Optimal control rules for random Boolean networks*

Matthew R. Karlsen and Sotiris K. Moschoyiannis

Department of Computer Science,
Faculty of Engineering and Physical Sciences,
University of Surrey, Guildford, Surrey, GU2 7XH, UK
{matthew.r.karlsen | s.moschoyiannis}@surrey.ac.uk

Abstract. A random Boolean network (RBN) may be controlled through the use of a learning classifier system (LCS) – an eXtended Classifier System (XCS) can evolve a rule set that directs an RBN from any state to a target state. However, the rules evolved may not be optimal, in terms of minimising the total cost of the paths used to direct the network from any state to a specified attractor. Here we uncover the optimal set of control rules via an exhaustive algorithm. The performance of an LCS (XCS) on the RBN control problem is assessed in light of the newly uncovered optimal rule set.

1 Introduction

Controlling complex networks is key in areas as diverse as biological systems and socio-technical systems such as transport networks. 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 [3]. Recent advances include work on network structure *control nodes* [15, 1, 16], its reconfiguration [8, 17, 18], but also on the network dynamics [3, 9] with application for example to transport [10]. *'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 states, where the system continues to perform its functions). Herein the network is said to be '*controlled*' when the rule set takes 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 [9], we have applied rule-based machine learning in the form of an "eXtended Classifier System" (XCS) [20, 21] to the problem of controlling random Boolean networks (RBNs) [12, 11]. RBNs have been used to model biological networks such as gene regulatory networks, e.g., see [13], [7]. We showed that XCS can evolve a rule set that takes an RBN from any state to an attractor. However, no indication of the 'ideal' or 'optimal' set of rules was supplied.

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 (contrasted to a weight of '1' for

---

[*] This research was partly funded by the Department for Transport, via Innovate UK and the AIR round 4 programme, under the *Onward Journey Planning Assistant (OJPA)* project and partly funded by EIT Digital under the *Real–Time Flow* project, activity 18387–SGA2018.

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 [5] (though they do not work with XCS).

The remainder of this paper is structured as follows. Section 2 briefly describes RBNs whilst Section 3 outlines key principles behind LCSs. In Section 4 we describe the central algorithm of the paper, which computes the optimal set of control rules for an RBN. Section 5 explains the experiments conducted to compare this algorithm with an XCS-based approach. Results are presented in Section 6, and discussed in Section 7. Conclusions and future work follow in Section 8.

2 Random Boolean Networks

Random Boolean Networks [12, 11] 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 [12, 11] 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 (enclosed within the box).

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.

Random Boolean networks can be controlled via timed bit flips at available targeted nodes (i.e. if the current Boolean value at the node is a 0 it can be flipped to a 1 or vice versa). This can be represented as an integer (starting at 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, see Fig. 1. 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.

Substantial previous work on the control of RBNs is covered in a review by Fornasini and Valcher [6]. This work differs in relation to previous works via (1) taking a programmatic approach rather than an algebraic approach and (2) providing the control instructions as a condensed set of ‘control rules’.

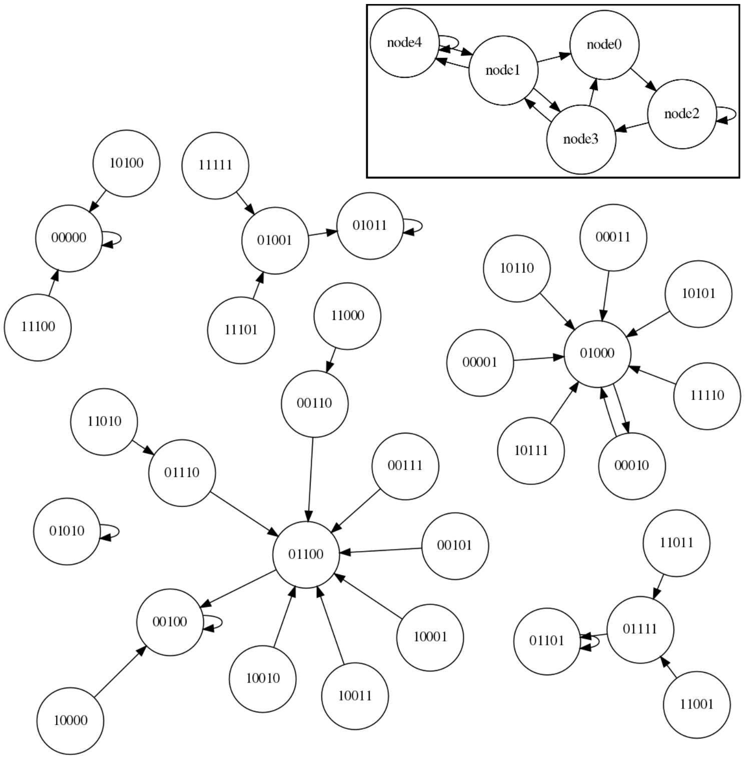


Fig. 1. An RBN with N=5,K=2 (enclosed in box). The resulting state space is also shown.

3 Learning Classifier Systems

A learning classifier system (LCS) [19] is a rule-based machine learning 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 propose 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.

The overall system for the XCS [20, 21] variant can be found in [9] (see Fig. 5 in [9]). The implementation used is that described in the algorithmic specification provided by Butz & Wilson [2], with a post-run rule set compression based on [22].

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
   (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

4 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
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 shortest path for the origin node
   (c) calculate total cost for the path (link weight sum) and store
   (d) calculate number of interventions (count of link with weight > 1) 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 [4]. 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 (see [9]). 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.

5 Experiments

Here we use the aforementioned optimal rule discovery algorithm to create the optimal rule sets for 10 different networks. We then run XCS on each of these networks 25 times and produce aggregate performance results (necessary due to XCSI's 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 [9]. 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 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 [9]). 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 [2]) 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 [14] 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 [9]. 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.

6 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

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
$doAsSubsumpt.$	true	Perform subsumption in the action set?
$doGaSubsumpt.$	true	Perform subsumption in the GA?

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 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 ‘/’. Note that the predicted payoff and fitness

Table 2. Control costs – Optimal vs XCS

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

Table 3. Results for IC = 2.0 (N=5,K=2)

Nw Num	Opt. Cost	OC Rule Count	XCS Avg Cost	XCS Rule Count	XCS 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 IC=2.5 (N=5,K=2)

Nw Num	Opt. Cost	OC 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

Table 5. Results for IC=3.0 (N=5,K=2)

Nw Num	Opt. Cost	OC 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

are related specifically to XCS and thus are not shown for the rules produced using the non-XCS algorithm.

7 Discussion

As can be seen from Table 2 the average cost for the optimal set of rules ranges from 2.61 for an N=5,K=2 network with an intervention cost of 2 to 3.99 for an N=7,K=2 network with an intervention cost of 3. In contrast, the XCS average cost is 5.11 for an N=5,K=2,IC=2.0 network, with a cost of 8.87 for an N=7,K=3,IC=3.0 network. The min. optimums can be substantially below the average optimums (i.e. one or more particular network structure of the 10 randomly generated ones are much easier to control than average).

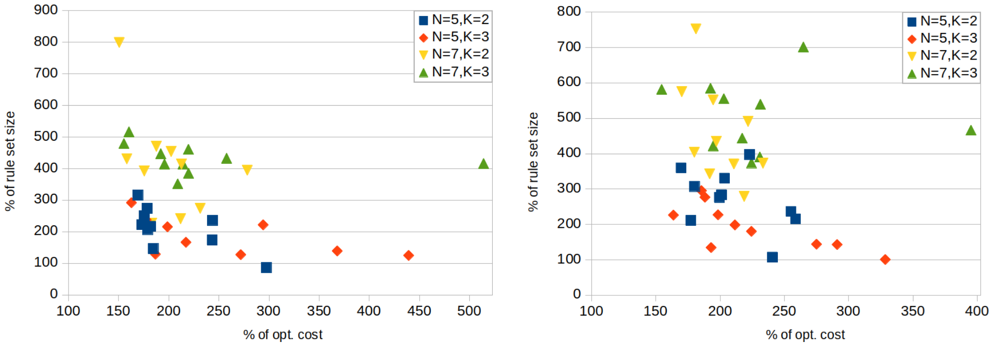
Table 6. Optimal versus two example rule sets for an N=5,K=2 network (IC =2.0)

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 above observation can be substantiated with Tables 3, 4, and 5 – substantial diversity of results can be seen across these tables. One of the major ways in which the 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).

Overall comparative results are presented in Figures 2a and 2b. 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.



(a) with intervention cost 2.0

(b) with intervention cost 3.0

Fig. 2. XCS relative performance

In Figure 2a 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 Figure 2b. 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).

Finally, 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. Due to space restrictions we do not include the control graphs here as in [9].

8 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 technique is currently only applicable to small networks. However, the ‘compressed’ nature of the final rule set suggests that we may be able to develop a more general technique applicable to larger networks.

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 [14]).

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.

References

1. Bianconi, G., Pin, P., Marsili, M.: Assessing the relevance of node features for network structure. Proceedings of the National Academy of Sciences **106**(28), 11,433–11,438 (2009). DOI 10.1073/pnas.0811511106. URL http://www.pnas.org/content/106/28/11433
2. Butz, M.V., Wilson, S.W.: An algorithmic description of XCS. In: International Workshop on Learning Classifier Systems, pp. 253–272. Springer (2000)
3. Cornelius, S.P., Kath, W.L., Motter, A.E.: Realistic control of network dynamics. Nature Communications **4**, 1942 (2013)
4. Dutot, A., Guinand, F., Olivier, D., Pigné, Y.: Graphstream: A tool for bridging the gap between complex systems and dynamic graphs. In: Emergent Properties in Natural and Artificial Complex Systems. 4th European Conference on Complex Systems (ECCS’2007) (2007)
5. Fornasini, E., Valcher, M.E.: Optimal control of boolean control networks. IEEE Transactions on Automatic Control **59**(5), 1258–1270 (2014)
6. Fornasini, E., Valcher, M.E.: Recent developments in boolean networks control. Journal of Control and Decision **3**(1), 1–18 (2016)
7. Gates, A.J., Rocha, L.M.: Control of complex networks requires both structure and dynamics. Scientific Reports **6**, 24,456 (2016)
8. Haghighi, R., Namazi, H.: Algorithm for identifying minimum driver nodes based on structural controllability. Mathematical Problems in Engineering **2015** (2015)

9. Karlsen, M.R., Moschoyiannis, S.: Evolution of control with learning classifier systems. Applied Network Science **3**(1), 30 (2018). DOI 10.1007/s41109-018-0088-x. URL https://doi.org/10.1007/s41109-018-0088-x
10. Karlsen, M.R., Moschoyiannis, S.: Learning condition–action rules for personalised journey recommendations. In: RuleML + RR, no. 11092 in LNCS, pp. 293–301 (2018)
11. Kauffman, S.: The Origins of Order. Oxford University Press, New York, NY (1993)
12. Kauffman, S.A.: Metabolic stability and epigenesis in randomly constructed genetic nets. Journal of Theoretical Biology **22**(3), 437–467 (1969)
13. Kim, J., Park, S.M., Cho, K.H.: Discovery of a kernel for controlling biomolecular regulatory networks. Scientific Reports **3**, 2223 (2013). DOI 10.1038/srep02223. URL https://www.nature.com/articles/srep02223
14. Kovacs, T.: XCS classifier system reliably evolves accurate, complete, and minimal representations for boolean functions. In: Soft computing in engineering design and manufacturing, pp. 59–68. Springer (1998)
15. Liu, Y.Y., Slotine, J.J., Barabási, A.L.: Controllability of complex networks. Nature **473**(7346), 167 (2011)
16. Moschoyiannis, S., Elia, N., Penn, A., Lloyd, D.J.B., Knight, C.: A web-based tool for identifying strategic intervention points in complex systems. In: Proc. Games for the Synthesis of Complex Systems (CASSTING'16 @ ETAPS 2016), *EPTCS*, vol. 220, pp. 39–52 (2016)
17. Savvopoulos, S., Moschoyiannis, S.: Impact of removing nodes on the controllability of complex networks. In: 6th Conf. on Complex Networks and Applications, pp. 361–363 (2017)
18. Savvopoulos, S., Penn, A., Moschoyiannis, S.: On the interplay between topology and controllability of complex networks. In: Conf. on Complex Systems (CCS'17) (2017)
19. Urbanowicz, R.J., Moore, J.H.: Learning classifier systems: a complete introduction, review, and roadmap. Journal of Artificial Evolution and Applications **2009**(1), 1–25 (2009)
20. Wilson, S.W.: Classifier fitness based on accuracy. Evolutionary Computation **3**(2), 149–175 (1995)
21. Wilson, S.W.: Generalization in the XCS classifier system. In: J. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. Fogel, M. Garzon, D. Goldberg, H. Iba, R. Riolo (eds.) Genetic Programming 1998: Proceedings of the Third Annual Conference. Morgan Kaufmann, San Francisco, CA (1998)
22. Wilson, S.W.: Compact rulesets from XCSI. In: International Workshop on Learning Classifier Systems, pp. 197–208. Springer (2001)