecies frequently has red feathers (90%) and the common subspecies infrequently has red feathers (7%); also assume the rarer subspecies takes up 1% of the bird species population. What is the likelihood of the bird being of the rarer subspecies with red feathers? Using the expanded version of Bayes' theorem:

$$\begin{aligned} P(A \mid B) &= \frac{P(B \mid A)\ P(A)}{P(B)} \\ P(rare \mid red) &= \frac{P(red \mid rare)\ P(rare)}{P(red \mid rare)\ P(rare) + P(red \mid common)\ P(common)} \\ &= \frac{0.9 * 0.01}{0.9 * 0.01 + 0.07 * 0.99} \\ &\approx 11.5\% \end{aligned}$$

By knowing the probabilities of the related facts being true, the probable value of the categorised bird being rare can be inferred. Given the scenario variables, there is approximately a 11.5% likelihood that the bird is of the rare subspecies and has red feathers.

All three authors of the Barnes et al. work have assisted in development of ABC-SysBio: a tool developed for parameter inference and model selection [50]. Using this tool, the parameters of the repressilator model were inferred. Barnes et al. noted the correlation of the results indicated the variables were difficult to infer; this is because statistical approaches assume independence in the inferred variables.

Due to the underlying nature of dynamic systems, only certain combinations of variables may be inferable. Consider an epidemiological system where individuals may enter the population through birth and then leave through death. If birth and death rates are inferred together, a range of acceptable values may be discovered because the variables are not independent. Discovering acceptable combinations of variables would be assisted by input from a domain expert.

3.4 NO FREE LUNCH

This chapter has detailed work related to the optimisation of rate values, the optimisation of model structures and also neighbouring fields such as ODE optimisation and inference. Beyond the illustrated examples, other optimisation techniques will exist further afield. In addition, continuing work will attempt to improve new and existing techniques. With constant improvement, it could be assumed that one technique will eventually master all optimisation problems.

The No Free Lunch (NFL) theorem was explored by Wolpert and Macready [69]. NFL posits that all optimisation algorithms that search a problem space have a cost for doing so. When averaged across all possible optimisation problems, all algorithms of a given cost perform equally well. For an algorithm to improve its performance to solve a particular problem, it must become less suited to solving problems in other areas; i.e. the cost must be paid from its performance to solve other problems.

The generic metaheuristic optimisation algorithms, such as GAs and GP techniques, illustrate this trade off. They are capable of finding strong solutions in large and complex search spaces with the trade off of not guaranteeing the global optimum solution will be located.

3.5 RELATED WORK SUMMARY

This chapter has presented the related work of the two major domains being bridged by the EPA framework: Evolutionary Computation (EC) techniques and Process Algebra (PA) Formal Modelling (FM) languages. Firstly, the parameter optimisation work which has been applied to PA models was discussed and the successes in this area were highlighted.

Next, the use of optimising PA model structures to develop fitter models was explored. This is a comparatively new field with work still in its infancy allowing for further potential to be unlocked.

The related work chapter concluded with discussion of neighbouring fields with specific examples including GP optimisation of Ordinary Differential Equations (ODEs). ODE optimisation is of growing interest due to the widespread use of ODEs in many domains; similar to the optimisation of Process Algebras this is relatively new.

Further neighbouring field examples included simulated annealing and Bayesian inference. Simulated annealing is a close neighbour to EC techniques and similarly attempts to find a strong solution if not the exact global optimum. Bayesian inference is a different track and utilises probability inference to solve optimisation problems.

4

EVOLVING PROCESS ALGEBRA FRAMEWORK

The Evolving Process Algebra (EPA) framework is the primary software application developed during this research. The EPA framework is designed to bridge the gap between Evolutionary Computation (EC) techniques and the Process Algebra (PA) models to which they will be applied.

On the Evolutionary Computation side, the framework is built on top of the Evolutionary Computation Java (ECJ) toolkit created by Luke and now maintained by Luke and a large team at George Mason University [53]. ECJ is a powerful and highly flexible research tool capable of running multiple types of Evolutionary Computation techniques. Written in Java, ECJ aims to be easily user-customisable whilst maintaining runtime efficiency for long and computationally expensive EC projects.

On the Process Algebra side, two plug-ins allow PEPA and Bio-PEPA to be controlled, manipulated and simulated within a Java environment. Both plug-ins were created at the University of Edinburgh [29, 21]. Further features include a graphical interface for the Eclipse IDE and error detection for their relevant Process Algebra.

The EPA framework has one particular bottleneck which slows its operation: the simulation of Process Algebra models, particularly PEPA models. To help alleviate this impedance and allow for multiple concurrent runs, the EPA framework is able to work with *HTCondor* (called *Condor* until 2012) [66]. HTCondor is a high throughput computing toolkit maintained by the University of Wisconsin-Madison and is designed to distribute experiments over multiple available nodes. In-house, experiments were conducted across approximately 200 *Linux* nodes, depending on availability.

Linux nodes are preferred for stability. A running EPA job may become suspended from an HTCondor node, most likely because a user requires the computer to which that node belongs. If a job becomes suspended during PA simulation on a non-*Unix* machine, there is a chance the EPA run will not be able to continue and will be aborted instead. This is due to how the HTCondor *Sigterm* (suspend) signal is handled on different architectures and its interaction with the Process Algebra plug-ins. Such an aborted job will still produce some output but is not likely to provide a solution as strong as a completed job.

A main driving goal behind the EPA framework is bridging the gap between domain experts and modelling. HTCondor is not aimed at this, so an assistant program was developed to ease the use of the framework across HTCondor nodes. The *EPA Utility Assistant*, presented in Appendix A, is designed for this purpose.

In this chapter, the structure of the EPA Individual (solution representation) is discussed next in Section 4.1, followed by the summary of default values used in the EPA framework in Section 4.2. Finally, in Section 4.3, the EPA lifecycle is described along with justification of the values used.

4.1 THE COMBINED INDIVIDUAL

The ECJ toolkit contains multiple core classes and interfaces which are utilised by the EPA framework. By extending known superclasses and implementing known interfaces, the framework can utilise the expansive predefined Evolutionary Computation library in the ECJ toolkit. The most important predefined construct is the *Individual*: the representation of a solution. Each Individual has an associated *species* which dictates the available representations, fitness functions and breeding operators. The ECJ *species* should not be confused with the Process Algebra *species* model segment.

The ECJ toolkit only allows for one type of Individual to exist in a population. However, at least two Individual types are required, one for the Genetic Algorithm (GA) representation of rate and population solutions, the other for the Genetic Programming (GP) tree-based representation of kinetic and species solutions. To overcome this restriction, the EPA framework utilises the *CombinedIndividual*, illustrated in Figure 4.1.

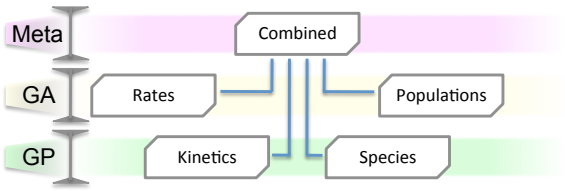


Figure 4.1: The EPA combined individual structure

The CombinedIndividual is a meta-Individual which contains four sub-Individuals; the two GA Individuals for rates and population solutions (middle row, highlighted in yellow), plus two GP Individuals for kinetics and species solutions (bottom row, highlighted in green). Representations for the GA and GP sub-Individuals are discussed in Section 4.1.1 below.

By utilising this meta-Individual approach, the EPA gains many advantages. Each sub-Individual can be handled independently, allowing for tailored Individual definitions, fitness functions and breeding operators. Each of these concepts are detailed later in this chapter with discussion of the EPA lifecycle in Section 4.3.

A further advantage of the meta-Individual representation is the ability to extract sub-Individuals and analyse them in isolation. This allows for sub-Individuals to be scored independently of the overall fitness of the CombinedIndividual. At its least involved, this could be used to give the domain-expert insight into the relative fitness of each of the evolved

Individuals. This data could also be used to weight certain sub-Individuals with a higher priority than others. Furthermore, complex selection or breeding operators could be developed to build offspring meta-Individuals from multiple sub-Individual parents should a problem require it. For easy reproducibility of experiments, such functionality has not been utilised during this work; it remains a logical next step for further EPA development.

4.1.1 *Representing Solutions*

The solution structure for an Evolutionary Computation technique is highly specific. It is specific to the EA itself and also the problem to which it is applied. This means the solution structure for one problem cannot be applied directly to a different problem. How these representations are varied during the breeding process is detailed later in Section 4.3.10 during the *Breed* step of the EPA lifecycle.

4.1.1.1 *Representing Genetic Algorithms*

The background descriptions of Genetic Algorithms (GAs) were detailed in Section 2.3.4. The traditional representation of solutions in Genetic Algorithms is as a string of binary digits. Figure 4.2 illustrates this traditional representation followed by the two specific implementations utilised by the EPA framework.

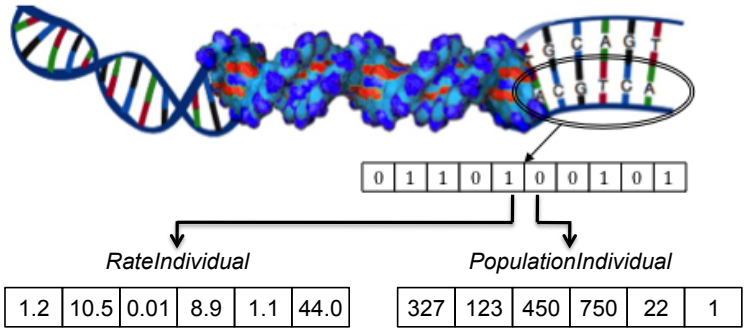


Figure 4.2: Translating a DNA strand to a binary string with example EPA individuals

The EPA framework uses a string of doubles to represent rate solutions and a string of integers to represent population solutions (lower left and lower right of Figure 4.2 respectively). In both representations, values are non-negative and can range from 0 (inclusive) to positive infinity (exclusive). This range can be restricted globally for an Individual or each locus may have its own specified range.

4.1.1.2 *Representing Genetic Programming*

Genetic Programming techniques utilise a tree based solution structure. The size and shape of the tree can vary hugely between different problem domains and within solutions to the same

problem. Due to the strict nature of the Bio-PEPA syntax, the GP syntax is different for *species* and *kinetics* definitions: Section 2.2.2 details the responsibilities of these PA segments and their definitions.

PEPA work concentrated exclusively on parameter matching problems and did not use any GP techniques, therefore there is no associated GP tree for the PEPA syntax.

The *species* portion of the Bio-PEPA model was the first section considered for optimisation using Genetic Programming techniques. The syntax is defined by Ciocchetta & Hillston [20], copied here for convenience. The syntax is displayed in ASCII as this is used by the EPA framework.

$$\begin{aligned} S &::= (\alpha, \kappa)\, \mathrm{op}\, S_{id} \mid S + S \\ \mathrm{op} &= \ll \mid \gg \mid (+) \mid (-) \mid (.) \end{aligned}$$

Figure 4.3 represents the tree structure used in this research to model species definitions. These definitions have a highly constrained syntax and will all have this structure. Note, solid connecting lines are required nodes and dashed lines are zero or more optional additional nodes.

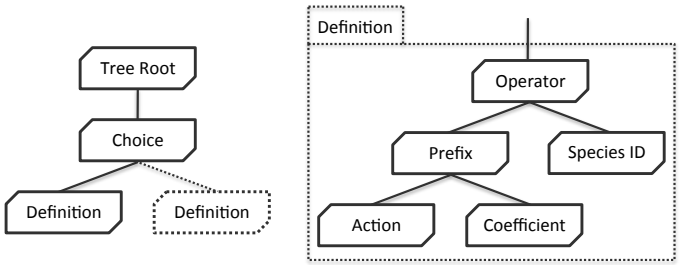


Figure 4.3: Bio-PEPA species syntax defined as a GP tree

The *Choice* and *Prefix* nodes always appear in the structure of the *species* GP tree, however their design includes a self-hiding mechanism when they are not required. In Section 2.2.2 the concept of syntactic shorthand was introduced which would hide the stoichiometry coefficient if it was equal to one; this causes the *Prefix* node to only display the contents of the *Action* node instead. Additionally, if there is no choice in a species definition (i.e., no +) the *Choice* node will be blank.

The *kinetics* definition was the second segment of the Bio-PEPA model to be evolved using Genetic Programming techniques. The syntax used by the EPA framework is more restrictive than the complete Bio-PEPA *kinetics* syntax. For the purposes of simplicity, trigonometric functions have not been included in the evolvable definitions: the work on kinetic evolution followed on from the species optimisation work, therefore it was applied to simpler systems, as discussed in Chapter 7. Similarly, only the function of mass action (*fMA*) has been included for simplicity.

The composite node (*c*) is an EPA addition to the Bio-PEPA *kinetics* syntax. These nodes were developed with utilising domain knowledge in mind and are specific to the epidemiological

domain. These nodes are built to model the density-based and frequency-based transmission contact rates in disease spread systems. By modelling these high level constructs, the EPA framework can be seeded with additional domain knowledge, see Section 4.3.1. The composite node is a form of library component and is only applicable to the domain where the knowledge is being utilised. Using these composite nodes comes with an associated cost, as described in Section 4.3.6. Additional problem-specific composite nodes could be developed and included for any domain.

Another important difference is the use of bytes instead of integer values. This limits the range of numbers from integer $-(2^{31})$ to $(2^{31}-1)$ down to byte $-(2^7)$ to (2^7-1); this is a practicality based choice and is designed to restrict the search space that the GP has to explore. Floating point values have also been omitted from being directly evolved. However, the combination of the *byte* and *division* nodes can form decimal values (i.e. $3/2 = 1.5$). Discrete floating point numbers are uncommon in kinetic functional rate definitions; this setup ensures their uncommon status while still making them possible. Note, this only influences the evolution of kinetic definitions: this does not affect user-specified kinetic definitions or *rate* variables in any way.

As above, the kinetics definition has been copied here for convenience.

$$
\begin{array}{lcl}
K & ::= & \mathtt{byte} \mid \mathtt{id} \mid \mathtt{c} \mid K\mathtt{m}K \mid K^K \mid \mathtt{f}MA(\mathtt{rate}) \\
\mathtt{id} & = & \mathtt{rate} \mid \mathtt{species} \\
\mathtt{m} & = & + \mid - \mid / \mid *
\end{array}
\qquad
\begin{array}{lcl}
\mathtt{c} & = & \beta SI \mid \beta SI/\mathtt{s} \mid \mathtt{s} \\
\beta & = & \mathtt{byte} \mid \mathtt{id} \mid \mathtt{c} \mid K\,\mathtt{m}\,K \mid K^K \\
\mathtt{s} & = & \sum \mathtt{species}
\end{array}
$$

Note, the *rate* and *species* are defined here as the identifier of any rate or species in the system. By default, *SI* in the epidemiological composite node *c* is defined as '$S * Inf$' but may be changed by the user to match the model specific identifiers for the susceptible and infected species in a system.

4.2 VARIABLE SUMMARY

The default set of variables utilised by the EPA framework are presented in Table 4.1. The values chosen as defaults attempt to balance the reliability of results and the computational power to achieve those results.

A brute-force systematic exploration of the search space would guarantee the optimum result(s), however this approach is intractable, especially as a problem's complexity increases. If computational resources allowed, variables such as population size, number of generations, number of internal or external replications and number of independent runs could be increased to improve the quality of results.

The concepts of *exploration* and *exploitation* were introduced previously in Section 2.3.1.2. In addition to balancing computational requirements with result quality, the default variables

attempt to strike a balance between exploration and exploitation. This balance may be maintained across multiple parameters; where one may lead to heavy exploitation, another may be very explorative.

EC Population Variables	
Population Size	100
Generations	200
EC Breeding Variables	
Selection type	Tournament
Tournament pool	5%
Elitism	1%
Crossover type	1-point
Crossover rate	100% / 90% $^{\Delta}$
Mutation rate	5% / 100% $^{\Delta}$
Mutation type ratio	1 : 1 $^{\Delta}$
Adaptive window size	5 $^{\Delta}$
Kinetic rebreeding	Rebuild $^{\Delta}$
Replacement type	Generational

$^{\Delta}$: kinetics only

EC Scoring Variables	
Fitness (GA & sub-)	Euclidean
Fitness (GP)	Tree Size $^{\times}$
Bloat type	Gradient $^{\times}$
Bloat penalty	~5% $^{\times}$
PEPA & Bio-PEPA Variables	
Simulation algorithm	Gibson-Bruck
Internal replications	10
External replications	1
Failure timeout	~10s $^{\diamond}$
EPA System Variables	
Independent runs	50
Memory allocation	510Mb (per run)

$^{\times}$: GP only $^{\diamond}$: Bio-PEPA only

Table 4.1: Default EPA variables by category

The EPA framework's user is able to override any value of the default variable set should they desire. This can be achieved from the command line or via the *EPA Utility Assistant* for novice users.

How these variables are utilised by the framework is described throughout the next section as each stage of the EPA lifecycle is discussed.

4.3 THE EVOLVING PROCESS ALGEBRA LIFECYCLE

The EPA framework follows a six-step lifecycle during execution, as illustrated in Figure 4.4. The *Start* and *Simulate* steps (highlighted in yellow) are in the Process Algebra domain and are accomplished using the relevant PA plug-in. The *Score*, *Select*, *Breed* and *Finish* steps (highlighted in green) are in the Evolutionary Computation domain and are executed in the ECJ toolkit or the EPA framework itself. A small portion of the *Start* step is also in the EC domain.

A single execution of the EPA lifecycle would yield 100 (the population size) possible solutions, of which at least one is hoped to be the (or close to the) global optimum. To achieve statistical significance, the EPA is executed multiple times with the results from each run collated together. If an individual solution appears in multiple runs it is more likely to be a high fitness individual. Also, multiple runs allow the EPA to avoid being entirely stuck

in a local optimum: while one system run may become stuck, many others will still finish successfully.

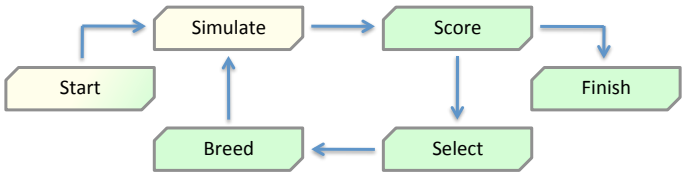


Figure 4.4: The EPA lifecycle

The EPA framework is designed to be parallelised using HTCondor, described above. Each full lifecycle, from *Start* to *Finish* steps, takes place on one HTCondor node (but may be moved to another node if suspended). Each separate lifecycle is executed on a separate node. The default number of EPA runs is 50, but this value could be increased depending on available nodes.

The remainder of this section elaborates on each of the six steps in the EPA lifecycle.

(i) START

The *Start* step is only done once per EPA framework execution as a preparation for the following steps. The first input to the framework is a prebuilt PEPA or Bio-PEPA model; this could be prepared freehand or developed in the appropriate plug-in tool. The second input is the target for the EC techniques to achieve; this commonly takes the form of a time series trace, but could be a desired set of behaviours (for instance, the cyclic behaviour of the repressilator system in Section 5.4), or both.

One or more segments of the model will be flagged for optimisation; this will be the rate and/or starting population values for GA evolution in PEPA models; or additionally, kinetic action and/or species definitions for GP optimisation in Bio-PEPA models. The flagged segments need not have values or be defined; only the segment name needs to exist in the initial model, for example:

$$\begin{aligned} X &\stackrel{def}{=} \ldots \\ Y &\stackrel{def}{=} \ldots \\ Z &\stackrel{def}{=} (alpha, 3) \gg Z \end{aligned}$$

This is a valid Bio-PEPA species definition snippet for the EPA framework only if the X and Y species are flagged for optimisation. If they are not, the framework will detect the incomplete model and report the error before terminating. Any existing definitions or part definitions for a flagged segment will be overwritten by the evolved definitions during the *Simulate* step. As such, Z may be flagged for optimisation, despite having a complete species definition.

In addition to accepting the model and target inputs, the *Start* step also prepares the initial population of the system, i.e. generation 0. This generation is generally a random distribution across the search space; the size of a search space is problem specific and may

become particularly large, especially kinetics experiments (Chapter 7). If desired, the starting population can be partially or wholly seeded with domain knowledge of the search space or continue from where a previous population finished.

4.3.1 *Utilising Domain Knowledge*

Previously, in Section 2.3, Evolutionary Computation techniques were described as guided trial-and-error methods for finding optimal solutions in a large search space. By utilising domain knowledge during the *Start* step, the search can start or focus in a more fruitful region of the problem space.

The EPA framework has no innate knowledge of what the species, the actions, or their interactions mean. The solution structures created are driven entirely by the *fitness function* (discussed in the *Score* step below). Therefore, if the solution is scored a good fitness, its domain-specific validity is irrelevant. This can be addressed through the use of domain knowledge.

Consider the G-protein system, described fully in Sections 5.5 and 6.4. The G-protein system comprises of two separate cycles, one on the outside of the cell and one on the inside, where the first has an excitatory effect on the second. Actions available to the first cycle are not available to the second and vice versa. For example, the *bind_RL* action (in Bio-PEPA notation) binds receptors and ligands which exist outside the cell, this action is not applicable to the other cycle inside the cell. A domain expert would know this, whereas the EPA framework would not.

When incorporating domain knowledge, the EPA framework can be given species-action *ban lists*. When generation 0 is seeded, each individual will not violate the restrictions of the ban list; species are restricted to only utilising kinetic actions not on their list. Checks are in place to ensure not all actions are restricted, thus leading to no valid solutions. This knowledge is also used in the *Breed* step to ensure subsequent generations do not violate these restrictions.

While utilising domain knowledge, caution should be exercised. Being overly restrictive could adversely affect the course of optimisation. This could either be by causing premature convergence, or by restricting the solutions to what the domain-expert wants rather than for allowing proper exploration of the search space.

(ii) SIMULATE

The *Simulate* step will normally be reached from the *Breed* step, with the exception being from generation 0, where the individuals being simulated will be those seeded during the *Start* step. In preparation for simulation, an individual is parsed to form a complete model. Elites, discussed below in the *Breed* step, do not need to be parsed in subsequent generations as their structure will not change.

Each individual comprises of sets of numeric values for the GA rate and population values, also GP tree structures for the kinetic and species definitions. The EPA framework uses multiline pattern-matchers to remove any whole or part definitions that may already exist in the precursor model, as described in the *Start* step.

RATE: ^\s*<*variable*>(\s*=\s*)\d*\.?\d*E?-?\d*;

KINETIC: kineticLawOf\s*<*variable*>\s*:\s*.*;

SPECIES: <*variable*>\s*=\s*(\w+|\(\w+,\s*\d\))\s*...?\s*<*variable*>
(\s*\+\s*(\w+|\(\w+,\s*\d\))\s*...?\s*<*variable*>)*;

POPULATION: \W\s*<*variable*>\s*\[\d+\]\s*

In each pattern, the <*variable*> term is replaced by the name of the segment being flagged. Through use of generic regular expressions (such as \s* for any amount of whitespace), poorly or differently formatted models may be handled safely. Also note, each backslash is duplicated in the Java environment, this is to escape the original backslash at compile-time.

After preparation, the Process Algebra model is supplied as a *String* to the plug-in for error checking and simulation. If errors prevent the model from being simulated, it will be flagged as poor for the *Score* step. The PA plug-ins can also detect poorly formed models, but these can still be simulated to termination.

Once past the error checks, the Process Algebra model is ready for simulation. The *Gillespie* PA simulator algorithm is a standard method of simulation. The *Gibson-Bruck* simulator algorithm builds on the Gillespie simulator for the purposes of time and computational resource efficiency [37]. As a default, it is the Gibson-Bruck simulator algorithm used by the EPA framework, however the Gillespie algorithm is also useable. The simulation is conducted from a start point, to an end point with a particular granularity; all these variables are user-specified and model specific.

4.3.2 *Replications*

Process Algebra simulation is a stochastic operation. For this reason, if a model is simulated multiple times it will result in different output time-series traces, even if simulated with identical parameter settings. This can cause a large amount of stochastic noise which will hide model behaviour.

In addition, Evolutionary Computation optimisation techniques can also be noisy, further hiding these behaviours. This stochastic noise must be reduced to discover model behaviour and accurately measure fitness.

4.3.2.1 *Internal Replications*

Consider Figure 4.5, three arbitrary traces. The left image presents a standard reduced-noise trace produced with 100 replications, notice the smooth curves along each trace series.

Compare this to the centre image, the same model run with only 1 replication, notice the more jagged and unrefined trace produced under these conditions.

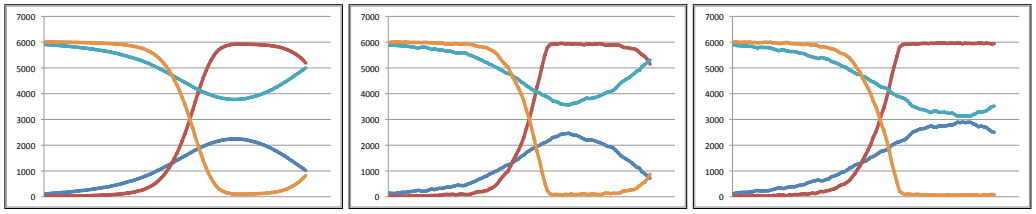


Figure 4.5: Three runs with 100 (left), 1 (centre) and 1 (right) internal replications

Lastly, consider the right image, this represents a trace produced from the same model with only 1 replication; the differences can be explained by the stochastic nature of PA simulation. Such a difference in the trace could cause a fit model to be scored poorly despite how well that model would normally perform. Conversely, a normally poor solution could be scored strongly because of these stochastic differences.

Internal replications are the primary method used by the EPA framework to reduce the impact of outlier results and stochastic noise. For each individual, the parsed model is run N times (where N is the number of internal replications, default 10). The time series traces produced from each of those runs is then averaged over N to give a singular final time series trace which will be scored.

4.3.2.2 *External Replications*

In some systems, the application of internal replications can cause further stochastic noise or entirely mask the system's behaviour. This is especially true for systems which exhibit cyclic or seasonal behaviour.

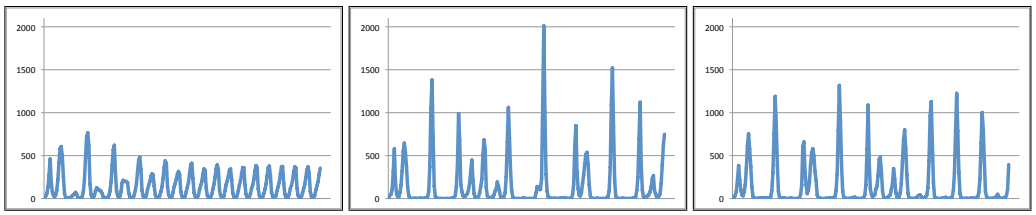


Figure 4.6: Three runs with 100 (left), 1 (centre) and 1 (right) internal replications

Figure 4.6 presents three results from one Susceptible-Exposed-Infected-Recovered (SEIR) model, each spike is indicative of an epidemic outbreak. The left image is produced with 100 internal replications, notice the multiple smaller peaks caused by averaging many runs. The centre and right images of this figure are examples of 1 internal replication run. This SEIR example exhibits cyclic behaviour where the pattern of infectious spikes is not consistent enough to average using internal replications.

Work with the repressilator system, discussed later in Section 5.4, inspired the implementation of *external replications*.

External replications follow a different view of the lifecycle, combining the *Simulate* and *Score* steps into an iterative cycle. The model is simulated and the resultant time series trace is then immediately scored. To reduce the impact of outliers and stochastic noise, this process is then repeated N times and the sum of these fitness scores is then averaged over N to produce the final fitness score.

Normally, external and internal replications will not be combined. If a model includes cyclic or seasonal behaviour, or otherwise warrants external replications, the number of internal replications will be set to one.

However, it is possible to use external and internal replications together. If both are used, the number of replications is calculated multiplicatively: for each external replication, the full number of internal replications are completed.

4.3.3 *Failure Timeout*

The failure timeout feature is only available for Bio-PEPA models which are running on threads. By default, each model simulation is dispatched on a separate thread and the results are collected together when the simulation ends. The time a simulation has been running on a thread is tracked.

It is possible to run each Bio-PEPA simulation sequentially without using separate threads, however this approach is very time-inefficient and is normally only used for the purposes of debugging.

Bio-PEPA models generally have a short simulation time which allowed for the introduction of the failure timeout variable. The exact value of this variable should be approximately 2.5 times the normal simulation time. If a simulation exceeds this duration then it is prematurely terminated and flagged as poor for the *Score* step. These simulations will have increased species levels beyond which the simulator can operate. Removing these solutions early reduces time wasted on known failures.

(iii) SCORE

The *Score* step is only reached from the *Simulate* step. The individual solution and associated raw time series trace are provided to the EPA scoring package where the solution will be assigned a fitness score representing its match to the target time series trace and any desired behaviours.

In general, a higher fitness score indicates a better solution. Evolutionary Computation (EC) techniques frequently use a minimising function, therefore a lower fitness score indicates a better solution.

An optimal score is zero, or no difference; indicating the solution provided an exact match to the model's target trace and any desired behaviour. The maximum fitness score is system dependent, but is normally positive infinity. A perfect score of zero is theoretically possible

but is impossible to achieve in practice. The stochastic nature of simulations will always add some element of noise to the raw time series trace, partially obscuring behaviour and causing the fitness score to be non-zero.

If an individual is passed into the *Score* step with a poor fitness flag, it is assigned the worst fitness possible for that system. This value is normally positive infinity. This flag can be caused by a solution containing errors or taking too long to simulate. Nothing more needs to be done during the *Score* step for these flagged individuals.

Choosing how to score a model is equally as important as the model itself. The *fitness function* accepts the time series trace from the *Simulate* step and objectively measures the individual's performance against the target trace and/or desired behaviours. The fitness function must be specific enough to capture all the desired behaviour without being overly constrained that the results prematurely converge.

In addition to being specific, the fitness function must be accurate: EC techniques are prone to taking any shortcuts available. For an example, consider a fitness function that penalises a GP tree structure based on size in order to control bloat (described fully in Section 4.3.5 below). If the penalty is too high, it will outweigh the score to match the target trace or desired behaviour. Instead, the fitness function will concentrate on reducing the size of solution trees, leading to poor yet high scoring solutions.

4.3.4 *Euclidean Distance Fitness Scoring*

The Euclidean distance measure is a basic fitness function which scores individuals based on the difference between two matrices, where one matrix is the simulated time series trace and the other is the target time series trace.

During preliminary incorporation of new models, the Euclidean distance fitness function was used; this was to avoid adding additional complexity through use of more complex fitness functions. During GP optimisation work, additional fitness functions were developed to score fitnesses using more complex measures other than distance: details follow in Sections 4.3.5, 4.3.6 an 4.3.7.

Consider Table 4.2 as an example: the target consists of two sets of time series data, the first column ranges from 0 to 10, and the second from 0 to 50. The values are first normalised to a range between 0 and 1, these normalised values are shown after the arrow in each column. This normalisation allows time series data to exist on different ranges and still be weighted equally and compared fairly.

Columns three and four are the time series trace produced from the evolved individual as passed in from the *Simulate* step, then normalised across the same range as columns one and two. The final column is the squared difference; calculated by taking the square of the differences between each series of the target trace and the corresponding value from the evolved trace. The Euclidean distance is the square root of the sum of the final column. It

is this value which is assigned as the fitness to the solution which created the evolved time series data.

Target Trace						Evolved Trace						Differences^2
0	→	0	0	→	0	0	→	0	0	→	0	$0^2 + 0^2 = 0$
4	→	0.4	12	→	0.24	3	→	0.3	9	→	0.18	$0.1^2 + 0.06^2 = 0.0136$
5	→	0.5	24	→	0.48	5	→	0.5	14	→	0.28	$0^2 + 0.2^2 = 0.04$
8	→	0.8	30	→	0.6	9	→	0.9	20	→	0.4	$-0.1^2 + 0.2^2 = 0.05$
10	→	1	50	→	1	10	→	1	25	→	0.5	$0^2 + 0.5^2 = 0.25$
raw value → normalised value												$\sum$Differences = 0.3536
												Distance (Fit) = 0.5946

Table 4.2: Euclidean distance fitness measure

A feature of using the Euclidean distance measure for fitness scoring is the ability to accurately calculate the maximum (worst) fitness value. With each value on every time series trace normalised to a range of 0 and 1, the greatest possible difference at each point is 1. Therefore the maximum fitness value is equal to the number of data points which is also equal to the time series trace length multiplied by the number of columns in that trace. So, $max\_fitness = num\_data\_points = timeseries\_length * num\_columns$. Knowing the maximum fitness value, the fitness score can be viewed along the scale of possible scores, thus giving an indication of the solution's overall performance.

The simple Euclidean distance measure can be built upon to capture additional desirable behaviour. To score solutions more accurately, fitness functions can be tailored to individual problems. This is especially desirable for large models or those describing particularly complex systems.

4.3.5 GP Tree Fitness Scoring

The kinetic and species definitions use Genetic Programming (GP) techniques for optimisation, full details can be found in Section 2.3.5. Each GP tree consists of multiple nodes to build up a definition, this immediately presents an additional criteria to score: the tree size. Ideally, the GP tree will adequately describe the model's behaviour in a concise manner. This is not always the case; since Genetic Programming techniques are prone to *bloat*.

Bloat is defined as an increase in the GP tree size with no appreciable improvement in that tree's fitness score or function. Bloat is principally controlled by the fitness function. The tree-size evaluator fitness function is built directly on top of the Euclidean distance fitness function detailed in the previous section. However, it can be built onto any other fitness function, now called the *sub-fitness function*. During GP tree fitness scoring, the sub-fitness

function provides a fitness score based on the target trace and/or desired behaviours, now called the *trace score*.

The tree-size evaluator assigns a score to the individual based on the GP tree structure regardless of the target trace, the desired behaviours or the trace score. The species GP definition trees are not overly susceptible to bloat; the maximum number of choices in a species definition has an upper bound equal to the number of kinetic actions. However, the kinetic GP definition trees are far more susceptible to bloat; the maximum number of nodes is bounded by a depth variable, this is 5 as an ECJ and Koza [47] default. The tree-size fitness score gets higher (poorer) proportionate to the GP tree sizes of the individual being scored. If species and kinetic trees are optimised together, they are weighted equally in terms of the tree-size fitness score.

The bloat penalties are based on the number of nodes used in a solution (*num_nodes*), the number of available kinetic actions available in the model (*num_kinetics*) and the number of children of the *Choice* node (*num_choices*).

The species bloat penalty is calculated as *species_bloat* = *num_choices* / *num_kinetics* and the kinetics bloat penalty is calculated as *kinetics_bloat* = *num_nodes* / $(2^{tree\_depth} - 1)$. Once calculated, the trace score is augmented by the tree-size fitness score. This is done additively. Doing so multiplicatively would skew the final fitness score, poor solutions would become even poorer. Furthermore, the tree-size fitness scores could not be fairly compared if applied multiplicatively.

The maximum bloat penalty should be approximately 5% of the maximum fitness range, or an appropriate amount if the maximum is positive infinity. Testing bloat penalties from 3% to 15% inclusive, 5% provided the strongest solutions in GP experiments which utilised the tree-size evaluator fitness function. If a concise definition is particularly important, the bloat penalty can be increased accordingly. This weighting allows the fitness function to pressure solutions into a more concise form, without overpowering the trace score and leading to small but useless solutions.

The tree-size evaluator fitness score is not always applied to the trace score; the EPA framework has four preprogrammed modes which can be used. *None* indicates there is no bloat penalty, the final fitness score is entirely based on the trace score. Conversely, *Always* will apply the whole bloat penalty for every individual from generation 0. The *Switch* mode comes with a user-defined generation between 1 and the number of generations, before this generation there is no bloat penalty (as with the *None* mode) and after this generation there is a full bloat penalty (as with the *Always* mode). Finally, *Gradient*; the bloat penalty is multiplied by (*current_generation* / *num_generations*) before being added to the trace score.

The *Gradient* mode is the EPA default. Using this formula the penalty is at its weakest during early iterations of the EPA lifecycle, allowing exploration, but becomes stronger towards the final iterations, which encourages exploitation of compact solutions.

4.3.6 GP *Node Weight Fitness Scoring*

The tree-size evaluator fitness function, described above in Section 4.3.5, scores a GP tree based on its size regardless of the tree's content or meaning. If a correct definition is large while correctly encapsulating the system's behaviour it may be scored poorly due to its size. The kinetics GP definition tree can utilise a second built in fitness function: the *node-weight evaluator*, which allows large definitions to score well based on their composition.

The node-weight fitness function is built on top of a sub-fitness function in the same design as the tree-size evaluator described above. During GP tree fitness scoring, the sub-fitness function provides a fitness score based on the target trace and/or desired behaviours, which is then augmented by its node weighting.

Table 4.3 presents the weights used by the EPA framework. Each weight is a positive non-zero value; the values chosen should be small in comparison to the trace score to not overshadow the original trace. In addition, the ratio of each weighting is proportional to the commonality of that node in kinetic definitions: i.e. *Math*, *Rate* and *Species* nodes are particularly common so their weight is small. Composite nodes, as described in Section 4.1.1.2, are high level solution structures and are used at an increased penalty; this encourages new solutions to be built rather than relying on the composite nodes.

Weight	Kinetic Node(s)
0.1	*Math*, *Rate*, *Species*
0.2	*Power*
0.3	*ByteVal*, *MassAction*
0.5	βSI, βSI / n, *Sum*

Table 4.3: GP Node Weights

Through use of the GP node-weight evaluator, larger structures can be evolved without being judged solely on their size. If desired, the node-weight evaluator and tree-size evaluator can be chained together so an individual can be scored on both size and structure. The scoring for both GP evaluators is additive, so neither takes precedence over the other.

The node-weight evaluator is not compatible with species definition trees. The structure of a species GP tree is far stricter than a kinetics GP tree. The children of a *Choice* node are identically structured (see Figure 4.3), therefore a node-weight fitness score would differ only by the number of these children in the same way as the species tree-size evaluator fitness function, described above.

4.3.7 *Additional Fitness Scoring*

The Euclidean distance, GP tree-size and GP node-weight fitness evaluators presented in this section are the most frequently used fitness functions used in this work. Should a user require

a more complex fitness function, the EPA framework can be extended. All fitness function are subclasses to an abstract *AbstractEvaluator* superclass; the abstract *getFitness* method must be implemented to return the fitness score to a provided individual. All inbuilt EPA fitness functions are stored in the internal *scoring* package. However, any custom fitness function may be utilised if addressed in the parameter file; detailed with the *EPA Utility Assistant* in Appendix A.

(iv) SELECT

The output of the *Score* step is an EPA *FitnessData* object which is passed into the *Select* step. The FitnessData contains all scoring data: the trace score, any penalty types applied, the penalty values applied and the final fitness score.

The role of the *Select* step is to choose which individuals will be parents and put their attributes forwards into the next generation in the form of offspring. Selection is one of the major influences between exploration and exploitation. Selection techniques which favour only the most fit individuals are highly exploitative, whereas techniques which allow poorer individuals to pass on their attributes are more explorative.

4.3.8 *Tournament Selection*

The background of the tournament selection algorithm was discussed in Section 2.3.1.2. The tournament selection function involves running several tournaments where each will yield an individual from the population with good fitness, but may not be the most fit individual. These individuals will be put forwards as parents.

A *tournament pool* is formed of a specified size; this size influences the exploration-exploitation balance. The larger the pool, the more likely the most fit individual will be encompassed, leading to exploitation. Conversely, a smaller pool may miss more fit individuals, allowing for more exploration of the search space. A pool of size one is effectively a random selection algorithm and a pool size equal to the population size is equivalent to a exploitative hill-climbing algorithm.

The tournament pool is populated by randomly choosing individuals from the population for entry. The same individual may not be duplicated in a pool; for this reason, the tournament pool cannot be larger than the population size. However, the same individual may appear in multiple pools and be selected multiple times.

Once the tournament pool is full, the most fit individual of the pool 'wins' and is put forwards as a parent for the *Breed* step. Each individual to be bred requires two parents, so this process is done twice for each such individual. Per generation the number of tournament pools required for selection is $(population\_size - num\_elites) * 2$. Elitism is discussed in Section 4.3.9.

Table 4.4 presents a hypothetical population of ten individuals with their fitness scores. These individuals are sorted in ascending order by fitness; this is for readability, in general

selection algorithms will not need to sort the population. Consider a tournament pool of size three which is randomly populated with individuals 2, 7 and 9; the winner for parent one is individual 2. A second tournament pool is randomly populated with individuals 3 and 7; tournament entries are unique therefore 3 cannot be duplicated, however individual 7 is permitted in pools for both parents. The final individual for the second pool is 10; the winner for parent two is individual 3.

#	Fitness	#	Fitness
1	0.0036	6	1.7438
2	0.0112	7	1.9942
3	0.0458	8	2.6745
4	0.1955	9	4.8340
5	0.5782	10	9.4372

Table 4.4: Tournament selection example, sorted in ascending order by fitness

It is possible, though unlikely, to get the same individual in both parent slots. The offspring can still differ from the parents due to the breeding operators, discussed in the *Breed* step. Notice, the most fit individual of the population was not entered into a tournament pool and is therefore ineligible for selection.

4.3.9 *Elitism*

Elitism is a modifier to the normal selection procedure which can be combined with any selection algorithm. Elitism involves selecting a number of *elite* individuals in a population which gain a 'bye'. That is, elite individuals immediately enter the next generation, unmodified, following normal replacement techniques. Possible variations and replacement techniques are discussed in the *Breed* step.

Elitism is a stabilising factor which allows selection algorithms to be more explorative without the risk of losing the current best individual; for instance, by not entering a tournament pool and being ineligible for selection. This ensures that the best individual can pass on their strong attributes to the next generation.

The more elites in a system, the heavier the exploitation of the current population. Care must be taken to not include too many elites; this would cause the elite individuals to dominate the system, leading to a lack of exploration and a premature convergence.

(v) BREED

Once high fitness individuals have been selected to pass on their attributes, the *Breed* step is responsible for improving upon those individuals, now called the *parents*. The selected parents represent relatively good solutions that have been found in the search space. It is assumed that good solutions will be neighboured by solutions of similar or higher fitness. By

combining the attributes of two parents, new and possibly fitter areas of the search space can be discovered.

The iterative nature of the EPA lifecycle aims to discover more and more fit solutions each generation, towards the global optimum. In actuality, combining two fit individuals can lead to poor solutions, especially when combining GP trees or complex individuals. This is because interconnected values are not always preserved during the breeding process. Poorer fitness children are permitted to replace their parents, this promotes further exploration of the search space.

4.3.10 *Variation*

Stagnation would quickly occur if all selected individuals were cloned into the child generation. Selected parents are bred to introduce variation into the created offspring. Breeding is represented through recombination (or crossover) and mutation operators.

The recombination operator, commonly called crossover, mixes the solution structure of two selected parents. This attempts to combine the fit attributes of two selected parents into an even fitter offspring.

The mutation operator has a chance of resetting the value at a random location on the solution structure. A higher mutation rate reduces the chance of becoming caught in a local optimum. However, if the rate becomes too high the Evolutionary Algorithm will have difficulty locating an optimised solution because solutions will bounce near-randomly around the search space.

An important note: if multiple segments of a Process Algebra model are flagged for optimisation, an individual will be comprised of different structures to optimise those flagged segments (Section 4.1): GA representations for rate and population levels and GP representations for kinetic and species definitions. During the *Breed* step, like can only be bred with like: rates with rates, kinetics with kinetics, species with species and lastly population levels with populations levels.

Varying GA and GP representations are discussed separately over the following sections. Many of the breeding examples display two offspring solutions being generated; this is to highlight the crossover and mutation operators. At runtime, only one of the child solutions of a breeding occurrence is passed into the offspring generation. This choice allows for more exploration of the search space by halving the exploitative influence of each breeding occurrence.

4.3.10.1 *Varying GAs*

The representation and variation of Genetic Algorithms was discussed previously in Section 2.3.4. The EPA framework implements two GA individuals: the *RatesIndividual* and *PopulationsIndividual*. Both these individuals use the linear chromosome representation of values.

The only difference is that rates are *double* values whereas populations are *integer* values (it is not possible to have a fractional member of a population).

The values held in the representation of a GA Individual do not affect the breeding operators available to that individual. Figure 4.7 illustrates a one-point crossover example on a *RatesIndividual* and a mutation example on a *PopulationsIndividual*, to the left and right respectively.

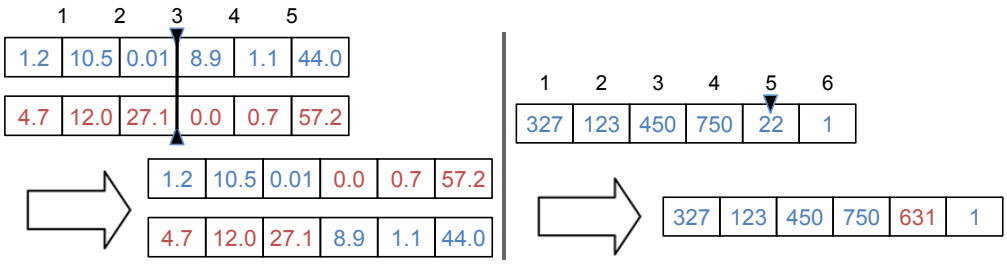


Figure 4.7: Crossover (left) and Mutation (right) operators on example EPA individuals

There are three forms of crossover operator: *one-point*, *two-point* and *uniform* crossover operators. One-point (as illustrated) stochastically selects a gap between loci and builds offspring solutions from the left portion from one parent and the right from the other parent. Two-point crossover operates similarly, except two arbitrary points are selected along the chromosome length. The offspring solution comprises of one parent between those two points with the remainder on each end from the other parent. Uniform crossover moves along the length of the chromosome; for each locus, the offspring's locus is randomly chosen from either of its parents. This effectively randomises each locus and is more explorative than the other crossover operators.

The mutation operator is simpler and has a chance of resetting the value at zero or more random locations on the chromosome. To the right of Figure 4.7, the fifth locus has been selected. The mutated value can be any number in the user-specified permissible range; in this example 22 is mutated to 631.

With Genetic Algorithms, the crossover operator is heavily exploitative. Conversely, the mutator operator is explorative as changes can move the solution into new areas of the search space.

Both GAs use the one-point crossover operator and the standard mutation operator. The crossover operator is applied to all GA breeding occurrences to reduce the likelihood of premature convergence. The mutation operator has a 5% chance to be applied to a locus, this further allows for exploration of the search space without being too high to become overly explorative.

The uniform crossover approach will arbitrarily mix the alleles of both parent solutions; increasing the probability of associations between neighbouring loci being lost. Using one-point crossover means any associations between neighbouring loci will likely be preserved.

The chromosome representation of a GA in the EPA framework is linear, not circular, so no extra benefit is gained by using two-point crossover.

4.3.10.2 *Varying GP - Species*

This section focuses on the GP species syntax tree along with the crossover and mutation operators available to it. Traditional GP tree structures, discussed in Section 2.3.5, allow the crossover operator to be performed where different edges (or connections) are chosen on each parent. Due to the strict Bio-PEPA syntax this is not permissible as it would lead to violations of the syntax and would automatically fail upon simulation. To prevent this, the same connection is chosen in both parents; this can be achieved because all species GP trees follow a strict structure as illustrated previously in Figure 4.3.

Some GP breeding operators allow multiple connections to be selected with multiple subtree crossovers; however, the EPA framework uses a single connection crossover technique akin to GA one-point crossover.

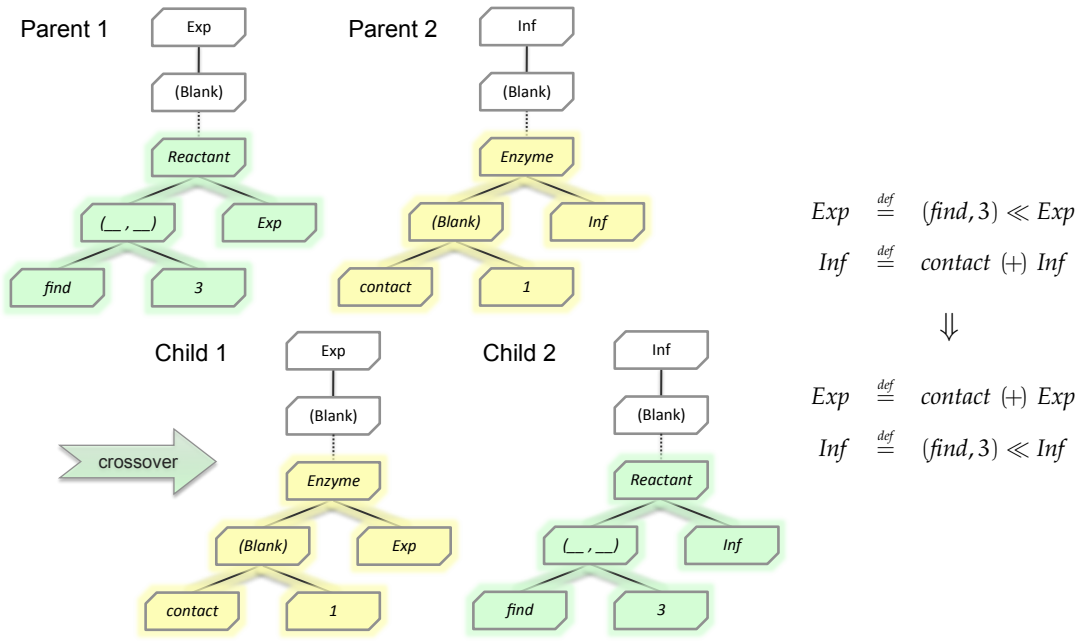


Figure 4.8: Crossover operator on a Bio-PEPA syntax GP species tree

When performing crossover on a GP tree structure the arbitrary point selected is a connecting branch. In Figure 4.8, one-point crossover is performed on the connection between the *Choice* and *Operator* nodes; denoted by the dashed lines. As mentioned above, the same branch is chosen on the second parent.

To satisfy the Bio-PEPA syntax, the *SpeciesID* node has its value changed to match the new *TreeToot* identifier. In this example, the *Exp* from *(find, 3) ≪ Exp* becomes a *Inf* to match the new *TreeRoot*; similarly the *Inf* from *contact (+) Inf* becomes *Exp*.

The crossover operator occurs for all species GP breeding occurrences. Due to the strict syntax of these trees, the crossover is not as destructive as in traditional GP crossover, allowing the EPA framework to be more explorative. This is not to say that all crossover operations will result in viable offspring.

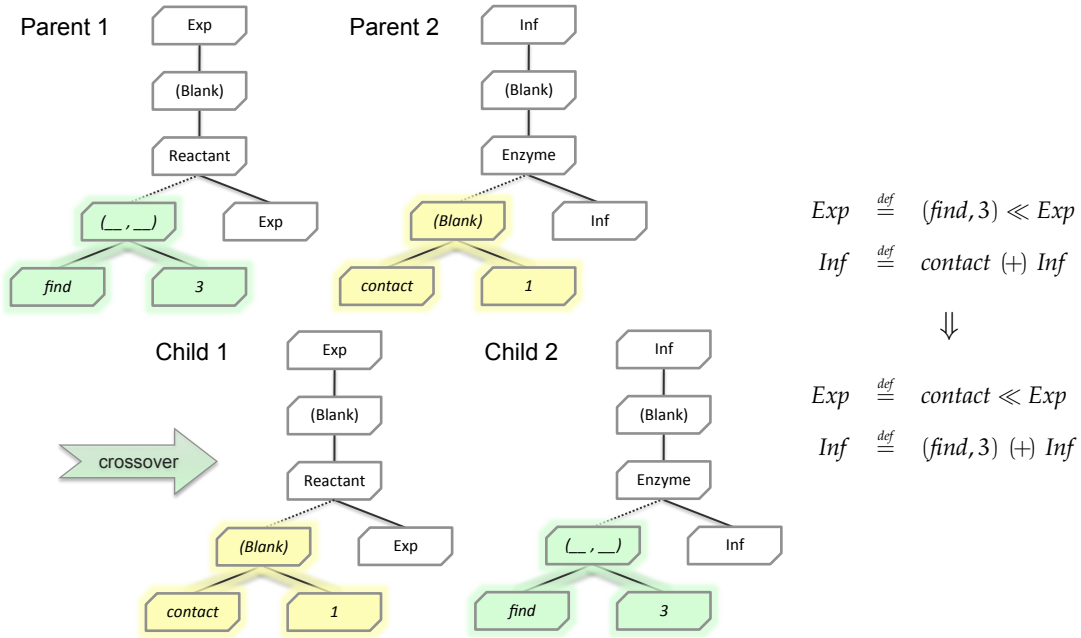


Figure 4.9: Crossover operator on a Bio-PEPA syntax GP species tree (with violation)

Figure 4.9 shows a crossover operation deeper in the GP tree; between the *Operator* and *Prefix* nodes: again, denoted by the dashed lines. This recombination is simpler because no node values need to be changed to ensure syntactical correctness.

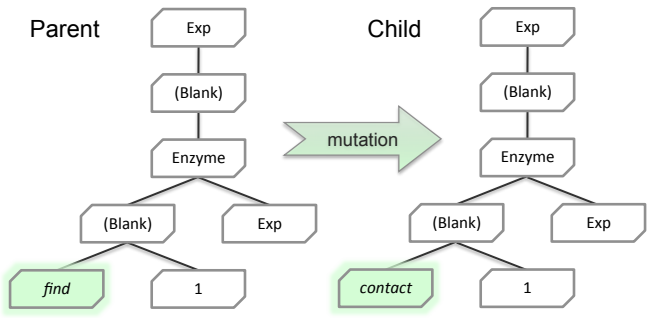


Figure 4.10: Mutation operator on a Bio-PEPA syntax GP species tree

However, in this example, the crossover will cause a violation in the Bio-PEPA syntax because a modifier (Section 2.2.2) cannot have a stoichiometry coefficient that is not equal to 1: this crossover has caused a value of 3 to be used. The EPA framework attempts to remove such erroneous offspring during the replacement process, described in Section 4.3.12. Figure 4.10

demonstrates the effect of the mutator operator on a GP tree structure. The mutator operator has a 5% chance to alter the state of a node in an offspring individual; as with the tree depth variable, this is an ECJ and Koza [47] default. This value allows a degree of exploitation of the parent GP tree structure, without being too exploitative or mutating too frequently to become overly explorative. One node is selected at random and is reset to any permissible value: an *Action* node can only contain actions, the *Operator* node can only contain operators, etc.

The mutator cannot be applied to *SpeciesID* or *Prefix* nodes. The *SpeciesID* node is a placeholder and can only contain the identifier of the *TreeRoot* node. The *Prefix* node is a container element designed to group the *Action* node with its stoichiometry coefficient in brackets, or remain hidden and only display the *Action* node if the stoichiometry coefficient is equal to 1. By removing these two nodes as possible mutation targets, the EPA ensures each mutation will have an impact on the resultant offspring.

Though not the most efficient approach, this simplified implementation while also keeping in line with ECJ design principles.

An additional special case is if the mutator is applied to the *Choice* node, where the number of child nodes will be altered. If the number is reduced, random subtree branches are removed to match this new lower value. Conversely, if the number is increased, new branches are generated and added to the GP tree. These generated branches are appended to the list of child nodes in the *Choice* node, the order of child nodes is unimportant. The blank *Choice* node contains +'s if it has more than one child node.

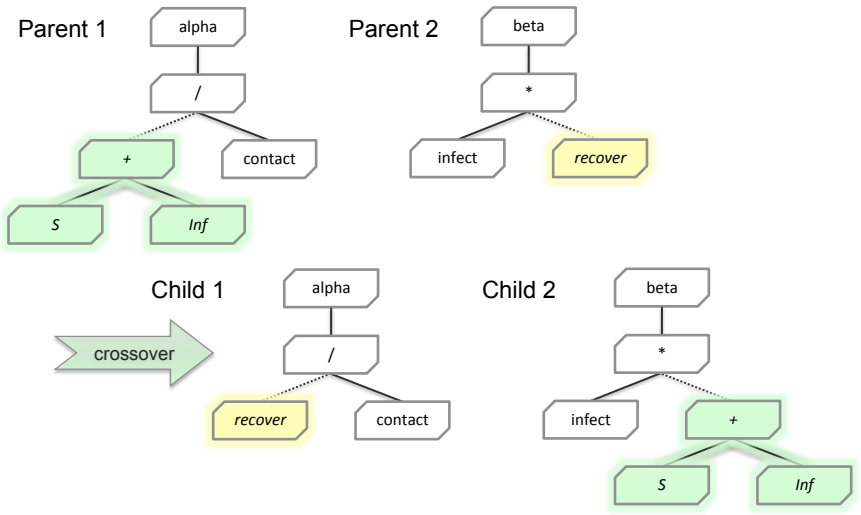
With Genetic Programming techniques, the roles of the crossover and mutator operators are reversed compared to GA operators. The crossover operator is heavily explorative and can be destructive too; two GP trees may individually have good fitnesses but combining the two will normally lead to a poor fitness region of the search space. The mutator operator is the exploitative influence; taking fit GP trees and mutating a single node is more likely to yield stronger offspring.

4.3.10.3 *Varying GP - Kinetics*

This section focuses on the Genetic Programming kinetics syntax tree along with the crossover and mutation operators available to it. The crossover operator for a kinetics GP tree is more indicative of traditional crossover (Section 2.3.5) than the species crossover operator, described above. This is because the Bio-PEPA syntax for kinetics definitions is not as strict as for species definitions.

Unlike crossover for species trees, the chosen node connection can be at different positions and depths on each parent; the size of the detached subtree can also be different, as in Figure 4.11. In addition, there are no special cases which would require a node to have its value changed: such as changing the *SpeciesID* node to match the *TreeRoot* node, as previously seen in Figure 4.8.

The likelihood of applying the crossover operator to a GP kinetics tree is reduced in comparison to the species crossover operator. Crossover is performed in 90% of kinetics breeding occurrences, the remaining 10% of breeding simply copies one of the selected parents into the offspring generation. The EPA framework implementation copies parent one for the offspring solution. This copying is not the same as elitism or cloning: copied individuals are still subject to possible mutation, described below. Kinetic GP crossover does not maintain the original tree dynamics of width, depth or number of nodes. Therefore, crossover in kinetics GP trees can quickly cause a large degree of *bloat* (discussed previously in Section 4.3.5); the crossover probability is reduced to aid in lowering bloat.



$$kineticLawOf\ \ alpha\ :\ (S + Inf)/contact \quad \Rightarrow \quad kineticLawOf\ \ alpha\ :\ recover/contact$$

$$kineticLawOf\ \ beta\ :\ infect * recover \quad \Rightarrow \quad kineticLawOf\ \ beta\ :\ infect * (S + Inf)$$

Figure 4.11: Crossover operator on a Bio-PEPA syntax GP kinetics tree

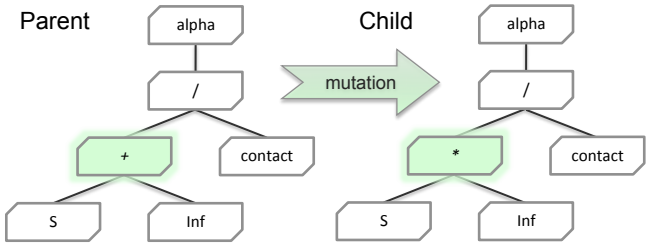
Like the species mutation operator, the kinetic mutation operator has a 5% chance of occurring. The mutation operator for the kinetics GP trees has two types: the *point* mutation and the *subtree* mutation (Section 2.3.5). When mutation occurs, one of these two types is utilised. The ratio of point to subtree mutation being selected is 1:1, so each has equal chance of being selected.

Mutation is the exploitative influence for GP tree variation. Point mutation is the more exploitative of the two operators available while subtree mutation is more traditional and highly explorative. While both mutation operators are exploitative, the ratio between point and subtree mutation allows a balance between exploitation and allowing additional exploration.

If an offspring individual is not subject to recombination for any reason, the chance of the mutation operator being applied is overridden to 100%. This could be because the 10% chance of copying was taken, or two identical parents were selected for breeding thus resulting in a crossover operation which produced an offspring identical to its parents. The exact mutation operator applied (point or subtree) is then determined based on the ratio described above. By

guaranteeing mutation when offspring solutions match the selected parents, the EPA aims to further avoid population stagnation or premature convergence.

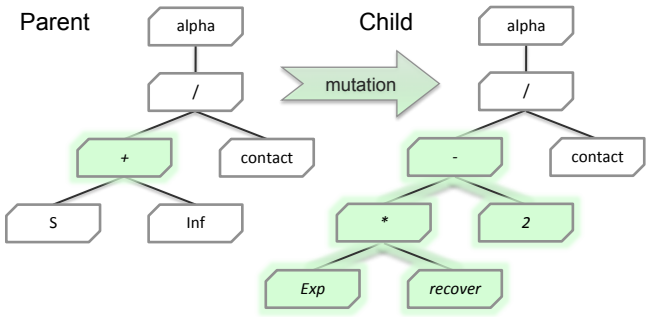
Figure 4.12 provides an example of the point mutation operator. When this occurs, a random node is selected anywhere on the GP kinetic tree. Unlike the species mutator operator, all nodes can be selected because all nodes impact the final definition. The selected node retains its type but has its value reset to any permitted value, a *Byte* node can only contain a byte value, etc.



$$kineticLawOf\ \ alpha\ :\ (S + Inf)/contact \quad \Rightarrow \quad kineticLawOf\ \ alpha\ :\ (S * Inf)/contact$$

Figure 4.12: Mutation (point) operator on a Bio-PEPA syntax GP kinetics tree

Figure 4.13 demonstrates the subtree mutation operator on the same parent GP tree as the point mutation above. Similarly to the point mutation operator, a random node is selected from anywhere on the GP kinetic tree. The key difference is what is mutated. Any existing child nodes (and their children) of the selected node are entirely removed. The chosen node is then rebuilt as a new subtree which can be a terminal or include child nodes, child-of-child nodes, etc.



$$kineticLawOf\ \ alpha\ :\ (S + Inf)/contact \quad \Rightarrow \quad kineticLawOf\ \ alpha\ :\ ((Exp * recover) - 2)/contact$$

Figure 4.13: Mutation (subtree) operator on a Bio-PEPA syntax GP kinetics tree

The point mutation operator can guarantee the tree size and structure will not change, whereas the subtree mutation has a low probability of retaining the original tree dynamics. Losing the original tree dynamics is not necessarily bad; it allows new areas of the search space to be explored.

To be considered well formed, the Bio-PEPA syntax enforces that kinetic action definitions may only include species identifiers if that species cooperates across the defined action, as

described in Section 2.2.2. If a kinetic action definition is flagged with no species segments, the permitted species identifiers for that kinetic action are constant and the Bio-PEPA syntactical requirement is easily adhered to.

In problems that flag at least one kinetic action and at least one species segment simultaneously, a new problem is presented during optimisation: the permitted species will be different from one individual to another. The permitted species from parents to offspring will also be different because of recombination and mutation operators. Ensuring syntactical correctness is handled by *kinetic rebreeding*.

4.3.10.4 *Kinetic Rebreeding*

The EPA framework employs *kinetic rebreeding* to ensure offspring solutions adhere to the Bio-PEPA syntax requirements. The rebreeding action was included in the EPA framework to help explore the far larger kinetics optimisation search space and to improve experiment results. The exact size of a search space is problem specific; an example of the size differences between species and kinetic optimisation is provided with the first kinetic optimisation experiment in Section 7.1.

The kinetic rebreeding action is multimodal and can be selected appropriate to the problem. Choosing *None* indicates the kinetic rebreeding actions are turned off: offspring individuals with syntax violations are permitted into the next generation. The *Replace* option is the basic rebreeding action: if a syntactic violation is detected, the individual is replaced by a new individual generated from scratch. Finally, the *Rebuild* option allows for a more precise detection technique: the syntactic violation is narrowed down to a specific subtree. The node above that subtree is then rebuilt using an operator similar to the subtree mutator operator (Section 4.3.10.3), thus removing the violating subtree.

Using the *Replace* or *Rebuild* kinetic rebreeding techniques may introduce new syntactic violations into the population. This is to be expected. These kinetic rebreeding actions are iteratively attempted three times to reduce syntactic violations. However, it is infeasible to always remove all violating individuals.

By detecting and resolving kinetic violations now, time and computational resources are saved later during the *Simulate* step. Furthermore, the newly created or updated individuals allow the EPA framework to explore new areas of the search space.

4.3.10.5 *Adaptive Depth Bias*

There is an additional aspect to consider for breeding operators on kinetic action definitions: the *depth* at which the operator is applied. The higher up the tree structure (or lower its depth) a chosen node exists, the larger the impact on the resultant offspring the operator will have. This is especially true for the subtree mutation operator; if a high level node is chosen, large portions of the tree will be rebuilt. In an extreme case the root node is selected: this causes the

entire tree to be rebuilt as if from scratch, thus losing any favourable traits which caused the parents to be selected.

When selecting a node for mutation or crossover, higher (or lower depth) selection leads to greater exploration whereas lower (or higher depth) selection tends towards exploitation. The EPA framework was extended for kinetic-specific GP work to include an adaptive depth bias. This is used to alternate between exploration and exploitation depending on which is most desirable at the time of selection.

The probability of a node being selected is $(node\_depth \ / \ tree\_depth) + adaptive\_offset$ where the tree root node is at depth 1. This gives a minimal (but not 0%) chance of choosing the root node and a 100% certainty of choosing the deepest terminal node. Disregarding the *adaptive offset*, this calculation will favour nodes deeper in the GP tree structure.

The adaptive offset is calculated based on a trailing window behind the current generation, the size of which is specified by the *adaptive window size* variable. If the overall fitness of the population has been improving, the standard deviation of generation-on-generation fitness in the trailing window will be high. However, if this deviation is low, the overall fitness of the population has plateaued. The adaptive offset is increased inversely to the standard deviation, bounded between 0 and 0.5; a probability greater than 1 is treated as 1. Therefore, a plateau will cause a spike of explorative behaviour, helping the EPA framework to avoid becoming stuck in a local optimum.

4.3.11 *Validity*

The entire offspring population is checked through the appropriate Process Algebra plug-in. This check will flag all individuals which contain errors that would prevent simulation. This is similar to the kinetic rebreeding action described above. The EPA framework will attempt to improve the validity of the offspring population by removing failed individuals. Each failure will be replaced by a newly created individual; this replacement process is iteratively attempted three times.

Similar to kinetic rebreeding actions (Section 4.3.10.4); by detecting and replacing individuals in this step, time and computational resources are saved later during the *Simulate* step. Furthermore, the newly created individuals are not bred from parent individuals; this gives the EPA framework an additional chance to explore the search space.

4.3.12 *Replacement*

Once breeding and validity checks are complete, the elites and offspring individuals must be introduced into the new population. The technique used for this introduction influences the balance between exploration and exploitation. There are different forms of replacement techniques, each with varying advantages and disadvantages.

The two techniques considered for use with the EPA framework are *steady-state* replacement and *generational* replacement. They both maintain a constant population size; this is a desirable

trait because it simplifies implementation and improves efficiency. Both techniques were introduced in Section 2.3.1.3.

The steady-state replacement technique breeds a small number of offspring individuals (as low as one) which will replace individuals in the current population. The replacement may be done arbitrarily or through a reverse-selection technique; i.e., the poorest fitness individuals are most likely to be selected for replacement. This approaches preserves the majority of the parent population each generation. This preservation allows for stability in subsequent generations but conversely makes any change harder to achieve because each new individual is competing with the majority of the previous generation.

The generational replacement technique discards the entire parent population. The new population comprises entirely of elites and offspring individuals bred during this step. This approach leans towards exploitation and is faster in finding optimal solutions compared to steady-state replacement. However, generational replacement is also more volatile because potentially strong solutions may be lost when a generation is discarded.

The EPA framework utilises a *generational* replacement technique. Once validity checks are complete, the entire parent population is discarded and replaced by bred offspring and elites. Due to the exploitative nature of generational replacement, this was desirable to balance the explorative influences of kinetic rebreeding and population validity checks.

With regards to Process Algebras, repeated scoring each generation reduces the risk of outliers or stochastic noise adversely influencing scoring. This challenges individuals and elites to maintain their good fitness generation-on-generation: if an individual is not a good fit, chances are low of it maintaining a fluke strong fitness.

(vi) FINISH

The *Simulate*, *Score*, *Select* and *Breed* steps continue their iterative cycle until a stopping criteria is met and the lifecycle moves into the *Finish* step. The most common stopping criteria is the number of generations to be completed. Certain care must be taken when selecting a number of generations; too few, and there will not be enough time to adequately explore the search space so finding an optimal solution is unlikely. Choosing too high and the optimal solution may be found but the lifecycle would continue, wasting time and computational resources. Estimating the correct number of generations for a particular problem can be difficult; it is better to have a number too high than too low.

Additional stopping criteria can be used. A fitness target or threshold can be defined that, once met or achieved, can bring the EPA lifecycle into the *Finish* step. As discussed in the *Score* step, a perfect fitness score of zero is an unobtainable goal, therefore a near-zero threshold is preferred.

Once the *Finish* step has been reached, the output information is finalised and result files are created. Output files include the most fit solution to the proposed problem, a trace produced

from that individual's simulation, a fitness score trace from generation 0 to see the progression of the EPA framework in solving the problem and finally a full list of all individuals in each generation to see which areas of the search space were explored.

If the EPA framework should be aborted for any reason, preliminary versions of all these files are created at the end of each lifecycle iteration. This allows some attempt at a solution even if a failure is encountered.

The preliminary and finalised output files are in basic file formats such as text (.txt) or comma separated value (.csv) files. This is so the EPA framework is compatible with HTCondor and other high-throughput or multi-node computation techniques. One function of the *EPA Utility Assistant* is to collate these files into more detailed output along with appropriate graphs. Full details of this tool are presented in Appendix A.

4.4 EVOLVING PROCESS ALGEBRA FRAMEWORK SUMMARY

This chapter has presented the implementation details of the Evolving Process Algebra (EPA) framework and introduces the *EPA Utility Assistant* which is fully detailed in Appendix A. This includes the implementation for the EPA meta-individual (the *CombinedIndividual*), the sub-individuals (*RatesIndividual*, *PopulationsIndividual*, *SpeciesIndividual* and the *KineticsIndividual*) and their associated syntax grammars and available breeding operators.

Additionally, the variable summary which is used as the default set for all coming experiments was presented in this chapter with justification for those variables discussed during the EPA lifecycle which occupies the majority of this chapter. The difficulties encountered during experimentation and the implemented solutions were also detailed as they are encountered in the EPA lifecycle.

5

PARAMETER MATCHING EXPERIMENTS

This chapter is the first of the experimental results chapters and details the rate value Genetic Algorithm (GA) parameter matching problems used to develop the Evolving Process Algebra (EPA) framework, as well as check the framework's functionality. All work presented in this chapter utilises the PEPA language as it was performed before the Bio-PEPA changeover for Genetic Programming (GP) optimisation work. This switchover was discussed previously in Section 2.2.3.

All Process Algebra models chosen for this research are complete descriptions of complex systems accompanied by time series data. By choosing complete models, experiments can be conducted by selectively removing aspects of the model. The Evolving Process Algebra (EPA) framework can then be challenged to rebuild the removed aspects using Evolutionary Computation (EC) techniques: in this chapter it is rate values which are removed to be rediscovered using GAs.

Evolved solutions can be checked qualitatively by comparing the individual to the original PA model and quantitively by comparing the evolved simulation trace to the original time series data and furthermore by comparing the evolved parameter sets to the known target parameter values.

It could be assumed the final result of an EPA framework execution would be a single value for each flagged rate which represents the singular best result. As such, it could be assumed that the final result of an EPA framework execution would be a single value for each flagged rate. However, this is an incorrect assumption. As mentioned in Section 2.3, EA techniques often find strong solutions which may not be the singular global optimum. Through multiple runs of the EPA framework, a range of values can be discovered, each which may provide a strong fitness.

Each singular best individual is taken from a different independent run of the EPA framework and collected into a histogram plot. If the spread of this graph for a particular rate is very wide or inconsistent, this demonstrates the Process Algebra model is not sensitive to the value of that rate. Conversely, if the spread for that rate is tightly clustered, this shows the model is particularly sensitive to the value of that rate. This allows qualitative analysis of the results which may give valuable insight to a domain expert by highlighting critical rates in the system.

Results for the Genetic Algorithm experiments in this chapter are presented with median and mean averages. These values are calculated by taking the singular best individuals from

each independent run then averaging those rates. Both the median and mean values are presented because both can be of use. If the spread of results follows a normal distribution, the mean is observed to provide a more accurate experimental output. However, in irregular or skewed distributions the median value is the more useful output as it is less affected by outliers.

Combining this knowledge with the histogram spreads; the mean value is the preferred result for loose clusterings, non-normal or low sensitivity variables, whereas the median value is the preferred result for sensitive, normal or highly clustered variables.

With the exception of the *repressilator* system (Section 5.4), all experiments in this chapter were performed using distance fitness functions, similar to the Euclidean distance fitness function as described in Section 4.3.4. This type of fitness function allows for an additional form of quantitative analysis. By discovering the maximum (worst) fitness score possible to a model, any evolved fitness scores can be calculated as a percentage. This provides insight into how well a solution performs on an objective scale: where 0% (or near-0%) is the best and 100% is the worst.

Every Process Algebra (PA) model described in this chapter has degrees of *organised* complexity and some measure of *disorganised* complexity; as discussed in Section 2.1. In PEPA models, the majority of the organised complexity exists in the species components and the system equation.

Specifically, the need for pools from which individuals can be recruited and to which they are eventually returned: this overcomes the inability of PEPA to create or destroy individuals. Complexity exists in the system equation because the recruitment pools must have lines of communication to the main components: i.e. the components which capture the described system's behaviour.

PEPA syntax includes a special rate $\top$ (or *infty* as ASCII for the plug-in tool), meaning infinity; a species defined with this rate is always available near instantaneously. Conversely, the special rate $\bot$ denotes a species which is rarely available.

In the plug-in tool, simulation cannot be accomplished using these special rates. Theoretically, their use could cause a model to enter a deadlocked state during simulation (specifically livelock, described with the first experiment in Section 5.1). This is because any action defined with rate $\top$ would dominate the system and prevents other actions from being performed. To prevent this situation, $\top$ becomes defined as a very large rate, such as $1,000,000.0$, instead of infinity. Similarly, $\bot$ is assigned a very small rate, such as 0.00001.

The remainder of this chapter is divided into sections which each describe a complex system, the full description of that system, the PEPA model and experiments derived from that system. Each experiment is presented with the model-specific parameter settings used to collect results. Unless specified otherwise, the default parameter values presented in Table 4.1 were used to obtain the presented results.

5.1 DINING PHILOSOPHERS

5.1.1 *Background & Model*

The *dining philosophers* problem is a resource allocation problem presented by Hoare [41]; it is commonly used to illustrate synchronisation issues such as *deadlock* and *livelock*, each discussed in turn later in this section.

The scenario posits five (but any number greater than one is acceptable) philosophers seated around a circular table. On the table is a bowl of spaghetti and a fork is placed between each pair of adjacent philosophers. The spaghetti is assumed to be in infinite supply. Each philosopher will alternate between thinking and eating. However, a philosopher can only start eating when in possession of both the forks to their immediate left and right. Each fork can only be held by one philosopher at a time, therefore a philosopher can only use a fork if their neighbour is not already using it. Once finished eating, a philosopher will put down both forks so others may use them. It is assumed that each philosopher does not know or care when other philosophers will eat or think.

To solve this problem, each philosopher must not starve; so philosophers should alternate infinitely between eating and thinking. This is achieved by managing the resources in the scenario: the forks. The spaghetti is not a limiting resource as it is assumed to be in infinite supply.

$$n = 5.62 \qquad h = 100 \qquad e = 1 \qquad r = 10 \qquad a = 10$$

$$\begin{aligned}
P_think &\stackrel{def}{=} (hungry, h).P_0 \\
P_0 &\stackrel{def}{=} (acquire, \top).P_1 \\
P_1 &\stackrel{def}{=} (acquire, \top).P_eat \;+\; (release, n).P_0 \\
P_eat &\stackrel{def}{=} (eating, e).P_r2 \\
P_r2 &\stackrel{def}{=} (release, r).P_r1 \\
P_r1 &\stackrel{def}{=} (release, r).P_think \\
\\
Fork_a &\stackrel{def}{=} (acquire, a).Fork_u \\
Fork_u &\stackrel{def}{=} (release, \top).Fork_a
\end{aligned}$$

$$(P\_think[20]) \underset{acquire,release}{\bowtie} (Fork\_a[15])$$

Figure 5.1: Dining philosophers model written in PEPA

Figure 5.1 describes the *dining philosophers* problem using PEPA taken from Benkirane's work [9]. The system initially contains 20 thinking philosophers (P_think) and 15 available forks ($Fork\_a$): values chosen to match Benkirane's experiment and also because the number of philosophers is greater than the number of required forks to eat, thus risking deadlock.

Note, the PEPA description of the *dinging philosophers* problem does not include a notion of adjacency between philosophers and associated forks; however, this does not impact the chance of deadlock.

At rate h, a thinking philosopher will become hungry and prepare for eating, entering a hungry state with no forks (P_o). The hungry philosopher will attempt to acquire an available fork (P_1) at rate a, this makes the fork unavailable ($Fork\_u$). Once a philosopher has one fork they will attempt to acquire a second available fork and begin eating (P_eat). A philosopher will finish eating at rate e and prepare to release both their forks (P_r2). The philosopher will then release their forks at rate r, first one fork (P_r1) then the other fork so they may return to a thinking state.

By following this transition of states, the system will inevitably end in a *deadlock* situation due to resource starvation. Deadlock is defined as a situation where two or more concurrent actions are waiting for the others to finish, therefore no actions can continue or finish. In the dining philosophers scenario, deadlock can occur when multiple philosophers each have one fork; each cannot release their fork until they finish eating, but cannot eat without another philosopher releasing a fork.

A solution to the deadlock problem could include a timer; after acquiring one fork the hungry philosopher would wait an arbitrarily defined time for a second fork, otherwise they would relinquish the one fork they have. This can cause a special case of resource starvation called *livelock*. Assuming all philosophers begin their thinking and eating cycle at the same time, they would all acquire one fork simultaneously and release it the arbitrary time period later. The cycle would then start over and continue infinitely. In livelock the state of the system is constantly changing, however no actions are progressing. A commonplace, real-world example of livelock involves two people meeting in a narrow corridor, each will politely move aside to let the other pass, thus resulting in both swaying side to side without progressing along the corridor.

The solution in the PEPA description of the dining philosophers system includes the special release rate n at which a philosopher with one fork (P_1) will relinquish it and return to a hungry (P_o) state thus making that fork available again ($Fork\_a$). The stochastic nature of Process Algebras will reduce the risk of the timed livelock scenario described above from occurring whilst also avoiding a deadlock scenario.

5.1.2 *Experiment* - n

The first experiment performed with the EPA framework was to rediscover the value for the special release rate n. When working with a single flagged variable, using an Evolutionary Computation optimisation technique is not the optimal approach. The search space is small enough that an empirical, iterative approach would suffice. However, this early stage of work

focused on the integration of Process Algebra models into the EPA framework and using their simulated time series trace as input for the GA fitness function.

Figure 5.2 displays an iterative sweep over the dining philosophers search space. Each point on the graph is the fitness score of a simulation for a value of the special release rate n in 0.01 incremental steps (0, 0.01, 0.02 ... 50). These simulations were each run 10,000 times to reduce stochastic noise when generating this search space sweep and then scored against the behaviour of maximising the number of eating philosophers.

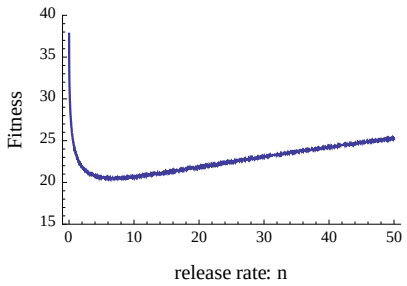


Figure 5.2: Trace of fitness for each value of n

Benkirane performed an ODE analysis to determine the global optimum of $n = 5.62$: this maximises the number of eating philosophers (P_eat) while avoiding deadlock. This global optimum is situated on a fitness plateau ranging from $n = 5.0$ and $n = 10.0$ of Figure 5.2. Any value of n in this plateau range is expected to obtain a particularly strong fitness score.

The fitness function used in this model is an unnormalised distance measure, similar to the Euclidean distance fitness measure discussed in Section 4.3.4. Normalisation is required to compare multiple time series traces with equal weighting. The dining philosophers model uses a single column target trace and does not require normalisation. The model contains 15 available forks, which equates to a maximum of 7 philosophers able to eat simultaneously while 1 fork remains available. If the system enters a deadlock or starvation state, 0 philosophers will get to eat; the worst possible difference at each time-step is 7 (maximum - minimum eating) squared, so 49. The fitness function uses a target time series trace with a length of 50; this equates to a maximum fitness value of 2,450. The iterative trace found that $n = 5.62$ scores a fitness of 20.535: in the percentage perspective the score is 0.83%.

Simulator Variables	
Start time	0
Stop time	49
Data points	50

EPA Override Variables	
Mutation rate	20%

Table 5.1: Dining philosophers (n) specific and EPA override parameter values

Table 5.1 illustrates the simulator parameters along with EPA framework overridden values. The standout override parameter is the *mutation rate*. With only a single rate flagged for

optimisation, recombination operators will have no effect. Instead, crossover will clone one parent solution; specifically parent 2 due to the EPA implementation. The rate of mutation is increased to compensate for the lack of exploration and exploitation influences normally provided by the recombination operator.

The EPA framework found a clustering of optimum solutions around the target of $n = 5.62$ but failed to match this exactly. This failure could be explained by the increased mutation rate skewing the balance towards exploration. The fitness scores discovered in the fitness plateau of the iterative sweep differed by less than 0.35 which is equivalent to 0.01% of the fitness score. Considering the stochastic nature of simulation combined with the insensitive nature of rate n these findings are acceptable.

The aim of this introductory experiment was not to match the value of rate n. Instead, the goal of integrating Process Algebras with the Evolutionary Computation techniques was attempted. This goal was achieved and the EPA framework was deemed compatible with applying Evolutionary Computation techniques to Process Algebra models.

5.2 INTERNET WORM

5.2.1 *Background & Model*

A *computer worm* is a class of standalone computer programs designed to exploit security weaknesses in operating systems with the intent of propagating themselves across a network. Similarly, an *Internet worm* aims to replicate itself across any computer connected via the Internet with the same goal of self propagation.

Unlike a virus, a worm does not need to execute itself inside an existing program to achieve this replication. Where viruses normally target computers for corruption or modification of files, a worm continues to target computers but to influence the network itself, even if only causing slowed operations by consuming bandwidth.

Many worms are designed only to duplicate themselves, but some include a *payload* designed to affect the host computer in some way. The majority of worm payloads have negative impacts on their host; the actual payload is worm specific. Some examples include erasing files from its host to disrupt computer operations, or hacking installed e-mail services to send spam messages and further propagate itself, or to install a backdoor entrance for easier access by the worm's creator.

Worms are considered malware because of the many negative variants in existence; however, some positive worms do exist. The *Nachi* family of worms exploit security vulnerabilities in operating systems in order to install patches to seal those same vulnerabilities. While good-intentioned, this modified the host computer without the owner's consent and also generated increased network traffic.

Figure 5.3 illustrates the Susceptible-Infected-Recovered (SIR) view of the Internet worm system as described in the paper by Bradley et al. [16]. This paper is co-authored by Hillston, the author of PEPA, and written for PEPA making it an ideal model for use with the EPA framework. Furthermore, the paper focuses on statistically analysing the performance of the proposed PEPA models; as such, these models are presented with rich data for comparison during fitness evaluation.

$$beta = 500 \qquad delta = 0.5 \qquad gamma = 0.1$$

$$\begin{aligned}
S &\stackrel{def}{=} (infect_S, \top).Inf \\
Inf &\stackrel{def}{=} (infect_Inf, beta).Inf + (infect_S, \top).Inf + (patch, gamma).R \\
R &\stackrel{def}{=} (recover, \perp).R \\
\\
Net_a &\stackrel{def}{=} (infect_Inf, \top).Net_u \\
Net_u &\stackrel{def}{=} (infect_S, beta).Net_a + (fail, delta).Net_a
\end{aligned}$$

$$(S[1000] \;||\; Inf[1]) \underset{infect\_S, infect\_Inf}{\bowtie} (Net\_a[200])$$

Figure 5.3: Internet Worm SIR model written in PEPA

The Internet worm system contain 1,001 computer nodes and is capable of supporting 200 concurrent and independent network connections between those nodes. 1,000 nodes start in a susceptible state (S) and only 1 starts in an infected state (Inf). A susceptible node can enter an infected state if a connection is available to an existing infected node. Once infected, the node can cooperate in infecting other susceptible nodes or can be patched at rate *gamma* to enter the recovered state (R).

The recovered state can be referred to as a removed state: once entered, the state cannot change. In the context of the Internet worm model, once a computer node has been patched it is no longer susceptible to reinfection. To satisfy the PEPA syntax, R is provided a definition; it will take the recover action at the slow rate $\perp$. The recover action requires no cooperation to complete and will not impact any other part of the model.

The Internet worm model follows a *frequency-based* spread of infection [8]. The EPA framework is later applied to systems which model the *density-based* transmission during Bio-PEPA work, see Section 7.2.

The connections between the computer nodes are modelled via the net resource pools in two steps. The first step secures an available connection (Net_a) for an infected node. Once an available connection has been obtained, the connection enters an unavailable state (Net_u). The obtained network connection attempts to infect a susceptible node at rate *beta*; this action can fail at rate *delta* either due to network contention or because of a lack of susceptible nodes to infect. Whether an infection attempt is successful or not is unimportant, the network connection is made available once the action is complete.

Bradley et al. noted that large scale worm infections tend not to waste time or resources by keeping track of which connected hosts are already infected. It is assumed that some attempts at infection will be made against already infected hosts: when this happens the infected host is unaffected.

5.2.2 *Experiment* - all rates

The Internet worms example was not the first Process Algebra model utilised by the EPA framework. However, it was the first system used with the primary objective of optimising flagged rates instead of testing the integration of PEPA optimisation code.

The experiment flags all variable rates for optimisation. These three rates are the infectivity rate *beta*, the infect timeout rate *delta* and the patch recovery rate *gamma*. The values of these three rates are known and presented in the Bradley et al. paper.

Simulator Variables	
Start time	0
Stop time	29
Data points	30

EPA Override Variables	
Independent runs	75

Table 5.2: Internet worm SIR specific and EPA override parameter values

A three variable search problem could have the values discovered through empirical, iterative sweeps. This number of variables is on the lower end of the scale for application of a Genetic Algorithm and presents a good starting point for the EPA framework. Furthermore, the Internet worm model has no special considerations and thus uses the default EPA framework parameters. Table 5.2 presents the simulator variables used for this experiment. Additionally, the increased number of *HTCondor* nodes available allowed for an increase in the *Independent runs* variable.

Variable	Target Value	Evolved Median	Evolved Mean	Std. Dev (σ)
beta (β)	500	548.65	578.01	103.89
delta (δ)	0.5	0.81	2.00	2.59
gamma (γ)	0.10	0.10	0.10	0.01

Table 5.3: Internet worm SIR results

The table of results for this experiment is illustrated in Table 5.3. One row per flagged rate which includes the target parameter value along with evolved median, evolved mean and associated standard deviation. As mentioned above, the target values are taken from the Bradley et al. paper, in addition these values were used to generate the target trace for the fitness function.

The results are presented in histogram form in Figure 5.4. Considering the asymmetrical shape and clustering of each of these histograms, it can be seen that the sensitivity of each of these three variable rates is very high. For this reason, the median values from Table 5.3 are considered the preferred results. It can be seen that the median values more closely match the target values than the mean values. This is particularly true for the infect timeout rate *delta*, where the mean result is increased due to the heavy right-side skew.

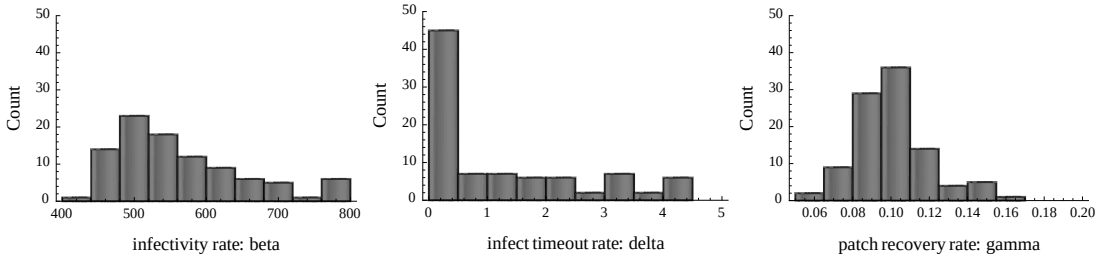


Figure 5.4: Histogram plots for Internet worm SIR: *beta*, *delta* and *gamma*

Considering the infectivity rate *beta*, the target value is 500 and the evolved median value is 548.65. The infect timeout rate *delta* is the weakest solution of the three evolved in this experiment; a target value of 0.5 and an evolved median value of 0.81. Conversely, the patch recovery rate *gamma* has a particularly strong solution; the target value is 0.1 and the evolved median (and mean) value is 0.1: a slight variation exists after the fifth decimal place.

The *delta* rate is responsible for causing an infect attempt to timeout and relinquish the network connection for future infectivity attempts. The insensitivity of the model to this rate could be caused by the available bandwidth. 200 nodes are available, as modelled by *Net_a*, for infected nodes to make connections with. With so many available nodes, the number of infection attempts and subsequent infections can be made early. If this number were reduced, the timeout rate would become more critical to achieving the target number of infections.

5.3 HIV

5.3.1 *Background & Model*

HIV (*Human Immunodeficiency Virus*) is a retrovirus of the lentivirus genus: a very slow replicating retrovirus. The virus is transmitted via contaminated bodily fluids, particularly by unprotected sexual intercourse or through the use of contaminated hypodermic needles. The HIV infection attacks the immune system directly, particularly vital immune cells such as *helper T cells* (specifically $CD4^+$ T cells), *macrophages* and *dendritic* cells. As the number of $CD4^+$ T cells declines, the host will lose their cell-mediated immunity: the immune response that does not involve antibodies. In time, HIV leads to the AIDS (*Acquired Immunodeficiency Syndrome*) disease.

Early stage HIV infection (also called *acute infection*) often presents with nonspecific influenza-like symptoms; such as fever, enlarged and tender lymph nodes, throat inflammation, rash and/or headaches. However, many hosts experience little to no symptoms in the early stages. Due to the nonspecific characteristics, HIV infection is often misdiagnosed as a more common infectious disease with overlapping symptoms: most frequently the common cold or influenza.

No matter the host's symptoms in stage one, the disease will progress into a stage of *clinical latency*. This period can last many years; between three and twenty, depending on treatment levels, with the typical duration being eight years. During this time the attack on the host's immune system continues to lower the $CD4^+$ T cells. At this stage the viral load of HIV cells in the host is detectable by blood tests.

A host is considered to have AIDS once the concentration of $CD4^+$ T cells in their blood falls below 200 cells per μL. While dangerous in itself, AIDS infected hosts are at major risk to opportunistic infections; infections which take hold because of the host's impaired immune responses. These may include bacterial, viral, fungal or parasitic infections, in addition to a risk of cancerous tumours developing.

HIV (and consequently AIDS) is considered a global pandemic with two distinct strains having been identified: HIV-1 and HIV-2. HIV-1 is global and more virulent than the HIV-2 strain which is less contagious and endemic to Africa. The pandemic status of the infection coupled with the difficulty in correct diagnosis during stage one infection has increased the interest in modelling HIV spread.

The HIV model was selected for use with the EPA framework as it is more complex than previous experiments in this chapter and presents a greater challenge for the framework. The HIV Susceptible-Exposed-Infected (SEI) set up also provides a different view of epidemiology models through lack of a recovered state. Treatments for HIV are focused on stemming the spread and alleviating the symptoms rather than curing the host. For this reason, there is no recovery and the model is SEI only. Figure 5.5 describes the spread of HIV through the United Kingdom. The model can be evaluated against publicly-available data on HIV [39], though population levels are scaled down for tractability. This example is adapted from the Weighted Synchronous Calculus of Communicating Systems (WSCCS) model presented by McCaig et al. [56] translated into a SEI PEPA model.

Classic to epidemiology models, individuals in the system can be susceptible, exposed to or infected by HIV (*S*, *Exp* and *Inf* states respectively). A susceptible individual can become exposed in two ways; by direct contact with an infected individual or by a contact external to the modelled components. Exposed individuals can contribute to the population by producing infected individuals at rate *pInfect* or susceptible individuals at that rate's inverse ($1 - pInfect$). Once an individual is infected it will continue to infect other exposed individuals in the system. As previously stated, individuals cannot enter a recovered state.

Individuals are able to enter and leave the system by birth and natural death at rates *pBirth* and *pDeath* respectively. Infection induced deaths are not included as the time period of the data covers the introduction of treatments which greatly reduced deaths due to HIV/AIDS induced complications. Individuals may also enter the system by immigration at rate *pImmigrate*.

$$pContact = 1 \qquad pImmigrate = 0.1 \qquad pInfect = 0.138$$
$$pBirth = 0.0119 \qquad pDeath = 0.01018$$

$$\begin{aligned}
S &\stackrel{def}{=} (contact, \top).Exp + (external_exposure, \bot).Exp \\
Exp &\stackrel{def}{=} (infect, pInfect).Inf_birth + (no_infect, 1 - pInfect).S_birth \\
Inf &\stackrel{def}{=} (contact, \top).Inf_birth \\
\\
S_birth &\stackrel{def}{=} (birth, pBirth).S_death + (no_birth, 1 - pBirth).S_death \\
Inf_birth &\stackrel{def}{=} (birth, pBirth).Inf_death + (no_birth, 1 - pBirth).Inf_death \\
S_death &\stackrel{def}{=} (death, pDeath).Pool + (no_death, 1 - pDeath).Sf \\
Inf_death &\stackrel{def}{=} (death_Inf, pDeath).Pool + (no_death, 1 - pDeath).Inff \\
\\
Resource &\stackrel{def}{=} (get_res, \top).Resource_NA \\
Resource_NA &\stackrel{def}{=} (regenerate, \top).Resource \\
\\
Sf &\stackrel{def}{=} (get_res, \top).S + (no_res_S, 1).Pool \\
Inff &\stackrel{def}{=} (get_res, \top).Inf + (no_res_Inf, 1).Pool \\
\\
Transmitter &\stackrel{def}{=} (contact, pContact).Transmitter + (death_Inf, \top).Pool2 + (no_res_Inf, \top).Pool2 \\
Dormant &\stackrel{def}{=} (infect, \top).Transmitter + (death, \top).Pool2 + (no_res_S, \top).Pool2 \\
Pool &\stackrel{def}{=} (birth_bis, \top).Sf \\
Pool2 &\stackrel{def}{=} (birth, \top).Pool22 + (immigration, pImmigrate).Pool22 \\
Pool22 &\stackrel{def}{=} (birth_bis, \top).Dormant
\end{aligned}$$

$$\begin{array}{c}
((Sf[5829] \;||\; Inff[2] \;||\; Pool[100]) \underset{get_res}{\bowtie} (Resource[100])) \\
\underset{no_res_S,no_res_Inf,infect,contact,birth,birth_bis,death,death_Inf}{\bowtie} \\
(Dormant[5829] \;||\; Transmitter[2] \;||\; Pool2[100])
\end{array}$$

Figure 5.5: UK HIV SEI model written in PEPA

The *Resource* and *Resource_NA* components are used to limit population growth. As previously discussed in Section 2.2.3, PEPA cannot have individuals being created or destroyed. To model movements in and out of the system, multiple recruitment pools are used to mirror the core components: *Sf*, *Inff*, *Transmitter*, *Dormant*, *Pool*, *Pool2* and *Pool22*.

The simulation time of PEPA models is the main bottleneck of the EPA framework. For tractability, the population sizes used during simulations were scaled down by 10,000 from the census values [39] to those detailed in the system equation of Figure 5.5. Note: the

experiment performed with this model was done before the EPA framework was extended to include ODEs.

5.3.2 *Experiment* - pInfect & pImmigrate

This system has a large jump in ordered complexity, as described in Section 2.1, therefore the number of rates to be flagged was reduced to two; the infectivity rate *pInfect* and the immigration rate *pImmigrate*. Also, rates such as *pBirth* and *pDeath* can be taken from the available census data.

By using an iterative sweep, a three-dimensional fitness landscape can be created to discover the target values for these two rates. This fitness landscape is presented in Figure 5.6; the x-axis *pInfect* ranges 0 - 0.5, the y-axis *pImmigrate* ranges 0 - 1 and the z-axis illustrates the fitness score for the simulation of the corresponding rate values. Each simulation was run with 100 internal replications to reduce stochastic noise during the iterative sweep. The *pInfect* rate range is smaller; by utilising domain knowledge it is known that values >0.5 cause massive HIV outbreaks in the population. This would be unrealistic and result in particular poor fitness scores, so is omitted from the search space.

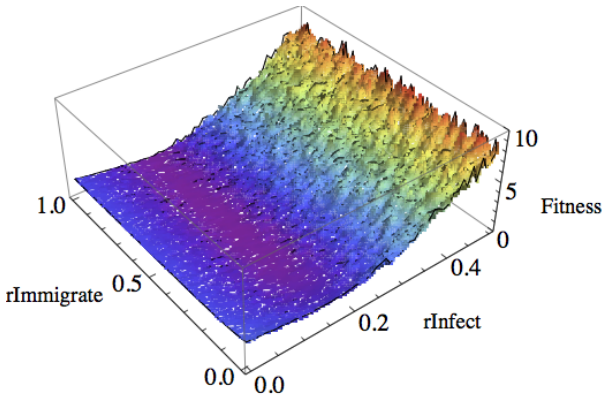


Figure 5.6: Trace of fitness for each value of *pInfect* and *pImmigrate*

Using this iterative sweep, the global optimum value can be found at $pInfect = 0.12$ & $pImmigrate = 0.48$ with a 4.23% objective fitness. Considering the shape of the fitness landscape, the system is more sensitive to the *pInfect* setting as indicated by the long valley along the y-axis.

McCaig et al. calculated *pInfect* to be 0.138 which differs from the 0.12 discovered during the iterative sweep described above. The PEPA model includes an immigration term to avoid deadlock during simulation. This *pImmigrate* term did not appear in the McCaig et al. work so its inclusion in the PEPA model will have affected the *pInfect* value.

The marked increased in ordered complexity requires additional time for simulation as well as keeping the number of time-steps to a minimum. The simulator values for the HIV system

are illustrated in Table 5.4. Beyond the time considerations, there are no additional aspects that affect the model parameters.

System Variables	
Start time	0
Stop time	10
Data points	11

Table 5.4: HIV SEI specific parameter values

Table 5.5 illustrates the results for the two variable rates being optimised along with the two targets ascertained previously using the iterative sweep of the landscape. Even before considering the sensitivity analysis, the results show excellent matches to the targets for both evolved mean and median values.

Variable	Target Value	Evolved Median	Evolved Mean	Std. Dev (σ)
pInfect	0.12	0.116	0.115	0.005
pImmigrate	0.48	0.485	0.487	0.050

Table 5.5: HIV SEI results

The histogram plots for these results are illustrated in Figure 5.7. Despite the range permitted for each value, both show highly clustered results, particularly in the peak for *pInfect*.

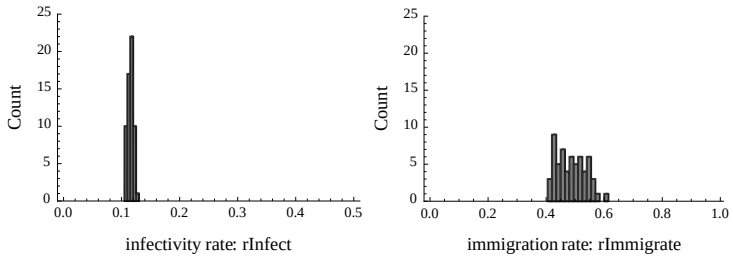


Figure 5.7: Histogram plots for HIV SEI: *pInfect* and *pImmigrate*

If both evolved results were considered to the second decimal place (like the target values), the differences become zero and negligible respectively. In either representation, evolved means and median average values for both rates are close to their targets and very acceptable.

5.4 REPRESSILATOR

5.4.1 *Background & Model*

The repressilator network is a genetic regulatory circuit which produces cyclic, oscillatory behaviour [12]. The repressilator is a stochastic and noisy system. It is a negative feedback

loop consisting of three repressor genes in *Escherichia coli* (*E. coli*) originally proposed by Elowitz and Leibler [30]. The first repressor gene causes transcription of the *LacI* protein which, in turn, inhibits the second gene responsible for transcription of the *TetR* protein. This protein inhibits the third gene responsible for transcription of λ *cI* which, completing the cycle, inhibits the first gene.

The PEPA model of the repressilator network employs a more generic approach. Figure 5.8 illustrates the network, note the *Gene_AB*, *Gene_BC* and *Gene_CA* components. These can be expressed as *Gene_XY*; meaning this gene is inhibited by protein *X* and, when expressed, transcribes protein *Y*. The inhibition is based on rate *con_X* and the transcription is at rate *rep_Y*. The rates and model structure are discussed later in this section.

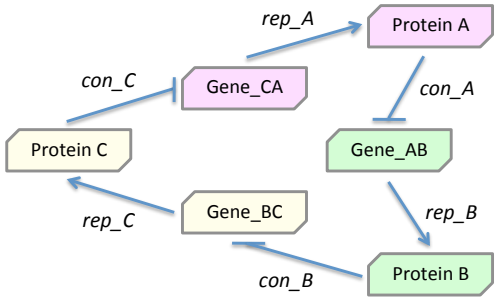


Figure 5.8: The repressilator synthetic genetic regulatory network

To satisfy the PEPA requirements for mirrored components, communication channels and recruitment pools, the repressilator model presented in Figure 5.9 contains more than the *Gene_XY* and *Protein_Y* components. The repressilator network is made up of three sections with identical behaviour.

When expressed, *Gene_XY* cooperates in action *rep_Y* to replicate *Protein_Y* at rate t from the recruitment pool (*Pool*). Once replicated (i.e. transcribed), *Protein_Y* will decay at rate d or produce the connecting component *Connect_Y* at rate b. The *Connect_Y* is consumed to inhibit the next gene in the cycle; for example, *Connect_A* is consumed to inhibit *Gene_AB* by moving it into the *Pause_AB* state. The inhibition will take precedence over the transcription because replication of *Gene_XY* occurs at rate t and inhibition at the faster rate $\top$. On consumption, the *Connect_Y* components are returned to the *Connect* recruitment pool. Once a gene has become inhibited, it will return to an uninhibited state at rate u.

The system equation of the PEPA models defines the sizes of the recruitment pools *Connect* and *Pool* as 300 and 100 respectively. Components *Gene_AB*, *Gene_BC* and *Gene_CA* are present in the system equation to define their communication channels but have no starting values, as such their values start at one. Other components, not defined in the system equation, also have a starting value of one.

When simulation commences, the repressilator network has no proteins present and no inhibited genes. All three genes will become active together and commence transcribing their

$$t = 100 \qquad d = 4.52576 \qquad u = 0.87173 \qquad b = 655.40285$$

$$\begin{aligned}
Gene_AB &\stackrel{def}{=} (rep_B, t).Gene_AB + (con_A, \top).Pause_AB \\
Pause_AB &\stackrel{def}{=} (delay, u).Gene_AB \\
Gene_BC &\stackrel{def}{=} (rep_C, t).Gene_BC + (con_B, \top).Pause_BC \\
Pause_BC &\stackrel{def}{=} (delay, u).Gene_BC \\
Gene_CA &\stackrel{def}{=} (rep_A, t).Gene_CA + (con_C, \top).Pause_CA \\
Pause_CA &\stackrel{def}{=} (delay, u).Gene_CA \\
\\
Protein_A &\stackrel{def}{=} (pro_A, b).Protein_A + (decay, d).Pool \\
Protein_B &\stackrel{def}{=} (pro_B, b).Protein_B + (decay, d).Pool \\
Protein_C &\stackrel{def}{=} (pro_C, b).Protein_C + (decay, d).Pool \\
\\
Connect_A &\stackrel{def}{=} (con_A, \top).Connect \\
Connect_B &\stackrel{def}{=} (con_B, \top).Connect \\
Connect_C &\stackrel{def}{=} (con_C, \top).Connect \\
Connect &\stackrel{def}{=} (pro_A, \top).Connect_A + (pro_B, \top).Connect_B + (pro_C, \top).Connect_C \\
Pool &\stackrel{def}{=} (rep_A, \top).Protein_A + (rep_B, \top).Protein_B + (rep_C, \top).Protein_C
\end{aligned}$$

$$\begin{array}{c}
((Pool[300]) \underset{rep_A,rep_B,rep_C}{\bowtie} (Gene_AB \parallel Gene_BC \parallel Gene_CA) \\
\underset{pro_A,pro_B,pro_C,con_A,con_B,con_C}{\bowtie} (Connect[100])
\end{array}$$

Figure 5.9: Repressilator system model written in PEPA

associated proteins. It is stochastically determined which protein will be transcribed first and in which order the genes will become active from that point forwards. The first gene to be transcribed is discounted when determining the transcription order. During the race to be the first active gene, that gene may not be fully expressed before becoming inhibited. The second, third and fourth active genes determine the order of activation.

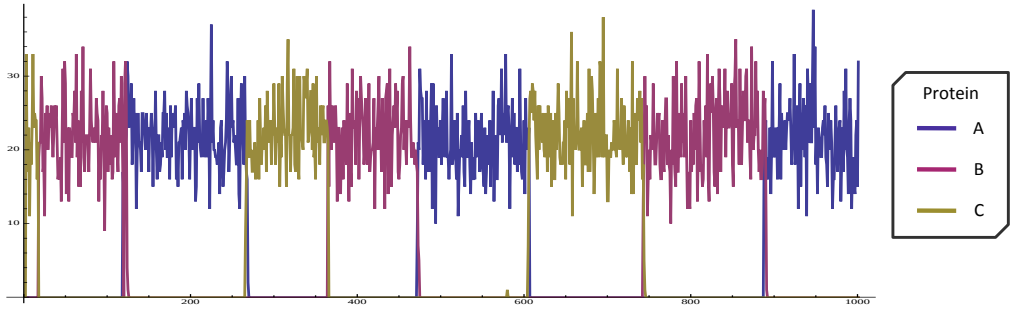


Figure 5.10: The protein expression trace of the repressilator system

Figure 5.10 is an example trace produced from the repressilator system. Consider, if simulation causes the protein transcription order to be *ABAC*, discounting the first protein, the order will be *BACBACBAC*, etc. until simulation ends. The expression of the second protein

onwards also have near-identical periodicity as the genes cycle, this periodicity is not set and may stochastically differ between simulations.

This noisy, cyclic and stochastic behaviour of the repressilator makes it a challenging system for parameter matching with the EPA framework. Models presented earlier in this chapter used a standard Euclidean distance measure as a fitness score, see Section 4.3.4. This method of scoring fitness could not be applied to the repressilator simulation trace. Due to the stochastic differences between simulations, before simulation commences the protein transcription order and periodicity of the inhibition cycle are unknown. Without a preprepared trace to match against, the Euclidean distance fitness function cannot be utilised.

The normal practice to reduce stochastic noise from a simulation trace is to run the simulation multiple times with the same GA derived variable set and average the trace (*internal replications*, Section 4.3.2.1). However, stochastic transcription order and periodicity would cause a mean trace to not be indicative of the repressilator behaviour. With more runs being averaged, the behaviour tends towards all proteins maintaining a similar level over the entire time series trace. This would hide the behaviours needed to score fitness correctly. A custom fitness function would be required that dynamically determines transcription order and periodicity.

In addition to creating a custom fitness function, the EPA framework had to be extended to reduce stochastic noise without the use of internal replications. During work with the repressilator network the EPA incorporated *external replications* for this purpose, see Section 4.3.2.2 for full details on their function and role.

As described above, there cannot be a preprepared target trace associated with the repressilator experiment. Figure 5.11 illustrates the pseudo-code of the repressilator fitness function.

```
1. Duration to discard d = proteinOneDuration()
2. Fitness fit = timeSeriesLength - d
3. for: Each time-series datapoint p after d
4.         if AdhereExpression(p) and AdherePeriodicity(p)
5.                 Decrement fit by 1
6. return fit
```

Figure 5.11: Pseudo-code for the repressilator fitness function

An evolved output trace is still produced which is analysed to match particular behaviour traits; the more of these traits which are satisfied, the stronger the fitness score. The repressilator fitness function makes a scan of the output time series trace until the first four gene expressions are completed.

The first expressed gene is disregarded because it is only a partial expression at the start of simulation and does not affect the final gene expression order. The duration of each gene expression is checked and should be equivalent to each other, this determines the period

at which each subsequent gene should be expressed. The *AdhereExpression* function ensures that the protein expressions follows the correct cyclic order. The *AdherePeriodicity* function checks that the protein expression duration remains consistent. The fitness score is based on the adherence to the gene expression order and the duration of each expression.

5.4.2 *Experiment* - all rates

The repressilator system contains four variable rates, all of which are flagged for optimisation in this experiment. The protein transcribing rate t, the protein decay rate d, the gene un-inhibition rate u and the connection binding rate b. The focus of this experiment was creating a model-specific fitness function and integrating it into the EPA framework.

Custom built, model-specific fitness functions are common when working with Evolutionary Computation techniques, as discussed later in Section 9.2.

Table 5.6 presents the simulator and EPA override parameter settings. The repressilator system presented a further challenge to the EPA framework due to its cyclic behaviour. The framework was extended during this experiment to include the *external replication* as detailed in Section 4.3.2.2. It would not be appropriate for this system to utilise internal and external replications simultaneously, therefore the former is reduced to one.

Simulator Variables	
Start	0
Stop	999
Data points	1000

EPA Override Variables	
Fitness	Repressilator
Internal replications	1
External replications	10
Independent runs	60

Table 5.6: Repressilator specific and EPA override parameter values

Table 5.7 presents the results of the repressilator experiment. At first glance, both the evolved mean and median results are notably different from their target values. The histogram distributions are provided later in Figure 5.12.

Variable	Target Value	Evolved Median	Evolved Mean	Std. Dev (σ)
t	100	542.453	548.883	273.611
d	4.526	26.560	25.931	15.682
u	0.872	0.077	0.449	1.530
b	655.4	1080.001	1036.268	276.592

Table 5.7: Repressilator results

With the poor evolved solutions, it would be expected for all four histogram spreads to be inconsistent and loosely distributed. This would indicate the EPA framework had failed to find an objectively strong fitting individual, for whatever reason. However, notice the clustered

peak for the gene un-inhibition rate u; across a range of 0 - 10, all but four best individuals clustered in the 0 - 1 range.

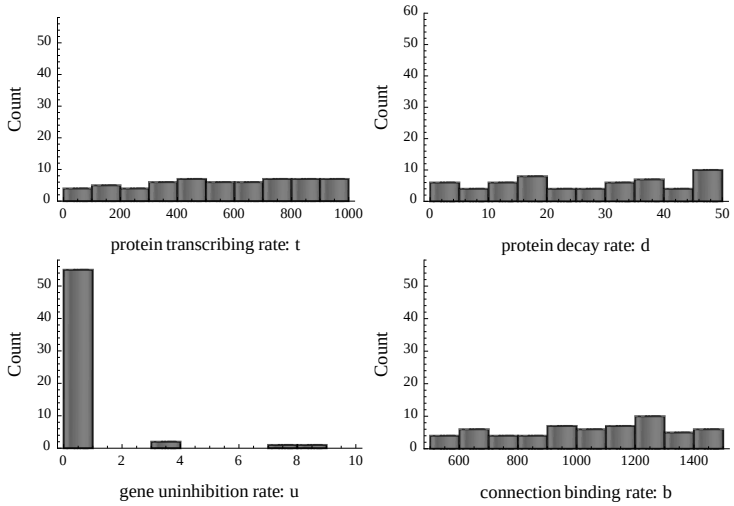


Figure 5.12: Histogram plots for repressilator: t, d, u and b

On closer inspection of the EPA framework and the repressilator fitness function class (*RepressilatorEvaluator*), discrepancies were found between the specified target behaviour of the repressilator system and the implemented target behaviour. A check was placed into the fitness function to ensure that the evolved solution would not cause a constant 0-value trace for all three expressed proteins; such an output could be caused by a poor combination of evolved rates and would be scored more poorly than normal. However, this caused an unexpected outcome in the direction of evolution.

The expected output of the repressilator system is a protein expression with a level of approximately 20. The evolved protein expression levels hovered at the 90 mark, clearly above the normal. However, this did not violate the behaviour requirements of the repressilator fitness function: a minimum protein expression level was defined to ensure the non-0 trace, but there was no maximum expression level.

As previously mentioned, an Evolutionary Computation technique will take any shortcuts which are made available to achieve a strong fitness score. As such, the evolved individuals had high protein expression levels with consistent ordering and good periodicity; the two desired behaviours and scored appropriately well.

The aspect of integrating custom fitness functions into the EPA framework during this experiment was a success. While the evolved values did not match the expected target, strong solutions were found which met the target behavioural criteria. Additionally, insight was given into the sensitive nature of the fitness function in getting the desired output.

A more accurate fitness function could be developed for the repressilator system, however the focus of this research is on the application of GP optimisation techniques directly to the Process Algebra model structure. Through combination of the EPA framework variables, the

runtime of a PA simulation could range from hours to days. Typically, the runtime of a complex simulation such as the Repressilator model aimed to be approximately 18 hours: overnight plus extra time while other work was conducted. Implementation and fine-tuning an upper cap on the protein expression could have become particularly time consuming because of this simulation time. Given the time requirements, work moved on towards this overarching goal.

5.5 G-PROTEIN (PEPA)

5.5.1 *Background & Model*

G-proteins, also known as *guanine nucleotide-binding proteins*, are binding proteins through which extracellular signalling molecules are transduced to intracellular signals as controlled by G-protein-coupled receptors. Many different stimuli can activate these signals; including biogenic amines, chemokines, hormones, ligands, lipids, light, neurotransmitters, nucleotides, and odorants [46].

G-proteins comprise of three subunits: *alpha* (α), *beta* (β) and *gamma* (γ), of which the β and γ can form the $\beta\gamma$ complex. The α subunit can be bound to a GDP (*guanosine diphosphate*) or GTP (*guanosine triphosphate*) molecule; the subunit is 'on' when bound to GTP and 'off' when bound to GDP. When activated by a ligand, a conformational change is triggered in the α subunit. This subunit will exchange the 'off' GDP for the 'on' GTP as well as dissociate from the $\beta\gamma$ complex. The α subunit will act on the target proteins inside the cell, after which the bound GTP will be hydrolysed to GDP before binding to the $\beta\gamma$ complex again. At this stage the cycle is complete and the receptor is ready for new stimuli [13, 46].

Bao et al. studies different modelling methods for describing the G-protein system [5]. Using their notation, seven components are required to describe the system. The receptor (R), the ligand (L), the bound receptor-ligand complex (RL), the deactivated α subunit which is bound to GDP (Gd), the $\beta\gamma$ complex (Gbg), the active α subunit which is bound to the $\beta\gamma$ complex (G) and finally the active α subunit which is bound to GTP (Ga).

Modelling methodologies used by Bao et al. include ODEs and Bio-PEPA; the latter being used by the EPA framework during Genetic Programming species optimisation, see Section 6.4. A PEPA description was developed from their models and is illustrated in Figure 5.13. The seven identified components can be found in the PEPA description, along with eighteen additional components required to encapsulate the G-protein system behaviour whilst still adhering to PEPA's requirement for mirrored components and recruitment pools.

Seven rates govern the PEPA model. Rate *rl* controls the binding of receptor and ligand components to form receptor-ligand complexes. Once bound, receptor-ligand complexes may dissociate at rate *rlm* or degrade at rate *rd1*. Receptor components alone may also degrade at rate *rdo*. Rate *ga* controls the disassociation of active α and the $\beta\gamma$ complex. Once dissociated,

the active α hydrolyses GTP to GDP and becomes inactive at rate $gd1$. The final rate $g1$ controls the association of α with the $\beta\gamma$ complex.

$$rl = 0.664 \qquad rlm = 0.002 \qquad rdo = 0.004 \qquad rd1 = 0.004$$
$$ga = 0.4 \qquad gd1 = 1.1 \qquad g1 = 10$$

$$\begin{aligned}
R &\stackrel{def}{=} (bind_L, rl).RL_o + (deg_R, rdo).Pool \\
RL_o &\stackrel{def}{=} (red_R, \top).RL \\
RL &\stackrel{def}{=} (split_RL, rlm).Make_R + (deg_RL, rd1).Pool + (activate, ga).RL \\
Make_R &\stackrel{def}{=} (gen_R, \top).L \\
L &\stackrel{def}{=} (bind_L\text{-}R, rl).Pool \\
Gd &\stackrel{def}{=} (bind_bg, g1).Gd_o \\
Gd_o &\stackrel{def}{=} (red_Gd, \top).G \\
G &\stackrel{def}{=} (transform_G, \top).Ga \\
Ga &\stackrel{def}{=} (deg_Ga, gd1).Gd \\
Gbg &\stackrel{def}{=} (bind_Gd\text{-}Gbg, \top).Pool
\end{aligned}$$

$$\begin{aligned}
Mirror_L &\stackrel{def}{=} (bind_L, rl).Connect_RL \\
Connect_RL &\stackrel{def}{=} (bind_L\text{-}RL, rl).Mirror_RL \\
Mirror_R &\stackrel{def}{=} (red_R, \top).Pool2 + (deg_R, rdo).Pool2 \\
Mirror_RL &\stackrel{def}{=} (red_RL\text{-}R, \top).Mirror_R + (deg_RL, rd1).Pool2 \\
Mirror_G &\stackrel{def}{=} (activate, ga).Connect_RL\text{-}G \\
Connect_RL\text{-}G &\stackrel{def}{=} (diss, \top).Make_Gbg \\
Make_Gbg &\stackrel{def}{=} (act_Gbg, ga).Mirror_Ga \\
Mirror_Ga &\stackrel{def}{=} (deg_Ga, gd1).Mirror_Gd \\
Mirror_Gd &\stackrel{def}{=} (red_Gd, \top).Mirror_G \\
Mirror_Gbg &\stackrel{def}{=} (bind_bg, g1).Connect_Gd \\
Connect_Gd &\stackrel{def}{=} (bind_Gd\text{-}Gbg, \top).Pool2 \\
Pool &\stackrel{def}{=} (gen_R, \top).R + (diss, \top).Pool1 \\
Pool1 &\stackrel{def}{=} (transform_G, \top).Gbg \\
Pool2 &\stackrel{def}{=} (split_RL, \top).Pool22 + (act_Gbg, \top).Mirror_Gbg \\
Pool22 &\stackrel{def}{=} (red_RL\text{-}R, \top).Mirror_L
\end{aligned}$$

$$\begin{aligned}
&((L[6022] \,||\, R[100] \,||\, G[70] \,||\, Gd[30] \,||\, Gbg[30]) \underset{gen_R,transform_G}{\bowtie} (Pool[150]) \\
&\underset{deg_R,bind_L,bind_L\text{-}R,red_R,deg_RL,split_RL,activate,diss,bind_bg,bind_Gd\text{-}Gbg,red_Gd,deg_Ga}{\bowtie} \\
&((Mirror_L[6022] \,||\, Mirror_R[100] \,||\, Mirror_G[70] \,||\, Mirror_Gd[30] \,||\, Mirror_Gbg[30]) \underset{act_Gbg,red_Rl\text{-}R}{\bowtie} (Pool2[150])))
\end{aligned}$$

Figure 5.13: G-protein system model written in PEPA

Due to the more succinct nature of Bio-PEPA, the same G-protein system is more elegantly expressed in Section 6.4. Also in that section, Figure 6.7 illustrates the system in the simpler Bio-PEPA representation.

Many pharmaceuticals are targeted at the G-protein-coupled receptors, making modelling their mechanisms important. The G-protein system was chosen for use with the EPA framework as it is larger and more complex than previous systems used, particularly in organised complexity (Section 2.1). Additionally, the G-protein system offers up to seven rates for GA optimisation.

More importantly, having the Bio-PEPA and ODE descriptions of the system allowed for forward preparation. Familiarity gained while working with the PEPA description during GA optimisation would prove useful when applying GP techniques in later work. Having the same model in different forms would also allow testing of the EPA framework by verifying results are consistent across those forms.

The G-protein system was the final PEPA model used to develop the EPA framework. This was the final GA parameter matching experiment performed. As described above, the G-protein system comprises of two behavioural cycles, one of which has an excitatory effect on the other: this adds an additional layer of ordered complexity to the system and poses a suitably challenging problem to the EPA framework.

5.5.2 *Experiment* - all rates

All seven of the variable rates in the G-protein system are flagged in this experiment. Four rates dictate the behaviour of the cycle outside the modelled cell: the receptor-ligand binding rate *rl*, the receptor-ligand dissociation rate *rlm*, the receptor-ligand decay rate *rd1* and the receptor decay rate *rdo*.

The remaining three rates dictate the behaviour of the cycle inside the cell: the α $\beta\gamma$ disassociation rate *ga*, the α hydrolysis rate *gd1* and the α $\beta\gamma$ association rate *g1*. Table 5.8 presents the simulator parameters for the G-protein PEPA system.

Simulator Variables	
Start	0
Stop	20
Data points	80

Table 5.8: G-protein (PEPA) specific parameter values

The results for this G-protein experiment are presented in Table 5.9. The results are split into the outside cycle rates (*rl*, *rlm*, *rdo* and *rd1*) above the dividing line and the inside cycle rates (*ga*, *gd1* and *g1*) underneath.

At first glance, the evolved mean and median results are far from the targets, up to orders of magnitude difference for several rates. The histogram spreads for all seven rates are illustrated in Figure 5.14: the outside cycle rates are on the top two rows and the inside cycle rates are on the final row.

Variable	Target Value	Evolved Median	Evolved Mean	Std. Dev (σ)
rl	0.664	0.3887	0.4101	0.2654
rlm	0.002	0.0493	0.0510	0.02539
rdo	0.004	0.0394	0.0435	0.0306
rd1	0.004	0.0500	0.0512	0.0264
ga	0.4	0.442	0.483	0.2811
gd1	1.1	4.976	4.866	2.871
g1	10	8.51	9.11	5.84

Table 5.9: G-protein (PEPA) results

During this experiment, the difference between target and evolved values has increased markedly. However, the fitness score of this experiment has been particularly strong: the highest (weakest) fitness score being 0.368%, indicating a near identical match to the target time series trace.

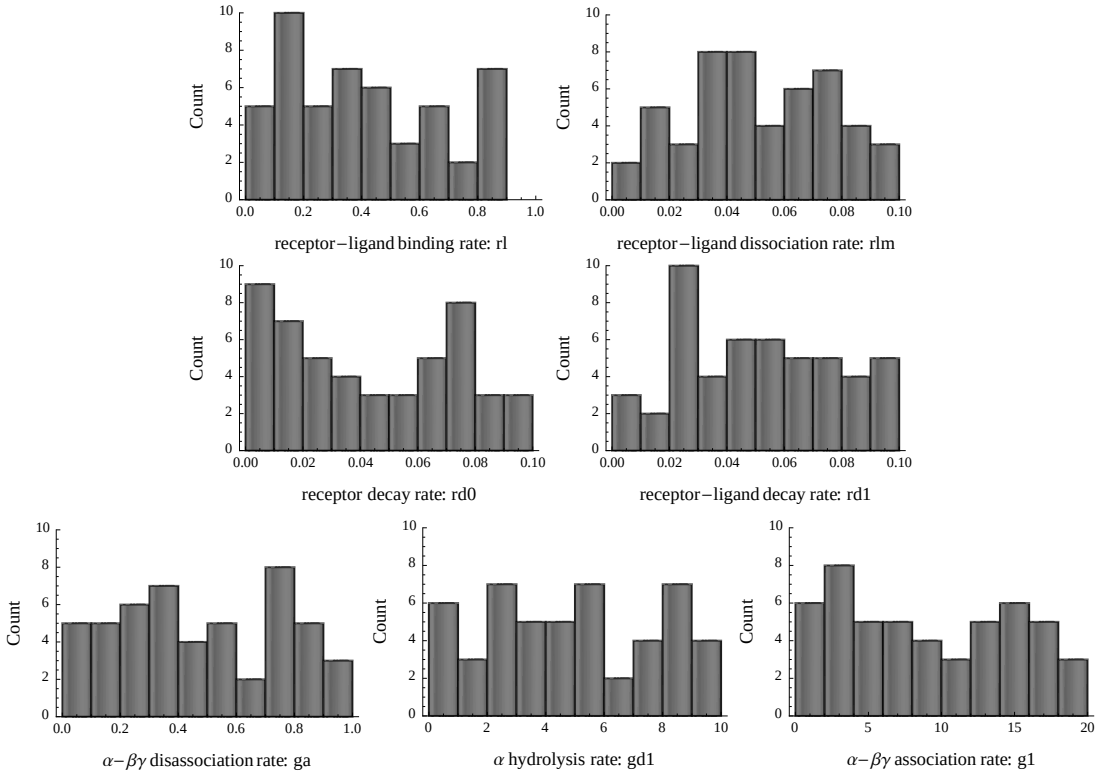


Figure 5.14: Histogram plots for G-protein (PEPA): *rl*, *rlm*, *rdo*, *rd1*, *ga*, *gd1* and *g1*

These results can be viewed in two ways. Negatively, the Euclidean distance fitness measure is not sophisticated enough to capture the complex behaviour of the G-protein dual cycle behaviour. This is likely true, a model-specific fitness function will always be preferred. Using the more generic Euclidean distance fitness function is a conscious choice to reduce the complexity during EPA framework experiments. Positively, there are different combinations of variable rate values which give strong matches to the target time series trace. This attests to

the robustness of the system rather than critical sensitivity to one particular parameter value or set of values. In complex systems with many facets, this is unsurprising. In the creation of formative models, providing results sets from outside the modeller's expected target may provide valuable insight into the model's operation.

This G-protein experiment provided experience with the described system; this would be valuable when applying Genetic Programming optimisation to the same system in future work. This specific problem could be solved to accurately match the target values, possibly with a custom fitness function, see Section 9.2.

5.6 PARAMETER MATCHING EXPERIMENTS SUMMARY

This chapter presented the application Genetic Algorithms (GAs) to models described in Performance Evaluation Process Algebra (PEPA) with the goal of parameter matching. PEPA models used during this chapter are complete and had some or all rates removed to be rediscovered. By using complete models, evolved solutions can be compared to the original PA model for qualitative and quantitive analysis.

The output of the EPA framework is a single strong solution which may not be the global optimum of the search space. Through multiple runs of the framework, the GA evolved values can be collated into a histogram spread to highlight critical rates to the user.

The performance of the EPA framework during the experiments of this chapter was strong overall. Difficulties became apparent during the latter experiments with the G-protein system (Section 5.5). When flagging a large number of parameters for optimisation, identifying critical rates becomes difficult in the histogram sensitivity analysis. Conversely, the number of different combinations of parameter values which provide strong fitness solutions attests to the robustness of the system.

Considering the time restraints, the strong fitness scores of solutions found and the experience gathered during these experiments; the focus moved onto Genetic Programming (GP) techniques and optimising Process Algebra model structures.

6

SPECIES STRUCTURE OPTIMISATION EXPERIMENTS

This chapter is the second of the experimental results chapters and details the Genetic Programming (GP) techniques utilised for direct model structure optimisation of species definitions. All experiments presented in this chapter utilise the Bio-PEPA language as it was performed after the Bio-PEPA changeover for Genetic Programming optimisation work, as discussed in Section 2.2.3.

All Process Algebra (PA) models chosen for this research are complete descriptions of complex systems accompanied by time series data. By choosing complete models, experiments can be conducted by selectively removing aspects of the model. The Evolving Process Algebra (EPA) framework can then be challenged to rebuild the removed aspects using Evolutionary Computation (EC) techniques: in this chapter it is species definitions which are removed to be rediscovered using GP techniques. Evolved solutions can be checked qualitatively by comparing the individual to the original PA model and quantitively by comparing the evolved simulation trace to the original time series data.

When working with Genetic Algorithm (GA) parameter matching experiments, the final result of each EPA execution was a numeric value for each rate flagged for optimisation. However, the results of species optimisation are GP trees and not number values. Due to this, results cannot be easily compared as a spread in the form of histogram plots. The qualitative sensitivity analysis is not so easily accomplished as observing the spread of values on such a plot.

Instead, a Process Algebra model is considered to be sensitive to a definition if that definition, or parts of it, appear frequently in strong fitness individuals. Conversely, a lack of sensitivity to a definition is indicated by that definition appearing infrequently in strong individuals or exclusively in weak individuals. This continues to allow a measure of qualitative analysis which may give insight to a domain expert by illustrating critical definitions in the described system.

To assist in this qualitative analysis, the *EPA Utility Assistant*, described in Appendix A, is capable of extracting this species component information. Each experiment result is presented with a bar chart to illustrate the spread of commonly appearing species segments, similar to the histogram spreads that provide insight into critical rates.

All the experiments performed in this chapter were performed using the tree-size evaluator fitness function built on top of the Euclidean distance sub-fitness function, as described in Section 4.3.5. The maximum (worst) fitness is based on the Euclidean distance sub-fitness

augmented by any scoring penalties. The method by which penalties are applied is taken into account when calculating the maximum fitness range. Once ascertained, the maximum fitness score can be used to observe how solutions score on the objective, percentage scale.

Model-specific organised complexity is important when choosing a system for GP optimisation techniques. The impact of changing a component definition in the Process Algebra model is greater than changing a rate value. For this reason, model complexity did not continue to increase from GA parameter matching work, instead it started anew with low complexity models. See Chapter 7 for kinetic definition GP optimisation or Chapter 8 for discussion of simultaneous Genetic Algorithm and Genetic Programming optimisation work.

6.1 BIOCHEMICAL REACTANT-PRODUCT

6.1.1 *Background & Model*

The biochemical reactant-product system is one of the simplest Bio-PEPA models. It represents the $2X + Y \rightarrow 3Z$ reaction, meaning two molecules of X and one molecule of Y are consumed (their levels both lower) to produce three molecules of Z. Figure 6.1 represents the Bio-PEPA model which describes this reaction: this is the same as Figure 2.4 in Section 2.2.2, copied here for convenience.

$$
\begin{aligned}
& r = 0.001 \\
& kineticLawOf\ alpha : fMA(r) \\
\\
& X \stackrel{def}{=} (alpha, 2) \ll X \\
& Y \stackrel{def}{=} alpha \ll Y \\
& Z \stackrel{def}{=} (alpha, 3) \gg Z \\
\\
& X[200] <alpha> Y[100] <*> Z[0]
\end{aligned}
$$

Figure 6.1: Biochemical Reactant-Product model written in Bio-PEPA

The biochemical reactant-product model consists of only a few segments: one rate (r), one kinetic action (*alpha*), three species (X, Y and Z) and the system equation defines communication across any action. Of those three species, only two Bio-PEPA operators are used: the reactant ($\ll$) and the product ($\gg$).

This model was chosen for use with the EPA framework because of the small number of components to work with. The stoichiometry coefficient is a non-zero, positive integer with an assumed maximum value of 10. In addition, each operator can be one of five values ($\ll$, $\gg$, $(+)$, $(-)$ or $(.)$). If all three species are flagged to be rebuilt together, that results in a solution space consisting of 150 points ($max\_coefficient * num\_operators * num\_species$). It is worth

noting that this search space is very small and would not normally warrant an Evolutionary Computation technique. Experiments derived from the biochemical reactant-product system were primarily used to test the EPA framework by ensuring GP mechanisms were correctly being applied to the Bio-PEPA syntax.

6.1.2 *Experiment* - all species

The first experiment in the GP domain flagged species X, Y and Z to be optimised. At first glance it seems adventurous to optimise all species components simultaneously; however, as mentioned above, this equates to a search space of 150 points.

Considering the default EPA parameters includes 100 individuals, it is reasonably likely the first generation will include the optimum solution. Such a small search space with a starting population which encompasses two-thirds of it would not normally warrant use of an EC approach. Similar to the simple dining philosophers experiment at the beginning of the GA parameter matching work (Section 5.1), this early stage of Genetic Programming work focused on the integration of the Bio-PEPA plug-in with the EPA framework. The biochemical reactant-product model was used to test this integration while compatibility with the PEPA plug-in was maintained.

Table 6.1 illustrates the simulator and EPA override parameter settings for the biochemical reactant-product model. Notice the use of a Euclidean distance fitness function instead of a tree-size evaluator; the biochemical reaction model contains only one kinetic action (which cannot be duplicated due to the Bio-PEPA restraints), therefore species definitions are not susceptible to bloat. This only applies to the species segments, later biochemical reaction experiments will utilise tree-size evaluators for the kinetic definitions, see Sections 7.1 and 8.1.

Simulator Variables	
Start	0
Stop	99
Data points	100

EPA Override Variables	
Fitness	Euclidean
Failure timeout	5s

Table 6.1: Biochemical (X, Y and Z) specific and EPA override parameter values

Correct definitions were discovered for this experiment in 50% (32 of 64) of EPA system runs. These correct definitions were an exact match to the $2X + Y \rightarrow 3Z$ structure of the original biochemical reactant-product model. Their fitness scores were all <0.005%, where 0% is no deviation from the expected time series trace. This demonstrates that obtaining a perfect 0% fitness score is infeasible; the slight deviations are attributed to the stochastic noise of Process Algebra simulations.

The poorer scoring solutions which made up the remaining 50% were only 0.001% higher in their fitness scores. These system runs contain example solutions of the correct structure,

but with different stoichiometry coefficients. For example, $4X + 2Y \rightarrow 6Z$ is a sensible solution which scored a good fitness, but a domain expert or human modeller would be able to easily refine this definition because the more elegant version is obvious. These solutions are likely because of a premature convergence at a local optimum.

Given more time for evolution or a higher evolutionary stress on lower stoichiometric coefficients, this could yield the solutions with a more elegant definition. No solutions resulted in terrible or nonsensical evolved definitions.

These results demonstrated the EPA framework had adequately been extended to incorporate Bio-PEPA species optimisation. Further experiments in this section aim to challenge the EPA with more difficult optimisation problems. It is worth noting that the objective fitness scores allow the user to gain knowledge of how fit a solution really is. Not every evolved solution will be acceptable, but the aim is that each solution will provide some insight to the framework's user.

6.2 measles (frequency-dependent transmission)

6.2.1 *Background & Model*

Measles is caused by a virus of the same name which belongs to the Morbillivirus genus within the family Paramyxoviruses, characterised by their negative-sense, single-stranded RNA structure. A positive-sense RNA can be directly transcribed into viral DNA, whereas a negative-sense RNA must first be translated into a positive-sense form. The virus is spread through contact with fluids from an infected host's mouth or nose, either directly or via airborne-spread such as sneezing.

The measles virus is highly contagious: greater than 90% of exposures without vaccine-primed immunity will result in infection. The three-part MMR vaccine was introduced to the UK in 1988, designed to vaccinate recipients against measles, mumps and rubella. The vaccine is designed to be administered at the age of 18 months with a booster at age 4 - 5. Despite the vaccine, many are still vulnerable to the virus, particularly in developing countries. A confirmed case of measles still warrants booster vaccinations in the immediate area to stem an outbreak.

Symptoms commonly include cough, fever, coryza, conjunctivitis and a distinctive full-body rash. There is currently no treatment beyond the preventative vaccine, as mentioned above. Most infected individuals will recover within fourteen days with bed rest and supportive treatment of the symptoms. In few cases the measles virus presents with complications. Complications can range in severity from moderate to serious, including diarrhoea, pneumonia and bronchitis. In rare cases, the infection can cause subacute sclerosing panencephalitis (SSPE): an untreatable, chronic swelling of the brain that ultimately results in death.

Benkirane et al. [11] model measles spread for the purposes of outbreak prediction, prevention and response. Their work includes a case study on Measles dynamics in England and Wales for 20 years between 1944 and 1964, with results focusing on Measles spread in Leeds during this period. Note, this is before the vaccine was introduced to the UK in 1968: this is the original vaccination before the MMR vaccine was developed. They do not include the four years between 1964 and 1968 (end of the study to introduction of the vaccine) due to administration changes which reduced reliability of the data.

Benkirane et al. present two SEIR models with their study: one PEPA and one Bio-PEPA; it is the latter which is of interest to this work. The full system includes cyclic, seasonal behaviour as the number of measles infections spike every few years. To keep the focus on the EPA's ability to rebuild removed species and kinetic definitions, models abstracting from the full feature set were used so the simpler Euclidean distance measure of fitness could be used, see Section 4.3.4.

This system was chosen for use with the EPA because of the rich wealth of data: 20 years worth across across 60 cities in England and Wales. Furthermore, measles is an SEIR system similar to the Internet worm model, as seen in Section 5.2. Though the SEIR behaviour is similar to the the SIR behaviour the notation is different in Bio-PEPA. Species in Bio-PEPA are viewed as levels which can increase or decrease through cooperation over actions. The number of susceptible individuals in the system will decrease as the number of exposed individuals increases. Note the use of $(.)$ (generic modifier) in defining the infected (Inf) species in Figure 6.2. This indicates that the infected individuals play a role in the *contact* action, but their numbers are not directly influenced by that action. This also increases the model's robustness and allows the action definition to satisfy the Bio-PEPA syntax requirement that actions derived from species must cooperate over that action, see Section 2.2.2.

$$rContact = 6.015 \qquad rInfect = 0.133 \qquad rRecover = 0.154$$

$$\begin{aligned}
&kineticLawOf\ contact : (rContact * S * Inf)/(S + Inf + R)\\
&kineticLawOf\ incubation : rInfect * Exp\\
&kineticLawOf\ recover : rRecover * Inf
\end{aligned}$$

$$\begin{aligned}
S &\stackrel{def}{=} contact \ll S\\
Exp &\stackrel{def}{=} contact \gg Exp + incubation \ll Exp\\
Inf &\stackrel{def}{=} contact\ (.)\ Inf + incubation \gg Inf + recover \ll Inf\\
R &\stackrel{def}{=} recover \gg R + contact\ (.)\ R
\end{aligned}$$

$$S[508000] <*> Inf[5] <*> Exp[0] <*> R[0]$$

Figure 6.2: Frequency-dependent transmission measles SEIR model written in Bio-PEPA

Frequency-based contact [8] assumes that individuals inside an epidemiological system will contact a fixed number of other individuals in a given time period, regardless of population

density. The chance of infection is dependent on the number of susceptible (S) and infected (Inf) currently in the system, as well as the total number of individuals in the system. Additional means of infection based on population density are not modelled; such as secondary transmission by coughing.

Figure 6.2 presents the frequency-based contact SEIR model derived from the Benkirane et al. paper. The *contact* kinetic action is derived from the rate *rContact* and the number of S and Inf, as well as the sum of the individuals in the system. This is commonly referred to as $\beta SI \,/\, N$, where N is all individuals in the system. The *incubation* action determines the period between being exposed (Exp) and becoming infected. Finally, the *recover* action governs the period it takes an infected individual to enter a recovered state (R).

6.2.2 *Experiment* 1 - Inf

In this first experiment, the Inf species in the model was flagged for optimisation. This component was chosen because it had the most complex behaviour of any species in the system and cooperates across all available actions. Table 6.2 presents the simulator and EPA override parameters for this experiment.

Simulator Variables	
Start	0
Stop	50
Data points	51

EPA Override Variables	
Failure timeout	10s

Table 6.2: Measles SEIR (Inf) specific and EPA override parameter values

Under these conditions, the original species definition for Inf was rediscovered with an excellent objective fitness score of 0.03%. Alternative evolved species definitions were also discovered with equally strong fitness scores. These definitions fully omit the *contact* action from the Inf species definition.

$$Inf_{\,original} \stackrel{def}{=} contact\,(.)\; Inf \;+\; incubation \gg Inf \;+\; recover \ll Inf$$

$$Inf_{\,evolved} \stackrel{def}{=} incubation \gg Inf \;+\; recover \ll Inf$$

These alternative definitions with strong fitness scores may be explained by the semantics of the $(.)$ operator. The generic modifier is included to satisfy the Bio-PEPA syntactic requirement of a species cooperating over an action to appear in that kinetic action's definition, see Section 2.2.2. When simulated these statements are identical because the kinetic definition which dominates the evolved behaviour, however the lack of a generic operator will cause a violation (not to be mistaken with an error which prevents simulation) in the Bio-PEPA syntax.

The species analysis for this experiment is presented in Figure 6.3. This provides a strong example of what is strived for by this form of analysis. The top three most occurring definition

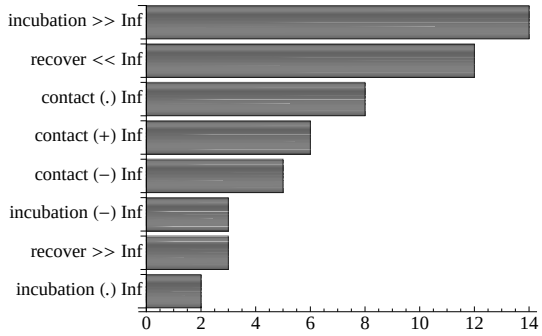


Figure 6.3: Species analysis for Measles SEIR (Frequency Version): *Inf*

segments (*'incubation* $\gg$ *Inf'*, *'recover* $\ll$ *Inf'* and *'contact* (.) *Inf'*) are the three segments of the target species definition. This highlights the sensitivity of the *Inf* species to these segments over any other definitions that are available.

6.2.3 *Experiment* 2 - Exp & Inf

Given the promising results obtained during the previous experiment while evolving *Inf* alone, a second experiment was run that challenged the system further by flagging both the *Exp* and *Inf* species. The *Exp* and *Inf* species were chosen because these two species contain the majority of the complex behaviour in the measles system. The setup for this experiment uses the same simulator and EPA override parameters as in the previous experiment, illustrated in Table 6.2.

The first attempt at this experiment did particularly poorly. Less than 10% of the EPA system runs returned with strong fitness solutions which were biologically valid. The fittest individual evolved was an exact match to the target *Exp* species definition. The *Inf* species definition omitted the *'contact* (.) *Inf'* segment, similar to strong fitness individuals in the previous experiment.

$$
\begin{aligned}
Exp_{\ original} &\stackrel{def}{=} contact \gg Exp \ + \ incubation \ll Exp \\
Inf_{\ original} &\stackrel{def}{=} contact\ (.)\ Inf \ + \ incubation \gg Inf \ + \ recover \ll Inf \\
Exp_{\ evolved} &\stackrel{def}{=} contact \gg Exp \ + \ incubation \ll Exp \\
Inf_{\ evolved} &\stackrel{def}{=} incubation \gg Inf \ + \ recover \ll Inf
\end{aligned}
$$

The generic modifier is used as a semantic reminder that a species participates over a kinetic action. This allows the species definition to satisfy the Bio-PEPA syntactic requirement that any kinetic action which includes a species must appear in that species. For this reason, the evolved definition for *Inf* will behave identically to the original, barring stochastic differences. A modeller using the EPA framework would understand the nature of the generic modifier and add the missing segment.

The remaining evolved solutions all included the *recover* action in the *Exp* species definition, most frequently in a *'recover* $\ll$ *Exp'* form. This indicates that exposed individuals are moving

directly into a recovered state without transitioning through an infected state. The measles SEIR system does not include any vaccine-like effect which would allow this to occur. The species analysis for both the *Exp* and the *Inf* species are both dominated by the *recover* action, so are not included here.

Looking at the trace generated from the *recover* $\ll$ *Exp* definition; the numbers of susceptible and exposed individuals decrease and the number of recovered individuals increases, the number of infected individuals remains unchanged. This seems to be a local optimum that the EPA framework prematurely converged into. Through use of the Euclidean distance sub-fitness function, this provided a reasonable fitness score: i.e. many of the datapoints are a match using a distance measure. Given more computational time, the EPA framework may have overcome this local optimum.

6.2.4 *Experiment* 3 - Exp & Inf *(with domain knowledge)*

An additional experiment was conducted to utilise known domain knowledge, see Section 4.3.1. The restricted action list was utilised for the *Exp* species and includes the *recover* action, therefore prohibiting individuals to move directly from an exposed state into a recovered state without first becoming infected. This experiment uses the same parameter settings as the original experiment, as detailed in Table 6.2.

Unexpectedly, this experiment evolved individuals with poorer definitions overall compared to the previous experiment which did not utilise domain knowledge. The solutions discovered were dominated by the infected definition '*recover* $\gg$ *Inf*'. This indicates that recovery increases the number of infected individuals which does not make biological sense.

The use of the restricted lists feature of the EPA framework did function correctly: the *recover* action did not appear in any *Exp* definitions. However, this feature did not lead to the improvement of fitness scores as originally expected.

Looking at the evolved solutions, the *Exp* definitions varied across the permissible actions whereas the *Inf* species definition frequently evolved the *recover* action as described above. Similar to the outcome of the second experiment, described above, the EPA framework seems to have converged prematurely. The number of recovered individuals will increase with the *recover* action because its definition remains constant, as illustrated in Figure 6.2. By having infected levels increase alongside recovered individuals, this provided a reasonable fitness score through the Euclidean distance fitness function: i.e., many of the datapoints are a match using a distance measure.

The restricted action list did function correctly but improvements could be made to increase the specificity. In this instance, such restrictions would not be appropriate because of the small number of actions available to the measles system. However, such a development could be beneficial to the future of the EPA framework, as discussed in Section 9.3.

6.3 GENERIC GENETIC NETWORK

6.3.1 *Background & Model*

A genetic network, also referred to as a gene regulatory network or genetic regulatory network, is a collection of DNA elements which interact intracellularly. The interaction is indirect via substances in the cell, particularly RNA and protein expressions which govern the levels of mRNA (*messenger RNA*). The mRNA molecules go on to produce specific proteins or sets of proteins. These proteins can be structural: gathering on the cell membrane or intracellularly for communication purposes.

Proteins may also be enzymes, which act as highly selective catalysts for many chemical reactions, such as food digestion or toxin breakdown. Other proteins may have an excitatory effect on genes, leading to their activation. Once activated these genes become transcription factors for specific proteins which, in turn, can activate other genes, leading to a cascade effect. Other proteins may have an inhibitory effect on other genes, preventing their associated protein transcription.

In single-celled organisms, regulatory genetic networks respond to external environmental factors, allowing the cell to adapt and survive in that environment. For example, a yeast cell in a sugar solution will activate genes which transcribe proteins responsible for metabolisation. This will digest sugars and produce carbon dioxide and ethanol (alcohol) under low-oxygen or anaerobic conditions; this fermentation process is key to producing alcoholic beverages.

$$
\begin{aligned}
&\mathit{kineticLawOf}\ \mathit{transcription}\ :\ 2.19/(356 + P2)\\
&\mathit{kineticLawOf}\ \mathit{translation}\ :\ \mathit{fMA}\ (0.043)\\
&\mathit{kineticLawOf}\ \mathit{degradation\_M}\ :\ \mathit{fMA}\ (0.0039)\\
&\mathit{kineticLawOf}\ \mathit{degradation\_P}\ :\ \mathit{fMA}\ (0.0007)\\
&\mathit{kineticLawOf}\ \mathit{dimerization}\ :\ \mathit{fMA}\ (0.025)\\
&\mathit{kineticLawOf}\ \mathit{dimerization\_Inv}\ :\ \mathit{fMA}\ (0.5)
\end{aligned}
$$

$$
\begin{aligned}
M &\stackrel{def}{=} \mathit{transcription} \gg M\ +\ \mathit{translation}\ (+)\ M\ +\ \mathit{degradation\_M} \ll M\\
P &\stackrel{def}{=} \mathit{translation} \gg P\ +\ \mathit{degradation\_P} \ll P\ +\ (\mathit{dimerization}, 2) \ll P\ +\ (\mathit{dimerization\_Inv}, 2) \gg P\\
P2 &\stackrel{def}{=} \mathit{transcription}\ (-)\ P2\ +\ \mathit{dimerization} \gg P2\ +\ \mathit{dimerization\_Inv} \ll P2
\end{aligned}
$$

$$
M[0]\ <\mathit{translation}>\ P[0]\ <\mathit{dimerization}, \mathit{dimerization\_Inv}>\ P2[0]
$$

Figure 6.4: Generic genetic network model written in Bio-PEPA

Ciocchetta & Hillston [20] present an example Bio-PEPA model of a generic genetic network, illustrated in Figure 6.4. The generic genetic network system is a negative feedback system through dimers, such as the biological control circuit in λ repressor protein *ci* of λ-phage in *Escherichia coli* (*E. coli*). *E. coli* is a single-celled bacterium commonly found in the intestines

of endotherms (warm-blooded organisms). It is one of the most widely studied creatures, particularly in the fields of biology and biochemistry. This is due to the quick, cheap and easy nature of the bacteria's growth. The repressilator model (Section 5.4) was also taken from a system inside the *E. coli* bacteria.

Consider the Bio-PEPA model and note the lack of rate variables. Experiments focused on species evolution, so no experiments derived from this model required explicitly separate rate variables. For conciseness, rate values have been amalgamated into the kinetic action definitions.

Three species exist in the genetic network; the mRNA (M), the transcribed protein in its monomer form (P) and the transcribed protein in its dimer form ($P2$). A single molecule of a protein is considered to be a monomer molecule of that protein. Two identical monomer molecules bound together are considered to be a dimer.

Starting with 0 levels of M, P and $P2$, some external stimuli will trigger the mRNA *transcription* action. Once transcribed mRNA can either degrade naturally via the *degradation_M* action or take the *translation* action to create the monomer form of the protein. The monomer protein may also degrade, via the *degradation_P* action. Through the *dimerization* action, two monomer proteins can form a dimer; the *dimerization_Inv* action is the reverse and returns the two proteins to their monomer form. As the levels of the dimer protein increase, the *transcription* action slows due to an inhibitory affect, denoted by the $(-)$ operator. Figure 6.5 illustrates the behaviour of the genetic network: boxes are states and edges are transitions caused by the labelled actions.

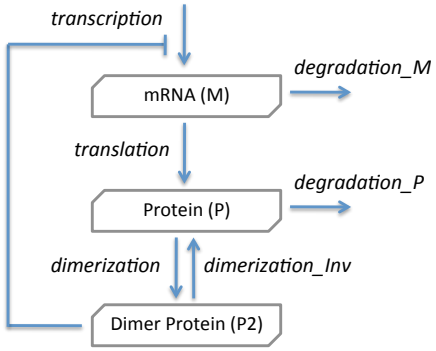


Figure 6.5: Processes of a generic genetic network

The genetic network model was initially chosen for use with the EPA framework because it was written by Ciocchetta & Hillston, the creators of Bio-PEPA, thus ensuring a valid and well formed model. This model also presents a further rise in complexity. The target model contains species each with at least three choices, thus challenging the EPA to create and match multiple GP trees per species. The $(-)$ modifier, which has not been previously used, is also part of the model. Finally, the system equation of the genetic network model specifies communication paths: i.e., communication paths are not all '$<*>$'s.

The mRNA (M) can only communicate with the monomer form protein (P) via the *translation* action which in turn can only communicate with the dimer form protein ($P2$) over the *dimerization* or *dimerization_Inv* actions. The remaining unspecified actions do not require cooperation so do not appear in the system equation.

6.3.2 *Experiment* - P2

Table 6.3 displays the simulator and EPA override parameters. The stop time is notably higher than other PA models, this is because it takes longer for the system to exhibit behaviours to compare to in the fitness function. The start time could be increased to lower the number of data points required for comparison. However, the time required to simulate the genetic network system is low, so the increased number of data points is acceptable.

Simulator Variables	
Start	0
Stop	10,000
Data points	10,001

EPA Override Variables	
Failure timeout	7s

Table 6.3: Genetic network specific and EPA override parameter values

Of the 50 EPA system runs, the original $P2$ definition was not rediscovered in its entirety. The most commonly evolved definition omits the *dimerization* action:

$$P2_{\ original} \stackrel{def}{=} transcription\ (-)\ P2\ +\ dimerization \gg P2\ +\ dimerization\_Inv \ll P2$$

$$P2_{\ evolved} \stackrel{def}{=} transcription\ (-)\ P2\ +\ dimerization\_Inv \ll P2$$

Considering the sensitivity analysis in Figure 6.6, the '$dimerization\_Inv \ll P2$' is the most frequently occurring definition segment. The *dimerization_Inv* action appears multiple times: this shows the sensitivity of the system to this action despite the accompanying operator or stoichometry coefficient. The *transcription* action also appears with unexpected operators, showing a lower sensitivity.

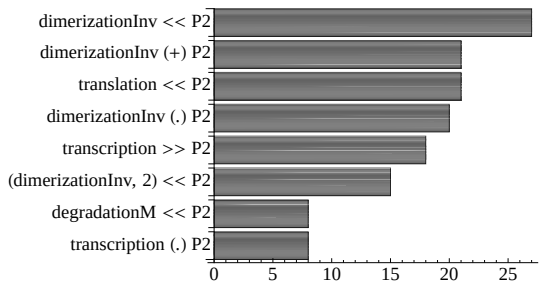


Figure 6.6: Species analysis for genetic network: $P2$

The sensitivity analysis is not as clear as the analysis of the measles SEIR experiment, illustrated previously in Figure 6.3. In noisy complex systems, evolved species definitions will

bloat and include extraneous segments which can occur frequently enough to appear in the sensitivity analysis. Combined with a domain expert's knowledge of a system, this analysis can provide insight into the critical species definition segments, even if the original definition is not rediscovered.

6.4 G-PROTEIN (BIO-PEPA)

6.4.1 *Background & Model*

The G-protein system was visited previously during Genetic Algorithm parameter matching work of the PEPA model in Section 5.5. To reprise, the G-proteins are controlled by G-protein-coupled receptors with the purpose to transduce extracellular signalling molecules into intracellular signals.

The PEPA version of the G-protein model appeared previously in Figure 5.13. The PEPA version was derived from the Bio-PEPA model presented by the Bao et al. study [5]. Their Bio-PEPA version of the model is illustrated in Figure 6.8, later in this section.

Using the Bao et al. notation, the G-protein system is comprised of seven species components. The receptor (R), the ligand (L), the bound receptor-ligand (RL), the deactivated α subunit which is bound to GDP (Gd), the $\beta\gamma$ complex (Gbg), the active α subunit which is bound to the $\beta\gamma$ complex (G) and finally the active α subunit which is bound to GTP (Ga). Unlike the PEPA model which requires mirror and recruitment pools, the Bio-PEPA description requires no extra species to accurately describe the system.

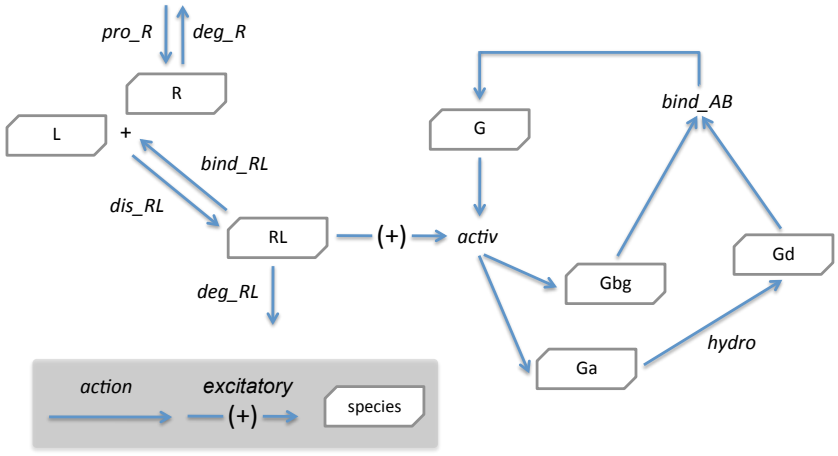


Figure 6.7: G-protein activation cycles

The G-protein system can be viewed as two separate cycles where one has an excitatory effect on the other. Figure 6.7 illustrates these cycles. To the left is outside the cell; receptor molecules can be produced (pro_R) or degrade (deg_R). With available ligands on the cell membrane,

receptors and ligands can associate (*bind_RL*) and once bound either become disassociated (*dis_RL*) or degrade (*deg_RL*). To the right of the figure is inside the cell, the α subunit and the $\beta\gamma$ complex begin in a bound state. As the levels of receptor-ligand associations increase, the α subunit disassociates from the $\beta\gamma$ complex because of an excitatory effect on the *activ* action. Once disassociated, the active α subunit hydrolyses (*hydro*) GTP into GDP and becomes inactive. Finally, the inactive α will re-associate (*bind_AB*) with the $\beta\gamma$ complex to reset the cycle.

Figure 6.8 illustrates the Bio-PEPA version of the G-protein model. Note, as with the generic genetic network model in Section 6.3, rate variables are not included because no experiments were derived involving them. Therefore, rate values have been incorporated into the kinetic action definitions.

$$
\begin{aligned}
&kineticLawOf\ bind\_RL\ :\ fMA\ (3.32e-6)\\
&kineticLawOf\ bind\_AB\ :\ fMA\ (1.0)\\
&kineticLawOf\ dis\_RL\ :\ fMA\ (0.01)\\
&kineticLawOf\ activ\ :\ fMA\ (1.0e-5)\\
&kineticLawOf\ hydro\ :\ fMA\ (0.11)\\
&kineticLawOf\ deg\_RL\ :\ fMA\ (4.0e-3)\\
&kineticLawOf\ deg\_R\ :\ fMA\ (4.0e-4)\\
&kineticLawOf\ pro\_R\ :\ 4.0
\end{aligned}
$$

$$
\begin{aligned}
L &\stackrel{def}{=} dis\_RL \gg L + bind\_RL \ll L\\
R &\stackrel{def}{=} dis\_RL \gg R + bind\_RL \ll R + deg\_R \ll R + pro\_R \gg R\\
RL &\stackrel{def}{=} dis\_RL \ll RL + bind\_RL \gg RL + deg\_RL \ll RL + activ\ (+)\ RL\\
G &\stackrel{def}{=} bind\_AB \gg G + activ \ll G\\
Ga &\stackrel{def}{=} activ \gg Ga + hydro \ll Ga\\
Gd &\stackrel{def}{=} bind\_AB \ll Gd + hydro \gg Gd\\
Gbg &\stackrel{def}{=} bind\_AB \ll Gbg + activ \gg Gbg
\end{aligned}
$$

$$
\begin{aligned}
&L[603,300] <*> R[10,000] <*> RL[0] <*> G[7,000]\\
&<*> Gd[3,000] <*> Gbg[3,000] <*> Ga[0]
\end{aligned}
$$

Figure 6.8: G-protein system model written in Bio-PEPA

Note the values in the Bio-PEPA system equation are 100 times larger than those previously seen in the PEPA model. This is because Bio-PEPA simulation time is much shorter, so the original Bao et al. values can be used.

It was known that the G-protein Bio-PEPA model would be used with the EPA framework, this is why the PEPA version of the model was explored previously. This was done so that experience could be gained with the system during GA parameter matching before moving onto the application of more complicated Genetic Programming optimisation techniques.

6.4.2 *Experiment* 1 - RL

The G-protein system was experimented with previously during Genetic Algorithm parameter matching work of the PEPA model in Section 5.5. This new experiment focuses on utilising the Genetic Programming aspect of the EPA framework to rediscover the definition for the receptor-ligand (*RL*) species. This species was chosen because it has the most complex behaviour of any in the system. Additionally, the *RL* species uniquely exists in the extracellular space and influences the intracellular system.

Simulator Variables	
Start	0
Stop	600
Data points	601

EPA Override Variables	
Failure timeout	10s

Table 6.4: G-protein (Bio-PEPA) (*RL*) specific and EPA override parameter values

Table 6.4 presents the simulator and EPA override parameters for this experiment. The G-protein system is comparatively slow to simulate, hence the longer failure timeout value. Beyond this, there are no special circumstances to consider.

Below is the most frequently evolved definition for the *RL* species. The majority of the definition has been rediscovered correctly with the exception of the operator associated with the *activ* action.

$$RL_{\,original} \stackrel{def}{=} dis\_RL \ll RL \;+\; bind\_RL \gg RL \;+\; deg\_RL \ll RL \;+\; activ\,(+)\,RL$$

$$RL_{\,evolved} \stackrel{def}{=} dis\_RL \ll RL \;+\; bind\_RL \gg RL \;+\; deg\_RL \ll RL \;+\; activ\,(-)\,RL$$

The target definition contains an (+) (activator) while the evolved definition contains an (–) (inhibitor): the opposite operator. The fitness score remains strong which indicates there is greater emphasis on the action chosen over the operator used. Therefore, the *activ* operator appearing with an inhibitory effect scores a better fitness than the action being fully omitted.

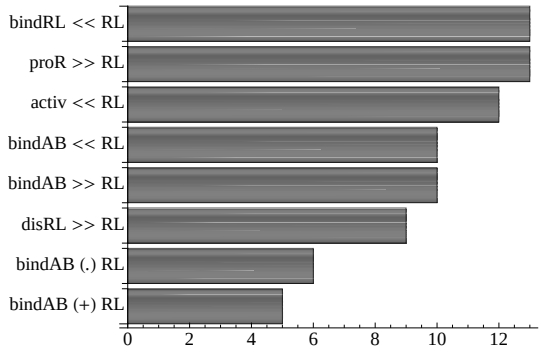


Figure 6.9: Species analysis for G-protein: *RL*

Figure 6.9 presents the species analysis for this experiment. Though the strongest solution was a very close match, the species analysis is less accurate. The '*bind_RL* ≪ *RL*' appears joint

frequently with the '*pro_R* ≫ *RL*' segment. In the G-protein system, the *bind_RL* action will bind receptors with ligands to increase the number of bound receptor-ligands. Additionally, the *pro_R* action does not play a role in the target *RL* species behaviour. Finally, the *bind_AB* action occupies half of this analysis despite that the *RL* species plays no role in the association of the inactive α with the $\beta\gamma$ complex.

The EPA framework can utilise domain knowledge as discussed in Section 4.3.1. The knowledge that the *RL* species exists in the extracellular space of the G-protein system is reflected using restricted action lists. The *RL* species definition is restricted to not include the *bind_AB* and *hydro* actions.

6.4.3 *Experiment* 2 - RL *(with domain knowledge)*

A second experiment was run with the parameters of the first G-protein: *RL* experiment, as detailed in Table 6.4. Additionally, the restricted action list was utilised for the *RL* species and includes the *bind_AB* and *hydro* actions. Figure 6.10 presents the species analysis of the restricted experiment.

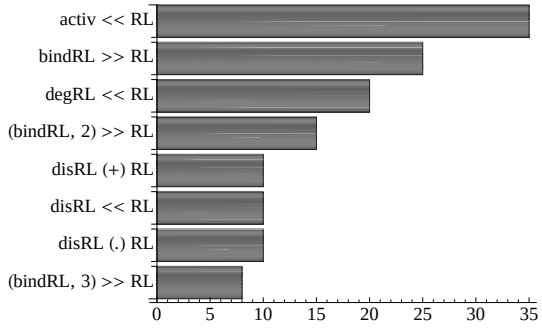


Figure 6.10: Species analysis for G-protein: *RL* with restrictions

Looking at this analysis, the *activ* action becomes the most frequently appearing action as a reactant to the *RL* species. By investigating the best evolved definitions, a different view of the system comes to light. The G-protein system allows receptor-ligand bindings to increase via binding, decrease via disassociation or degradation and finally has an excitatory effect on the *activ* action. The evolved view of the system allows receptor-ligand bindings to increase via binding three at a time, decrease via disassociation or via the *activ* action.

$$RL_{\,original} \stackrel{def}{=} dis\_RL \ll RL \;+\; bind\_RL \gg RL \;+\; deg\_RL \ll RL \;+\; activ\,(+)\,RL$$

$$RL_{\,evolved} \stackrel{def}{=} dis\_RL \ll RL \;+\; (bind\_RL, 3) \gg RL \;+\; activ \ll RL$$

This difference can be explained by the reactant operator for the *activ* action instead of the activator operator. To make up for the increased drain on the *RL* species, the *bind_RL* species has an increased stoichometry coefficient and the *deg_RL* degradation action is no longer required and is therefore omitted.

The G-protein system is a known complex system therefore the biological mechanisms are understood. If used to explore a newly modelled system, this form of alternative insight could give a fresh perspective to the domain expert into the mechanisms of that system.

6.4.4 *Experiment* 3 - RL & G

This next experiment extends the *RL* species only experiment with the additional flag of the *G* species. The receptor-ligand (*RL*) species was chosen again because of the success with the previous experiment, additionally it has the most complex behaviour of any species in the system. The *RL* species uniquely exists in the extracellular space and influences the intracellular system. The *G* species (active α subunit bound to the $\beta\gamma$ complex) was chosen because it exists in the intracellular system and the *activ* action causes its disassociation, as influenced by the levels of the *RL* species.

Simulator Variables	
Start	0
Stop	600
Data points	601

EPA Override Variables	
Internal replications	5
Failure timeout	13s

Table 6.5: G-protein (Bio-PEPA) (*RL* & *G*) specific and EPA override parameter values

Table 6.5 presents the simulator and EPA override parameters for this experiment. Notice the increased failure timeout value and the reduced number of internal replications. Both factors are to control the increased simulation time required during this experiment.

The first attempt at this experiment was done without the restricted action lists, so the EPA framework had no knowledge of the separate cycles and which actions could appear in each. Under these conditions, the *G* species was correctly rediscovered to be *'bind_AB* $\gg$ *G* + *activ* $\ll$ *G'* in many of the EPA runs.

However, other evolved solutions contained actions which biologically should not be available in the intracellular space. The *RL* species definitions were dominated by the *bind_AB* action just as in the previous experiment which evolved the *RL* species alone, as described in Section 6.4.2.

6.4.5 *Experiment* 4 - RL & G *(with domain knowledge)*

The fourth experiment with the G-protein system built upon the previous experiment by including domain knowledge. The restricted action list was utilised for the *RL* species and includes the *bind_AB* and *hydro* actions; just as with experiment two in Section 6.4.3. Additionally, the *G* species had a restricted action list containing the receptor-ligand related actions

(*bind_RL*, *dis_RL* and *deg_RL*). The experimental settings continue to use the parameters of the previous experiment, detailed in Table 6.5

Figure 6.11 illustrates the species analysis for *G*. This analysis shows the optimisation was weaker than for the *RL* half of the experiment, described below. The '$bind\_AB \gg G + activ \ll G$' target definition for *G* was not rediscovered in any of the EPA system runs. Also, the target *bind_AB* action was rarely used in solution structures, instead the *activ* and *hydro* action were commonly utilised.

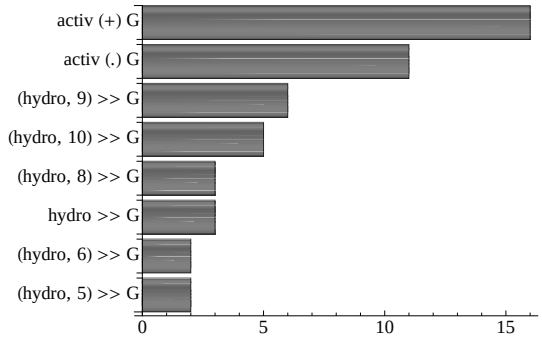


Figure 6.11: Species analysis for G-protein: *G* with restrictions

The *RL* target was also never rediscovered in its entirety, but the majority of solutions were close. This led to the *RL* species analysis in Figure 6.12 to be more accurate than for *G*, above.

$$RL_{\ original} \stackrel{def}{=} dis\_RL \ll RL + bind\_RL \gg RL + deg\_RL \ll RL + activ\ (+)\ RL$$

$$RL_{\ evolved} \stackrel{def}{=} dis\_RL \ll RL + bind\_RL \gg RL + deg\_RL \ll RL + activ\ (.)\ RL$$

$$RL_{\ evolved} \stackrel{def}{=} bind\_RL \gg RL + deg\_RL \ll RL + activ\ (+)\ RL$$

$$RL_{\ evolved} \stackrel{def}{=} dis\_RL \ll RL + bind\_RL \gg RL + deg\_RL \ll RL$$

The *bind_RL* and *deg_RL* actions dominate the analysis which is acceptable because these actions are core to the behaviour of the receptor-ligand species and appear with the target operators. The next most frequent appearing definition segments '$activ\ (+)\ RL$' and '$dis\_RL \ll RL$' which also appear in the target *RL* species definition.

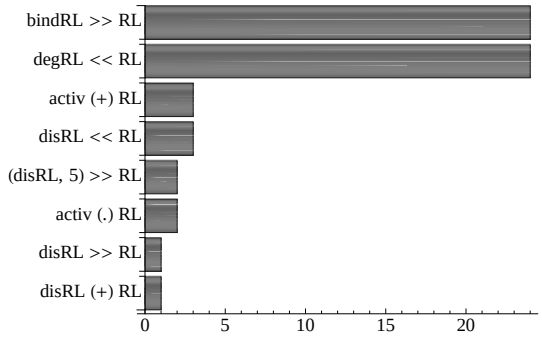


Figure 6.12: Species analysis for G-protein: *RL* with restrictions

Curiously, the *activ* action appears less frequently than in the species analysis of *RL* when evolved alone, see Figure 6.10. This can be explained because the *RL* and *G* species are flagged

together; both species heavily influence the *activ* behaviour in the original model. Should a *G* species definition be evolved to not include the *activ* action is is less likely for the *RL* species definition to retain the *activ* action.

6.5 SPECIES STRUCTURE OPTIMISATION EXPERIMENTS SUMMARY

This chapter presented the application of Genetic Programming (GP) optimisation techniques to Bio-PEPA models with the goal of evolving fitter species definition structures. Bio-PEPA models used during this chapter are complete and had some or all species definitions removed to be rediscovered during optimisation. By using complete models, evolved solutions can be compared to the original Process Algebra model definitions.

During Genetic Algorithm (GA) parameter optimisation work (Chapter 5), analysis could be performed on histogram spreads of parameter values. Species definitions do not use numeric values so this same analysis cannot be performed on GP experiments. To overcome this, the EPA framework was extended to produce bar charts representing the most frequently appearing species definition components. High sensitivity is indicated by a definition appearing frequently in strong solutions, whereas a lack of sensitivity is indicated by infrequently appearing definitions or appearing exclusively to weak individuals.

The performance of the Evolving Process Algebra (EPA) framework during the GP experiments of this chapter was strong overall. The frequency-dependent transmission of measles (Section 6.2) and the G-protein system (Section 6.4) highlighted deficits in the EPA framework's performance. The measles system frequently became stuck in a local optimum with biologically invalid definitions. The ability to utilise domain knowledge (Section 4.3.1) assisted in improving these results.

The G-protein system included a complex underlying structure that could not easily be rediscovered using a simple Euclidean distance measure fitness function. The requirement of custom fitness functions for complex problems is a common concern with Evolutionary Computation optimisation techniques, this is discussed in Section 9.2.

7

KINETICS STRUCTURE OPTIMISATION EXPERIMENTS

This chapter is the third of the experimental results chapters and details the Genetic Programming (GP) techniques utilised for direct model structure optimisation; focusing on kinetics, though some experiments are extended to include species as well. All work presented in this chapter utilises the Bio-PEPA language exclusively as it was performed after the Bio-PEPA changeover for Genetic Programming optimisation work, as discussed previously in Section 2.2.3.

All Process Algebra (PA) models chosen for this research are complete descriptions of complex systems accompanied by time series data. By choosing complete models, experiments can be conducted by selectively removing aspects of the model. The Evolving Process Algebra (EPA) framework can then be challenged to rebuild the removed aspects using Evolutionary Computation (EC) techniques: in this chapter it is kinetic action definitions which are removed to be rediscovered using GP techniques. Evolved solutions can be checked qualitatively by comparing the individual to the original PA model and quantitively by comparing the evolved simulation trace to the original time series data.

The previous two results chapters had additional forms of qualitative analysis: histogram distributions for Genetic Algorithm (GA) parameter matching and sensitivity analysis for species GP definition optimisation work. There is no additional form of analysis for evolved kinetic definitions. Forms of kinetic qualitative analysis could be included with future development of the EPA framework, this is discussed later in Section 9.3.

Experiments conducted in this chapter were performed using tree-size or node-weight evaluator fitness functions, as described in Sections 4.3.5 and 4.3.6 respectively. For either function, it is built on top of the Euclidean distance sub-fitness function, see Section 4.3.4. The maximum (worst) fitness is based on the sub-fitness augmented by any scoring penalties. The method by which penalties are applied is taken into account when calculating the maximum fitness range. Once ascertained, the maximum fitness score can be used to observe how solutions score on the objective, percentage scale.

Models used for kinetic definition optimisation were chosen from the repertoire of complex systems previously utilised with the EPA framework. Without additional forms of qualitative analysis, this choice allowed for experience gained with those systems to guide the development of the framework. This choice also allowed the first simultaneous kinetics and species experiment to be performed in Section 7.2.3 as a stepping stone towards simultaneous GA and GP optimisation in the next chapter.

7.1 BIOCHEMICAL REACTANT-PRODUCT

7.1.1 *Background & Model*

The biochemical reactant-product system is one of the simplest Bio-PEPA models, previously explored during species optimisation work in Section 6.1. It represents the $2X + Y \rightarrow 3Z$ reaction, meaning two molecules of X and one molecule of Y are consumed to produce three molecules of Z. The model was also presented previously in that section in Figure 6.1 and with the Bio-PEPA language description in Figure 2.4.

This model was previously chosen for use with the EPA framework because of the small number of components to work with and the associated limited search space. Starting a new level of work, this model was revisited as a baseline for comparison. Additionally, the experience gained with the system during the species optimisation experiment would assist here.

When working with the species optimisation of X, Y and Z, the search space consisted of 150 data points ($max\_coefficient * num\_operators * num\_species$). However, when considering the search space for the kinetic definition tree, the size is considerably larger. By using the default maximum tree-depth of 5, the maximum size of a kinetics tree is $2^{tree\_depth} - 1$ nodes, which is 31 in this case.

Excluding composite nodes which were implemented after this experiment, each node may have one of five types which affect their value and number of children, as detailed in Section 4.1.1.2. Of these five types, the *m* (math) node may have four values: addition, subtraction, multiplication or division. Furthermore, the *id* node may be any *rate* or *species* inside the system: rate r and species X, Y and Z in the biochemical reactant-product system. Finally, the *byte* node can have any value ranging from $-(2^7)$ to $(2^7 - 1)$ and any non-*formula* node may be repeated in a kinetic definition.

Through combination of these facts, the search space becomes approximately 2^{41} points. The tree-size evaluator, as described in Section 4.3.5, is used to restrain the evolved definitions and avoid bloat.

7.1.2 *Experiment* - alpha

The first experiment performed on the kinetic definition of a Bio-PEPA model flagged the *alpha* action. The target definition of '$fMA(r)$' is simple and consists of only two nodes. This may appear simple but, as described above, the search space is large. This experiment focused on integration of the new code into the EPA framework for kinetics GP definition optimisation.

Unlike the species experiment in Section 6.1, the larger search space of the kinetics optimisation offers a challenge for the original *alpha* definition to be correctly rediscovered.

Table 7.1 presents the simulator and EPA override parameter settings for the biochemical reactant-product model.

Simulator Variables	
Start	0
Stop	99
Data points	100

EPA Override Variables	
Failure timeout	5s

Table 7.1: Biochemical (*alpha*) specific and EPA override parameter values

The correct definition was discovered for this experiment in 40% (20 of 50) of EPA system runs. These correct definitions were an exact match to the *'kineticLawOf alpha : fMA(r)'* structure of the original biochemical reactant-product model. The remaining 60% of EPA system runs all returned the definition as *'kineticLawOf alpha : r'*. The evolved definitions of all results had a fitness score of <0.048%.

This strong set of fitness scores demonstrates that either definition can offer a strong match to the target trace and could offer insight to the user about the mass action function in relation to the biochemical system.

These results demonstrate the EPA framework has adequately been extended to incorporate Bio-PEPA kinetics optimisation. Additionally, the correct definition was evolved in the large search space. Further experiments in this section aim to challenge the EPA with more difficult optimisation problems.

7.2 MEASLES (DENSITY-DEPENDENT TRANSMISSION)

7.2.1 *Background & Model*

The measles virus system has been explored previously in Section 6.2, during which the frequency-dependent transmission model by Benkirane et al. [11] was introduced. This section uses the same measles system but with a density-dependent transmission variant model. Background information on the measles virus can be found with the frequency-dependent version in the previous chapter: the density-dependent model and important differences shall be detailed here.

This is not a commentary on the transmission dynamics of the measles virus; by moving from frequency-dependent to density-dependent transmission the definition of the *contact* action becomes simpler (described below) and offers a more suitable lead in to EPA kinetics optimisation.

Frequency-based contact assumes that individuals inside an epidemiological system will contact a fixed number of other individuals in a given time period, regardless of population density or secondary contacts. Conversely, density-dependent contact assumes that the more

individuals in a system, the more likely a contact will occur. The rate of an infectious contact occurring is dependent on the number of susceptible (S) and infected (Inf) currently in the system.

Figure 7.1 illustrates the density-dependent transmission variant of the measles Susceptible-Exposed-Infected-Recovered (SEIR) model. This differs from the frequency-dependent in two ways. The first is the definition of the *contact* action which is derived from the *rContact* rate and the number of S and Inf: this is commonly referred to as βSI. The second difference is in the recovered (R) species definition; since the number of recovered individuals'a do not influence the *contact* action, the action does not appear in the R definition.

$$rContact = 6.015 \qquad rInfect = 0.133 \qquad rRecover = 0.154$$

$$\begin{aligned} &kineticLawOf\ contact : rContact * S * Inf \\ &kineticLawOf\ incubation : rInfect * Exp \\ &kineticLawOf\ recover : rRecover * Inf \end{aligned}$$

$$\begin{aligned} S &\stackrel{def}{=} contact \ll S \\ Exp &\stackrel{def}{=} contact \gg Exp + incubation \ll Exp \\ Inf &\stackrel{def}{=} contact\ (.)\ Inf + incubation \gg Inf + recover \ll Inf \\ R &\stackrel{def}{=} recover \gg R \end{aligned}$$

$$S[508000] <*> Inf[5] <*> Exp[0] <*> R[0]$$

Figure 7.1: Density-dependent transmission measles SEIR model written in Bio-PEPA

The density-dependent transmission variant of the measles model was selected for use with the EPA framework during kinetic definition experiments because of the simpler form of the *contact* action:

$$\begin{aligned} contact_{\ density} &: rContact * S * Inf \\ contact_{\ frequency} &: (rContact * S * Inf)\ /\ (S + Inf + R) \end{aligned}$$

Without the use of composite nodes, as discussed in Section 4.1.1.2, the density version contains five GP nodes compared to the frequency version's eleven GP nodes. Through use of composite nodes the difference is lessened with two nodes required for the density definition and five nodes required for the frequency version. However, the composite node was not introduced to the EPA framework until the full measles experiment in Section 7.3, after this set of experiments.

The search space of this experiment is slightly larger than that of the biochemical reactant-product model in the previous section. This is because there is an additional species (four instead of three) and two additional rates (three instead of one) which may be used to construct kinetic action definitions. These increases led to the search space of this problem approaching approximately 2^{42} data points.

7.2.2 *Experiment* 1 - contact

The first experiment performed on the density-dependent transmission version of the measles SEIR model flagged only the *contact* action for optimisation. The *contact* action was chosen because it has the most impact on the measles model as well as having the most complex definition of the three kinetic actions in the system.

Simulator Variables	
Start	0
Stop	50
Data points	51

EPA Override Variables	
Failure timeout	10s

Table 7.2: Measles SEIR (*contact*) specific and EPA override parameter values

Table 7.2 presents the simulator and EPA override parameters for this experiment. This is the same as the frequency-dependent transmission measles experiments in Section 6.2.

The correct definition was rediscovered in 10% (5 of 50) of EPA system runs. The remaining 90% of runs resulted in definitions which did not match the target but continued to score strong fitness. The majority of these definitions are illustrated below. Whilst not matching the target definition they are biologically possible; the rate of transmission in the latter two evolved solutions would be markedly high but could provide insight to the user. Solutions are presented in the form they were evolved; the $\rightarrow$ indicates a simplification for readability.

$$
\begin{aligned}
contact_{\ original} &: rContact * S * Inf \\
contact_{\ evolved} &: 7 * S * Inf \\
contact_{\ evolved} &: rContact \\
contact_{\ evolved} &: Inf * S * Inf \ \rightarrow \ Inf^2 S \\
contact_{\ evolved} &: ((S * S * Inf) * S * Inf) \ \rightarrow \ Inf^2 S^3
\end{aligned}
$$

The *'kineticLawOf alpha : 7 * S * Inf '* is particularly close because the actual value of *rContact* is 6.015 which is one *byte* index from the evolved value. A domain expert viewing this output may notice this similarity and adjust the kinetic definition accordingly.

The EPA framework does not have an equality engine which could equate a value close to a rate to be that rate automatically. This was not implemented because it could find equalities where none exist, transforming numbers into rate variables which would not be biologically valid. Further discussion of the equality engine and other EPA potential developments are in Section 9.3.

7.2.3 *Experiment* 2 - contact, S, Exp & Inf

Following the *contact* only experiment described above, this second experiment flagged the *contact* kinetic action in addition to the *S*, *Exp* and *Inf* species. The *contact* action was chosen for

the same reasons as in experiment one, described above, along with the additional experience gained during that experiment. The work done with species optimisation of the frequency-dependent transmission measles model variant, as described in Section 6.2, culminated with the *Exp* and *Inf* species being flagged. In order to challenge the system, the same species were flagged for optimisation, with the addition of the *S* species. Overall, this covers each GP model segment that influences the *contact* action.

The focus of this experiment is the rediscovery of the four flagged sections of the measles model. Additionally, this is the first experiment which combines optimisation of kinetic and species segments, so there is a measure of integration during this experiment. The simulator and EPA override parameter set is the same as for the previous experiment, illustrated in Table 7.2 above.

No singular EPA system run matched the target definitions entirely; however, the returned individuals held a strong fitness score. During experiments in the previous chapter that flagged more than one species segment, it was discussed that the evolved definition for one segment could be reflected in a poorly evolved definition for the other.

Given more time to evolve, the EPA framework could have explored more of the search space to discover the target definition. The last discussion of time restrictions was for the Repressilator experiment in Section 5.4. The time for simulations increased from hours to days as the number of flagged segments increased and the search space becomes larger. Simulations at this stage tended to be approximately three days long.

Viewed individually, the evolved model segments have a stronger fitness than the overall fitness of the solution. The *contact* kinetic action was commonly evolved to be *rContact* alone.

$$contact_{\ original} \quad : \quad rContact * S * Inf$$

$$contact_{\ evolved} \quad : \quad rContact$$

This is not the target definition but still leads to a strong fitness, particularly as a sub-fitness, which is why the same evolved definition appeared in the previous experiment. The evolved definitions of the species segment will influence the evolved solutions for the kinetics segments, perhaps keeping from the target definition being discovered.

Examining the species segments, the evolved solutions were closer to the target definitions. Two EPA system runs resulted in an exact match to the definitions of *S*, *Exp* and *Inf* as seen in Figure 7.1. A further six runs returned exact matches to the *S* and *Exp* definitions while the *Inf* definition omitted only the '*contact* (.) *Inf*' term. The *contact* definition which accompanied these species matches were particularly poor. The fitness function weighted kinetic and species influences equally, therefore this indicates that the species definition has a larger impact on the resultant fitness of an individual.

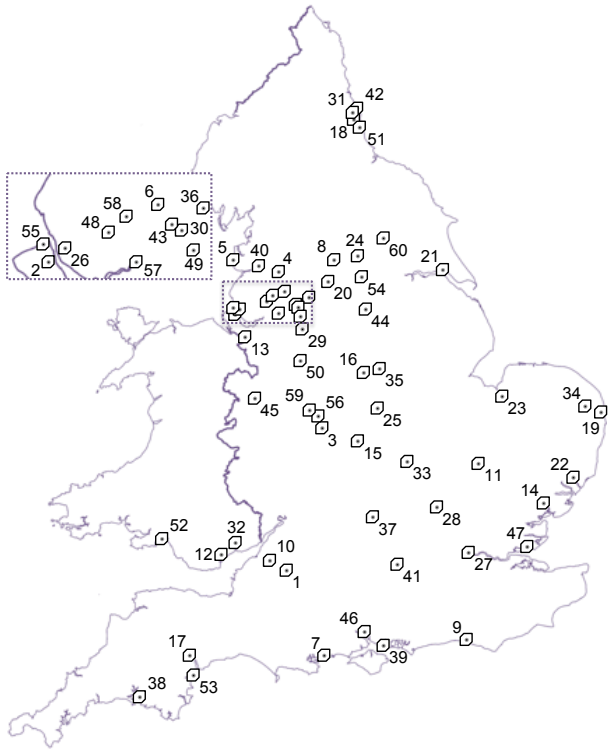
To not bias the optimisation route, domain knowledge was not used. This decision was made because the secondary focus was to test the integration of simultaneous optimisation of the two different GP segments. However, this caused the *recover* action to appear prevalently in

the evolved definitions, just as in the first two experiments of Section 6.2. The experiment could be repeated using domain knowledge, however the results from this experiment were stronger than in the previous chapter and the integration was a success. The domain knowledge will be used in the following section when the measles model is built upon.

7.3 MEASLES (FULL)

7.3.1 *Background & Model*

The background of the measles virus itself has been described previously in Section 6.2. The reason it is of interest to modellers is the complex behaviours which involve recurrent outbreaks in small populations and cyclic outbreaks in larger populations [60], also features such as transmission, population growth, seasonality and immigration. Furthermore, a large set of observational data is freely available for the number of reported cases for measles in England and Wales for over twenty years between 1944 and 1964 inclusive.



#	location	#	location
1	Bath	31	Newcastle
2	Birkenhead	32	Newport
3	Birmingham	33	Northampton
4	Blackpool	34	Norwich
5	Blackpool	35	Nottingham
6	Bolton	36	Oldham
7	Bournemouth	37	Oxford
8	Bradford	38	Plymouth
9	Brighton	39	Portsmouth
10	Bristol	40	Preston
11	Cambridge	41	Reading
12	Cardiff	42	South Shields
13	Chester	43	Salford
14	Colchester	44	Sheffield
15	Coventry	45	Shrewsbury
16	Derby	46	Southampton
17	Exeter	47	Southend
18	Gateshead	48	St. Helens
19	Great Yarmouth	49	Stockport
20	Huddersfield	50	Stoke
21	Hull	51	Sunderland
22	Ipswich	52	Swansea
23	King's Lynn	53	Teignmouth
24	Leeds	54	Wakefield
25	Leicester	55	Wallasey
26	Liverpool	56	Walsall
27	London	57	Warrington
28	Luton	58	Wigan
29	Macclesfield	59	Wolverhampton
30	Manchester	60	York

Figure 7.2: The 60 England & Wales cities in the Grenfell et al. study

Figure 7.2 illustrates the cities where this data is recorded from. The data is recorded fortnightly (once per fourteen days) which also approximates the lifecycle duration of the

measles virus. Fine & Clarkson [31] extensively investigate the measles data; their work includes a hypothesis that extrapolating daily measles infections could lead to finer granularity studies. However, they also disprove this hypothesis, demonstrating that daily infections could be misleading because of complications caused by the weekend-weekday influences of mixing critical school populations.

This same data was studied by Schenzle in attempts to make alternative representations of the standard Susceptible-Exposed-Infected-Recovered (SEIR) model [63]. One presented model included a seasonality based on the school year instead of the calendar year. The school transport and communal classrooms are likened to effective exposure chambers for respiratory-borne pathogens.

Schenzle adjusted the contact behaviour to match outbreaks to academic semesters. Specifically, infectious behaviour is reduced during school holidays and at weekends. Given additional time, the EPA framework would have been applied to this version of the measles SEIR model after the following version, see Section 7.3.3 for further discussion.

$$crs = 3.046 \qquad crw = 6.015 \qquad ir = 0.133 \qquad rr = 0.154$$
$$br = 0.000047 \qquad dr = 0.000047 \qquad imr = 0.0396$$

$$start\_winter = 9 \qquad end\_winter = 4 \qquad month = floor(time/30)$$
$$season\_time = H(((month - 12 * floor(month/12)) - end\_winter) * (start\_winter - (month - 12 * floor(month/12))))$$

$$kineticLawOf\ contact : ((crw * S * Inf)/N) * (1 - season\_time) + ((crs * S * Inf)/N) * (season\_time)$$

$$kineticLawOf\ birth : br * N \qquad kineticLawOf\ die\_S : dr * S$$
$$kineticLawOf\ immigration : imr \qquad kineticLawOf\ die\_Exp : dr * Exp$$
$$kineticLawOf\ incubation : ir * Exp \qquad kineticLawOf\ die\_Inf : dr * Inf$$
$$kineticLawOf\ recover : rr * Inf \qquad kineticLawOf\ die\_R : dr * R$$

$$S \stackrel{def}{=} contact \ll S + birth \gg S + die\_S \ll S$$
$$Exp \stackrel{def}{=} contact \gg Exp + incubation \ll Exp + birth\ (.)\ Exp + die\_Exp \ll Exp$$
$$Inf \stackrel{def}{=} contact\ (.)\ Inf + incubation \gg Inf + recover \ll Inf + birth\ (.)\ Inf + immigration \gg Inf$$
$$R \stackrel{def}{=} recover \gg R + birth\ (.)\ R + die\_R \ll R$$

$$S[16,967] <*> Inf[5] <*> Exp[28] <*> R[491,000]$$

Figure 7.3: Full measles SEIR model written in Bio-PEPA

Figure 7.3 presents the full Bio-PEPA measles model. The Bio-PEPA model is taken from Benkirane et al. [11]. Bokler performed extensive mathematical modelling of the measles virus from an epidemiological and ecological perspective [14]. The seasonality function has been simplified to assume that each month is 30 days long. Also, winter is assumed to last eight months rather than a six-six split to roughly represent academic semesters; this is a coarser representation than Schenzle's fine-grained study.

This version of the measles model contains two aspects that have not previously been tested on the EPA framework. Firstly, the use of variables *month* and *season_time* are a different way of representing rates. Bio-PEPA is capable of defining rates algebraically instead of only numerically (as they are evolved by the EPA framework). Secondly, the H step (*Heaviside*) function used in the definition of the *season_time* variable. This is a form of switch; it allows the model to behave differently in summer or winter time. The result of $H(n)$ is 1 if $n > 0$ otherwise $H(n)$ is 0; in this model, 1 is summer (term time) and 0 is winter (holiday time).

The remainder of the model follows familiar Susceptible-Exposed-Infected-Recovered dynamics as seen in previous epidemiological examples. Susceptible individuals (S) will enter an exposed state (Exp) after participating in the *contact* action. Once exposed to the measles virus, an individual will become infected (Inf) after an *incubation* action is completed. Finally an individual can enter a recovered (R) state after performing the *recover* action. Individuals can enter the system through *immigration* or *birth*. Also, individuals in all four states can leave the system through a death action (*die_S*, *die_Exp*, *die_Inf* or *die_R*).

The measles model starts with 16,967 susceptible, 5 infected, 28 exposed and 491,000 recovered individuals. These figures are taken from the historical data for the city of Leeds (Figure 7.2, Index 24). Leeds was selected because of its midrange size, not being a capital and not being a port city. Capital cites and port cities have a more constant level of reported infections due to their higher levels of immigration. Leeds follows the predicted cyclic behaviour of having a measles outbreak approximately every two years, as illustrated in Figure 7.4.

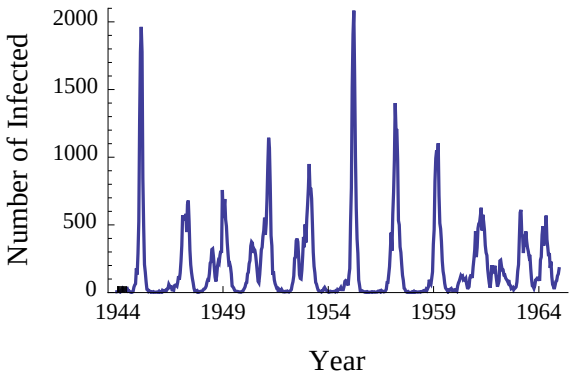


Figure 7.4: The number of measles incidents in Leeds between 1944 and 1964

Looking at the incidence of infection, though the cyclic nature is clear, the peaks do not reach a consistent height. For this reason, experiments with this model use external replications, as described in Section 4.3.2.2. This adds an additional challenge during these experiments; to build a custom fitness function capable of capturing these measles outbreaks. A target fitness trace is not used, because acceptable behaviour allows for these measles outbreaks once every one to three years. Evolved models that cause no outbreaks to occur, too many outbreaks to occur, or outbreaks to occur with a poor periodicity will all be scored a poor fitness.

Figure 7.5 illustrates the pseudo code for the custom measles fitness function. An outbreak is detected through measurement of the area underneath the incidence curve. First, the average area under this curve is calculated. Next, a travelling window scans across the incidence curve to calculate the local area under the curve. The default window size is seven datapoints; of the values between three and nine inclusive, seven correctly identified the most peaks in the Leeds infection data. This window size can be specified by the user. A spike is detected by: $spike = (local\_area > last\_local\_area)$ and $(local\_area \ / \ 2) > average\_area$; if true, a spike is recorded at the time point $((window\_size - 1) \ / \ 2)$ datapoints prior to detection (i.e., the centre of the window).

```
1.  Average a = AverageAreaUnderIncidenceCurve()
2.  Window size w = 7 (by default)
3.  Spikes s = null
4.  for: Each time-series datapoint p after w
5.          if SpikeDetected()
6.                  s = s + Spike(p, w / 2)
7.  if s is empty
8.          return fit
9.  for: Each spike in s
10.         Duration between spikes d = Duration(s_i, s_i+1)
11.         if d < 2 years → Decrement fit by 0.3
12.         else if d < 3 years → Decrement fit by 1
13.         else if d < 4 years → Decrement fit by 0.5
14. return fit
```

Figure 7.5: Pseudo-code for the full measles fitness function

Beyond the need for a custom fitness function, the full version of the measles model offers additional challenges. The number of rate variables and kinetic actions is larger than previous experiments, the complexity of the species definitions is also increased.

The *floor* and *season_time* variables are not included in this list, the EPA framework is not able to evolve the algebraic structure for rate variables. The framework was not extended to do so. This is because the *season_time* definition (and *floor* inside that definition) for calculating the time of year is highly complex and utilises domain knowledge, it would not be expected for this sort of structure to be evolved. Strictly, this sort of structure could be evolved but would require a complex fitness function to incorporate the cyclic behaviour; this could be a possible future experiment, see Section 7.3.3.

7.3.2 *Experiment* - contact

The first experiment performed with the full measles system was to flag the *contact* action alone for optimisation. This is a step back from the simultaneous species and kinetics experiment performed on the abstracted measles system, described in Section 7.2.3. With the sizeable

increase in complexity, this was a prudent step. Both complexity types increase (Section 2.1); the organised complexity increases due to the model description with the inclusion of seasonal and cyclic behaviour and also the disorganised complexity increases with the larger sizes of the starting populations.

Simulator Variables	
Start	0
Stop	7665
Data points	7666

EPA Override Variables	
Fitness	Measles
Internal replications	1
External replications	10
Independent runs	200
Failure timeout	15s

Table 7.3: Measles SEIR (*contact*) specific and EPA override parameter values

Table 7.3 illustrates the simulator and EPA override parameters for this experiment. As discussed with the presentation of the full measles model above, the fitness scoring for this experiment requires a custom fitness function and external replications. The failure timeout is the highest amongst experiments run with the EPA framework; this is due to the increase in organised model complexity as well as the increased number of individuals in the system. Lastly, the number of independent runs is increased because of a larger number of available computer nodes at the time.

The EPA framework was unable to locate an appropriate match to the target behaviour. The strongest evolved individual scored an objective fitness of 44.46% which is orders of magnitude worse than fitness scores discovered in previous experiments. If using a Euclidean distance fitness function, this would indicate that almost half of the datapoints in this trace of the evolved individual were unfit.

The measles experiment uses the *MeaslesEvaluator* which records spikes, the maximum number of spikes is dependent on the window size, as described above. Under these conditions, the objective fitness score is better at 40.44%. The actual worst fitness score is positive infinity which indicates a violation of the evaluator requirements. No EPA system run resulted in such a failed individual.

The best evolved individual of the 200 EPA runs is illustrated below alongside the original target *contact* definition. Even at first glance, these two definitions are markedly different:

$$\begin{aligned} contact_{\ original} \quad : \quad & ((crw * S * Inf)/(S+E+I+R)) * (1 - season\_time) \\ & + ((crs * S * Inf)/(S+E+I+R)) * (season\_time) \end{aligned}$$

$$\begin{aligned} contact_{\ evolved} \quad : \quad & S^{S^{-2crs^{Exp}+Inf\,(\,-crw+Inf\,(\,Exp+Inf+R+S\,)+Exp+Inf+R\,)+Exp+7\,5Inf+R-S^{rr}+S}} \\ & + \left(crs^{Exp}\right)^{rr} - Exp^{Exp} + 2Exp - 57Inf + 2R - 52^{S^{rr}} \end{aligned}$$

The evolved definition makes extensive use of the *Power* (x^a) math node despite an increased penalty being applied to it through the node-weight fitness function (Section 4.3.6). The *Power* node is scored equivalent to a *Composite* node: 2.5x normal and 5x other *Math* nodes, see Figure 4.3. This increased penalty for *Power* nodes was implemented because of their

prevalence in previous SEIR experiments, whereas the operator is rarely utilised in target definitions. This increased penalty reduced the number of *Composite* nodes in the evolved solutions, however the *Power* node is still frequently used.

This experiment is considered a failure because none of the 200 EPA system runs could correctly evolve a close match to the target definition. A small measure of success can be taken from zero of these runs resulting in a solution which violated the *MeaslesEvaluator* requirements to cause at least one measles outbreak.

Given the time restrictions, no further work was done with the full measles model. This experiment still yielded important developments to the EPA framework as well as knowledge in the domain, as discussed in the following section.

7.3.3 *Results Discussion*

Given more time, the EPA framework would have been challenged with the normal progression of experimentation. Starting with *contact* flagged for optimisation alone and then building towards more segments being optimised together especially from different portions of the Bio-PEPA model, as with previous experiments to develop the framework.

Beyond this normal progression, original plans for the full measles system included a series of experiments to utilise more of the expansive data set available for the measles virus. The model presented in Figure 7.3 is tuned to the Leeds dataset as illustrated in Figure 7.4. After successfully evolving a definition based on this dataset, the intention was to verify and validate the evolved model on cities of similar size by predicting their outbreaks.

The comparison of other features of the dataset was also considered, such as cities with defining characteristics (ports) or distance components. Grenfell et al. studied the 'travelling wave of infection': neighbouring cities will suffer an outbreak shortly after a source city suffers an outbreak due to immigration. Due to difficulties with the *contact* experiment and imposing time restraints these planned experiments were not conducted.

Despite this setback, the full measles system was important for developing the EPA framework. The first significant expansion to the framework was the development of the composite node (*c* as discussed in Section 4.1.1.2). Consider the definition of the *contact* action which is constructed of 33 nodes. This is a representation of the target solution in the EPA framework GP node structure, with and without composite (*c*) nodes:

$$
\begin{aligned}
contact_{\ original} \quad &: \quad ((crw * S * Inf)/(S+E+I+R)) * (1 - season\_time) \\
&\qquad\qquad +((crs * S * Inf)/(S+E+I+R)) * (season\_time) \\
contact_{\ nodes} \quad &: \quad ((r\,m\,s\,m\,s)\,m\,(s\,m\,s\,m\,s\,m\,s))\,m\,(b\,m\,r)\,m\,((r\,m\,s\,m\,s)\,m\,(s\,m\,s\,m\,s\,m\,s))\,m\,(r) \\
contact_{\ nodes \setminus wc} \quad &: \quad (c\,m\,c)\,m\,(b\,m\,r)\,m\,(c\,m\,\ c)\,m\,(r)
\end{aligned}
$$

Where *r* is a rate node, *m* is a math node, *s* is a species node and *b* is a byte node. Firstly, this is over the 32 node limit for a depth five tree (the EPA framework default), increasing the

depth to six would allow 63 nodes and cause more bloat in solutions. By incorporating the composite nodes both the complexity and size of the solution is reduced, the target definition now contains 13 nodes which could be recreated in a depth four tree.

Even through use of the composite nodes, the tree size is still sizeable compared to previous experiments; biochemical-reactant product '$alpha \ : \ fMA\ (r)$', two nodes; and the density-dependent measles system '$contact \ : \ rContact * S * Inf$', five nodes. The penalty applied by the GP tree-size evaluator fitness function would apply a higher bloat penalty because of the definition size and could pressure the EPA framework into evolving smaller definitions. Understanding that smaller does not always result in fitter solutions, the GP node-weight evaluator fitness function was developed, as described in Section 4.3.6. This development allowed for kinetic action definitions to be scored based on sensible use of nodes instead of size alone. The individual weights can also be adjusted by the user to guide the course of evolution.

During this experiment, all existing fitness functions were modified so they could be applied in serial; i.e. multiple fitness functions could be applied to one solution. This provides further choice to the user of the EPA framework by allowing any number fitness functions to be combined. The penalties applied by fitness functions are done so additively, so the order in which fitness functions are applied is irrelevant to the final fitness score. This serial design of fitness functions is in keeping with the ECJ toolkit's design.

7.4 KINETICS STRUCTURE OPTIMISATION EXPERIMENTS SUMMARY

This chapter presented the application of Genetic Programming (GP) optimisation techniques to Bio-PEPA models with the goal of evolving fitter kinetic action definition structures. Bio-PEPA models used during this chapter have been encountered in previous chapters; they are complete and had some or all kinetic definitions removed to be rediscovered during optimisation. By using complete models, evolved solutions can be compared to the original Process Algebra (PA) model definitions. To further test the Evolving Process Algebra (EPA) framework, some kinetics experiments were run with kinetics and species flagged for optimisation simultaneously.

The performance of the EPA framework during the GP experiments of this chapter was weaker than in previous experiments. The level of organised complexity (Section 2.1) increases when working with kinetic actions. This stretched the abilities of the Euclidean distance fitness measure. The need for custom fitness functions is discussed in Section 9.2.

Original plans for the EPA framework during this chapter included more experiments than were accomplished in the allotted time. Important developments included the composite node *c* (Section 4.1.1.2), the node-weight evaluator fitness function (Section 4.3.6) and the serialisation of the fitness functions for scoring based on multiple criteria simultaneously.

8

PARAMETER MATCHING & STRUCTURE OPTIMISATION

Work detailed in previous results chapters focused individually on Genetic Algorithm (GA) parameter matching followed by Genetic Programming (GP) structure optimisation, the focus then moved to combining the two approaches to function simultaneously. The Bio-PEPA Process Algebra (PA) continued to be used exclusively during this stage of work, however the EPA framework remained capable of simulating PEPA models.

Though not mutually exclusive, using Genetic Algorithms and Genetic Programming techniques simultaneously has inherent difficulties which must be overcome. Consider a species definition S that can take action A at rate R. If R evolves a zero or near-zero value, A's impact on the individual's resultant fitness will be zero or negligible, despite A's definition. In resultant generations, S could have the entire definition of A removed, along with any latent behaviour associated with A. While not necessarily incorrect, this interaction between GA parameter matching and GP optimisation techniques could lead to failed individuals or premature convergence on local optima.

During this transition, it was a simple step to extend the GA parameter matching behaviour to the system equation values. The EPA framework is able to evolve the starting population levels (also called initial conditions) of a system. Many systems will have known starting values, rendering this extension redundant. However, some systems may have a snapshot state of a system and require working backwards to the starting population levels, allowing domain experts insight into the system.

Model specific organised complexity is more important than with previous work. The changes that simultaneous GA parameter matching and GP model optimisation techniques bring can have a large impact on the resultant model's behaviour. One must not overpower the other; in addition to balancing exploration and exploitation, the GA and GP influences must also be balanced.

8.1 BIOCHEMICAL REACTANT-PRODUCT

8.1.1 *Background & Model*

The biochemical reactant-product system is one of the simplest Bio-PEPA models, previously explored in Section 6.1 for species optimisation and again in Section 7.1 for kinetic action optimisation. It represents the $2X + Y \rightarrow 3Z$ reaction, meaning two molecules of X and one

molecule of Y are consumed to produce three molecules of Z. The model was also presented previously in Figure 6.1.

This model was previously chosen for use with the EPA framework because of the small number of components to work with and the associated limited search space. Starting a new level of work, this model was revisited for the same reasons.

When working with species optimisation, the search space consisted of 150 points (Section 6.1). Later, when working with kinetics optimisation, the search space consisted of approximately 2^{41} points (Section 7.1). The size of the search space for rate optimisation is dependent on the number of rates flagged for optimisation, the permissible range for these rates and their granularity.

The EPA framework is capable of storing *double* values to sixteen decimal places. Likewise, the search space for population optimisation is dependent on the number of population values flagged for optimisation and their permissible ranges. The size of the search space is larger when working with simultaneous optimisation as it will combine the search spaces of two or more of the search spaces described above.

8.1.2 *Experiment* - all segments

The first experiment performed for the simultaneous GA and GP definition optimisation flagged all segments of the biochemical reactant-product model: one rate, the release rate r; one kinetic action, *alpha*; three species, X, Y and Z; finally, three initial populations for these species. As noted above, the search space for this experiment is sizeable. In practice, it would be unlikely that an EPA framework user would flag every evolvable segment of a model simultaneously. A domain expert would have some knowledge of a system's function that could be defined in a model. However, this experiment focused on integration of the code required to simultaneously utilise the GA and GP aspects of the EPA framework and the underlying ECJ toolkit.

Unlike the species experiment in Section 6.1, but similar to the experiment in Section 7.1, the large search space offers a challenge for the original definition segments to be correctly rediscovered.

Simulator Variables	
Start	0
Stop	99
Data points	100

EPA Override Variables	
Fitness	Euclidean
Failure timeout	7s

Table 8.1: Biochemical (*alpha*) specific and EPA override parameter values

Table 8.1 presents the simulator and EPA override parameter settings for the biochemical reactant-product model. The model specific settings are unchanged from previous experi-

ments. The Euclidean distance fitness function is used, this keeps the focus on integration of simultaneous optimisation into the EPA framework.

Considering the size of the search space, it is unsurprising that the target definition was not rediscovered in its entirety. The EPA system runs returned solutions ranging from 0.40% (best) to 1.21% (worst) objective fitness scores; these scores are not as strong as previous biochemical experiments but appear to be reasonable matches to the target trace. In actuality, the traces do not exhibit the expected reduction in X and Y and increase in Z species levels. The Euclidean distance fitness measure matches enough of the datapoints to consider the results to be a strong match despite this.

Starting with rate r, there was no consistent answer across the solutions from the 50 EPA system runs. The solutions were distributed across the available range with a dip in the 2.5 to 3.5 range, as illustrated in Figure 8.1.

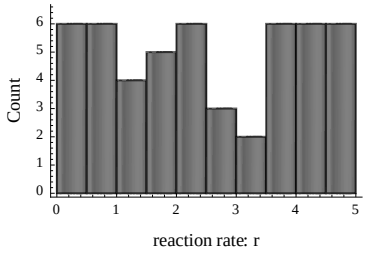


Figure 8.1: Histogram plots for biochemical reactant-product: r

The lack of consistency in the histogram can be explained by the use of rate r. Normally the biggest influence on the course of evolution is the fitness function. However, the use of the rate r in a kinetic action definition will have a bigger influence. Consider a kinetic action being flagged for optimisation which uses a rate, how that rate is utilised will vary from individual to individual. This will lead to an inconsistent spread, as observed in Figure 8.1.

The optimisation of the kinetic action *alpha* was far more successful. The most frequently evolved definitions used the mass action function on rate r or the rate alone:

$$\begin{aligned} alpha_{\,original} &: fMA\,(r) \\ alpha_{\,evolved} &: r \\ alpha_{\,evolved} &: fMA\,(r) \end{aligned}$$

Curiously, another commonly evolved solution uses the '$r\,m\,r$' structure, where m is a math node (+, -, / or *). The selected math operator caused the accompanying evolved rate to approach the original target rate. This could indicate the optimisation was attempting to recreate the 'r' definition as resulted from other EPA system runs.

Continuing on to the evolved species definitions, these are the strongest segment of this experiment. This could be explained by the comparatively small search space for species compared to the other segments. However, should the other segments evolved around the species result in a poor fitness score, the target definition may be evolved but not be selected

for. The target X definition was rediscovered in 70% (35 of 50) and the target Y definition was rediscovered in 88% (44 of 50) of EPA system runs, therefore both correct definitions dominate their respective species analysis.

The target definition for Z was rediscovered only in 18% (9 of 50) of runs, Figure 8.2 illustrates the Z species analysis. It would be expected that the rediscovery rate of species Z would be higher considering the success with the X and Y species. However, the starting populations (initial conditions) of the species will influence the definitions evolved. This is because of the how the Euclidean distance fitness function operates. Instead of X and Y levels lowering to produce Z, Z levels could start high due to the optimisation of initial conditions. The aim of the function is to match the species populations at each time step; if the starting population for Z starts high, the majority of the Euclidean distance score will be strong.

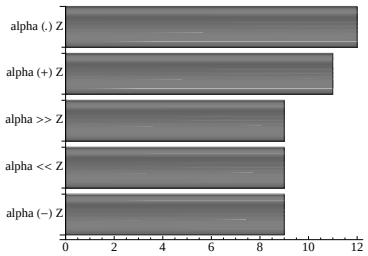


Figure 8.2: Species analysis for biochemical reactant-product: Z

The evolved values of the initial conditions was the weakest part of this experiment. Ideally the original values ($X = 200$, $Y = 100$, $Z = 0$) would have been rediscovered; if not that, then the ratio of 2:1:0 instead. However, the EPA runs did not return any such consistency. Given more generations in the system run, it could be possible to evolve this pattern. Commonly, this sort of desired behaviour would be selected for by the fitness function; however, the Euclidean distance fitness function was used in this experiment for the focus to remain on integration.

8.2 KINETICS STRUCTURE OPTIMISATION EXPERIMENTS SUMMARY

This chapter presented the simultaneous application of Genetic Algorithms (GAs) and Genetic Programming (GP) optimisation techniques to Bio-PEPA models with the goal of evolving rate and population values in tandem with kinetic action and species definitions. Only a single experiment was performed during this chapter with the emphases on ensuring correct operation with all segments flagged.

The integration of simultaneous GA and GP optimisation was a success. It is unlikely that a domain expert would flag every evolvable segment. However, if done, the framework can still provide general insight to the user. The more domain knowledge provided, the more specific the EPA framework's results will be. Moving forwards with simultaneous optimisation, more encompassing fitness functions will be required to guide the path of evolution.

9

CONCLUSION

Through the course of this research, the combination of Evolutionary Computation (EC) techniques and Process Algebra (PA) modelling have been combined into the Evolving Process Algebra (EPA) framework. The framework supports two Process Algebras for model descriptions: Performance Evaluation Process Algebra (PEPA) and the more Biologically-Inspired PEPA (Bio-PEPA). The EPA implements Genetic Algorithms (GAs) for parameter value fitting of rate and/or initial population levels. In addition, Genetic Programming (GP) optimisation techniques are used to evolve the structures of Bio-PEPA models for fitter species and/or kinetic action definitions.

The EPA framework is built on top of the Evolutionary Computation Java (ECJ) toolkit to make use of the extensive library of standard EC operators. This includes the framework of a population of individuals spanning multiple generations, as well as the tournament selection, one-point crossover and generational replacement operators as described in Chapter 2.

The EPA framework was successfully extended with many additional features; extension was done in an abstract, modular way in keeping with the ECJ toolkit's design. Through abstraction, implemented extensions may be further extended for minor changes. Modularity allows for entire sections to be swapped as required by the user; for instance, custom fitness functions.

Further important developments and contributions include the GP tree structures to represent Bio-PEPA species and kinetic action definitions. Due to the strict nature of species definitions, the EPA framework had to restrict crossover functionality to identical edges in parents (Section 4.3.10.2). Initially, the kinetic action definition experiments performed poorly in comparison to previous experiments. The EPA framework was extended with novel methods of improving kinetic experiments: the point kinetic mutation operator (Section 4.3.10.3), incorporating domain knowledge through ban lists (Section 4.3.1), adaptive depth bias (Section 4.3.10.5) and kinetic rebreeding (Section 4.3.10.4).

Pertinent to all EPA experiments, the framework also attempts to reduce the number of known failures before wasting time on simulation, thus increasing validity of results (Section 4.3.11). Finally, the EPA framework novelly handles external replications to reduce stochastic noise in cyclic systems while simultaneously reducing obfuscation of system behaviours (Section 4.3.2.2).

The remainder of this chapter is split into three sections. Firstly, in Section 9.1, the progress of the work presented with this research is compared to the original thesis statement proposed

at the work's commencement. Secondly, in Section 9.2, a discussion of the requirements of custom fitness functions for higher complexity problems while considering the No Free Lunch (NFL) theorem [69]. Finally, possible improvements and extensions to the EPA framework and *EPA Utility Assistant* are detailed in section 9.3.

9.1 THESIS STATEMENT

The primary aim of this research was the development of a bridging framework to combine Evolutionary Computation techniques with Process Algebra models with the goal of assisting domain experts. The development of the EPA framework is the result of this and has successfully applied GA and GP techniques to models written in PEPA and Bio-PEPA.

The EPA framework was to include prebuilt constructs to improve experimental results in the domains of biological and epidemiology. The composite node (*c*, as described in Section 4.1.1.2) successfully fulfils this role by modelling commonly appearing system qualities. This includes density-dependent transmission, frequency-dependent transmission and the sum of individuals in a population.

To assist in the use of the EPA framework, a command-line entity, a user interface needed to be created. The *EPA Utility Assistant*, as detailed in Appendix A, provides two forms of support in using the framework. This is a complimentary program that successfully provides a user interface for the domain expert to use the EPA framework.

9.2 CUSTOM FITNESS FUNCTIONS

Towards the latter experiments conducted in the course of this research, it became apparent that basic or 'catch-all' fitness functions would not suffice to accurately capture all of the target behaviours in an evolved Process Algebra definition. The fitness functions in the EPA framework are modular in design and can be applied in serial. This serialisation allows for multiple fitness functions to be applied to one individual to score additively on multiple criteria. The modularity allows for the simpler fitness functions (such as Euclidean distance measure fitness function 4.3.4) to be built upon to satisfy model specific requirements.

The No Free Lunch (NFL) theorem was discussed in Section 3.4. To reprise, NFL posits that all optimisation algorithms that search a problem space have a cost for doing so. When averaged across all possible optimisation problems, all algorithms of a given cost perform equally well. For an algorithm to improve its performance to solve a particular problem, it must become less suited to solving problems in other areas; i.e. the cost must be paid from its performance to solve other problems.

The Euclidean distance measure fitness function is all-purpose so does not invest cost into any particular domain. Consider the GP tree-size fitness function (Section 4.3.5) which scores

solutions based on their size to reduce bloat. This fitness function specialises by forfeiting performance in solving non-GP tree problems in order to score individuals based on their definition size.

It's impractical to assume one fitness function will be equally successful for all problems it is applied to. The EPA framework and the included fitness functions do not promise to solve all problems. The framework has a collection of fitness functions designed to offer a user strong scoring solutions to a problem which may not necessarily be the global optimum.

Due to the size of search spaces associated with complex problem, finding the global optimum to a problem becomes intractable. Custom fitness functions can alleviate some of this difficulty with the associated cost of specialisation. Particularly complex problems in this research have warranted custom fitness functions: the Repressilator system (Section 5.4) and the final measles system (Section 7.3).

The fitness function is one of the major influences on the course of evolution. Even a well-considered and properly implemented custom fitness function can lead to unexpected outcomes, this is due to EC techniques and their propensity to exploit shortcuts. Care must be taken when creating a custom fitness function to avoid such shortcuts; for instance, the lack of a maximum protein expression limit in the Repressilator fitness function.

If creating a fitness function is so tricky, the question could be asked as to why creating one is worth the effort, such effort could be put into writing Process Algebra models by hand. By writing a model by hand, the result will entirely be the writer's intuition. Using an assistive tool such as the EPA framework allows for the search space to be explored and provide possible insights to the user. A custom fitness function allows for a middle ground of searching a problem space and garnering insights whilst also incorporating a domain expert's intuition and knowledge.

9.3 POTENTIAL DEVELOPMENTS

The EPA framework was developed for this research to apply Evolutionary Computation techniques to models written in Process Algebras to assist domain experts in modelling. As discussed with the thesis statement in Section 9.1, this goal has been successfully achieved.

However, through the course of this research possible improvements have arisen. Not all such improvements could be implemented in the given timescale, the remainder of this section discusses possible future developments which could improve the EPA framework.

9.3.1 *Supported Representations*

The first obvious extension to the EPA framework would be to increase the number of supported representations. Due to the modular design of the framework, one interface class is responsible

for interacting with the model representation. This choice attempts to make future extension easier. GA operators for optimisation rate variables would function immediately, whereas GP techniques would require a tree structure to be defined.

Beyond Process Algebras, the first choice of representation would be Ordinary Differential Equations (ODEs). The application of GP techniques to ODE models is discussed with the related work in Section 3.3.1. From this discussion the approach is feasible, incorporating it into the EPA would allow greater flexibility for the user.

9.3.2 *Smarter Ban List*

The action ban list is a function designed to utilise domain knowledge when working with species GP definition trees (Section 4.3.1). The ban list functions by associating a species identifier with a list of kinetic actions that it cannot utilise during optimisation.

During the frequency-dependent transmission measles model experiment (Section 6.2), this version of the ban list revealed a weakness. Due to how the Euclidean distance fitness function scored individuals in that experiment, the results became saturated with $Inf \stackrel{def}{=} recover \gg Inf$. This means the recovery action increases the number of infected individual; this is counter-intuitive and biologically inaccurate. However, the *recover* action could not be added to the ban list for the infected species because the action is required but with a different operator: '$recover \ll Inf$', so recover reduces the number of infected.

The solution to this problem would be reworking the ban list to increase its flexibility. Instead of pairing species identifiers with a list of kinetic actions, the species would be paired with a list of kinetic actions which in turn would be paired with Bio-PEPA operators. Only the combination of all three would be restricted, therefore other combinations would still be permissible.

9.3.3 *Equality Engine*

The equality engine was first mentioned in the density-dependent transmission measles model which flagged the *contact* kinetic action (Section 7.2). An equality engine would operate on kinetic GP definition trees by associating raw numbers (*byte* nodes) with rate values which closely match. If the number is within a user-specified tolerance, the *byte* node would be altered to a *rate* node defined as the matched rate.

The benefits of such an engine include assisting the optimisation of definitions which attempt to recreate a rate variable with a number variable. Additionally, this could also reduce the size of the kinetic definition tree. The decimal 1.5 (represented as 3/2) consists of three nodes (*byte math byte*). If this value was replaced with a rate variable, the tree size would reduce from three nodes to one, helping to alleviate bloat in kinetic GP definition trees.

A negative side-effect of this could be false equivalencies being discovered and the *rate* nodes causing poorer solutions to be created by interrupting the normal course of optimisation. This problem would be compounded if combined with the algebraic simplification engine, as described in the following subsection. Despite these negatives, the equality engine could improve fitness scores in kinetics problems if used with care, similar to the utilising domain knowledge feature (Section 4.3.1).

9.3.4 *Algebraic Simplification Engine*

The algebraic simplification engine was contemplated from the beginning of kinetic action definitions because of their likelihood to bloat. This could be a possible solution to reduce bloat as well as make the final evolved definition more human readable. Consider an example kinetic action with a simple but repetitively evolved structure:

$$\textit{contact} \quad : \quad S + S + S \quad (5\,\text{nodes}) \quad \rightarrow \quad 3 * S \quad (3\,\text{nodes})$$

The algebraic simplification engine would reduce the number of nodes from five to three. This is a simplified example but is still useful. The most extreme example would affect a kinetic action definition with a lot of *byte* nodes:

$$\textit{contact} \quad : \quad \text{<a set of } \textit{math} \text{ \& } \textit{byte} \text{ nodes>} * 0 \quad (n\,\text{nodes}) \quad \rightarrow \quad 0 \quad (1\,\text{node})$$

In this example the number of nodes is reduced by n-1. Whilst making kinetic GP definition trees smaller and more readable, this example illustrates the biggest negative of a algebraic simplification engine: by refactoring portions of the GP tree, the number of possible nodes available for breeding operators (Section 4.3.10.3) is reduced. This could cause mutator operators to have a larger effect than expected and could cause crossover operators to move smaller but more influential subtrees.

When discussing the equality engine above, it was noted that combining the equality engine with the algebraic simplification engine a problem could arise. If a *byte* node (or collection of *byte* nodes) is equated to a rate variable, that rate variable could then be regressed into a simpler form. This change could hide or entirely remove the traits which made the original individual desirable. Similar to the equality engine, if implemented, care should be taken if using the algebraic simplification engine.

9.4 MOVING FORWARDS

During the course of this research, the two main software developments have been the Evolving Process Algebra (EPA) framework and the *EPA Utility Assistant*, introduced and detailed in Chapter 4 and Appendix A respectively. The previous section discussed possible

extensions to both to provide more possible functionality to domain experts; who these experts are and how they would utilise the EPA framework are important questions.

Through the experiments conducted in this work, particularly in GP optimisation experiments, the focus was on biological and epidemiological domains. The EPA framework remains generic enough to be applied to many other domains and its modular design would allow for domain-specific changes to be incorporated. Experiments performed focused on integration and testing of features and how well the framework was able to rediscover removed segments of complete models. Interactions and input from domain experts at this point have been low; work to this point could be considered a preparatory step before focusing on delivering the EPA framework's functionality.

The future of this work is in modelling unknown systems to aid domain experts in the field. This would be done working in tandem with those experts. One avenue considered was working with local fish farms in modelling parasite spread in and between tanks. Work would proceed collaboratively to develop the functionality of the EPA framework and the interface of the *EPA Utility Assistant*.

The final goal of the EPA framework would be to have its functionality and customisability entirely accessibly to the domain expert. This would be without any requirement of Java libraries, knowledge of ECJ or either Process Algebra plug-ins, or the input of a coder.

APPENDIX A: EPA UTILITY ASSISTANT

The *EPA Utility Assistant* was first described in Chapter 4 with the implementation details of the EPA framework and its relation to the *HTCondor* system as used for parallelisation. HTCondor (called *Condor* until 2012) is a high throughput computing toolkit maintained by the University of Wisconsin-Madison and is designed to distribute experiments over multiple available nodes [66]. HTCondor and the EPA framework are both designed to run at the command-line level and so have no graphical user interface.

The EPA framework is designed to assist domain experts in modelling complex systems and as such should require no knowledge of command-line interfaces or HTCondor's operation. The *EPA Utility Assistant* is designed to bridge the gap between the domain expert and the backend functionality of the EPA framework plus the parallelisation of the HTCondor toolkit. Since the majority of functionality offered by the framework is for Bio-PEPA models this is what the assistant program supports.

This assistant program is a simple graphical user interface split into two tabs. The *EPA Condor Assistant* is designed to assist with preparing experiments and posting them to HTCondor nodes. The *EPA Results Collator* is designed to collate the various results files produced by the framework from across those nodes. Both tabs are discussed in detail over the following two sections.

A.1 EPA HTCONDOR ASSISTANT

The HTCondor toolkit offers the ability to parallelise EPA system runs and thus allowing to significantly cut down on the time requirements of Process Algebra simulations. The downside of this approach is the setup required by the HTCondor toolkit by the user. Additionally, the EPA framework and ECJ toolkit both also require setup. The following files are required for a typical experiment:

EPA FRAMEWORK: EPA archive (.jar), Bio-PEPA model (.biopepa), target trace (.txt)

ECJ LIBRARY: ECJ archive (.jar), Parameters (.params)

HTCONDOR: Submission (.sub)

The HTCondor toolkit does not support a multilayer folder structure, so all files for a particular experiment must be stored in one folder. The java archive (.jar) files for the EPA framework and ECJ toolkit must also be included. The EPA framework requires the Bio-PEPA

model: this may be complete or partially complete, as described in Section 4.3. Normally, a target trace is required which encapsulates the target behaviour. The EPA framework may not require a target trace file, depending on the fitness function used; for instance, the *repressilator* system (Section 5.4) cannot be matched to a trace due to the cyclic and unpredictable properties of the system, therefore no trace file is required.

The ECJ toolkit accepts user specified variable settings through parameter files. In keeping with this format, the EPA framework uses parameter files to take in parameter values: the default parameter set is illustrated in Section 4.2.

Finally, the HTCondor toolkit requires a submission file which lists all files required for an experiment along with the behaviour of the HTCondor nodes and their outputs. The behaviour includes the memory allocation, operating system requirements and the number of nodes to post separate experiments to. How the results are returned and processed is discussed in Section A.2.

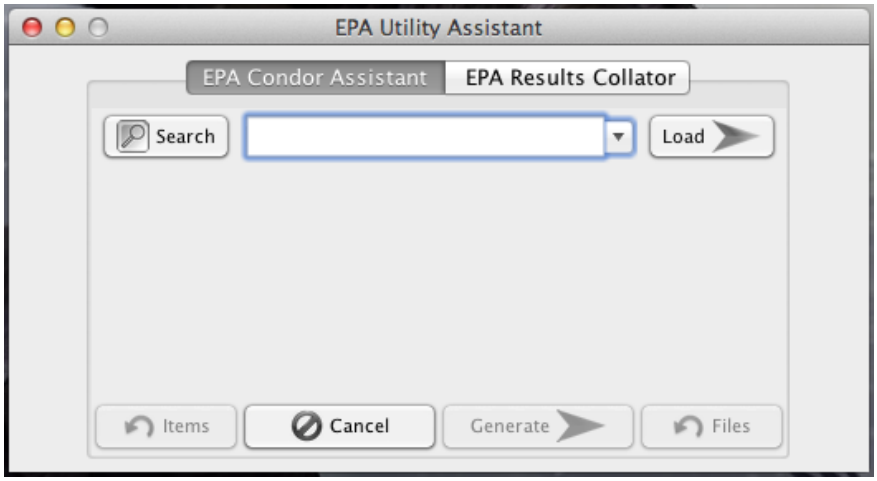


Figure A.1: The *EPA Utility Assistant* - HTCondor assistant view

Figure A.1 illustrates the *EPA Utility Assistant* when first launched in the default HTCondor assistant tab. The first step in using the assistant is to select a complete or partially complete Bio-PEPA model file (.biopepa). This can be done by manually typing the file location in the input field or using the search function. Additionally, the input field has an autocomplete function (triggered by *ctrl + space*) which acts as a search if utilised when blank. If multiple Bio-PEPA files are found, they will be listed in the drop-down portion of the input field: this list persists until the assistant is reset so further experiments can be created after the first without having to search again.

Once a Bio-PEPA file has been selected, the model will be scanned and the individual components will be identified. Figure A.2 illustrates the component selector window of the assistant. The identified components are listed vertically by type and sorted in the order that they spprst in the Bio-PEPA model. Sorting this way allows for components to be found quickly by those familiar with the model. In this example, the release rate r and species definitions X,

Y and *Z* from the biochemical reactant-product (Section 6.1) example have been flagged for optimisation.

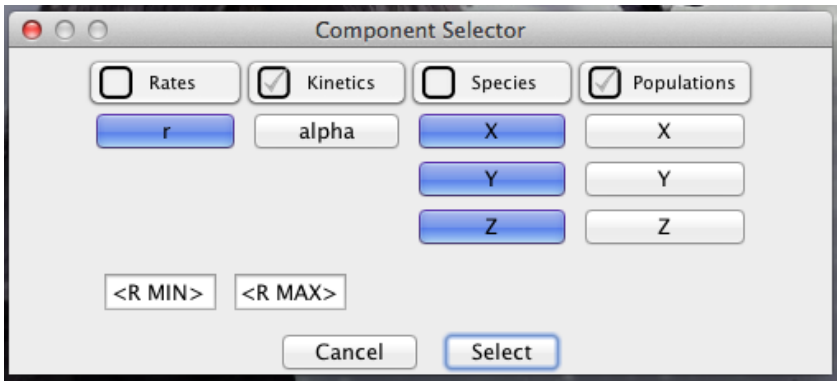


Figure A.2: The EPA *Utility Assistant* - component selector view

If a rate or population is flagged for optimisation, the minimum and maximum bounds for that segment need to be specified: the <*R MIN*> and <*R MAX*> fields in this example. The bounds of kinetic and species definitions can be inferred from the Bio-PEPA model. Note, this simplified view assumes the permissible range for each rate is the same to not overcomplicate the interface. If this assumption is not true, the ranges can be specified individually in the parameter selector.

Once the components to be flagged have been chosen, the experiment parameters need to be defined; Figure A.3 illustrates the parameter selector interface. By default, the file name is the same as the Bio-PEPA file from which the model was scanned, the default target name appends "Target.txt" and the project folder name appends the flagged components. The target trace is assumed to be required and in the same folder as the Bio-PEPA model; unchecking the trace box removes this requirement the search feature can be used to find a trace located elsewhere.

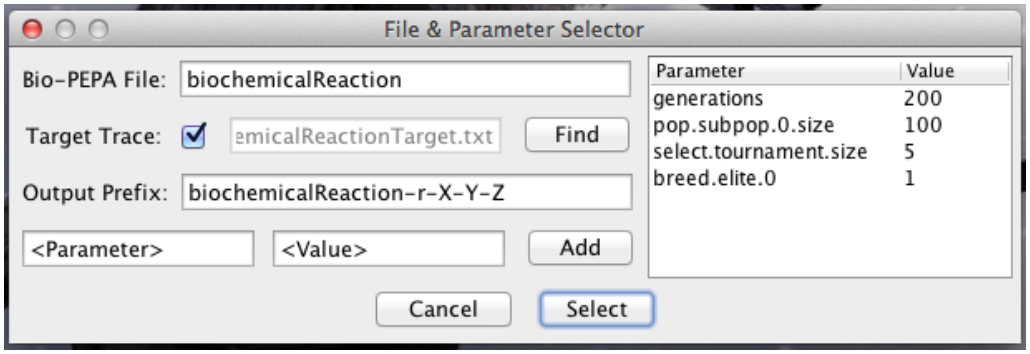


Figure A.3: The EPA *Utility Assistant* - file parameter selector view

The <*Parameter*> - <*Value*> paired input fields allow for the user to specify any parameter, these will then appear in the table to the right-side. This feature is aimed at users with a higher knowledge of the EPA framework and/or ECJ library, the format of the parameter names

is highly specific: some examples are given in the example table. This can be done to specify new parameters or override existing ones; if a parameter is specified multiple times, the most recent occurrence will take precedence. Inside the *EPA Utility Assistant* structure there is a file which contains a list of default parameters and their values. This file can be appended or changed if the user frequently uses a particular set of parameters.

After the Bio-PEPA model, flagged components and any parameters have been chosen, the final stage of the HTCondor assistant is the file generation, as illustrated in Figure A.4. Before generating files any choices made can be amended, the system can go back to the search result list or the assistant can be reset entirely.

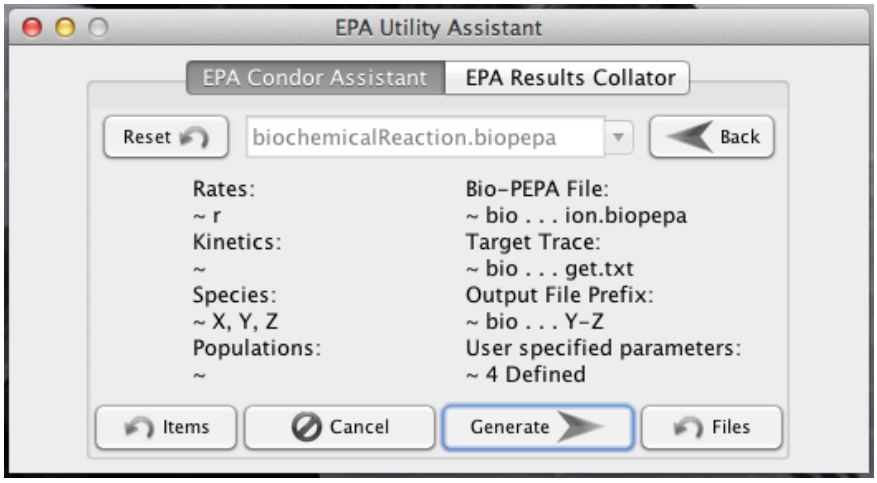


Figure A.4: The *EPA Utility Assistant* - HTCondor assistant ready view

Once finalised, all files required for an experiment are generated in a new folder. If the folder name is already taken, an increasing numeral is appended until the name is available. In addition to the required files, a read-me file is generated with detailed directions in how to submit the experiment to the HTCondor nodes.

A.2 EPA RESULTS COLLATOR

The second half of the *EPA Utility Assistant* is designed to assist the user with results collation. The output from one HTCondor node is four results files: as discussed in Section A.2.1. The collator extracts the typically most useful aspects of the outputs and creates up to three (four with species optimisation) final results files, see Section A.2.2.

By default, an optimisation experiment is run independently on 50 HTCondor nodes which equates to 200 raw results files. The collator was originally developed to reduce the amount of transfer time required to pull results from a remote server: reducing the number of files from 200 to three. The key usefulness of the collator is in automating the post-experiment result processing, which is why it was added to the *EPA Utility Assistant*.

Figure A.5 illustrates the *EPA Utility Assistant* when first switched to the results collation tab. The first step is to select a set of results to collate, this is done by choosing a filename prefix. The standard EPA framework output has each result file prefixed with a specified name, thus allowing results from a particular experiment to be identified. Similar to the HTCondor Assistant tab, this input field has an autocomplete feature which functions as a search if performed on an empty field. If more than one set of results is found, the drop down list will show them all. The default output prefix is the same as the input prefix and the number of files found is provided as well.

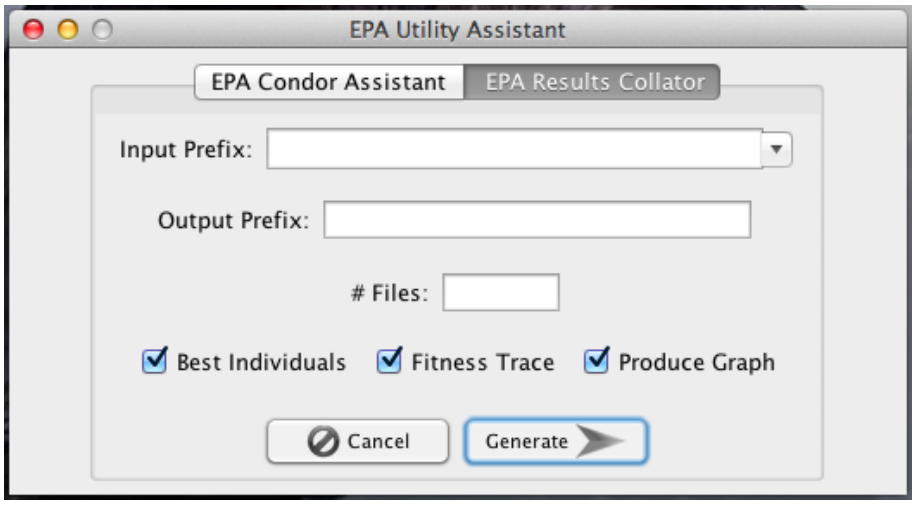


Figure A.5: The *EPA Utility Assistant* - results collator view

The amount of result processing can be controlled by the three checkboxes towards the bottom of the interface. The files created by each option will be detailed in Section A.2.2. First, the raw results collected for collating will be discussed.

A.2.1 *Raw Result Files*

Each EPA system run has four files which are created. The detail and number of files can be controlled by the user through parameter values but the default is for all output to be recorded. The following sections detail the four output files, the headers are the suffix appended to the output prefix.

A.2.1.1 *-Fitness.csv*

The fitness file records the fitness outputs from the best individual of each generation for each subpopulation. The fitness score is recorded with the trace score and any penalties which are applied.

Additionally, the number of successful versus failed individuals in the generation is recorded. This is after any validity increasing action to reduce the number of failed individuals, as

described in Section 4.3.11. In experiments with kinetic actions flagged for optimisation, the number of kinetic violations (Section 4.3.10.3) is also recorded.

A.2.1.2 -*GenerationalOutput.txt*

The generational output log records every individual which appears in an EPA system run, organised by subpopulation then generation. All details about an individual are recorded, including the fitness score, trace score, any penalties, the individual's success status, whether it includes a kinetic violation and the definitions for all flagged segments, sorted by segment type.

Fitness	Trace	Bloat	Suc?	Vio?	Kinetics:	alpha
2983.19	2983.19	3.97E-05	true	false	|	(X + Y + Z)
0.048534	0.048455	7.94E-05	true	false	|	fMA(r)
infinity	~	~	false	false	|	(Z - X)
588.8584	588.8582	1.19E-04	true	false	|	((X + Y + Z) / 5)
1.661178	1.661059	1.19E-04	true	false	|	(r / 24)

In addition to the example output, compiler messages are also recorded in the generational output. These messages come in three categories of rising severity: *info*, *warning* and *error*. An *info* message does not impact the validity or simulation of an individual but may be of interest to the user: `Info: Definition declared but not used`. A *warning* message refers to the validity of an individual but can still be simulated, these will include Bio-PEPA syntax violations and other possible impacts to the simulated results: `Warning: The rate alpha does not rely on X. Population could decrease below zero`. An *error* message indicates the individual contains some fatal error which will prevent simulation, due to the validity checks (Section 4.3.11) in the EPA framework these are not common: `Cannot evaluate the expression`.

The generational log is primarily for debugging purposes but can be used to track the areas of the search space which have been explored during the course of evolution.

A.2.1.3 -*Subpop-x-Solution.txt*

The solution file is the best individual of a given experiment; one per subpopulation. The x in the suffix is the subpopulation to which the individual belonged. If the experiment run is interrupted (such as by an HTCondor node being required by another user), the solution file notes which generation had been reached prior to interruption. If the experiment later resumes, the solution file is amended.

This result file includes the modes utilised during optimisation; i.e. which general segments of the Bio-PEPA model have been flagged. The fitness score, including a penalty breakdown and sub-fitness scores if the experiment is multimodal, see Section 4.1. Finally, a list of evolved segments for each flagged component is included.

A.2.1.4 *-Subpop-x-Trace.csv*

The trace file is the companion to the solution result file discussed above. Similarly, there is one file per best individual per subpopulation and the x in the suffix is the subpopulation to which the individual belonged. Also like the solution file, the trace result file records which generation has been reached, in case of interruption.

The trace file repeats the fitness score of the individual, along with the fitness penalty breakdown. The primary use of the trace file is to record the output trace from simulating the evolved individual. This can be used to debug the fitness function and also provide insight to the evolved behaviour to the user.

A.2.2 *Processed Result Files*

The *EPA Utility Assistant* extracts information from the raw results files created from the EPA executions on the HTCondor nodes, as discussed in Section A.2.1. This is done to produce the finalised results files. Which processed results files are to be created is controlled by the user via the checkboxes, as seen in Figure A.5. By default, all checkboxes are selected. As with the raw results files described in the previous section, the following section headers are the suffix appended to the output prefix.

A.2.2.1 *-Results.txt*

If the *Best Individuals* checkbox is selected, the results file is created. This file will consist of the singular best individual from every EPA system run: one individual per line where each column is a definition for a flagged segment, sorted by segment type. The EPA system run identifier number is also included, so the raw results files for a particular run can be further investigated if required.

If a rate or population segment is flagged, the assistant can produce additional output in the form of mean, median and standard deviation values. Kinetic and species segments cannot be so easily analysed, for species specific analysis output see Section A.2.2.3.

A.2.2.2 *-FitnessTrace.csv & -Chart.png*

The fitness trace and fitness chart are related files but controlled separately through the *Fitness Trace* and *Produce Graph* checkboxes respectively. The fitness chart produces the generational best fitness scores: one EPA system run per row.

The fitness graph is produced based on this trace; an example from the biochemical reactant-product experiment is illustrated in Figure A.6. The graphical processing behind the fitness graph is supplied by the *JFreeChart* open source graphical Java library [51]. The graph displays the average fitness scores over the number of generations and includes the weakest recorded

best individual, the fittest recorded best individual and the median point with quartiles: colour coded red, yellow and green respectively.

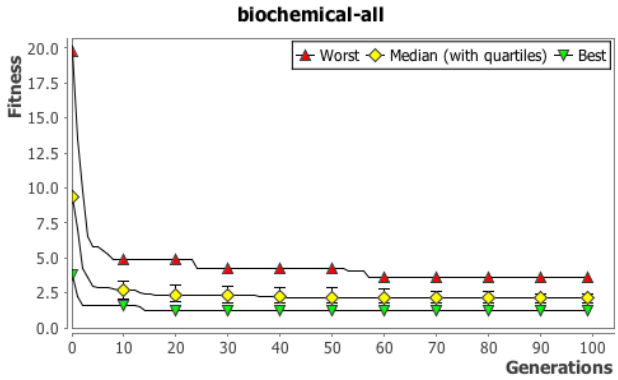


Figure A.6: The *EPA Utility Assistant* - fitness graph output

The data for the fitness graph is taken from the fitness trace data. If the fitness graph is selected for production without the fitness trace file, the *EPA Utility Assistant* performs the calculations internally without creating the fitness trace file.

A.2.2.3 *-SpeciesAnalysis.txt*

The species analysis file is only applicable to experiments where one or more species segments are flagged for optimisation. This results file is created with the best individuals index, as detailed in Section A.2.2.1, if the *Best Individuals* checkbox is selected.

The species analysis file contains the processed data of how many of each definition segment appears across best individuals from all EPA system runs. The species analysis images which appear in Chapter 6 have been further processed with the *Mathematica 9* library.

BIBLIOGRAPHY

[1] Ágoston E Eiben and Thomas Bäck. Empirical Investigation of Multiparent Recombination Operators in Evolution Strategies. *Evolutionary Computation*, 5(3):347–365, 1997.

[2] Thomas Bäck, David B Fogel, and Zbigniew S Michalewicz. *Evolutionary Computation 1: Basic Algorithms and Operators*. Taylor & Francis, 2000.

[3] Thomas Bäck, David B Fogel, and Zbigniew S Michalewicz. *Evolutionary Computation 2: Advanced Algorithms and Operators*. Taylor & Francis, 2000.

[4] Jos C M Baeten. A Brief History of Process Algebra. *Theoretical Computer Science*, 335 (2/3):131–146, 2005.

[5] Yifei Bao, Adriana B Compagnoni, Joseph Glavy, and Tommy White. Computational Modeling for the Activation Cycle of G-proteins by G-protein-coupled Receptors. In *Proceedings of the 4th Membrane Computing and Biologically Inspired Process Calculi*, pages 39–53. Open Publishing Association, 2010.

[6] Chris P Barnes, Daniel Silk, and Michael P Stumpf. Bayesian design strategies for synthetic biology. *Interface Focus*, 1(6):895–908, 2011.

[7] Nils Aall Barricelli. Esempi numerici di processi di evoluzione. *Methodos*, pages 45–68, 1954.

[8] Michael Begon, Malcolm Bennett, Roger G Bowers, Nigel P French, S M Hazel, and J Turner. A clarification of transmission terms in host-microparasite models: numbers, densities and areas. *Epidemiology and Infection*, 129:147–153, 2002.

[9] Soufiene Benkirane. *Process algebra for epidemiology: evaluating and enhancing the ability of PEPA to describe biological systems*. PhD thesis, University of Stirling, 2011.

[10] Soufiene Benkirane, Jane Hillston, Chris McCaig, Rachel Norman, and Carron Shankland. Improved Continuous Approximation of PEPA Models through Epidemiological Examples. *Electronic Notes in Theoretical Computer Science*, 229(1):59–74, 2009.

[11] Soufiene Benkirane, Rachel Norman, Erin Scott, and Carron Shankland. Measles Epidemics and PEPA: An Exploration of Historic Disease Dynamics Using Process Algebra. In *FM 2012: Formal Methods*, volume LNCS 7436, pages 101–115. Springer Berlin Heidelberg, 2012.

[12] Ralf Blossey, Luca Cardelli, and Andrew Phillips. A Compositional Approach to the Stochastic Dynamics of Gene Networks. *Transactions on Computational Systems Biology*, 3939:99–122, 2006.

[13] Jol Bockaert, Philippe Marin, Aline Dumuis, and Laurent Fagni. The 'magic tail' of G protein-coupled receptors: an anchorage for functional protein networks. . *FEBS Letters. Signal Transduction Special Issue*, 546(1):65–72, 2003.

[14] Benjamin Bokler. Chaos and complexity in measles models: a comparative numerical study. *Mathematical Medicine and Biology*, 10:85–93, 1993.

[15] Jeremy T Bradley and Thomas Thorne. Stochastic Process Algebra Models of a Circadian Clock. In *Simulation and Verification of Dynamic Systems*. Internationales Begegnungs und Forschungszentrum für Informatik (IBFI), 2006.

[16] Jeremy T Bradley, Stephen T Gilmore, and Jane Hillston. Analysing Distributed Internet Worm Attacks Using Continuous State-Space Approximation of Process Algebra Models. *Journal of Computer and System Sciences*, 74(6):1013–1032, 2008.

[17] Hans J Bremermann. Optimization Through Evolution and Recombination. In *Self-Organizing Systems*. Spartan Books, 1962.

[18] Hongqing Cao, Lishan Kang, Yuping Chen, and Jingxian Yu. Evolutionary Modeling of Ordinary Differential Equations for Dynamic Systems. In *GECCO 1999: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 959–965, 1999.

[19] Giuliano Casale and Mirco Tribastone. Process Algebraic Modelling of Priority Queueing Networks. In *Proceedings of the 9th Workshop on Process Algebra and Stochastically Timed Activities*. PASTA, 2010.

[20] Federica Ciocchetta and Jane Hillston. Bio-PEPA: a Framework for the Modelling and Analysis of Biological Systems. *Theoretical Computer Science*, 410:3065–3084, 2008.

[21] Federica Ciocchetta, Adam Duguid, Stephen Gilmore, Maria Luisa Guerriero, and Jane Hillston. The Bio-PEPA Tool Suite. http://homepages.inf.ed.ac.uk/jeh/Bio-PEPA/, 2009.

[22] Irun R Cohen and David Harel. Explaining a complex living system: dynamics, multi-scaling and emergence. *Journal of the Royal Society*, 4:175–182, 2007.

[23] Jacques Cohen. The Crucial Role of CS in Systems and Synthetic Biology. *Communications of the ACM*, 51:15–18, 2008.

[24] Icosystem Corporation. Icosystem. http://www.icosystem.com/demos/thegame.htm, 2001.

[25] Nichael Lynn Cramer. A Representation for the Adaptive Generation of Simple Sequential Programs. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 183–187. L. Erlbaum Associates Inc., 1985.

[26] Jack L Crosby. *Computer Simulation in Genetics*. Jown Wiley & Sons, 1973.

[27] Charles Darwin. *The Origin of Species*. P. F. Collier & son, 1909.

[28] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 26(1):29–41, 1996.

[29] Adam Duguid, Stephen Gilmore, Michael Smith, and Mirco Tribastone. The PEPA Eclipse Plug-in: A modelling, analysis and verification platform for PEPA. http://www.dcs.ed.ac.uk/pepa/documentation/, 2010.

[30] Michael B Elowitz and Stanislas Leibler. A synthetic oscillatory network of transcriptional regulators. *Nature*, 403:335–338, 2000.

[31] Paul Fine and Jacqueline Clarkson. Measles in England and Wales – I: An Analysis of Factors Underlying Seasonal Patterns. *International Journal of Epidemiology*, 11(1):5–14, 1982.

[32] David B Fogel, editor. *Evolutionary Computation: A Fossil Record*. Wiley-Blackwell, 1998.

[33] Lawrence J Fogel. *On the Organization of Intellect*. PhD thesis, University of California, 1964.

[34] Alex S Fraser. Simulation of Genetic Systems by Automatic Digital Computers. I. Introduction. *Australian Journal of Biological Sciences*, 10:484–491, 1957.

[35] Alex S Fraser and Donald Burnell. *Computer Models in Genetics*. McGraw-Hill Education, 1970.

[36] Nil Geisweiller. EM-PEPA, A Software to Find the Most Likely Rates Inside a PEPA Model. http://empepa.sourceforge.net/, sep 2006.

[37] Michael A Gibson and Jehoshua Bruck. Efficient Exact Stochastic Simulation of Chemical Systems With Many Species and Many Channels. *Journal of Physical Chemistry A*, 104: 1876–1889, 2000.

[38] David E Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.

[39] Health Protection Agency. New HIV Diagnoses Surveillance Tables: UK data to the End of December 2007. www.hpa.org.uk/web/HPAwebFile/HPAweb C/1208763421275, 2008.

[40] Jane Hillston. *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.

[41] Charles Antony Richard Hoare. Communicating Sequential Processes. *Communications of the ACM*, 21(8):666–677, 1978.

[42] John H Holland. *Adaptation in Natural and Artificial Systems*. MIT Press, 1975.

[43] Hitoshi Iba and Erina Sakamoto. Inference Of Differential Equation Models by Genetic Programming. In *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 788–795, 2002.

[44] Cliff Joslyn and Luis M Rocha. Towards Semiotic Agent-Based Models of Socio-Technical Organizations. In *Proceedings. AI, Simulation and Planning in High Autonomy Systems*, pages 70–79, 2000.

[45] James Kennedy and Russell C Eberhart. Particle Swarm Optimization. In *IEEE International Conference on Neural Networks IV*, pages 1942–1948. IEEE, 1995.

[46] Brian Kent Kobilka. G protein coupled receptor structure and activation. *Biochimica et Biophysica Acta (BBA) - Biomembranes*, 1768(4):794–807, 2007.

[47] John R Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1972.

[48] John R Koza. John R Koza Collection. http://www.johnkoza.com, 2007.

[49] Pedro Larrañaga and Jose A Lozano, editors. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Kluwer Academic Publishers, 2002.

[50] Juliane Liepe, Chris Barnes, Erika Cule, Kamil Erguler, Paul Kirk, Tina Toni, and Michael P H Stumpf. ABC-SysBio – approximate Bayesian computation in Python with GPU support. *Bioinformatics*, 26(14):1797–1799, 2010.

[51] Object Refinery Limited. JFreeChart. http://www.jfree.org/jfreechart/, 2013.

[52] Laurence Loewe, Stuart Moodie, and Jane Hillston. Quantifying the implicit process flow abstraction in SBGN-PD diagrams with Bio-PEPA. In *CompMod*, pages 93–107, 2009.

[53] Sean Luke. ECJ: A Java-based Evolutionary Computation Research System. http://cs.gmu.edu/ eclab/projects/ecj/, 2013.

[54] Sean Luke. Essentials of Metaheuristics. http://cs.gmu.edu/~sean/book/metaheuristics/, 2013.

[55] Jean-Michel Marin, Pierre Pudlo, Christian P Robert, and Robin J Ryder. Approximate Bayesian computational methods. *Statistics and Computing*, 22(6):1167–1180, 2012.

[56] Chris McCaig, Rachel Norman, and Carron Shankland. From Individuals to Populations: A Symbolic Process Algebra Approach to Epidemiology. *Mathematics in Computer Science*, 2(3):139–155, 2009.

[57] Davide Prandi. Particle Swarm Optimization for Stochastic Process Calculi. In *Proceedings of the 9th Workshop on Process Algebra and Stochastically Timed Activities*, pages 77–82. PASTA, 2010.

[58] Corrado Priami. Algorithmic Systems Biology: An Opportunity for Computer Science. *Communications of the ACM*, 52(5):80–88, 2009.

[59] Dean Garner Pruitt and J P Gahagan. Campus Crisis: The Search for Power. In *Perspectives on Social Power*, pages 349–392. Aldine Pub. Co [Chicago], 1974.

[60] Ottar N Bjørnstad, Bärbel F Finkenstädt, and Bryan T Grenfell. Dynamics of Measles Epidemics: Estimating Scaling of Transmission Rates Using a Time Series SIR Model. *Ecological Monographs*, 72(2):169–184, 2002.

[61] Brian Ross and Janine Imada. Evolving Stochastic Processes Using Feature Tests and Genetic Programming. In *11th Annual Conference on Genetic and Evolutionary Computation*, pages 1059–1066. ACM, 2009.

[62] Sauro Lab. Systems Biology Workbench. http://sbw.kgi.edu/sbwWiki/doku.php?id=sysbio:sbw, 2011.

[63] Dieter Schenzle. An age-structured model of pre- and post- vaccination measles transmission. *IMA Journal of Mathematics Applied in Medicine and Biology*, 1(2):169–191, 1984.

[64] Erin Scott. *PEPA or Bio-PEPA: A Comparison*. Undergraduate honours thesis, University of Stirling, 2011.

[65] SynthSys, University of Edinburgh. Systems Biology Software Infrastructure. http://www.sbsi.ed.ac.uk/, 2012.

[66] The HTCondor Team. HTCondor: High Throughput Computing. http://research.cs.wisc.edu/htcondor/index.html, 2013.

[67] Kyaw Tun. Nonlinear System Identification using Genetic Programming. Master's thesis, National University of Singapore, 2003.

[68] Warren Weaver. Science and Complexity. *American Scientist*, 36(4):536–544, 1948.

[69] David H Wolpert and William G Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.