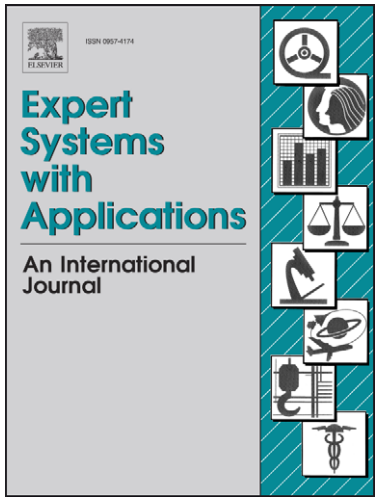
Accepted Manuscript



Modelling Epistasis in Genetic Disease using Petri Nets, Evolutionary Computation and Frequent Itemset Mining

Michael Mayo, Lorenzo Beretta

PII: S0957-4174(10)01019-5
DOI: 10.1016/j.eswa.2010.09.062
Reference: ESWA 5261

To appear in: *Expert Systems with Applications*

Please cite this article as: Mayo, M., Beretta, L., Modelling Epistasis in Genetic Disease using Petri Nets, Evolutionary Computation and Frequent Itemset Mining, *Expert Systems with Applications* (2010), doi: 10.1016/j.eswa.2010.09.062

Modelling Epistasis in Genetic Disease using Petri Nets, Evolutionary Computation and Frequent Itemset Mining

Michael Mayo[1] (corresponding author), Lorenzo Beretta[2]

[1]Deptartment of Computer Science

University of Waikato

Hamilton, New Zealand

Ph. ++64-7-8384402 Fax ++64-7-8585095

Email mmayo@cs.waikato.ac.nz

[2]Referral Center for Systemic Autoimmune Diseases

Fondazione IRCCS Ospedale Maggiore Policlinico di Milano

Via Pace 9, 20122 Milan, Italy

Email lorberimm@hotmail.com

Abstract

Petri nets are useful for mathematically modelling disease-causing genetic epistasis. A Petri net model of an interaction has the potential to lead to biological insight into the cause of a genetic disease. However, defining a Petri net by hand for a particular interaction is extremely difficult because of the sheer complexity of the problem and degrees of freedom inherent in a Petri net's architecture.

We propose therefore a novel method, based on evolutionary computation and data mining, for automatically constructing Petri net models of non-linear gene interactions. The method comprises two main steps. Firstly, an initial partial Petri net is set up with several repeated sub-nets that model individual genes and a set of constraints, comprising relevant common sense and biological knowledge, is also defined. These constraints characterise the class of Petri nets that are desired. Secondly, this initial Petri net structure and the constraints are used as the input to a genetic algorithm. The genetic algorithm searches for a Petri net architecture that is both a superset of the initial net, and also conforms to all of the given constraints. The genetic algorithm evaluation function that we employ gives equal weighting to both the accuracy of the net and also its parsimony.

We demonstrate our method using an epistatic model related to the presence of digital ulcers in systemic sclerosis patients that was recently reported in the literature. Our results show that although individual "perfect" Petri nets can frequently be discovered for this interaction, the true value of this approach lies in generating many different perfect nets, and applying data mining techniques to them in order to elucidate common and statistically significant patterns of interaction.

Keywords

Petri net; genetic algorithm; frequent itemset mining; epistasis; genetic disease

PACS Codes

82.20.Wt Computational Modeling; Simulation

87.19.xk Genetic Diseases

87.55.de Optimization

1. Introduction

Genetic diseases can be roughly divided into two categories. The first category consists of all those diseases caused by a mutation to a single gene. These are the so-called Mendelian diseases. The gene associated with diseases of this class can often be identified by performing a parametric statistical analysis on genetic data obtained from patients with and without the disease. That is, the mutant gene's presence should be highly correlated with the presence of the disease in the sample of patients.

The other category of genetic disease is that for which no single genetic cause can be identified. When in a population the supposedly causative genes are examined in isolation, none of them correlates strongly at all with the disease. Rather, it is the peculiar biochemical interaction between a set of two or more genes that leads to the disease. This is known as epistasis or non-linear gene interaction, and it is thought to be the predominant cause of genetic disease (Moore, 2003; Phillips, 2008).

Non-linear gene interactions are difficult to model by traditional parametric approaches, and hence other computational tools, essentially based on data mining techniques, have been developed to identify the set of interacting genes responsible. The advantage of these techniques is their ability to deal with sparseness of data in the multidimensional space (often referred to as the "curse of dimensionality"). The Multifactorial Dimensionality Reduction (MDR) algorithm (Moore, 2004) is one such algorithm that has been highly successful in this area and is routinely used to identify non-linear interactions, not only in genetics but also in a wide range of other application areas.

The problem addressed in this paper, which is the next step after the data mining, is how to take a set of interacting genes that have been identified and automatically build a computational model of how these genes may be interacting. In other words, how could the genes, each of which produce a substance in some quantity determined by genotype, lead to the epidemiological observation of a high risk of the disease in some individuals, and a low risk in others?

Producing a model of this interaction should help scientists understand how and why the disease occurs. The models produced, however, must be biologically analogous and be capable of incorporating prior biological and common sense knowledge in order to be useful. The typical output of a machine learning algorithm, for example a decision tree, would not meet this criteria.

One class of computational model that has been usefully applied in the field of biochemical interaction modelling is the Petri net (Reisig, 1985). To illustrate, Petri nets are critical in the construction of models of gene regulatory networks for both cancer

(Cheng et al., 2007; Lin et al., 2007) and diabetes (McGarry, 2007). They are so widely used by computational biologists that they are also included in common biological data analysis software packages such as BioCAD (Lee et al., 2007).

Most of the time, however, scientists needing a Petri net model of biochemical interactions must construct their models by hand. This approach has two significant drawbacks.

Firstly, in all but the simplest of cases, designing a Petri net by hand is an extremely tedious trial-and-error process. If, for example, there are many "rules" governing the interaction that must be modelled, then it may be very difficult to find one model that covers all of them; often one will find a model that captures some of the interactions but not all, and an attempt to change the model to capture a new facet of the interaction inevitably invalidates some of the others.

Secondly, a manual process of building a model such as a Petri net leads only to a single solution that may be biased by preconception. For example, the scientist may believe that the interaction occurs in a certain way, and then look for Petri nets that only behave this way, thereby excluding other possibilities. In reality, there may be many possible different solutions, some more likely than others, some more parsimonious than others.

What is needed, therefore, is an automatic method of learning Petri nets for a given interaction. If there are many different possible solutions to this problem, an automatic method could be set up to identify them, and these multiple solutions could then be ranked by likelihood or parsimony. The scientist may then choose to select the single best

solution, or perhaps analyse all of them and look for patterns. Such an approach is clearly not feasible if the Petri nets are designed manually.

We therefore describe here a novel method for automatically building biologically analogous Petri nets for gene-gene interactions. We focus specifically on deterministic Petri nets, and use a genetic algorithm (Goldberg, 1989; Meffert, 2009). We then show how multiple solutions to the same problem can be generated and analysed in a statistical fashion, so that the most commonly occurring patterns of interactions can be identified.

2. Materials and methods

In this section, we briefly introduce Petri nets in the context of modelling in biochemistry, and then present our approach for non-linear gene modelling and show how we intend to learn Petri nets for given epistatic models.

2.1 Theory of Petri Net Modelling and Learning in Biochemistry

Petri nets are abstract computational models of dynamic, concurrent processes (Reisig, 1985). Although they were originally developed to model the interacting behaviour of concurrent software components - and that is still their main use - they have recently found application in the fields of biochemistry, genetics and medicine. For example, see Cheng et al. (2007), McGarry et al. (2007) and Lin et al. (2007) amongst other references in the literature.

Why are Petri nets relevant to these fields? The answer is that they can be considered abstract model of biochemical processes, which like interacting software components, are also concurrent. Specifically, Petri nets capture succinctly the notion of substances

dynamically interacting via reactions to produce further substances that lead to further reactions, and so on. The concentration of substances changing over time can be modelled and observed. A Petri net can therefore be given some "input" substances, "executed" for a period of time, and the "output" substances can be measured.

For those unfamiliar with the notion of a Petri net, Figure 1 depicts an example of a simple Petri net. Petri nets are directed bipartite graphs with two classes of node, places (circles) and transitions (rectangles). Places represent substances, and the concentration of a substance is modelled discretely as an integer associated with each place. Thus, in Figure 1, the concentration of the substance P_0 is 5, and the concentration of P_2 is 2. The discrete integer units are called tokens. The concentrations of all of the substances in the net in Figure 1 can be represented by the vector <5,0,2,1>, which is called the "marking" of the Petri net.

Transitions are used to model reactions between substances. They have a number of input substances, depicted by the transition's incoming arcs, and a number of output substances, depicted by the outgoing arcs. For example, in Figure 1, T_0 has a single input, P_0, and a single output, P_2. (It also has an inhibitor arc as an input which will be discussed shortly.) T_1, on the other hand, has two inputs and one output.

The amount of a substance consumed or produced by a transition is given by the weights on the arcs. Where there is no weight assigned to an arc in a diagram, a default weight of 1 is assumed. For example, if T_0 "fires", it will reduce the concentration at P_0 by 1, and increase the concentration of P_2 by 2. Thus the overall marking will change to <4,0,4,1>. On the other hand, if T_1 fires instead of T_0, the marking will change to <2,0,1,5>.

A transition can fire only if it has sufficient inputs available – otherwise, it is disabled. Likewise, it also cannot fire if there is insufficient free capacity at one of its output places; for example, in Figure 1, if the maximum capacity of P_3 is 4, then T_1 cannot fire because there is already one token at P_3, and there is not enough capacity available at P_4 to take an additional 4 tokens.

Some arcs are inhibitory, which means that they basically disable transitions regardless of whether the transition has sufficient inputs or sufficient output capacity. For example, the arc from P_1 to T_0 is an inhibitor. If the marking of P_1 is greater than zero, then T_0 can never fire regardless.

To make our Petri nets deterministic, we enforce the rule that whenever two transitions can fire simultaneously, the lowest numbered transition always fires first. This corresponds to the arbitrary ordering rule for deadlock resolution suggested by Reising (1985). In Figure 1, therefore, T_0 will always fire before T_1, unless T_0 is disabled (for example, by the presence of inhibitory substance P_1, or a lack of input substance P_0).

Generally speaking, once a Petri nets starts executing, it will either run until no more transitions can fire, or it will enter an infinite loop, in which case the simulation of the network may halt after sufficient time has passed.

There has been limited work to date on the problem of automatically learning Petri nets. The earliest attempt appears to be Reid (1998), who applied genetic search to build a Petri net given observations of its interaction with some environment. The approach does not seem to have been entirely successful, however, due to various problems. Bourdeaud'huy & Yim (2002) extended this approach, using a genetic algorithm to

construct a Petri net controller for a hexapod robot. Mauch (2003) generalised it, showing that Petri nets can be usefully employed as "genome" representations in genetic programming.

More recently, Zhao et al (2007) provides an interesting account of Petri net learning in the domain of fault diagnosis. Their method consists of applying a set of rules to update the structure and weights of a Petri net given observations. Similarly, Durzinsky et al. (2008) describe a method for learning a Petri net from time series data. Their approach enumerates all possible Petri nets, ordered by complexity up to some limit, that can explain the data of interest. All possible network structures can therefore be explored. It is not entirely evident, however, how scalable this approach is to different and larger problems.

In the field of gene-gene interaction modelling, Moore and Hahn (2003)'s approach is the first attempt to describe a technique using a genetic algorithm to learn a Petri net model of a non-linear gene interaction. However, the particular variety of genetic algorithm ("grammatical evolution") that they employ appears to produce only very simple Petri nets.

Mayo (2005) applied multi-start random hill climbing to evolve the arcs and weights of a Petri net with a fixed number of places and transitions. Although this method produced perfect models of the genetic interaction, the Petri nets were scored only by accuracy and did not include prior expert knowledge about the problem; hence the resulting nets tended to be more complex and less biologically relevant than they needed to be.

Nummela and Julstrum (2005) also provide an approach to learning Petri nets, this time of metabolic pathways. Similar to previous approaches, they use a genetic algorithm, but in their case the search is divided into two steps: the initial search is for the best Petri net structure, and the second search is for the best set of weights once the structure is finalized. Like Reid (1998), they appear to get mixed results, with the method sometimes working and sometimes failing.

2.2 Petri Net Learning: A New Approach Incorporating Expert Knowledge and Parsimony

In this section, we describe our new method. Like prior approaches, we also use a genetic algorithm. However, the key differences between our approach and previous approaches are (i) that we include prior expert knowledge in the net, in the form of "gene units" and constraints on the structure of the Petri net that can be used to encode biological and common sense knowledge, and (ii) we score our Petri nets not only by their performance or accuracy as a model of the genetic interaction, but also by their parsimony (simplicity), on the grounds that given two different solutions, the simplest is more likely to be the correct one. Simpler solutions also tend to be more easily understood by humans, who must examine the outputs of the algorithm.

In addition to these key differences, our model also routinely finds multiple different solutions, although this can take considerable computational effort as the complexity of the interaction being modelled increases.

2.2.1 Petri Net Configuration for Statistical Epistasis Modelling

The Petri nets that we employed in this study are required to incorporate certain expert knowledge, and therefore conform to a specific structure. Because we are interested in how genes interact with one another, it is necessary to define a "gene unit" or a sub-Petri net whose structure is fixed and cannot be modified by the genetic algorithm.

A full model of non-linear gene interaction, therefore, consists of several of these fixed gene units, with the gene units connected to each other via other non-gene unit places and transitions. Figure 2 depicts the structure of a gene unit.

As the figure shows, each gene unit consists of a single transition. It has exactly one input substance, that being the substance that "activates" the gene, and exactly one output substance, the "product" of the gene. There is also an inhibitor substance that may prevent the gene from producing the product substance if it is activated.

Within each gene unit, the rate at which the activating substance is consumed by the transition is one token per firing; therefore, due to the fixed maximum capacity of the activating place, a gene unit cannot be continuously active -- although the activating substance could potentially be continuously replenished by another transition. The product substance is produced at a rate g, and the key point here is that g is always determined by genotype.

This is quite different to other Petri net learning approaches where it is assumed that the arc weights are uniformly fixed, and must be optimised globally. In our case, there is one arc per gene unit that is variable and to be determined by genotype, and there will be many different genotypes (and therefore different values of g) in a population.

We use the "gene dose effect rule" to determine the value of *g*. Each genotype consists of an unordered pair of alleles, with each allele being either wild-type (*A*) or a mutant (*a*). Genotypes may therefore be either fully wild-type (hereafter labelled *AA*), heterozygous for the mutant allele (*Aa*), or homozygous for the mutant allele (*aa*).

Following several biological observations, it is assumed in our nets that the mutant allele *a* causes the gene unit to overproduce more of the product substance than the non-mutant version *A* at some fixed ratio *n* (Pociot *et al*, 1995; Fishman *et al*, 1998; Hoffmann *et al*, 2001). Thus, genotype *AA* will have the least production when the gene is activated; *Aa* will have a medium rate of production, and *aa* will have the greatest rate of production.

How are the exact values of *g* computed? We start with two parameters, *x* and *n*, and compute the *g* according to the rules given in Table 1. The values of *x* and *n* are fixed for a particular gene unit, and for the duration of its lifetime, but they may vary between gene units and between Petri nets. In our simulations, we select $x \in \{1,2,3,4,5\}$ and $n \in \{2,3,4,5\}$ randomly, with the constraint that $2nx$ must be less than or equal to the maximum capacity of the product place (or else the gene unit will be unable to fire at all).

Now that the gene units have been defined, we can specify the overall architecture of our Petri net models of gene-gene interactions. Let us define an (*i,j,k*) architecture as a Petri net consisting of the following:

- one input place, P_0, representing the initial source of tokens, or, from a biological point of view, the trigger event that starts the chain of reactions that lead to disease; P_0 may only be a transition input

- one output place, P_1, representing the toxic disease-causing substance that builds up for high risk genotypes; P_1 may only be a transition output
- *i* additional places, representing intermediate substances
- *j* additional transitions
- *k* gene units as defined above, each gene unit comprising an activation and a product place along with a transition; the inhibiting substance, if the gene has one, is assumed to be one of the other places in the network

There are a number of constraints that each network in this configuration must abide by. Firstly, no other places can connect directly to the transitions of gene units. The gene units are meant to be the fixed "building blocks" of the net, and therefore access to them is only via the activation place, the product place, or the inhibition place.

The second constraint is that there can be no transitions that are "sources" or "sinks". A source is transition with output arcs but no inputs, and a sink is the converse, a transition with inputs but no outputs. Because we want our nets to model the way in which a finite amount of substance is transformed by the gene units, it makes sense not to allow infinite sources or sinks of substance.

Thirdly, the places have a fixed uniform maximum place capacity, which in our simulations was set to 10.

Fourthly, the arc weights also have a fixed maximum value of 10. (Inhibitor arcs, however, do not have an arc weight.)

Finally, and perhaps most importantly, there is a threshold t that is used to determine whether the concentration of substance at P_1 should be assigned "high risk" or "low risk". High risk assignments are assumed to lead to a high risk of the disease being modelled, and low risk assignments lead to either low or no risk of the disease. The value for t is arbitrary, and is set at a randomly chosen value between 0.5 and 0.9 of the maximum capacity of the output place P_1. Once t is selected, however, it is fixed for an entire run of the genetic algorithm.

Figure 3 gives an example of Petri net with three genes that conforms to this architecture and was discovered using our method. The net is composed of an input place (P_0), an output place (P_1), five additional transitions (one of them, T_4, ended up not being used), and three gene units. The gene units are P_2-T_5-P_3, P_4-T_6-P_5, and P_6-T_7-P_7, and in this particular example, none of them ended up needing an inhibitory place. The arc weights are 1 unless otherwise stated.

Note that the transitions in the gene units all have the lowest firing priority and therefore all other non-gene unit transitions will fire before them if enabled.

2.2.2 Execution and Evaluation of Gene-Unit-based Petri Nets

A genetic algorithm requires candidate solutions to be evaluated in order to determine the solution's fitness, and if the candidate solutions are Petri nets, then obviously they must be executed during the evaluation step. How is this achieved?

Firstly, the markings of all the places except for P_0 are initialised to 0. For P_0, we set the initial marking to an arbitrary value, which is 10 in our experiments. This is the only initial quantity of tokens available in the network.

Secondly, the values of g_o, $g_{1...}g_{k-1,}$ where k is the number of genes involved in the epistatic model, must then be computed according to the gene dose effect rule described in the previous section. Once the value of these genotype-dependent arc weights has been determined and set as arc weights within the gene units, the Petri net is ready for execution.

The network is then executed until one of the three following halting conditions occur: (i) the number of tokens at output place P_1 exceeds the threshold t, in which case the assignment is automatically "high risk", or (ii) no more transitions can fire, in which case the assignment is "low risk", or (iii) 100 transitions fire without the threshold at P_1 being exceeded, in which case the execution is aborted and a "low risk" assignment is assumed.

Thus, for any particular Petri net, we would expect that different genotypes lead to different Petri net execution dynamics and therefore different disease risk assignments.

The purpose of our study here, therefore, is to start from a set of genotype combinations that are divided into high and low risk (as determined by a genetic study of patients with and without the disease of interest), and find a way of building automatically a full Petri net model that predicts or explains the correct risk assignment for each genotype. It is extremely important that only the g values vary between genotypes; the rest of the net must be fixed and independent of genotype.

Since there are three genotypes for each gene, there will be 3^2=9 genotype combinations where two genes are interacting, and 3^3=27 combinations where three relevant genes are involved. However, some genotype combinations may not occur in a real population, and therefore the actual number of observed combinations may be slightly less.

2.2.3 Genetic Algorithm for Evolving Petri Nets

Genetic algorithms are a well known, widely used, and effective solution to the problem of local search and optimisation (Goldberg, 1989). Although we do not have the space to describe genetic algorithms fully here, it is worthwhile giving a brief overview.

The basic idea is to start with a population of randomly generated "solutions". In our case, the solutions are Petri nets adhering to the structure and constraints specified in Section 2.2.1. Because the initial nets are generated randomly, they are not likely to perform particularly well; that is, they may only produce the correct risk assessment for, say, 50% of the genotype combinations. This is the level of accuracy that would be expected due to chance.

The problem, then, is to "evolve" the population of Petri nets to cover more of the genotype/risk assessment rules, aiming to find a net that accurately covers 100% of the genotypes. This can be then returned as a solution, and is what we refer to as a "perfect" Petri net for a given problem.

However, not all perfect Petri nets are equal: some nets may be 100% accurate but have more arcs, places and transitions than another net that is also 100% accurate. We would therefore prefer the simpler solution.

Similarly, one solution may on average require, for example, 50 transitions to fire before a risk assessment is made, whereas another solution may require only 5 transitions to fire. Again, the simpler solution is more desirable because the net becomes more "understandable".

In both cases, we want the search algorithm to focus on the simpler, less complex, and more parsimonious solutions. Such Petri nets are more likely to be comprehended and interpreted by human beings.

How can a genetic algorithm evolve such a Petri net? The answer requires us first to define a representation for the Petri nets. A direct graph representation is quite inefficient and overly complex from a computational standpoint. One simpler, more computational-friendly, representation is the 2D integer array. One simply defines a row for each place, a column for each transition, and inserts an integer into each cell representing the arc weight. To illustrate, Table 3 shows the Petri net depicted in Figure 1 after conversion into a 2D array representation.

In Table 3, a positive array entry indicates a place to transition arc and a negative array entry indicates a transition to place arc. Zero indicates no arc. There is a special integer code i (chosen to be beyond the range of normal arc weights) to indicate the presence of an inhibitor arc, which always goes from place to transition.

For all (i,j,k) Petri nets as defined in the previous section, then, the dimensions of the equivalent 2D array will be the same as long as the values of i, j and k remain the same. We can now view the problem of Petri net learning and one of optimising a 2D integer array.

How does a genetic algorithm find solutions? It has two basic operators: (i) crossover, in which two solutions “mate” with probability proportional their value or fitness, producing offspring solutions; and (ii) mutation, in which offspring are randomly changed.

With regard to Petri nets, a solution that is the result of crossover will have half of its array entries coming from one parent, and half from the other. A solution that has been mutated will have a single entry changed, either to 0 (deleting the arc), to another non-zero weight (which may reverse the direction of the arc if the sign changes), or to *i*, turning a normal or non-existent arc into a new inhibitor arc.

By continuously crossing over and mutating Petri nets, our genetic algorithm can gradually construct a Petri that both covers all of the genotype/risk assessment rules, and is at the same time parsimonious.

The fitness or value function used in our genetic algorithm will now be defined. Let *p* be an arbitrary (*i*, *j*, *k*) Petri net as defined previously. Suppose we have *r* genotype combinations that we want the Petri net to model, each genotype combination leading to a particular risk assessment. Let *numCorrect(p)* be the number of rules that the Petri net actually models correctly, i.e. the number of genotype combinations for which the eventual risk assessment after executing the Petri net is correct. This will be a number between 0 and 3^r.

Likewise, let *numArcs(p)* be the number of arcs (both normal and inhibitor) in the net, and let *numFires(p)* be the average number of transition firings required to predict a risk assessment given a genotype (the rules for determining when to stop executing the Petri net were given in Section 2.2.2). These functions measure the parsimony of the Petri net.

Another function is used to compute the fitness, that being the maximum possible number of arcs in a net, *maxArcs(p)*, which is computed by multiplying the number of places by the number of arcs.

Our final fitness function, therefore, is defined by Equation 1. There are three main components to this equation: (i) the rule coverage component, which has an overall maximum contribution to the fitness of $\frac{1}{2}$; (ii) a transition firing component, which increases as the average number of transition firings per rule decreases (recall that the maximum allowed number of firings is 100), and has maximum contribution of $\frac{1}{4}$, and (iii) an arc complexity component that is maximised when the number of arcs is minimised, which also has a maximum contribution to the overall fitness of $\frac{1}{4}$.

$$Value(p) = \frac{1}{2}\left(\frac{numCorrect(p)}{r}\right) + \frac{1}{4}\left(1 - \frac{numFires(p)}{100}\right) + \frac{1}{4}\left(1 - \frac{numArcs(p)}{maxArcs(p)}\right) \quad (1)$$

Thus, the total fitness function ranges theoretically from 0.0 (the absolutely worst Petri net along all of the dimensions of correctness and parsimony) to 1.0 (a perfectly correct, highly parsimonious net).

In practice, however, the fitness values tend to fall within a fairly narrow range within the 0.0 to 1.0 theoretical limits. For example, the Petri net depicted in Figure 3 correctly predicts the risk assessment for all of the rules it was trained on, it has a low number of transition firings on average, and also a minimal number of arcs required to model the interaction, yet its fitness value is approximately 0.65.

In all our experiments, the genetic algorithm's population size is set to 500 and it runs until 500 generations elapse without any further improvement. If the best current solution happens to cover all of the rules, i.e. it is 100% accurate in predicting risk assessment

from genotype, then this best solution is returned; otherwise it the genetic algorithm restarts with a new, random population.

This latter measure, specifically restarting if no further gains can be made and if a perfect net has not been found, is necessary to avoid the genetic algorithm from converging to local fitness function maxima.

3. Results

In this section, we evaluate our approach for automatically building Petri net models of epistasis.

3.1 Experimental Setup

A recently discovered disease-causing non-linear gene interaction is used as a test-bed (Beretta et al., 2009). This model, depicted in Figure 4, describes the risk of developing digital ulcers in a population of 200 Italian systemic sclerosis patients. It was built using the MDR kernel over a set of 22 cytokine Single Nucleotide Polymorphisms (SNPs) and 3 Human Leukocyte Antigens (HLAs), genotyped by polymerase chain reaction with sequence-specific primers as previously described (Beretta *et al*, 2009). The genes are HLA-B*3501 (hereafter referred to as B35), IL-2 and IL-6**.**

In each cell of Figure 4, the left bars indicate the frequency of patients (cases) with digital ulcers, and the right bars indicate the frequency of patients without digital ulcers (the controls). If the ratio of cases:controls exceeds a certain threshold, patients are labelled as “high-risk” (they dark-shaded cells), otherwise they are “low-risk” (the light-shaded cells).

We desire a perfect Petri net that explains this model. Such a net should ultimately produce, after execution, a token count exceeding the threshold *t* at the output place for high risk genotypes, but it should not exceed the *t* for the low risk genotypes.

We consider two different approaches to using our method to understand these interactions. A first approach is simply to run the algorithm a number of times and select by hand (using biological knowledge and intuition) one of the nets, whichever seems the most suitable or parsimonious.

In the second approach, the algorithm can be run multiple times to construct multiple perfect Petri net models. These Petri nets can then be analysed statistically to determine the most frequent interactions. However, the output in this case is not an executable model: rather it is a set of commonly occurring patterns that would represent the most interesting biochemical pathways that intervene in gene-gene interactions.

To perform the statistical analysis of multiple perfect Petri nets we used frequent itemset mining (Argawal et al., 1994; Lui et al., 1998). Frequent itemset mining in its simplest form enumerates all patterns of a particular size, ranking them by confidence. In our case, the data is composed of Petri nets. For each net, we determined the number of direct connections between places. That is, if two places in a given net were connected by at least a single transition, a direct connection between the places is assumed; on the other hand, if the two places did not have a single transition between them, but rather a sequence of more than one pair of intermediate transitions and places, a direct connection was not assumed.

This approach allowed us to assign to each possible pair of places in each net a binary indicator, with 1 denoting the presence of direct connection, and 0 the absence. Therefore each net could be converted into a fixed length binary vector, and frequent itemset mining could be applied on the entire dataset.

We performed two types of analysis on this place-place connection data: firstly we determined the most frequent single connections, and secondly we determined the most frequent largest pattern of connections (i.e. the largest group of single connections that co-occur as a group).

3.2 Experimental Results

In the first experiment, we generated 400 Petri nets for the digital ulcers model depicted in Figure 4. As described in Section 2.2.1, the Petri nets are constrained in various ways. The g values for each gene are genotype dependent and computed according to the gene dose rule described in Table 1. The value function to be optimised in each run of the multi-start genetic algorithm is that described previously in Equation 1.

Two examples of Petri nets learned by the genetic algorithm along with the g values required in order to execute them are given in Figure 5. Figure 3 is also an example of a perfect net that was found.

To check that these nets are indeed perfect Petri nets, it is possible to trace the execution of the Petri nets for given genotypes. For example, according to Figure 4, the genotype B35=*aa*, IL-2=*Aa* and IL-6=*Aa* (the darkly shaded middle cell of the first 3×3 square in the figure) should lead to a high risk assessment. The g values are, in the case of Figure

5(a), g_{B35}=6, $g_{IL\text{-}2}$=5 and $g_{IL\text{-}6}$=6; and in the case of Figure 5(b), they are g_{B35}=6, $g_{IL\text{-}2}$=5 and $g_{IL\text{-}6}$=4.

After the g values for this particular genotype are set, the input place P_0 is initialised to 10 tokens and the net is executed deterministically. Because this genotype is high risk, the number of fired transitions should not exceed 100 before the threshold the output place P_1 is exceeded.

Table 3 lists the sequence of transitions fired in order to exceed the threshold at P_1, for both of the nets depicted in Figure 5. Note that the sequences are quite short despite up to 100 transition firings being permitted; this is clearly an effect of including the average number of transition firings into the value function.

The single place-place connections found via frequent itemset mining are shown in Table 3, ranked by frequency. As it can be observed, the most frequent connection is from the product place of the B35 gene (P_7) to the output place P_1, which occurs in 382 out of 400 of the Petri nets. This is by far the most frequent of the connections to the output place; the next most frequent connections are from the input place P_0 to the activating places of the gene units, P_2, P_4 and P_6. The fourth most frequent connection is from P_3 (the output of the IL-6 gene unit) to P_1, which occurs 288 times. And finally, there are two interesting "feedback" connections that frequently occur: specifically the output of the IL-2 gene unit (P_5) to the input of the IL-6 gene unit (P_2), and the output of the IL-6 gene unit (P_3) to the input of the B35 gene unit (P_6).

Single connection analysis can reveal the frequency of different place-place connections in general, but because of variations in the perfect Petri nets that are constructed with

each run of the algorithm, there may be larger patterns consisting of sets of multiple connections that co-occur. Exploring these would therefore give a greater qualitative understanding of the gene-gene interaction dynamics.

In the next analysis, we used frequent itemset mining to determine the most frequent overall sub-structure occurring in the 400 Petri nets we found. That sub-structure is as follows: {$P_0 \rightarrow P_2$, $P_0 \rightarrow P_4$, $P_0 \rightarrow P_6$, $P_3 \rightarrow P_1$, $P_7 \rightarrow P_1$, $P_5 \rightarrow P_2$, $P_5 !\rightarrow P_4$}. This sub-structure of 7 co-occurring place-place connections occurs in 115 out of 400 of the Petri nets, and for the most part the present connections (e.g. $P_0 \rightarrow P_2$) are the same as the single most frequent connection analysis in Table 4. However, the symbol "!→" indicates absence of a connection rather than the presence; and so one can infer the additional conclusion that the IL2 gene unit (P_5) only feeds back and activates the IL6 gene unit (P_2) when there is no feedback from IL2 to itself (i.e. no connection from P_5 to P_4).

4. Discussion

We have described a novel method based on a genetic algorithm for automatically building Petri nets, with application to the modelling of non-linear gene interactions. Our Petri nets have a unique architecture, that being one in which any finite number of interacting gene units can be modelled. Our method also constructs multiple perfect Petri nets ranked by parsimony for a problem, which are then amenable to statistical analysis in order to discover common interaction patterns. These patterns may form the basis for future experimental investigation in a laboratory.

Constructing Petri nets manually or using a different automatic method that produces only a single solution excludes the possibility of this kind of aggregate analysis.

There are two main further avenues future refinement of this method. Firstly, more biological or common sense knowledge could be incorporated as constraints.

Secondly, a greater focus could be placed on the aggregate analysis side, in which significant patterns across many solutions are identified. Tools such as data and graph mining can be applied fruitfully here. The drawback of this approach, of course, is that the output is not an executable model as a normal Petri net typically is; rather, it would be a set of qualitative graphical patterns (for example, "the output of genes X and Y combine together to activate gene Z").

Yet, biologists and geneticists are somewhat more interested in reducing epistatic models to a limited number of simple reactions invariant to randomness that can eventually be tested *in vivo* or *ex vivo*, rather than having a complex, albeit mathematically-perfect model that is difficult to test by translational research experiments. These "core" hypotheses could, for instance, be verified by perturbation experiments in simple organisms, a well-known and successful strategy in dissecting biological epistasis (Phillips, 2008).

To achieve this goal, it will be necessary to develop a more intelligent and therefore faster method for learning Petri nets, especially when the size of the desired Petri net is larger. A genetic algorithm, although powerful, is still a heuristic search strategy and therefore is not guaranteed to always find an optimal solution. Random mutations to arc presence, weight and direction take little account of Petri net dynamics, and our observations have shown that the fitness function frequently does not vary smoothly with minor changes or mutations to the net's structure. Future research should therefore also

be directed towards devising a more efficient method of searching the space of possible Petri nets.

Tables

Table 1

Genotype	**Rule for computing *g***
AA	2*x*
Aa	*x*+*n*
aa	2*nx*

Table 2

	T_0	**T_1**
P_0	1	3
P_1	i	0
P_2	-2	1
P_3	0	-4

Table3

Petri Net depicted in…	**Transitions Fired**
Figure 5(a)	$T_3\ T_5\ T_6\ T_6\ T_1\ T_5\ T_6\ T_6\ T_1\ T_7\ T_7\ T_3\ T_6\ T_6\ T_1\ T_5\ T_0$
Figure 5(b)	$T_4\ T_7\ T_7\ T_0\ T_5\ T_2\ T_6\ T_6\ T_5\ T_5\ T_3\ T_7\ T_0\ T_1$

Table 4

Connection	**Frequency**
$P_7 \rightarrow P_1$	382 (96%)
$P_0 \rightarrow P_6$	360 (90%)
$P_0 \rightarrow P_2$	353 (88%)
$P_0 \rightarrow P_4$	323 (81%)
$P_3 \rightarrow P_1$	288 (72%)
$P_5 \rightarrow P_2$	276 (69%)
$P_3 \rightarrow P_6$	240 (60%
$P_7 \rightarrow P_2$	227 (57%)
$P_5 \rightarrow P_6$	214 (54%)
$P_5 \rightarrow P_1$	204 (51%)
$P_7 \rightarrow P_4$	196 (49%)
$P_3 \rightarrow P_4$	184 (46%)
$P_3 \rightarrow P_2$	167 (42%)
$P_7 \rightarrow P_6$	163 (41%)

Figures

Figure 1

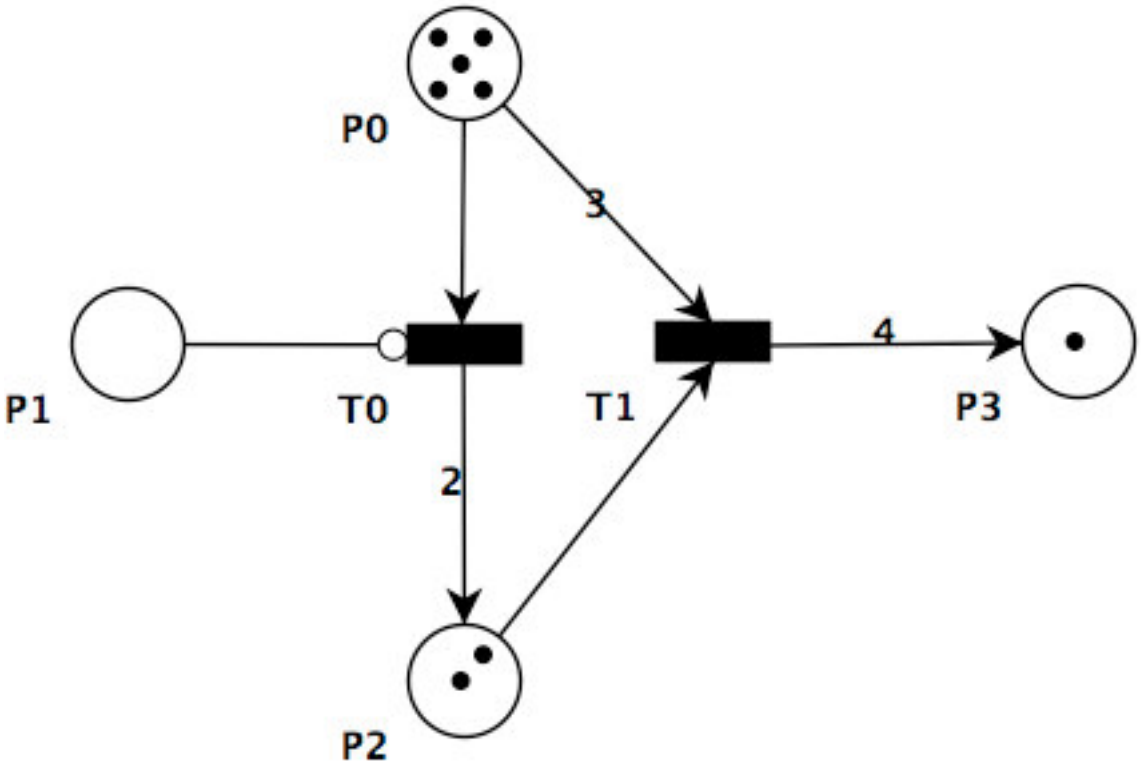


Figure 2

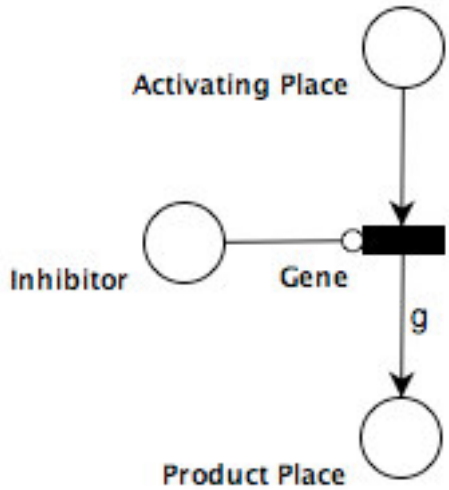


Figure 3

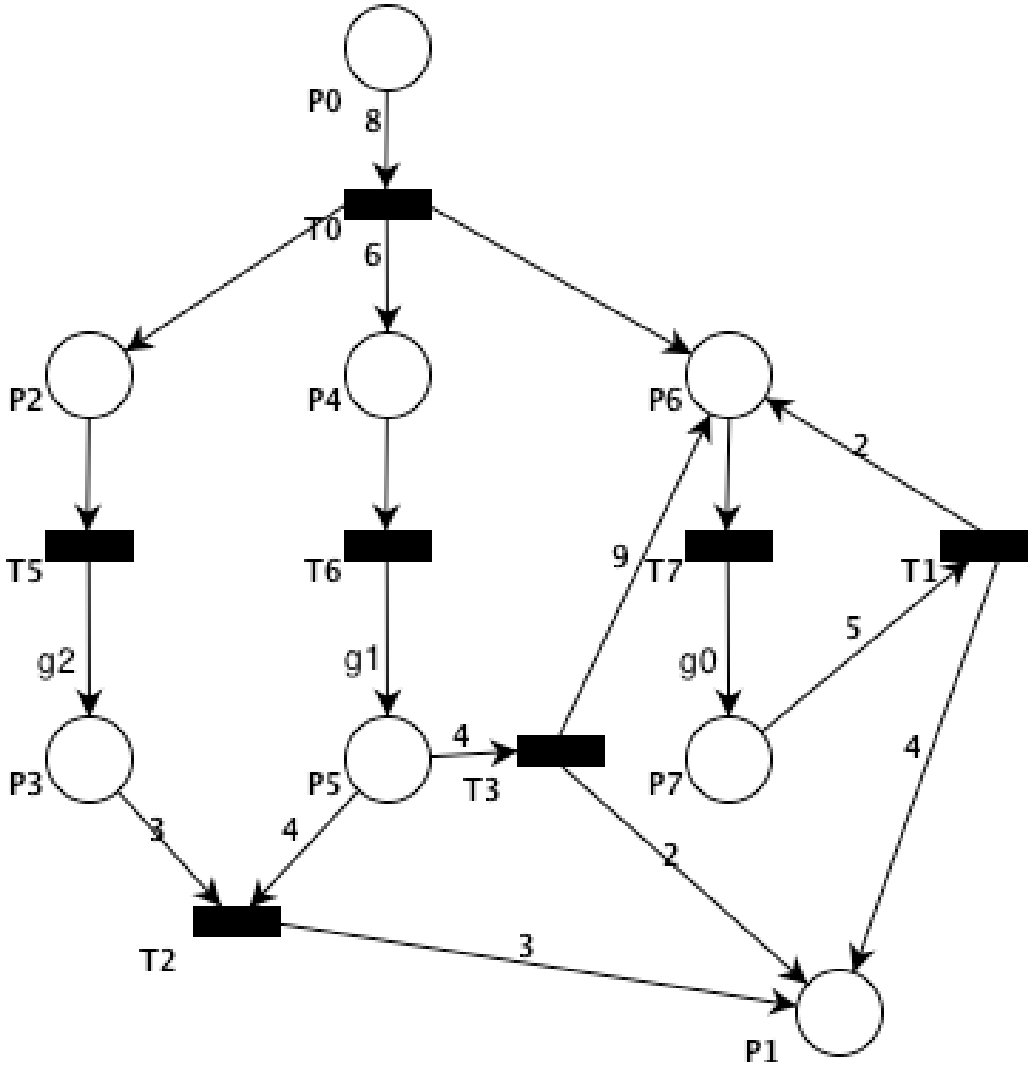


Figure 4

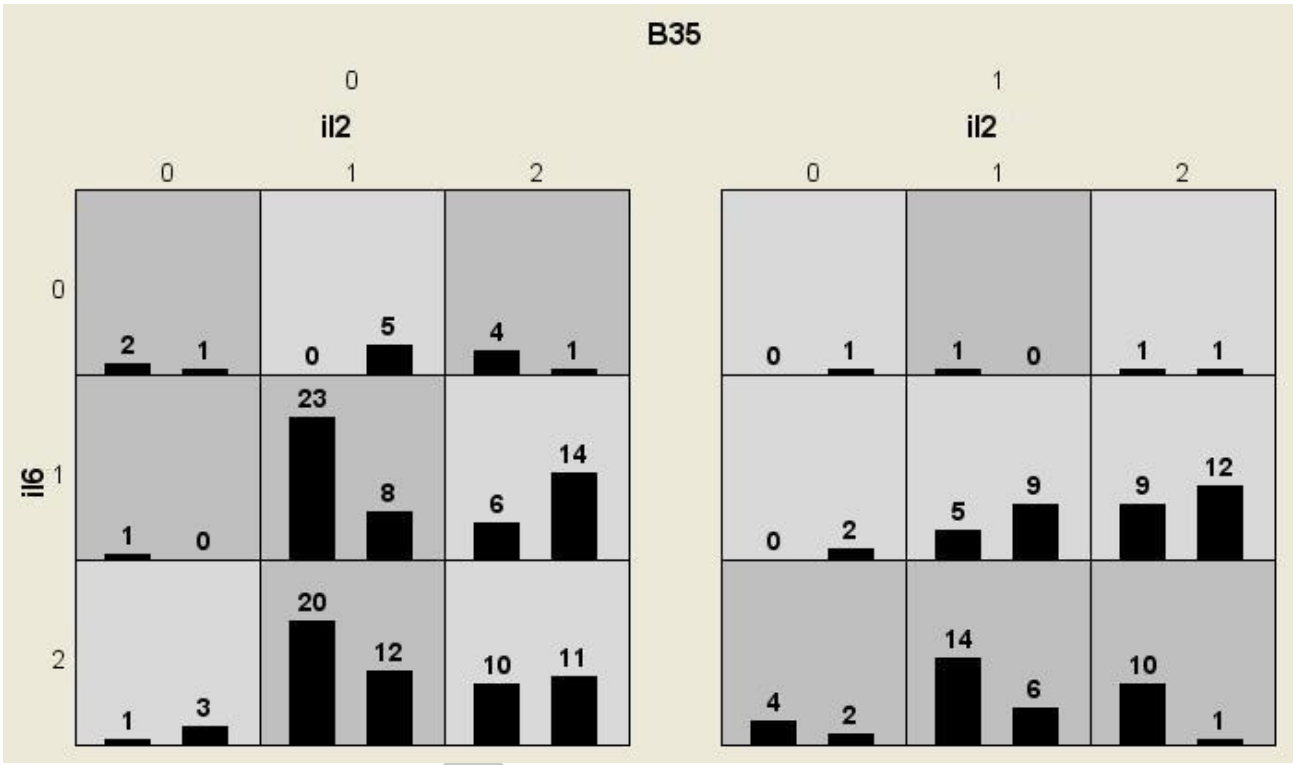
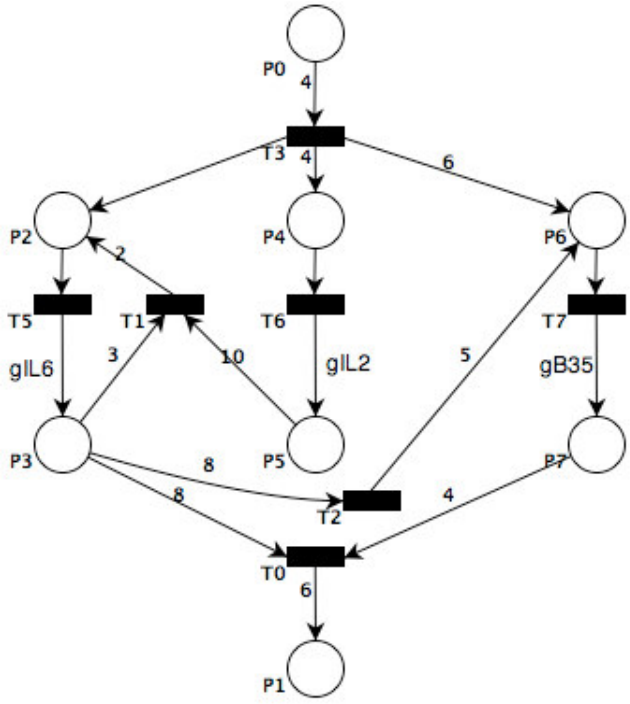


Figure 5(a)

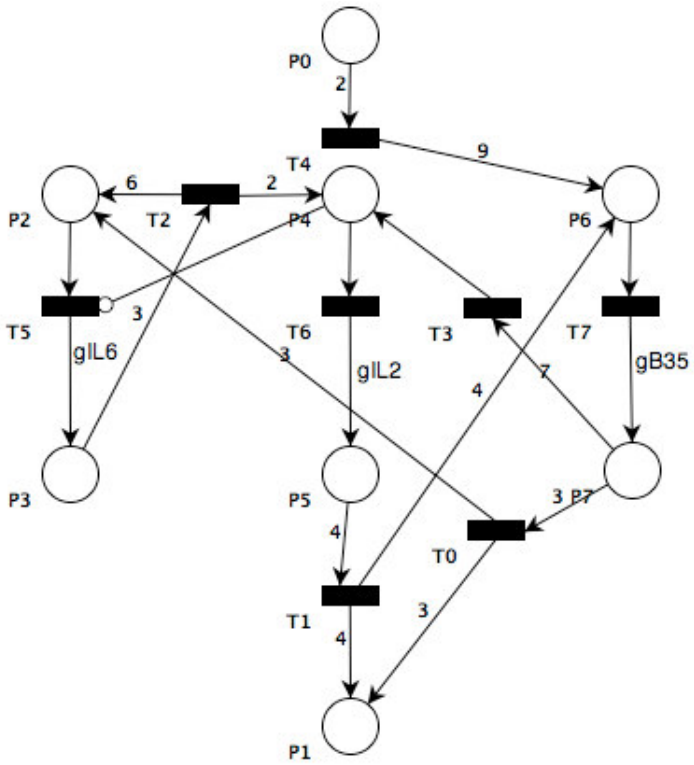


P_1 threshold: 0.5

g values:

	B35	**il2**	**il6**
aa	6	8	8
Aa	--	5	6
AA	2	2	4

Figure 5(b)



P_1 threshold: 0.7

g values:

	B35	**il2**	**il6**
aa	6	8	6
Aa	--	5	4
AA	2	2	2

Legends to tables and figures

Table 1. Rules for computing *g* for each genotype.

Table 2. 2D array representation of the Petri net depicted in Figure 1.

Table 3. Sequence of transitions fired by two different perfect Petri nets in order to exceed the threshold at P_1. The initial marking for P_0 is 10 tokens.

Table 4. Variable single place-place connections and their frequency of occurrence in 400 Petri nets for the simplified digital ulcers model.

Figure 1. An example of a Petri net with four places and two transitions.

Figure 2. Schematic representation of a gene. When tokens are in the activating place (AP), the gene produces at a fixed rate *g* in the product place (PP) unless inhibited.

Figure 3. A Petri net consisting of three gene units.

Figure 4. Multifactor dimensionality reduction (MDR) model of non-linear gene-gene interaction. The model describes the risk of developing digital ulcers in Italian systemic sclerosis patients (Beretta et al. 2009). Key: For IL-2 and IL-6, cell indices 0, 1 and 2 denote genotypes *AA*, *Aa* and *aa* repectively. For B35, cell index 0 indicates *AA*/*Aa* and index 1 indicates genotype *aa*.

Figure 5. Two different but perfect Petri net explanations of the interaction defined by the simplified digital ulcers model in Figure 4.

References

Agrawal R, and Srikant R (1994.) Fast Algorithms for Mining Association Rules in Large Databases. In Proc. 20th International Conference on Very Large Data Bases, pp. 478-499.

Beretta L, Cappiello F, Moore J, Barili M, Greene C. and Scorza R. (2008). Ability of Epistatic Interactions of Cytokine Single-Nucleotide Polymorphisms to Predict Susceptibility to Disease Subsets in Systemic Sclerosis Patients. Arthritis & Rheumatism 59:7, pp 974–983

Beretta L, Santaniello A, Mayo M, Cappiello F, Marchini M and Scorza R (2009.) Genetic and Biological Models of Epistasis to Predict Digital Ulcer Occurrence in Italian Systemic Sclerosis Patients. Article In Submission to Annals of Human Genetics.

Bourdeaud'huy T, Yim P (2002.) Petri net controller synthesis using genetic search. In: Proceedings of the 2nd IEEE Int. Conf. on Systems, Man and Cybernetics (SMC'02) Vol. 1, pp 528-533.

Cheng S, Yeh H, Lin Y, Lin S, Soo V. (2007). Inferring Gene Regulatory Networks from Microarray Data Based on Transcription Factor Analysis and Conditional Independency. BIOCOMP 2007: 65-71

Durzinsky M, Wagler A, Weismantel R, and Marwan W. (2008.) Automatic reconstruction of molecular and genetic networks from discrete time series data. Biosystems 93:3 pp. 181-190.

Fishman D, Faulds G, Jeffery R, et al. (1998) The effect of novel polymorphisms in the interleukin-6 (IL-6) gene on IL-6 transcription and plasma IL-6 levels, and an association with systemic-onset juvenile chronic arthritis. J Clin Invest. 102:1369-76.

Goldberg, D (1989.) Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley.

Hoffmann SC, Stanley EM, Darrin Cox E, et al. (2001) Association of cytokine polymorphic inheritance and in vitro cytokine production in anti-CD3/CD28-stimulated peripheral blood lymphocytes. Transplantation. 72:1444-50.

Lee D, Kim S, and Kim Y. (2007.) BioCAD: an information fusion platform for bio-network inference and analysis. BMC Bioinformatics 2007, 8(Suppl 9):S2.

Lin Y, Yeh H, Cheng S, and Soo V. (2007). Comparing Cancer and Normal Gene Regulatory Networks Based on Microarray Data and Transcription Factor Analysis. In Proc. of the 7th IEEE International Conference on Bioinformatics and Bioengineering, BIBE 2007, pp. 151-157.

Liu B, Hsu W, Ma Y (1998.) Integrating Classification and Association Rule Mining. In Proc. Fourth International Conference on Knowledge Discovery and Data Mining, pp. 80-86.

Matsuno H, Doi A, Nagasaki M, et al (2000). Hybrid Petri net representation of gene regulatory network. Pac Symp Biocomput, 5:338-49.

Mauch H (2003) Evolving Petri nets with a genetic algorithm. In Proc. of Genetic and Evolutionary Computation (GECCO 2003), Lecture notes in computer science 2724. Springer, Berlin, pp 1810-1811.

Mayo M. (2005). Learning Petri net models of non-linear gene interactions. BioSystems, 82(1), 74-82.

McGarry K, Loutfi M and Moscardini A. (2007). Stochastic Simulation of the Regulatory Pathways involved in Diabetes using Petri-nets. In Proc. of the International Conference on Computer Theory and Applications (ICCTA2007), Alexandria, Egypt.

Meffert K, et al. (2009.) JGAP - Java Genetic Algorithms and Genetic Programming Package. URL: http://jgap.sf.net

Moore J, and Hahn L. (2003.) Petri net modelling of high-order genetic systems using grammatical evolution. BioSystems 72, 177–186

Moore J. (2003.) The Ubiquitous Nature of Epistasis in Determining Susceptibility to Common Human Diseases. International Journal of Human and Medical Genetics 56:1-3

Moore J. (2004.) Computational analysis of gene-gene interactions using multifactor dimensionality reduction. Expert Review of Molecular Diagnostics 4:6, pp. 795-803.

Nummela J, and Julstrom B. (2005.) Evolving Petri nets to represent metabolic pathways. In Proceedings of the 2005 Conference on Genetic and Evolutionary Computation, pp. 2133 – 2139.

Phillips, PC. (2008) Epistasis--the essential role of gene interactions in the structure and evolution of genetic systems. Nat Rev Genet. 9:855-67.

Pociot F, Molvig J, Wogensen L, Worsaae H, Nerup J (1995). A TaqI polymorphism in the human interleukin-1β (IL-1β) gene correlates with IL-1β secretion in vitro. Eur J Clin Invest. **22**:396–402.

Reid D. (1998.) Constructing petri net models using genetic search. Mathematical and Computer Modelling 27:8 pp. 85-103.

Reisig W. (1985.) Petri nets: an introduction. In: EATCS Monographs on Theoretical Computer Science. Springer-Verlag.

Zhao X, Zhou J and Liu H. (2007.) Research of Self-Learning Petri Nets Model for Fault Diagnosis Based on Rule Generation. In Proc. International Conference on Machine Learning and Cybernetics, Vol 2 pp. 1106-1110.