

RESEARCH

Open Access



Learning versus optimal intervention in random Boolean networks

Matthew R. Karlsen¹, Sotiris K. Moschoyiannis^{1*} and Vlad B. Georgiev²

*Correspondence:

s.moschoyiannis@surrey.ac.uk

¹Department of Computer Science,
Faculty of Engineering and Physical
Sciences, University of Surrey, GU2
7XH Guildford, Surrey, UK
Full list of author information is
available at the end of the article

Abstract

Random Boolean Networks (RBNs) are an arguably simple model which can be used to express rather complex behaviour, and have been applied in various domains. RBNs may be controlled using rule-based machine learning, specifically through the use of a learning classifier system (LCS) – an eXtended Classifier System (XCS) can evolve a set of *condition-action* rules that direct an RBN from any state to a target state (*attractor*). However, the rules evolved by XCS may not be optimal, in terms of minimising the total cost along the paths used to direct the network from any state to a specified attractor. In this paper, we present an algorithm for uncovering the optimal set of control rules for controlling random Boolean networks. We assign relative costs for interventions and ‘natural’ steps. We then compare the performance of this optimal rule calculator algorithm (ORC) and the XCS variant of learning classifier systems. We find that the rules evolved by XCS are not optimal in terms of total cost. The results provide a benchmark for future improvement.

Keywords: Controllability, Complex networks, Rule-based machine learning, XCS, Optimality, Intervention cost, Weighted graphs

Introduction

Research on networks and network control has been motivated by the ability to better understand, and intervene in, large-scale complex systems (Krause et al. 2009) such as gene regulatory networks, transportation networks, and the Internet. Applications range from medical interventions and ecosystems management to power grids management. Perturbations may cause the network to spontaneously go to a state that is less desirable than others, e.g., perturbations in metabolic networks may indirectly lead to non-viable strains (Cornelius et al. 2013). Recent advances include work on network structure *control nodes* (Liu et al. 2011; Bianconi et al. 2009; Moschoyiannis et al. 2016), its reconfiguration (Haghighi and Namazi 2015; Savvopoulos and Moschoyiannis 2017; Savvopoulos et al. 2017), but also on the network dynamics (Cornelius et al. 2013; Karlsen and Moschoyiannis 2018a) with application for example to transport (Karlsen and Moschoyiannis 2018b). ‘*Controllability*’, is here measured by the extent to which we have the ability to direct the network from any (possibly ‘bad’) state to an *attractor* (possibly a ‘good’ state or cycle of states, where the system continues to perform its functions). Herein the network is said to be ‘*controlled*’ when the technique applied can take the network from any state to the target attractor. We focus here on developing sets of ‘control rules’ able to achieve control for a given target attractor.

In previous work (Karlsen and Moschoyiannis 2018a), we have applied rule-based machine learning in the form of an “eXtended Classifier System” (XCS) (Wilson 1995; 1998) to the problem of controlling random Boolean networks (RBNs) (Kauffman 1969; 1993). XCS is an accuracy-based Learning Classifier Systems (LCS) developed originally by Butz and Wilson (2001) to target reinforcement learning problems. So XCS can function by receiving only the utility of the function effected on the environment (*reinforcement learning*). The learning framework evolves around rules - matching, predicting, updating, discovering. A rule follows an IF : THEN expression and comprises two parts; the *condition* (or *state*) and the *action* (or *class* or *phenotype*). The fact they are based on rules means the model and the outcome can be interpreted by both humans and machines. This in turn allows to understand why XCS produces a specific outcome and how it arrives at it - what is often referred to as *explainable AI*.

RBNs present a simple model which can relatively quickly start to exhibit rather complex behaviour. They have been used to model biological networks such as gene regulatory networks, e.g., see Kim et al. (2013); Gates et al. (2016). Control of Boolean networks is therefore relevant to understanding how to intervene in biological systems such as gene regulatory networks and direct them towards a desired state and away from undesired states. In Karlsen and Moschoyiannis (2018a), we showed that XCS can evolve a rule set that takes an RBN from any state to a specified attractor. Therefore, XCS has been successfully applied to *multi-step* problems where the reward from effecting the selected action onto the environment is not known immediately, as in *single-step* problems, but several steps later, i.e., once the network has reached the specified attractor. However, no indication of the ‘ideal’ or ‘optimal’ set of rules was supplied.

XCS provides an interesting approach to controlling BNs due to the ternary structure of the conditions of the condition–action rules used within XCS. This ternary structure (using the symbols 0, 1 and #) permits the creation of a compressed human-readable set of control rules for a given network. There are also potential interesting tie-ins with studies of canalisation, such as the work of Marques-Pita and Rocha (Marques-Pita and Rocha 2013). Like other heuristic approaches, XCS does not require full knowledge of the state space in order to control the network.

In this paper, we provide an algorithm that derives an optimal set of control rules for a given network and a given intervention cost, which is represented by a weight greater than one for intervention links (in contrast to a weight of ‘1’ for a ‘natural step’). This enables a comparison between the XCS rules evolved to control the network and the ‘ideal’ or ‘optimal’ set of rules that does the job. The idea for taking a cost-based approach rather than restricting the operations available to XCS can be attributed to Fornasini and Valcher (Fornasini and Valcher 2014) (though they do not work with XCS). The rule sets uncovered may be considered optimal in terms of total cost (as explained after the algorithm specification below). We currently do not aim to *optimise the number of rules* in addition to optimisation of the total cost, though we do endeavour to produce a relatively compact rule set.

Whilst RBNs are often considered from a ‘macro sense’, focusing on their overall statistical properties for given values of N and K , here we focus on RBNs in a ‘micro sense’, considering the control of given generated instances with specific values of N and K . Indeed, since an algorithm must run against a concrete network instance, analysis at the micro level is critical before proceeding with a more aggregated approach. (Clearly

such an aggregate approach would require significantly higher levels of computational resources.)

The remainder of this paper is structured as follows. “[Random Boolean Networks](#)” section briefly describes RBNs, and then considers control of RBNs, whilst “[Learning Classifier Systems](#)” section outlines key principles behind LCSs. In “[Optimal Control Rules](#)” section we describe the central algorithm of the paper, which computes the optimal set of control rules for an RBN. “[Experiments](#)” section explains the experiments conducted to compare this algorithm with an XCS-based approach. Results are presented in “[Results](#)” section. We consider the scalability of the proposed optimal rule calculator algorithm in “[Scalability](#)” section, and briefly illustrate the control of a BN derived from a real-world system in “[Application to a Boolean network model of a GRN](#)” section. Discussion is provided in “[Discussion](#)” section. Conclusions and future work follow in “[Conclusions and Future Work](#)” section.

This work is an extension of the previous paper of Karlsen and Moschoyiannis (Karlsen and Moschoyiannis 2018c) via the addition of the critical control diagrams omitted in the original work as well as additional detail regarding the prior state of the art. We also provide an application to a Boolean network model of a real-world system, namely the cell cycle of fission yeast (Wang et al. 2010) to show the merit of the proposed approach to real-world applications. Finally, the scalability aspects of the optimal rule calculator are also considered in this paper.

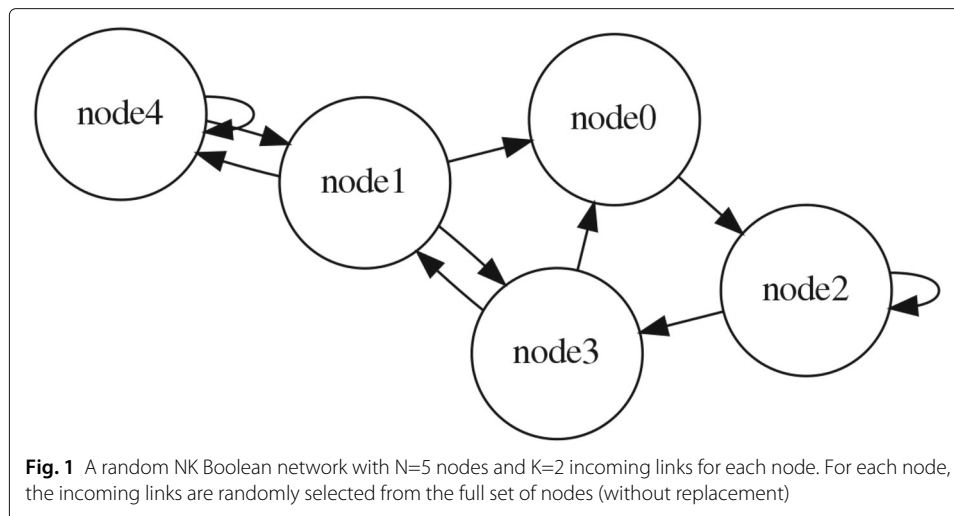
Random Boolean Networks

Random Boolean Networks (Kauffman 1969; 1993) are networks with Boolean values and an update function at each node, where the network structure and/or the update functions are specified at random. Here we focus on the NK Boolean Network (Kauffman 1969; 1993) with N nodes and K inputs per node. In this model the origin node of each input link is randomly drawn from the set of N nodes such that each of the N nodes is affected by K nodes in the same set (a node may have an input from itself). Each node holds a Boolean variable, initialised randomly with the value 0 or 1. The update functions for each node are randomly determined Boolean functions such that each unique combination of input values supplied by the input links (0,0; 0,1; 1,0; or 1,1; when $K=2$) updates the node to a new value (either 0 or 1), overwriting the previous value. An example RBN is shown in Fig. 1.

Random Boolean networks may be asynchronous or synchronous. In an asynchronous network one node is randomly selected to update first, its inputs are read, and a new value is calculated and set at that node. The new value then propagates down any outward links attached to that node, updating further nodes, and so on, until a stable state or state cycle is reached. In a synchronous RBN all nodes are updated simultaneously. Given the current state of the network – the Boolean value at each node – the new Boolean values are calculated without being set. Once all the calculations are complete, a new Boolean value is set at each node. Herein we consider networks that update synchronously.

Control of Boolean Networks

This section focuses on how Boolean networks may be controlled. Note that there are papers which focus on Boolean networks *as control systems* – we are not focusing on such aspects in this section.



As described (above) for networks more broadly, RBNs can be controlled via interventions at available targeted nodes *or* via inputs to external nodes attached to the system. Intervention at targeted nodes is in the form of timed bit flips. A bit flip simply alters the Boolean value at a given node – if the current value is a 0 it can be flipped to a 1 or vice versa. This can be represented as an integer (starting at index 1) indicating which node to bit flip. The state of the network can be shown as a bit string consisting of the state of each node in the network in a fixed order, as shown in Fig. 2. Thus if you have, for instance, 5 nodes then the state may be 00101 and the action could be 4 (shown as 00101 : 4). In this instance flipping the 4th bit from a 0 to a 1 would result in the state 00111. Fig. 2 shows the state space of the RBN ($N=5$, $K=2$) shown earlier in Fig. 1.

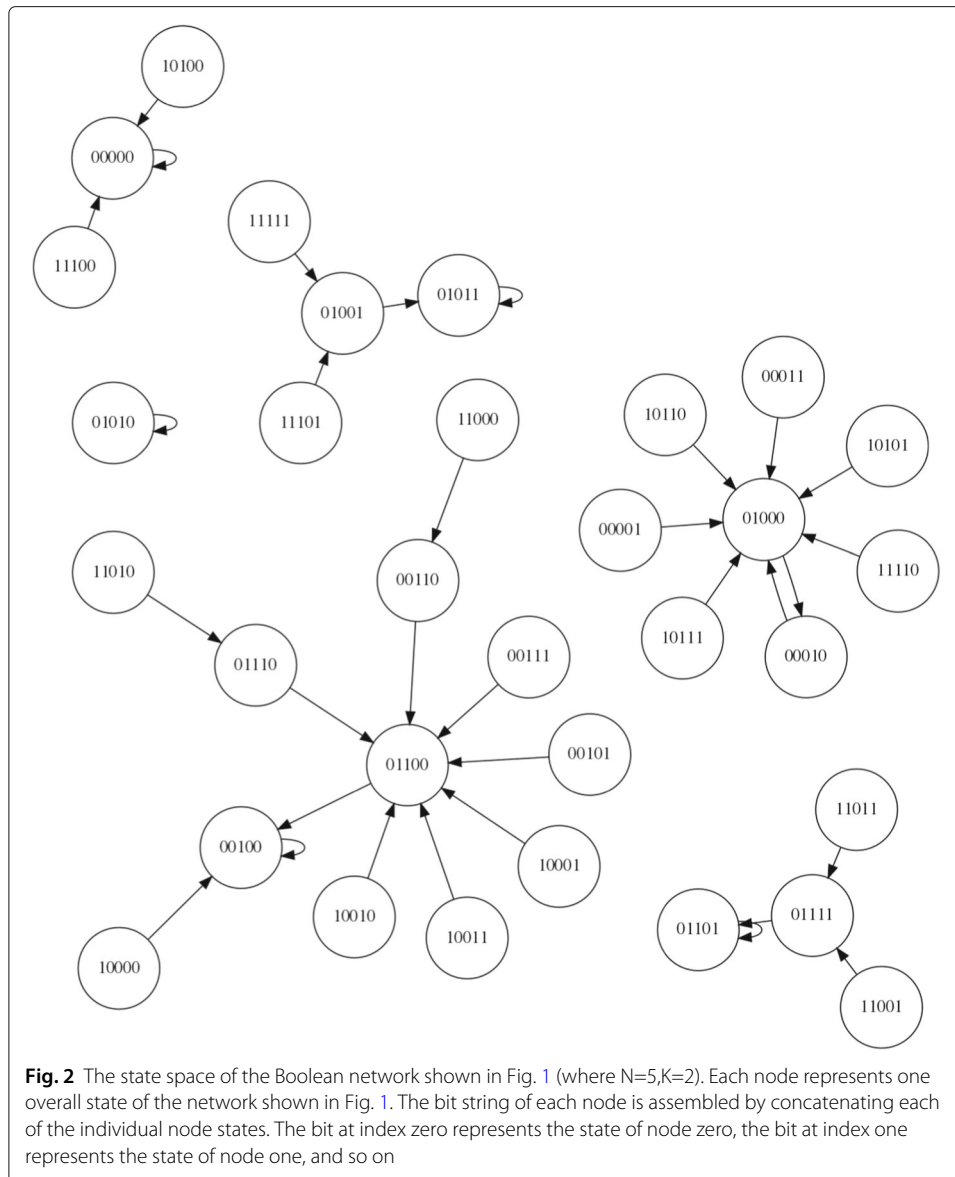
Before the work on control of Boolean networks is considered we can briefly note some of the key dimensions that separate the works. These are as follows:

- a focus on synchronous networks or asynchronous networks
- the definition of control used
- the levers considered available to achieve control
- the provision of optimal versus good solutions
- the maximum size of the networks the technique is applicable to

We divide the works below into the two over-arching categories of ‘Heuristic and Sub-Optimal Control Techniques’ and ‘Optimal Control Techniques’ and then consider the relevant papers in chronological order. (Other organisational schemes could also be applied.)

Heuristic and Sub-Optimal Control Techniques

Following initial formative papers on Boolean networks (Kauffman 1969), substantial work was conducted on understanding the properties of such networks. However, comparatively little work was aimed at controlling such networks. Luque and Solé (Luque and Solé 1997) is one exception, though their notion of control differs from later works aimed at controlling Boolean networks. The focus is on intervening in a synchronous Boolean networks with ‘chaotic’ unpredictable paths (typically produced with larger values of K)



to produce a predictable state cycle. The intervention is direct, in that a certain proportion of node values are altered periodically. Simulations involve networks with a number of nodes in the low hundreds.

Major work on controlling Boolean networks appears to have *started* with E.R. Dougherty and collaborators on the Boolean network super-set of ‘Probabilistic’ Boolean networks (Akutsu et al. 2007). See for example, the work of Shmulevich et al. (Shmulevich et al. 2002) (and Datta et al. (Datta et al. 2003), discussed below in the ‘optimal’ section).

The work of Shmulevich et al. (Shmulevich et al. 2002) focuses on control of synchronous networks via modification of the Boolean functions themselves, in sharp contrast to many of the papers described here. Networks of five nodes are considered and the technique used involves a genetic algorithm.

Focus on Boolean networks (rather than Probabilistic Boolean Networks) specifically appears to have started with the work of Akutsu et al. (Akutsu et al. 2007). Akutsu et al.

focus on synchronous networks and have a notion of control that involves arriving at state Y on a certain time step, having started at state X . The levers considered are in the form of ‘external control nodes’ (equivalent to the ‘control input’ nodes of Datta et al. (Datta et al. 2003)). This initial work focuses on controlling a structural subset of all Boolean networks (networks with a tree structure or small networks with only one or two loops) making the applicability of the technique limited. However, their work does recognise control of most Boolean networks as an NP-hard problem, with solution algorithms unable to produce solutions in polynomial time frames unless the network has a tree structure.

Shi-Jian and Yi-Guang (Shi-Jian and Yi-Guang 2011) return to the notion of control put forward by Luque and Solé (Luque and Solé 1997), developing a technique that reduces the number of node modifications required to achieve this notion of control. The sensitivity of Boolean function can be measured as the number of input bits where a change of value would lead to a change of the output value. The authors introduce a new control strategy based on the expected average sensitivity of the acting nodes and evaluation of the value of γ_c^{max} , i.e. the fraction of nodes to be frozen in order to reduce the chaotic behaviour of the system. This is possible after freezing the nodes with high sensitivity. The analysis is then tested on a standard RBN of size 1000 and $K = 3$.

Whilst Cornelius et al. (Cornelius et al. 2013) does not focus on Boolean networks, they consider the control of biological systems from which BNs may be constructed and their work must therefore be recognised as relevant. Their notion of control focuses on successfully shifting the network from a single state to another target state. The levers they consider are bit flips of the nodes themselves, with focus on a ‘control set’ (subset of all the nodes). Their biological example network includes 60 nodes.

Kim et al. (Kim et al. 2013) focus on discovering ‘control kernels’ (subsets of the nodes in the network that can be used to control the system) in biological Boolean networks. Their approach involves genetic algorithms. Their investigations support the notion that the controllability of a network depends on both structure and behaviour (i.e. the logic gates used, in the Boolean network instance). The control notion used aims at controlling the network from any state to any target attractor. Their control levers are bit flips applied to the control kernel. The networks they consider are quite large (139 nodes, for example).

Li et al. (Li et al. 2015) consider both the controllability and observability of a Boolean network. Boolean networks are presented in n -dimensional polynomial form and the controllability and observability of a network is tested by leveraging powerful tools using computational algebraic geometry, theory of commutative algebra, and the Gröbner basis algorithm. They use the Liu et al. (Liu et al. 2011) definition of controllability – from any initial state to any target state in finite time. The three problem networks considered range from 8 node networks to 58 node networks. The levers in each problem are external nodes.

Gates and Rocha (Gates et al. 2016) analyse the control of many biological Boolean networks, making use of the idea of ‘network motifs’. They argue strongly that control approaches must consider both structure and function (in a similar vein to Kim et al. (Kim et al. 2013)). Their notion of a CTSG (controlled state transition graph) forms a key building block of the optimal control algorithm described within this paper.

Zañudo et al. (Zañudo et al. 2017) argue that feedback vertex set control would be the most suitable one for control of biologically inspired systems. A feedback vertex set (FVS) is the set of nodes that would cause the digraph to become acyclic if were to be removed.

The central idea is to obtain control over the FVS nodes whilst they still have influence on the rest of the nodes in the network. In contrast to other methods, to apply such control we do not need all system's parameters and detailed information about the system's dynamics. This is purely a structure-based control approach.

Paul et al. (Paul et al. 2018) provide a strong heuristic method for control of asynchronous Boolean networks. The idea underlying the decomposition based approach is to use the strongly connected components to divide the network into blocks which then form a directed acyclic graph. The target attractor is then projected on to those blocks and the blocks are re-combined to compute the global basin of attraction. Strongly connected components are the sets of nodes in which a path exists for every nodes pair. The technique described is suitable for quite large networks, and is applied to both real-life biological networks and random generated networks.

Finally, Taou et al. (Taou et al. 2018) take an interesting genetic algorithm-based approach to the problem. They use the GA to evolve a Boolean control network that is then used to control a target Boolean network model.

Optimal Control Techniques

In contrast to the work of Shmulevich et al. (Shmulevich et al. 2002), Datta et al. (Datta et al. 2003) focus on an intervention method of additional external 'control input' nodes to control synchronous Boolean networks. (From this point onward this is a common means of intervention in the network considered.) Their notion of control involves bringing about a desired state sequence and, more precisely, the minimisation of a cost function. The approach is based on the use of Markov chain analysis to formulate a traditional control problem that is then solved via dynamic programming. They apply their technique to a seven node network (reduced from ten nodes via a preprocessing technique).

Cheng and Qi (Cheng and Qi 2009) tackle Boolean networks without focusing on a simplified subset of problems. They use a 'discrete-time dynamics' approach on synchronous Boolean networks. Their notion of control involves controlling the network from chosen initial state X to a second state Y . They consider, separately, both common lever types – external nodes and modification of the state of internal nodes. However, the analysis seems to focus on only three-node networks and the technique used may not be applicable to larger networks.

Kobayashi and Hiraishi (Kobayashi and Hiraishi 2012) take a Petri Net and integer dynamics approach to controlling *asynchronous* Boolean networks. This appears to be the first paper that considers asynchronous networks, that tend to be harder to control due to their more complex state space graphs (since a given state, say 010, is no longer sufficient to work out the next state of the network – we must also know which node is going to update next). Control is again considered as the ability to take the network from a given initial state to a given target state. They focus on three node networks and, as they acknowledge, their approach is only suitable for small networks.

Fornasini and Valcher (Fornasini and Valcher 2014) also take an analytical approach to optimal control of RBNs. The levers that they consider are external input nodes. Fornasini and Valcher also provide a recent review of a subset of the Boolean network control literature and also consider the problem of finding control paths *whilst avoiding certain dangerous or undesirable states in the system* (Fornasini and Valcher 2016).

The Approach Herein

The work herein differs in relation to previous works on *optimal* control of RBNs by (1) taking a programmatic approach rather than an analytical approach and (2) providing the control instructions as a condensed set of ‘control rules’. The work also provides a comparison between an optimal non-machine learning-based approach and an approach using rule-based machine learning (RBML) or genetics-based machine learning (GBML). We recognise that our approach uncovers optimal rule sets (in terms of minimisation of the control cost) and is therefore not directly competitive with some of the most recent heuristic approaches that can work out *reasonable* solutions to networks with many more nodes. Our primary intent is to establish a baseline with which we can compare XCS solutions and establish a ‘road map’ for improvement of RBML-based solutions to the BN control problem, rather than provide an approach that is directly superior to the heuristic methods in the review above.

Learning Classifier Systems

A learning classifier system (LCS) (Urbanowicz and Moore 2009) is an RBML technique comprising a population of rules, a reinforcement (or supervised) learning mechanism and a genetic algorithm. Further additional components include a number of filters, a ‘covering mechanism’ (to generate new rules when needed), a prediction array (to assess the quality of proposed actions) and an action selector (to choose which of the potential actions actually gets implemented in the environment).

At the heart of an LCS is a number of IF <condition> THEN <action> rules in a *population*. The rules in this instance are represented as follows: 10100 : 3 (a bit string condition followed by a colon separator, followed by an integer representing an action). The condition consists of 0s, 1s and # symbols (where the # wildcard indicates ‘don’t care’). The environment state (represented as a bit string) matches a rule’s condition when at each index the symbol either matches or the condition is #. For example, 11101 will match 111#1 : 5. ##### : 1 will match *any* environment, taking action 1. This rule format is essentially the same as that used by Marques-Pita and Rocha in their study of canalisation (Marques-Pita and Rocha 2013).

The overall system for the XCS (Wilson 1995; 1998) variant can be found in Karlsen and Moschoyiannis (2018a) (see also Fig. 3). The implementation used is that described in the algorithmic specification provided by Butz & Wilson (Butz and Wilson 2000), with a post-run rule set compression based on (Wilson 2001). The description of XCS that we provide below is a somewhat compressed summary of the specification in (Butz and Wilson 2000) – a paper that we strongly recommend to those aiming to implement the XCS algorithm. (Though we firmly believe in the value of self-contained articles we are unfortunately unable to describe XCS in full here without essentially reiterating all of Butz and Wilson (2000).)

The overall steps of the algorithm are as follows:

1. The RBN state is read in as a bit string (e.g. 10100)
2. The subset of rules that match the RBN state are selected from the rule population
3. The size of this ‘match set’ is considered. While match set too small:
 - (a) Generate a new rule
 - (b) Add the rule to the population

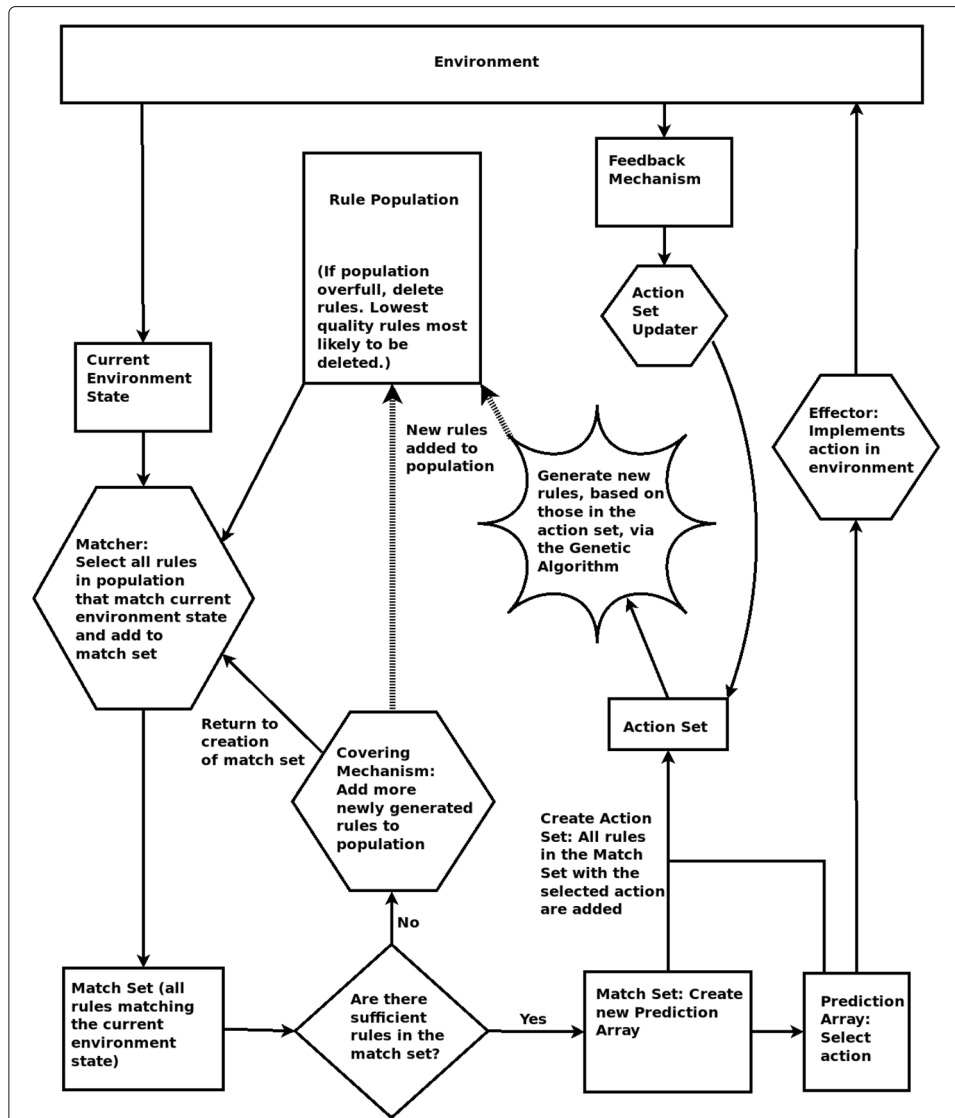


Fig. 3 A diagrammatic representation of the XCS system, adapted from (Karlsen and Moschoyiannis 2018a). The XCS system consists of the entire diagram, excluding the uppermost box which represents the environment that the XCS system interfaces with (in this instance, the Boolean network). The current environment state is in the form of a bit string, such as 10010. The feedback supplied to the feedback mechanism is a reward value (either 0 or 1000 in this instance). The effector instruction consists of simply an index value, which instructs the effector to perform a bit flip on the bit string value at that index

(c) Regenerate the match set

4. Create a prediction array (contains fitness-weighted payoffs for each match set action)
5. Select an action (random with a certain % probability, or else best action picked from action set)
6. Effect action in the environment (i.e. alter one state bit within the RBN)
7. Derive the action set from the match set (the subset that suggest the effected action)
8. Apply the genetic algorithm to the action set, adding new rules to the population
9. Repeat the above steps

We now explain the workings of XCS in further detail. First, the RBN state is converted to a bit string by the classifier system’s ‘detectors’. Since we number each node in the RBNs that we generate, the conversion of the RBN node values to a bit string is performed by simply concatenating the bit values of each node in sequence. If node one’s state is 1, node two’s state is 0, node three’s state is 1, and so on, then the overall RBN state is simply 101...

Once the state of the environment is acquired, this is compared to each of the rules within the rule population. Those rules that match (as described earlier in this section) are added to a ‘match set’. The number of *distinct* actions present within the rules of the match set are then considered. If there is one action per bit to flip (i.e. at least one rule with the flip bit n action, for all possible values of n), then the algorithm proceeds to the next step (prediction array creation). If not, a covering mechanism is used to generate more rules that are added to both the current match set and the population. The covering mechanism generates a new rule by setting the rule’s action to match the current environment state and then attaching an action not present in the current match set. For each index of the new rules condition a random number is drawn in the range $[0, 1)$ and compared to the $P_{\#}$ parameter. If less than this parameter, the value at that index is converted to a hash symbol. After each addition of a new rule, the exit criterion is again assessed, until it succeeds (proceeding to prediction array creation). If at any time the number of rules in the population, via this process, exceeds the maximum permitted number, selection operates on the population to remove rules and return the population to its maximum permitted size (see the paragraph proceeding the GA explanation below for details).

The prediction array is used to decide which action is selected for implementation in the environment (and which rules are selected for introduction to the innovation generation mechanism (the genetic algorithm)). The rules in the match set are split by action. A fitness-weighted sum of the predicted payoffs is then created for each action (i.e. the predicted payoff of each rule is multiplied by its fitness and added to the total). Each sum is then divided by the sum of the fitness values for the action in question (Butz and Wilson 2000), to produce an overall predicted payoff for the action. Here, the fitness of a rule is an indication of its ‘reproductive potential’. The fitness update algorithm (Butz and Wilson 2000) proceeds as follows. Each classifier in the action set is assigned an accuracy of one. Classifiers in the action set with error greater than or equal to the parameter ϵ_0 have this value overwritten with the value $\alpha * (\epsilon_{cl}/\epsilon_0)^{-\nu}$ where ϵ_{cl} is the classifier error (see Table 1 for parameter explanations). An accuracy sum is calculated by summing the accuracy of each classifier multiplied by the numerosity of each classifier. Fitness f_{cl} is then calculated as in Eq. 1, adapted from Butz and Wilson (Butz and Wilson 2000, p10).

$$f_{cl} = f_{cl} + \beta * (\text{classifier accuracy} * \text{classifier numerosity} / \text{accuracySum} - f_{cl}) \quad (1)$$

During the learning phase of the algorithm, an action is then chosen randomly 50% of the time (explore) or the action with the highest predicted payoff is selected 50% of the time (exploit). The selected action (in this case, an index indicating a bit to flip) or 0 (for no-operation) is then effected in the environment. All rules with the chosen action are retained within a new ‘action set’.

Once an action set is created, it is retained until feedback for the step is acquired from the environment (the system operates with discrete steps). If the action was a bit flip, the new state of the environment is assessed immediately after the bit flip. If the action was a no-operation the RBN is permitted to unfold for a step according to its own pattern and

Table 1 Parameter settings and brief descriptions

Parameter	Value	Brief Description
R	1400	Rule population size
γ	0.71	Discount rate
θ_{mna}	$N+1$	Min. number of actions in match set
$P_{\#}$	0.33	Probability of hash
p_i	0.0	Initial payoff
ϵ_i	0.0	Initial error
F_i	0.0	Initial fitness
ϵ_0	10	Error threshold
θ_{ga}	5.0	Genetic algorithm frequency
θ_{del}	20.0	Deletion threshold
β	0.2	Affects update of p, ϵ , and action set size for classifiers
α	0.1	Affects fitness updates
ν	5.0	Affects fitness updates
χ	0.8	Likelihood of GA crossover operation
μ	0.05	Likelihood of GA mutation operation
δ	0.1	Mods. effect of fitness on the 'deletion vote' of a classifier
θ_{sub}	30.0	Subsumption threshold
p_{explr}	0.5	Likelihood of exploring
<i>doAsSubsumpt.</i>	true	Perform subsumption in the action set?
<i>doGaSubsumpt.</i>	true	Perform subsumption in the GA?

is then assessed. If the action has resulted in reaching a state of the desired attractor a reward of 1000 is supplied to the feedback mechanism. If the action has not resulted in reaching a state of the desired attractor the reward is 0. This value (1000 or 0) is fed in to the action set updater which adjusts the predicted payoff, the payoff error, the 'action set size estimate', and the fitness of each rule in the action set (Butz and Wilson 2000).

Once the action set has been updated based on the feedback obtained from the environment, the action set is supplied as input to the genetic algorithm. This GA functions similarly to a regular GA that is not part of an LCS, except that the GA is clearly operating on the action set, rather than the whole population. The GA selects two rules out of the population (random selection with replacement). These rules become the 'parent' rules. These parent rules are then crossed with each other to produce two new novel combinations with pieces from each parent. Each index (locus) of each of the novel combinations is then mutated with a small probability. After this process, subsumption is checked against the parent rules (see below). If the offspring rules survive this process they are added to the population. If the population becomes over-full, selection operates on the population.

When selection is activated the population is reduced by one (if greater than the maximum population size). The selection technique has a tendency towards removal of the poorest performing rules, though the technique is stochastic and thus this is not guaranteed. For details of this process, see Butz and Wilson (2000).

Subsumption is a technique to reduce the number of rules in the population by removing rules that are superfluous (due to the covered states completely dealt with by a more general rule that points to the same action). There are two points in which subsumption may be used, both within the action set creation and at the end of the generation of new rules within the GA. The subsumption process (detailed within Butz and Wilson (2000))

essentially looks for rules sharing the same action where the condition of one is completely 'enveloped' by the condition of another. These enveloped rules are deleted and the numerosity of the enveloping rule (the count of that rule in the population) is incremented by one.

Optimal Control Rules

An overview of the algorithm for producing optimal control rules is as follows:

Construct initial rules

1. Construct the original state space graph of the RBN (i.e., with no interventions)
2. Assign each original link a weight of 1.0
3. For each state-space node, create one 'intervention' link to each nearest neighbour with an intervention-level weight (2.0, 3.0, etc).
4. For each state-space node:
 - (a) run the Dijkstra shortest path algorithm to each attractor node in the specified attractor
 - (b) store the path with the total least cost (if more than one path exists with the same total cost, the first to be returned by [the GraphStream implementation of] Dijkstra's algorithm is stored)
 - (c) calculate total cost for the path (link weight sum) and store
 - (d) calculate number of interventions (count of link with weight > 1.0) and store
 - (e) for each state node in the path:
 - (i) create a new classifier with condition == current node and action as required (to point to the next step on the shortest path; note that the classifier contains no # symbols)
 - (ii) if no classifier exists with the new classifier condition, add the new classifier to population
5. calculate average statistics (average cost and average number of interventions)

Compress rules (*optional*)

6. Run rule merger:
 - (a) Group classifiers by action
 - (b) For each action group:
 - (i) For each classifier in action group, for every other classifier in action group, check subsumption
 - (ii) If classifier subsumes other classifier, delete classifier to be subsumed and increment numerosity of subsumer
 - (c) For each action group; while all combinations not explored without a merge:
 - (i) iterate through all pair combinations of classifiers
 - (ii) if 'is adjacent or overlaps' then merge the two rules

We shall now explain some steps in greater detail. Construction of the original state space graph of the RBN is performed by initialising the network at each state of the network and then evolving the network to an attractor. For each path traced out in this way, the subset of the state space explored is recorded. When complete, this process yields the full state space of the network. Each link in the original state space network is given a weight of 1.

Once the original state space has been constructed all possible intervention links are added to the graph. These 'nearest neighbour' links are created as follows. For each state space node the neighbouring nodes are identified as those that differ from the selected node by a single bit flip. Then, a directed weighted link is created from the originally selected node to each neighbouring node. The precise weight used depends upon the cost of an intervention.

Next, the Dijkstra shortest path algorithm is applied from every node to every node in the target attractor, using the GraphStream library (Dutot et al. 2007). The algorithm was selected because it is capable of considering weighted edges, unlike the A* algorithm (also in GraphStream). Once the Dijkstra shortest path has been worked out for each of the nodes in the network state space, for each node on each path we create a new classifier with a condition equal to the current node (with no wildcards) and the action required to alter the state of the network to the next state on the shortest path. For instance, if the current state is 11011 and we required a shift to 10011 and the only link connecting the two nodes was a link with weight > 1 then we work out which bit must be flipped (bit 2 in this instance) and then construct the rule 11011 : 2. In contrast, if the current state was 00110 and the next desired state was 00111 and there was an original (weight 1) link connecting the two states then the rule 00110 : 0 would be created where 0 is the 'no operation' action, indicating that no action needs to be taken to alter the state. Upon completion of the classifier, we add the new classifier to the population *only if* no classifier already exists with the same condition.

Note that it is possible for the shortest path algorithm to find more than one shortest path from a node to the attractor. For this reason, we only add a classifier if its condition is not already present in the population.

The rule merger (used for rule set compression, as outlined in the itemised steps above) first groups classifiers by action (this is a quick way of reducing combinations later in the process). Once grouped by action, we check subsumption for each possible pair combination of classifiers (as described within the overview of the LCS structure, above).

Following the subsumption, the main merging code is then initialised, using a while loop that exits when no further modifications are possible. This loop iteratively checks whether the classifier pairs have a matching action and matching condition components, *less one, which is adjacent or overlaps* (i.e. one rule has 0 and the other 1; or one or both have # at the specified location) – if this condition is met the rules are merged.

Post compression, further steps are performed to produce a state-space diagram, displaying both the 'natural' links (in green) and the interventions (in red). Note that the cost of interventions affects the resulting network.

The sets of control rules produced using the above technique are (one) optimal set of control rules for the given network. The rule set is optimal in terms of the control cost, and *may or may not* be optimal in terms of the number of rules generated. Since we do not evaluate every possible 'optimal' path produced by Dijkstra's algorithm, it may be possible that there is a different combination of rules that is more compact than those produced

here. However, to re-iterate, the algorithm does produce one set of control rules that *is* optimal in terms of the total cost of controlling the system (to one target attractor, from all other nodes that are not part of the target attractor).

Whilst we do not provide a formal proof, an intuitive understanding of the optimality of the algorithm can be gained. The variant of Dijkstra's algorithm that we use here (as implemented in GraphStream) considers the link weight (intervention cost) of each link and returns the path with the least total cost (in terms of these link weights). We know that this algorithm unfailingly provides the least-cost path from any node to any other target node. Once we have added all possible intervention links in to the graph, we divide the overall problem in to finding a shortest path from each initial node to the attractor, which consists of one or more nodes. The least-cost path to the attractor (taking in to account the differing cost of 'natural' links and intervention links) is uncovered simply by running Dijkstra's algorithm from the origin node to each of the target attractor nodes, which will unfailingly reveal the least cost path (out of the one or more possible paths). This means that so long as we encode the paths with the rules accurately, we can see that, hinging on the well known quality of the Dijkstra algorithm, a cost-optimal rule set will be created.

Experiments

Here we use the aforementioned optimal rule discovery algorithm to create the optimal rule sets for 10 different networks. We aim to control the network to consistently reach the 'first' attractor in the list of known attractors (i.e that with the lowest value when the bit string representation of the attractor is interpreted as a binary number) with the smallest cost possible. We then run XCS on each of these networks 25 times (with the same target attractor) and produce aggregate performance results (necessary due to XCS' stochastic components). These results are then critically compared to the solutions produced via the optimal rule discovery algorithm. We evaluate 3 levels of intervention cost: 2.0, 2.5 and 3.0.

There are some changes to the way XCS is used in Karlsen and Moschoyiannis (2018a). Rather than enforcing a mandatory 'natural step' after each intervention we introduce the weighting of links such that intervention links are more 'expensive' than 'natural step' links. This necessitates changes to the reward function of the XCS implementation. Reward acts to alter the predicted payoff, payoff error and, indirectly, classifier fitness, of each classifier rule active in the step resulting in the acquired payoff. The payoff and error adjustments provide the classifier system with more accurate knowledge whilst the adjusted fitness enables the GA to better select, and 'reproduce' with, those rules that are advantageous to developing a solution. Reward is given as 0 for an action that does not reach an attractor and 1000 as the reward for an action that reaches the attractor (even if said action is a 'no-op'). Two interventions are therefore permitted in succession. However, because both incur the increased cost this is deemed acceptable. The XCS parameters used are shown in Table 1 (the brief descriptions are based on those in the original table in Karlsen and Moschoyiannis (2018a)). Due to the simplification of the more complex reward structure used in the previous paper, we find that the 'default' XCS parameters (as described by Butz and Wilson (Butz and Wilson 2000)) function suitably. We use these parameters for these initial results, with the advantage that the parameter combination used is not randomly generated by a parameter explorer and thus is more human-comprehensible.

A condensation technique (Kovacs 1998) is used towards the end of the XCS run to reduce the number of rules present in the population. This is applied before the optional compression step as described in Karlsen and Moschyiannis (2018a). After 10,000 steps the XCS algorithm evaluates the solution and proceeds to the condensation phase. If a solution is not present, it continues to run and re-evaluates every 1000 steps. In the condensation phase θ_{mna} is set to 1, and the GA mutation and cross-over are disabled. The XCS algorithm then runs for 20,000 additional steps and evaluates the solution. If the solution is viable the program writes the output and halts. If the solution is not viable the condensation algorithm runs for a further 10,000 steps. It is technically possible that the algorithm will never halt and will require a restart. This is comparatively rare: on the 3004 runs conducted this occurred only 4 times.

It should be noted that the control rules developed, both via the optimal control algorithm and the XCS approach are applicable only for a specific control challenge with a specific attractor. If a different target attractor is selected then a different set of control rules must be developed to direct the unfolding state of the network towards that attractor. Similarly, different values of N and K will result in substantially different network instantiations. Each instantiation requires its own set of control rules. This is a common limitation for techniques controlling BNs – it is not likely to be specific to just our proposed solution.

Results

Table 2 presents a highly condensed summary of the results. Each unique combination of N , K and IC (intervention cost) represents 10 distinct networks of 25 runs using XCS and a single computation of optimal results using the algorithm described earlier in the paper for each distinct network. Each network has an optimal control cost associated with it. The 'Opt. Min' here shows the minimum optimal control costs from the 10 different control costs for the networks. The 'Opt. Avg.' shows the average of these 10 control costs. Finally, the 'Opt. Max' shows the maximum optimal control cost from the 10 different control costs for the networks. The XCS columns are equivalent to the optimal columns except that they apply to the 25 XCS runs per network rather than the application of the optimal rule algorithm (above).

Tables 3, 4 and 5 present the detailed results for the 10 $N=5, K=2$ networks (with intervention costs 2.0, 2.5 and 3.0 respectively). The 'Opt Cost' is the average cost of navigating from any state to one of the target attractor states. To produce this measure the cost of navigation from every node in the network is summed (at a cost of 1.0 per 'natural' link and the specified cost per 'intervention' link) and then divided by the total number of states. 'OC Rule Count' is the number of rules required to achieve the Optimal Cost outcome. 'XCS Avg. Cost' is the average value of all the average costs for each of the 25 runs on the network in question. The 'XCS Rule Count' is the average rule count required to achieve the XCS average cost. The 'XCS time' is the average time (in seconds) required for the 25 runs on a given network.

For each of the networks shown in Tables 3, 4 and 5, there is an associated optimal set of control rules and 25 distinct sets of control rules evolved by XCS. Whilst displaying them all is not feasible due to space constraints, an example set of optimal rules is shown in Table 6, along with two example sets of XCS-evolved rules. The square brackets next to each rule display the final predicted payoff and fitness of each rule, separated by a '/'

Table 2 The minimum, maximum and average costs of the optimal solutions and XCS solutions

N	K	IC	Opt. Min.	Opt. Avg.	Opt. Max.	XCS Min.	XCS Avg.	XCS Max.
5	2	2.0	1.72	2.61	3.59	3.69	5.11	6.42
5	2	2.5	1.63	2.59	3.59	3.73	5.88	7.35
5	2	3.0	1.94	3.13	4.56	4.53	6.43	8.91
5	3	2.0	1.75	2.68	3.34	4.63	6.27	7.69
5	3	2.5	1.78	2.63	3.34	5.10	7.08	9.35
5	3	3.0	2.38	3.45	4.47	6.15	7.55	9.45
7	2	2.0	2.46	3.30	4.44	5.21	6.48	9.24
7	2	2.5	2.41	3.21	4.44	5.72	7.25	10.06
7	2	3.0	2.88	3.99	5.94	6.30	7.93	10.50
7	3	2.0	2.61	3.19	3.71	4.94	7.24	13.41
7	3	2.5	2.55	3.08	3.52	6.03	7.98	13.75
7	3	3.0	3.12	3.87	4.77	6.05	8.87	14.45

Each row summarises the results of 10 different network structures (with 25 XCS runs per structure)

Note that the predicted payoff and fitness are related specifically to XCS and thus are not shown for the rules produced using the non-XCS algorithm.

Figure 4 shows the optimal control diagram for the example RBN of Fig. 1 when intervention cost is 3.0 whilst Fig. 5 shows an XCS controller evolved to control the same network.

Scalability

Questions of scalability are of key importance – we need to know which networks this technique may be reasonably applied to. This section presents brief experiments aimed at uncovering the maximum network size that may be realistically controlled by the optimal rule algorithm, for $K = 2$ networks.

To test which networks may be controlled by the optimal rule algorithm we first applied the algorithm five times to an $N = 5, K = 2$ network and recorded the results (resulting number of rules, and time to compute). We then proceeded to repeatedly increment N by 2 and, at each increment, applied the algorithm five times and recorded the results. This was repeated until constraints were encountered. The procedure was carried out on a computer with a i7-7500U processor and 32GB of RAM.

Table 3 Results for 10 networks with $N=5, K=2$ and cost 2

Network Number	Optimal Cost	Cost-Opt. Rule Count	XCS Avg. Cost	XCS Rule Count	XCS Time
1	3.41	15	5.99	37.52	73.16
2	1.72	13	4.18	22.6	55.68
3	2.00	13	3.69	19.04	47.28
4	3.25	13	5.92	28.24	70
5	1.91	19	5.67	16.48	88.12
6	2.25	13	5.49	30.72	77.36
7	2.16	7	3.86	14.52	49.32
8	2.69	14	4.65	31.16	63.2
9	3.59	13	6.42	35.64	90.04
10	3.13	13	5.29	41.04	118.72

Table 4 Results for 10 networks with N=5,K=2 and cost 2.5

Network Number	Optimal Cost	Cost Opt. Rule Count	XCS Avg. Cost	XCS Rule Count	XCS Time
1	3.41	18	7.35	43.32	72.6
2	1.63	12	4.35	21.8	53.44
3	2.00	5	3.73	17.36	47.16
4	3.25	11	6.82	22.04	70
5	1.91	19	6.20	15.88	85.72
6	2.19	12	6.43	27.72	97.04
7	2.16	7	4.72	18.64	48.64
8	2.69	13	5.52	32.32	63.48
9	3.59	9	7.30	40.32	87.96
10	3.13	10	6.40	35.56	112.6

The results of this experiment may be seen in Fig. 6. Five readings are presented for each of N and K combinations, with the exception of the N=13,K=2 network where only four results are presented (one of the algorithm runs took greater than 40 min and was aborted). The results show that the algorithm easily copes with networks with values of N up to 11 (producing the output within minutes, using a standard spec personal computer). Applying the algorithm to networks with N set to 13 is feasible in the majority of cases but some networks of this scale pose problems. (The specific problem encountered was a shortage of RAM on the machine running the algorithm.) These results suggest that networks where N is less than 13 may be reliably covered by the algorithm. For those larger than this, an alternative technique should be used.

Application to a Boolean network model of a GRN

Here we consider application of the technique to a Boolean network model of a real-world system – the cell-cycle of fission yeast as presented by Wang et al. (Wang et al. 2010). The underlying network is shown in Fig. 7. The rules governing the updates are different within this model than Kauffman NK Boolean networks. Nevertheless, both XCS and the optimal rule calculator are able to control the system. For details of the update rules please refer to the work of Wang et al. (Wang et al. 2010).

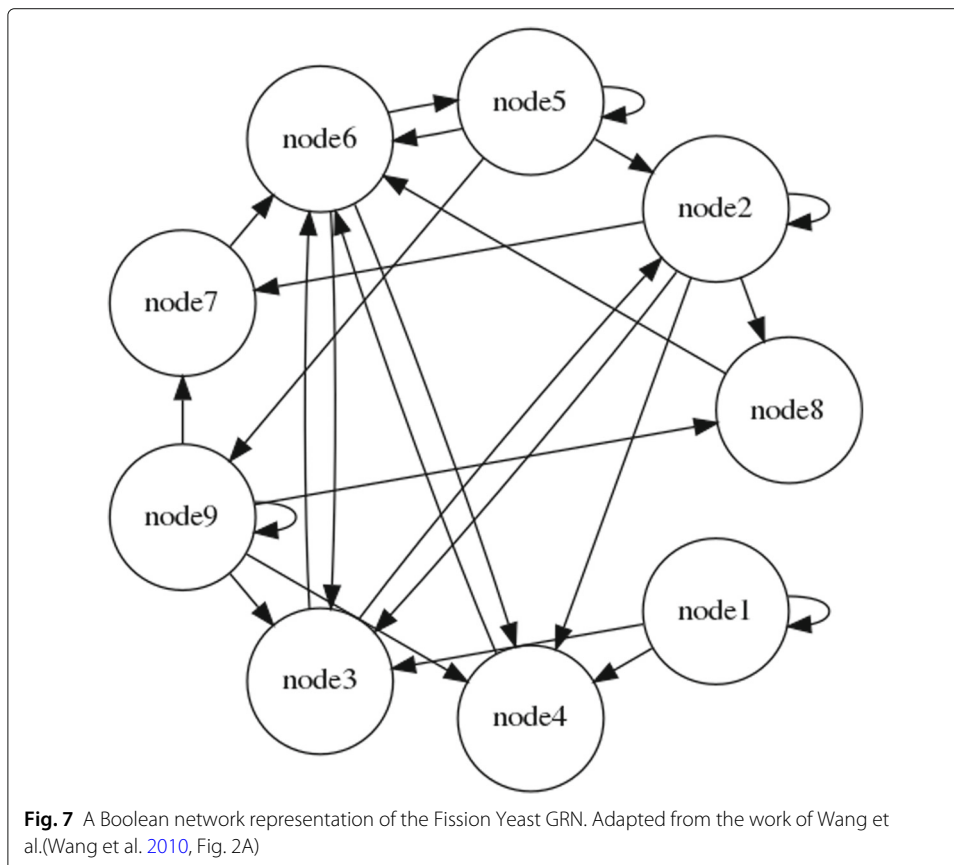
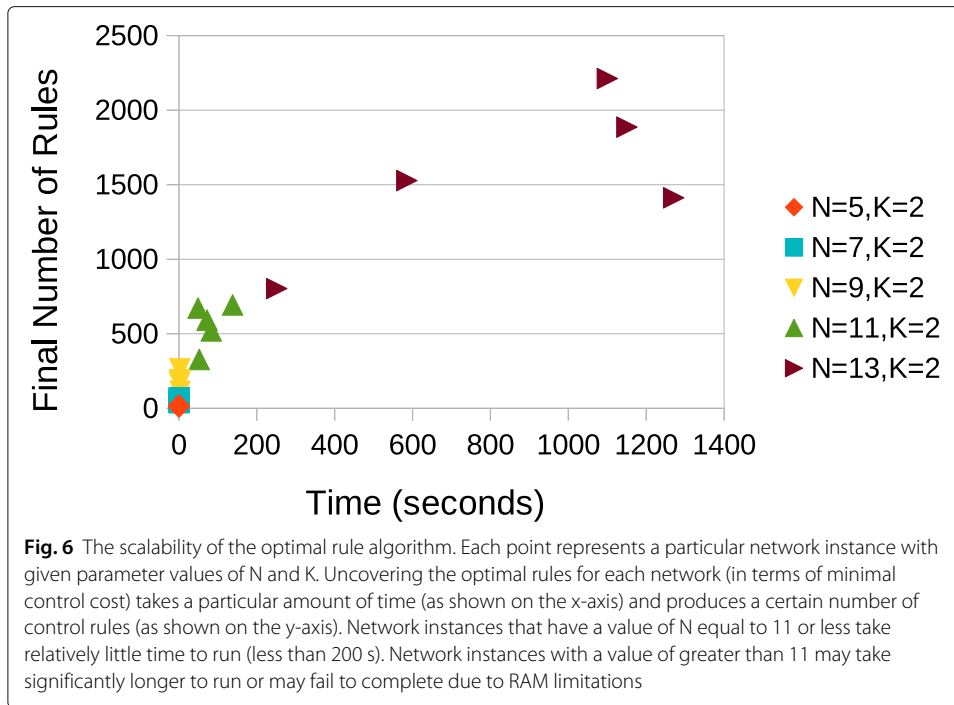
Table 5 Results for 10 networks with N=5,K=2 and cost 3.0

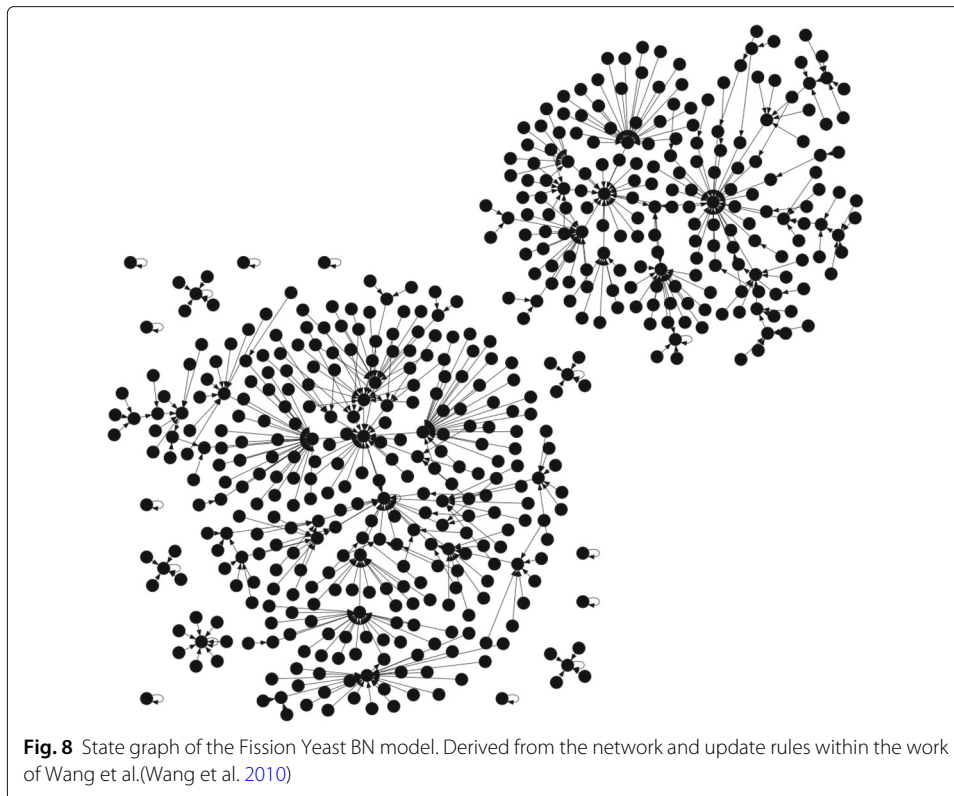
Network Number	Optimal Cost	Cost Opt. Rule Count	XCS Avg. Cost	XCS Rule Count	XCS Time
1	4.56	19	8.09	40.08	71.84
2	1.94	9	4.94	21.28	54.72
3	2.03	5	4.53	19.88	46.92
4	3.81	11	7.67	31.2	69.28
5	2.53	19	6.09	20.4	83.08
6	2.75	14	7.12	30.16	77.92
7	2.66	7	4.78	21.52	50.36
8	2.84	11	5.79	36.36	63.56
9	4.47	13	8.91	35.92	87.08
10	3.75	12	6.36	43.16	115.36

Table 6 Optimal versus two example rule sets for an N=5,K=2 network (IC =2.0)

Optimal Rule Set	XCS Example Rule Set 1	XCS Example Rule Set 2
01000 : 2	#0### : 0 [826.14/0.9978]	0#### : 0 [610.71/1]
00100 : 3	01000 : 2 [1330.26/1]	01000 : 2 [1277.48/1]
01010 : 4	##1#0 : 3 [1090.68/1]	00### : 2 [792.98/0.7836]
11101 : 5	0#000 : 1 [779.48/1]	0#010 : 4 [963.7/1]
11010 : 3	#0### : 2 [792.56/0.7614]	0#00# : 1 [645.64/0.9968]
00#11 : 0	0#000 : 4 [785.99/1]	###11 : 5 [669.3/1]
#1111 : 0	0#0#0 : 5 [677.82/1]	1#10# : 0 [1241.56/1]
100#1 : 0	01100 : 1 [1043.01/1]	0#000 : 4 [652.45/1]
#11#0 : 0	0#000 : 3 [735.1/1]	0#001 : 5 [933.07/1]
101## : 0	10000 : 1 [1111.64/1]	0##00 : 5 [621.72/1]
100#0 : 1	##010 : 3 [688.34/1]	0#01# : 3 [542.8/1]
#1011 : 2	1#000 : 3 [999.7/1]	0#000 : 3 [664.91/1]
1100# : 3	##100 : 4 [757.19/1]	1#000 : 3 [865.67/1]
00#10 : 4	0#010 : 4 [1329.22/0.6272]	10##0 : 1 [1096.53/0.9996]
0##01 : 5	#1#10 : 2 [811.6/1]	11##0 : 1 [732.65/0.9979]
	0#101 : 5 [747.06/1]	##1#0 : 3 [802.9/0.315]
	011#0 : 0 [751.29/1]	##010 : 5 [499.36/0.9975]
	11100 : 0 [1386.61/1]	###01 : 3 [577.9/1]
	##110 : 4 [788.72/1]	####1 : 1 [612.58/0.9038]
	010#0 : 0 [792.48/1]	1#0## : 0 [648.04/1]
	1#0## : 4 [551.88/0.8454]	####1 : 4 [538.15/0.4199]
	####1 : 3 [537.19/1]	1#110 : 0 [948.36/1]
	####1 : 1 [535.8/0.8495]	0#1## : 4 [545.14/0.9135]
	1##10 : 1 [758.33/1]	00010 : 1 [690.26/1]
	01##1 : 0 [534.25/1]	0#110 : 3 [724.95/0.9887]
	0##1# : 1 [552.97/0.7678]	###10 : 2 [685.01/0.978]
	0#100 : 5 [570.96/1]	0101# : 1 [547.31/0.9862]
	0#001 : 5 [996.12/0.1555]	1#000 : 2 [639.82/1]
	##11# : 5 [673.12/0.9958]	1#01# : 3 [656.55/1]
	0#011 : 5 [740.69/1]	01100 : 3 [987.87/0.9938]
	11#0# : 2 [722.05/0.8468]	0#101 : 5 [658.58/1]
	1#100 : 5 [618.54/1]	11111 : 0 [694.67/1]
	0#100 : 2 [753.52/0.9925]	1##10 : 4 [676.44/1]
	01010 : 4 [1032.16/0.9781]	####1 : 2 [567.87/0.743]
	1#100 : 1 [751.96/0.9846]	##1## : 2 [605.81/0.4683]
	#1#01 : 4 [510.13/0.8903]	1##01 : 5 [677.38/1]
	0#011 : 4 [746.69/0.9999]	0#110 : 5 [515.22/1]
	0010# : 1 [705.39/0.9299]	0#100 : 1 [787.68/1]
	110## : 0 [598.88/1]	##111 : 3 [536.47/1]
	01001 : 5 [984.98/0.9288]	1#00# : 4 [539.28/0.5748]
	1#1#1 : 5 [774.49/0.3823]	1#10# : 4 [646.66/0.8809]
	1#001 : 5 [719.66/1]	##110 : 1 [656.67/0.9729]
	##1#1 : 4 [561.8/0.8403]	10111 : 0 [942.59/1]
	111#1 : 0 [736.48/1]	

The Boolean network shown in Fig. 7, combined with the update rules described within Wang et al. (2010), results in the state space as shown in Fig. 8. As can be seen, there are two major basins of attraction and several much smaller basins.





complexity of the control problem varies is in the number of attractors. The networks with a higher number of attractors tend to have a higher control cost.

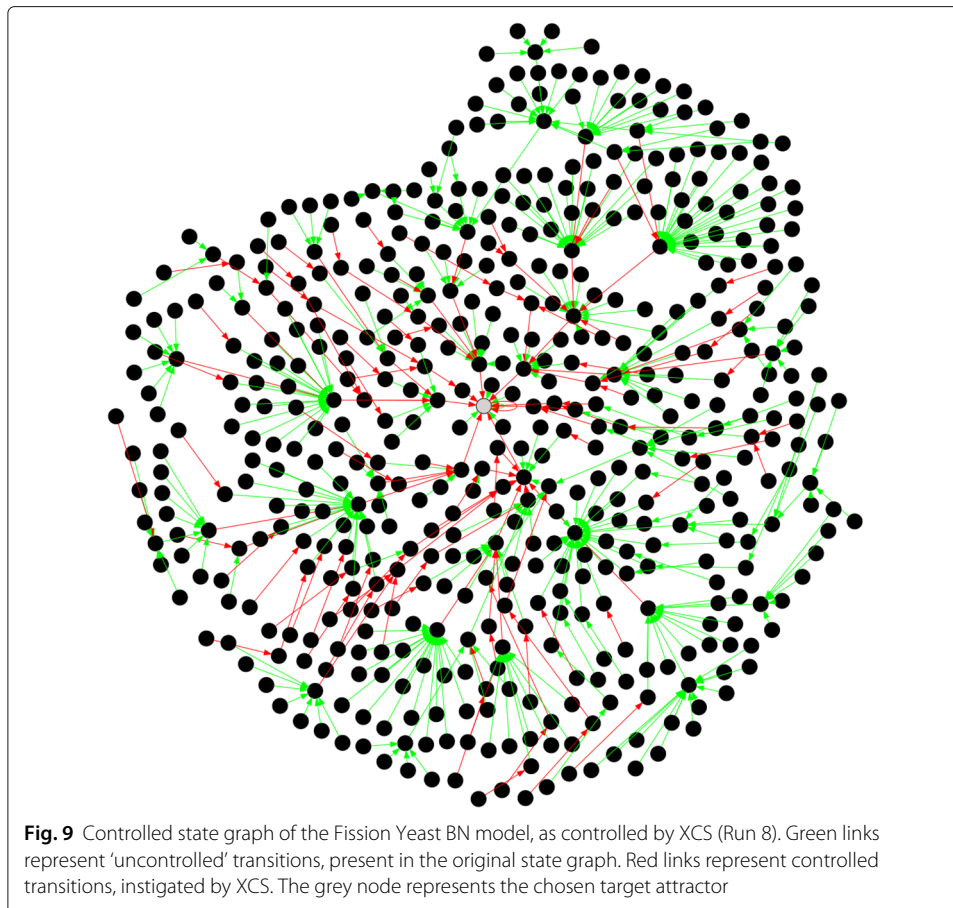
Table 6 shows us the optimal rule set and two of the rule sets that were actually evolved for the network. We can see that the entire XCS procedure has produced rule sets that are about 2.5 times the size of the optimal rule set. We can also see that the XCS rule sets do not contain the optimal rule set as a subset of their rules. In terms of exact matches, the rule 01000 : 2 is found in all three sets whilst 01010 : 4 is found in one of the XCS rule sets. That said, whilst the number of exact matches is few, we can see a number of rules in the two XCS rule set examples are close to the rules in the optimal rule set (for instance there may be one too many # symbols – the classifiers may be too general).

The rule sets produced can be studied in the vein of the research conducted within Marques-Pita and Rocha (2013). It may be that there is a relation between canalisation and the active intervention rules ‘introduced’ in to the network.

Overall comparative results are presented in Figs. 12 and 13. The x-axis displays the average cost of the XCS intervention strategies for each network as a percentage of the optimal cost, whilst the y-axis shows the average rule set size for the XCS intervention strategies as a percentage of the number of rules required to achieve the optimal strategy.

In Fig. 12 we can see that for the intervention cost of 2.0 the XCS solutions range between approximately 150% and 300% of optimal cost (excluding 3 outliers) whilst rule set size is between approximately 120% and 510% (excluding one outlier).

With cost 3.0 performance is more varied across the various network configurations, as shown in Fig. 13. The cost of the XCS rule sets falls between approximately 150% and

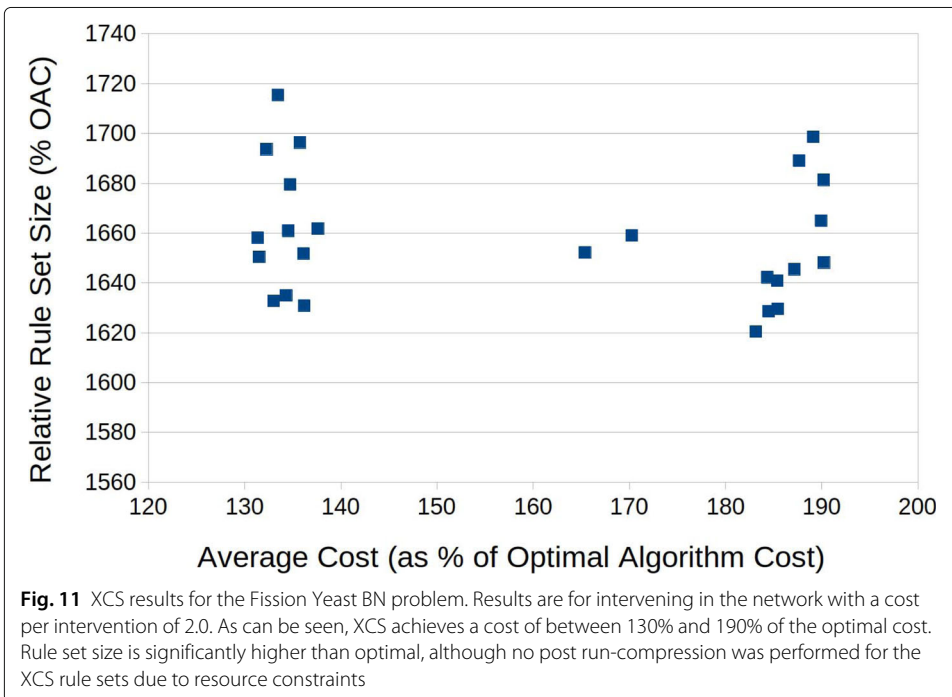
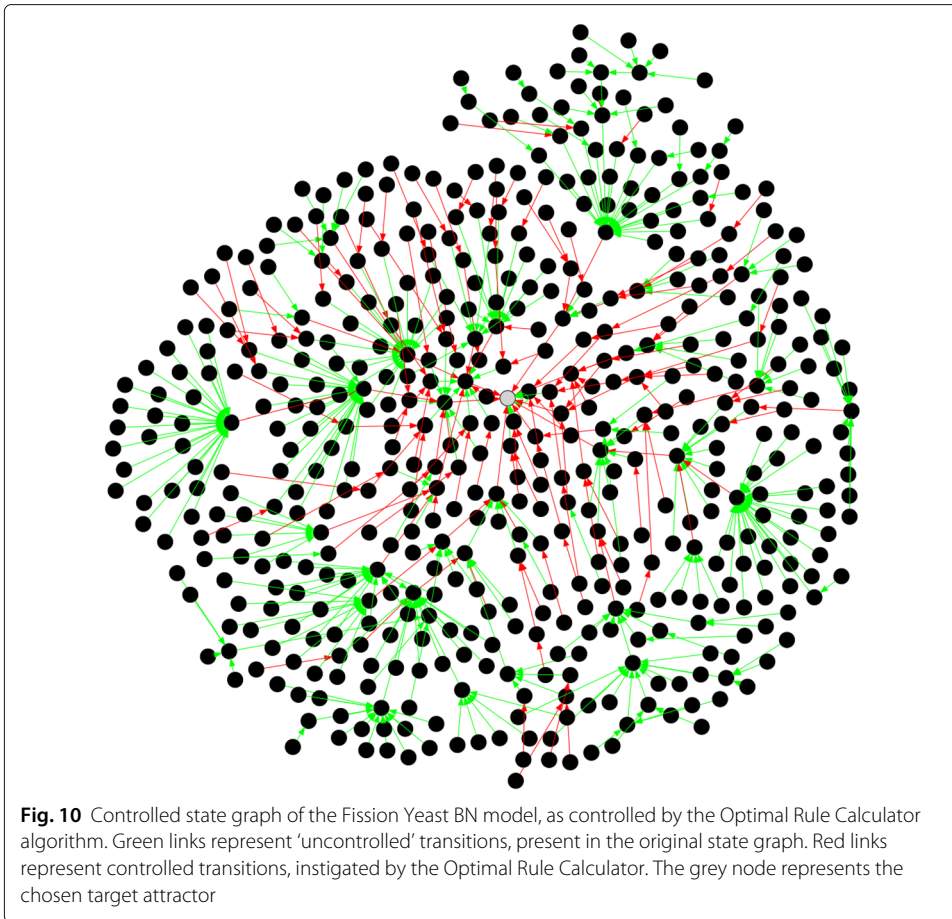


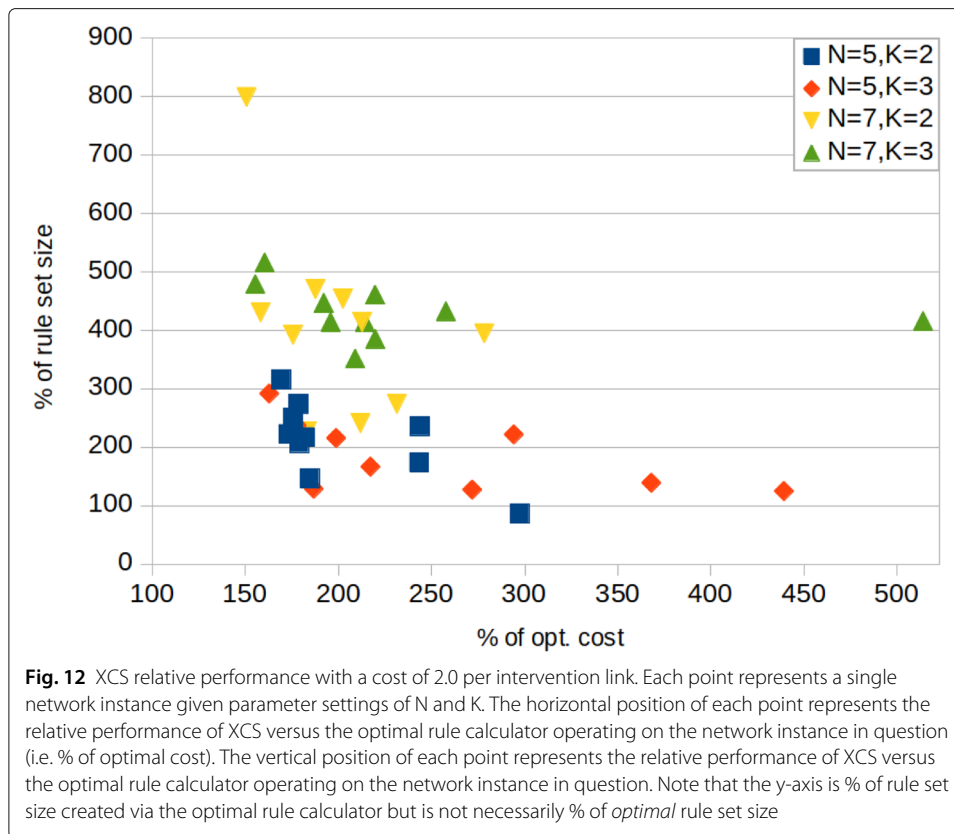
300% of optimal (excluding two outliers) whilst the rule set size falls between 100% and 600% of optimal (excluding two outliers).

It should be noted that the different costs may bring about substantially different control graphs – when intervention cost is 3.0 the number of intervention links is notably lower than when intervention cost is 2.0.

In “[Scalability](#)” section we considered the extent to which the ORC approach can scale to larger networks, finding that the technique is applicable to networks where N is less than 13. This limitation may be overcome through the addition of more than 32GB of RAM. For ORC to progress we require some technique that will avoid the necessity of storing the full state space of the RBN in memory. In contrast, XCS is limited by the ‘long action chain problem’ (Barry 2002) and has difficulty with networks that ORC is able to handle. XCS is therefore currently unable to scale to networks of hundreds of nodes that other recent heuristic techniques are able to handle. On the XCS side, overcoming the long action chain problem is a key area for future work.

As indicated in the introduction we emphasise that the main advantage of both approaches herein is the human-readable nature of the rules produced. When enhanced by the inclusion of the third ‘don’t care’ symbol (i.e. #) the rule set produced can be substantially compressed whilst maintaining human readability. This structure is also compatible with canalisation studies, as used in the work by Marques-Pita and Rocha (Marques-Pita and Rocha 2013).



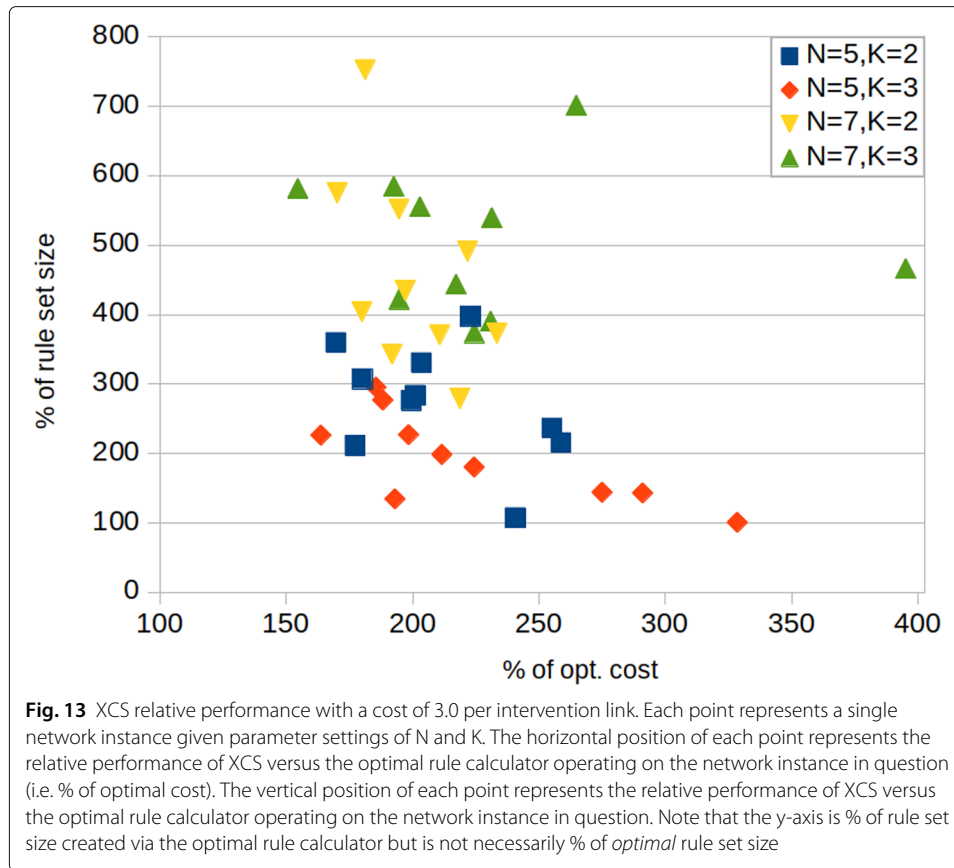


Conclusions and Future Work

We presented an algorithm for uncovering the optimal set of control rules for random Boolean networks. We compared the performance of this optimal rule calculator algorithm and the XCS variant of learning classifier systems when applied to these networks. We find that, whilst XCS presents a viable means of evolving control rules for Boolean networks, the currently produced rules are not optimal in terms of cost or rule set size. Conversely, the optimal rule calculator algorithm produces very strong performance in controlling relatively small Boolean networks. Further work is required to narrow the gap between the XCS solution and the optimal solution.

There a number of directions for future investigation.

Firstly, the study herein suggests that optimal rulesets can be developed for networks of small to medium scale. The approach has been successfully applied to real world networks for policy-making (Moschoyiannis et al. 2016), such as the transition of the South Humber Bank regional economy from fossil fuel-based to bioenergy-based products which involved a network of 8 nodes. It may not be immediately applicable to large-scale networks such as long-running transactions in digital ecosystem architectures (Razavi et al. 2009) or intelligent routing of Electric Vehicles (EVs) in city road networks (Kosmanos et al. 2018), but concentrated effort and/or additional computational resources shall explore this further. Preliminary analysis shows that the ‘compressed’ nature of the final rule set suggests that a more general technique applicable to larger networks is possible.



Secondly, XCS could be modified to increase performance, via parameter adjustments or structural modification of XCS itself (incrementally or via replacement of XCS with another classifier system variant).

Thirdly, improvement of the condensation or compression algorithm could also be possible – variants exist (see Kovacs (1998)).

There is also the question of whether gaining a precise understanding of the canalisation already present in the original network can assist us in developing more compact rule sets (Marques-Pita and Rocha 2013). Canalisation occurs when a single bit input in to a node governs the new bit set at that node (i.e. the remainder of the rule’s indexes are wildcard # symbols) (Marques-Pita and Rocha 2013). Canalisation can exist by degree, based on the number of wildcards present, less non-wildcard index that governs the next state (Marques-Pita and Rocha 2013).

We also note that the larger Fission Yeast network starts to bring up a challenging problem for XCS. The average distance between any node and the target attractor increases with increment of the number of nodes in the network. Thus, more interventions are needed to bring the network from any possible attractor to a desired one. This means that XCS must learn a bigger state-action-payoff mapping. Barry (Barry 2002; 2003) studied the problem of long action-chains and hypothesised that there will be a point in time when the payoff of both correct and incorrect action sets will have insufficient differences, which would be enough to prevent XCS from developing correct classifiers. This is possible due to the discount factor γ when the predictions for payoff in the early stage

of a run are likely to be very close for all sets of actions taken so far and that similarity is being further enhanced by the discounts. Thus, with a sufficiently long action-chain we can observe the phenomenon since there will be enough time for XCS to develop incorrect rules which overgeneralise (Kovacs 2001) and have sufficient time to be deemed as “accurate” if there is no convergence of correct classifiers earlier. Furthermore, due to their over-generalisation and inflated fitness, those classifiers would have a high number of opportunities to take part in the GA and “flood” the population, which leads to them controlling the predictions in the next stages. XCS therefore has its scalability limitation when controlling larger BNs, which is an important challenge to address in future.

Finally, the different network structures could be investigated further. This could be in the form of larger NK Boolean networks or networks with higher values of K , or network structures produced with different generators entirely, such as Erdős–Rényi model-based graphs or the Barabási–Albert model. The techniques considered herein may also be applicable to controlling small system dynamics models in the vein of Schoenenberger and Tanase (Schoenenberger and Tanase 2018). We hope, in future, to develop variants that will scale to networks of even greater size.

Abbreviations

BN: Boolean Network; CTSG: Controlled State Transition Graph; EV: Electric Vehicle; FVS: Feedback Vertex Set; GA: Genetic Algorithm; GB: Gigabytes; GBML: Genetics-Based Machine Learning; GRN: Gene Regulatory Network; IC: Intervention Cost; LCS: Learning Classifier System; OC: Optimal Cost; Opt; Avg: The average optimal control cost; Opt; Cost Optimal Cost; Opt; Max: The maximum optimal control cost; Opt; Min: The minimum optimal control cost; ORC: Optimal Rule Calculator; RAM: Random Access Memory; RBML: Rule-Based Machine Learning; RBN: Random Boolean Network; XCS: eXtended Classifier System

Acknowledgements

The graph figures within the paper are generated using GraphViz (Gansner and North 2000). We thank four anonymous reviewers for insightful comments and suggestions. We are particularly indebted to these reviewers for drawing our attention to the related canalisation work (Marques-Pita and Rocha 2013) and suggesting that we include an example of the control of a BN derived from a real-world system.

This research was partly funded by the Department for Transport, via Innovate UK under the *Onward Journey Planning Assistant (OJPA)* project, and partly by the EPSRC IAA project AGELink (EP/R511791/1).

Authors' contributions

MRK coded the RBN implementation, the LCS implementation and the optimal control algorithm. MRK also wrote the preliminary version of the paper. SKM contributed substantial guidance during the research and provided many refinements to the paper. VBG contributed to “Control of Boolean Networks” section and wrote the concluding paragraph on long action chains. MRK constructed the initial versions of “Scalability” section and “Application to a Boolean network model of a GRN” section, that were then subsequently refined by SKM. All authors read and approved the final manuscript.

Funding

This research was partly funded by the Department for Transport, via Innovate UK under the *Onward Journey Planning Assistant (OJPA)* project, partly funded by EIT Digital IVZW under the *Real-Time Flow* project (activity 18387–SGA2018), and partly funded by EPSRC IAA under the AGELink project (EP/R511791/1).

Availability of data and materials

The configurations used, RBN structures generated, output control rules, and output graph diagrams are available on reasonable request.

Competing interests

The authors declare that they have no competing interests.

Author details

¹Department of Computer Science, Faculty of Engineering and Physical Sciences, University of Surrey, GU2 7XH Guildford, Surrey, UK. ²Thought Machine Albert House, Old Street, London EC1V 9DD, UK.

Received: 20 March 2019 Accepted: 10 December 2019

Published online: 30 December 2019

References

- Akutsu T, Hayashida M, Ching WK, Ng MK (2007) Control of Boolean networks: Hardness results and algorithms for tree structured networks. *J Theor Biol* 244(4):670–679
- Barry A (2003) Limits in long path learning with xcs. In: Genetic and Evolutionary Computation Conference. Springer, Berlin. pp 1832–1843

- Urbanowicz RJ, Moore JH (2009) Learning classifier systems: a complete introduction, review, and roadmap. *J Artif Evol Appl* 2009(1):1–25
- Wang G, Du C, Chen H, Simha R, Rong Y, Xiao Y, Zeng C (2010) Process-based network decomposition reveals backbone motif structure. *Proc Natl Acad Sci* 107(23):10,478–10,483
- Wilson SW (1995) Classifier fitness based on accuracy. *Evol Comput* 3(2):149–175
- Wilson SW (1998) Generalization in the XCS classifier system. In: Koza J, Banzhaf W, Chellapilla K, Deb K, Dorigo M, Fogel D, Garzon M, Goldberg D, Iba H, Riolo R (eds). *Genetic Programming 1998: Proceedings of the Third Annual Conference*. Morgan Kaufmann, San Francisco
- Wilson SW (2001) Compact rulesets from XCSI. In: *International Workshop on Learning Classifier Systems*. Springer, Berlin. pp 197–208
- Zañudo JGT, Yang G, Albert R (2017) Structure-based control of complex networks with nonlinear dynamics. *Proc Natl Acad Sci* 114(28):7234–7239. <https://doi.org/10.1073/pnas.1617387114>

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Submit your manuscript to a SpringerOpen[®] journal and benefit from:

- ▶ Convenient online submission
- ▶ Rigorous peer review
- ▶ Open access: articles freely available online
- ▶ High visibility within the field
- ▶ Retaining the copyright to your article

Submit your next manuscript at ▶ [springeropen.com](https://www.springeropen.com)
