Condition–action rules in controlling complex systems

Sotiris Moschoyiannis
on joint work with Matthew R. Karlsen and Vlad Georgiev

University of Surrey, England

RuleML Webinar @ Skype

29th March 2019

Outline

Introduction

OJPA Project

Onward Journey Planning Assistant

Personalised recommendations for:

- customers planning a journey
- customers experiencing disruption

Overall challenge

- Input:
  - environment factors:
    - train, taxi, tube, boat and bus [**0 or 5**]; weather [**0 to 5**]
  - journey-specific factors:
    - current delay, delay on current mode(s), onward delay [**0 or 5**]
  - passenger preferences:
    - value, speed, comfort, shelter [**0 to 5**]
- Output:
  - "Correct" single integer recommendation:
    - no change (**0**), single taxi (**1**), shared taxi (**2**), bus (**3**), boat (**4**), tube (**5**), train (**6**)

Population of Rules (the Knowledge)

Condition	:	Action
[0,0] [0,0] [0,0] [5,5] [5,5] [#,#] [#,#] [5,5] [0,1] [#,#] [2,5] [#,#] [#,#]	:	1
[0,0] [0,0] [0,0] [5,5] [5,5] [#,#] [#,#] [5,5] [0,1] [2,5] [#,#] [#,#] [#,#]	:	1
[0,0] [0,0] [0,0] [5,5] [5,5] [#,#] [#,#] [5,5] [2,3] [#,#] [4,5] [#,#] [#,#]	:	1
[0,0] [0,0] [0,0] [5,5] [5,5] [#,#] [#,#] [5,5] [2,3] [4,5] [#,#] [#,#] [#,#]	:	1
[0,0] [0,0] [0,0] [5,5] [5,5] [#,#] [#,#] [5,5] [0,1] [0,1] [2,3] [#,#] [#,#]	:	2
[0,0] [0,0] [0,0] [5,5] [5,5] [#,#] [#,#] [5,5] [0,1] [2,3] [0,1] [#,#] [#,#]	:	2
[0,0] [0,0] [0,0] [5,5] [5,5] [#,#] [#,#] [5,5] [0,1] [2,3] [2,3] [#,#] [#,#]	:	2
[0,0] [0,0] [0,0] [5,5] [5,5] [#,#] [#,#] [5,5] [2,3] [0,1] [2,3] [#,#] [#,#]	:	2
[0,0] [0,0] [0,0] [5,5] [5,5] [#,#] [#,#] [5,5] [2,3] [2,3] [2,3] [#,#] [#,#]	:	2
[0,0] [0,0] [0,0] [5,5] [5,5] [#,#] [#,#] [5,5] [4,5] [0,1] [2,3] [#,#] [#,#]	:	2
[0,0] [0,0] [0,0] [5,5] [5,5] [#,#] [#,#] [5,5] [4,5] [0,1] [4,5] [#,#] [#,#]	:	2
[0,0] [0,0] [0,0] [5,5] [5,5] [#,#] [#,#] [5,5] [4,5] [2,3] [2,3] [#,#] [#,#]	:	2
[0,0] [0,0] [0,0] [5,5] [5,5] [#,#] [#,#] [5,5] [4,5] [2,3] [4,5] [#,#] [#,#]	:	2
[0,0] [0,0] [0,0] [5,5] [5,5] [#,#] [#,#] [5,5] [4,5] [4,5] [0,1] [#,#] [#,#]	:	2
[0,0] [0,0] [0,0] [5,5] [5,5] [#,#] [#,#] [5,5] [4,5] [4,5] [2,3] [#,#] [#,#]	:	2
[0,0] [0,0] [0,0] [5,5] [5,5] [#,#] [#,#] [5,5] [4,5] [4,5] [4,5] [#,#] [#,#]	:	2
[0,0] [0,0] [0,0] [5,5] [5,5] [#,#] [#,#] [5,5] [0,1] [0,1] [0,1] [#,#] [#,#]	:	3
[0,0] [0,0] [0,0] [5,5] [5,5] [#,#] [#,#] [5,5] [2,3] [0,1] [0,1] [#,#] [#,#]	:	3
[0,0] [0,0] [0,0] [5,5] [5,5] [#,#] [#,#] [5,5] [2,3] [2,3] [0,1] [#,#] [#,#]	:	3
[0,0] [0,0] [0,0] [5,5] [5,5] [#,#] [#,#] [5,5] [4,5] [0,1] [0,1] [#,#] [#,#]	:	3
...	:	...

Rule Matching

Figure 1: Simple rule matching example

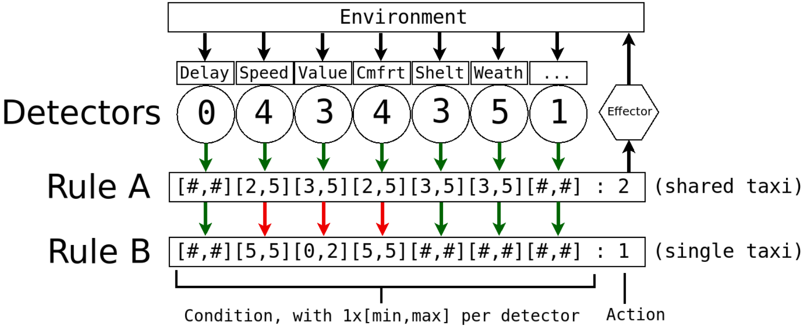
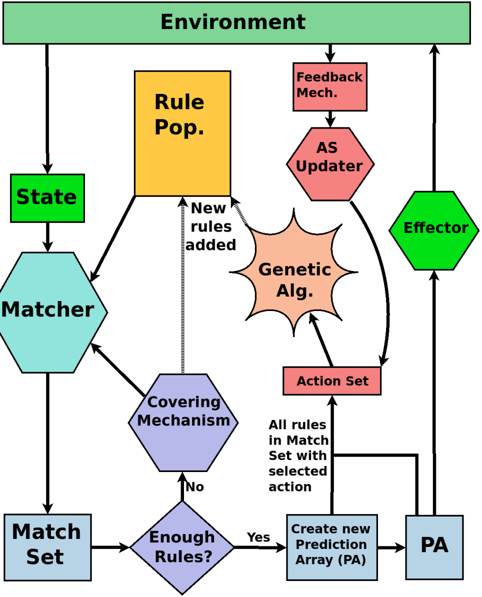


Figure 2: XCSI overview



Effector and Feedback Mechanism

- Effector outputs:
  - no change (**0**), single taxi (**1**), shared taxi (**2**), bus (**3**), boat (**4**), tube (**5**), train (**6**)
- Feedback Mechanism, receives:
  - 1000 for a correct suggestion
  - 0 for an incorrect suggestion
- This feedback is used to update the action set rules

XCS details in: Butz, Martin V., and Stewart W. Wilson. "An algorithmic description of XCS." International Workshop on Learning Classifier Systems. Springer, Berlin, Heidelberg, 2000.

Experiments

- Simulation based on London tube network
- 300 artificial ‘passengers’ with randomised:
  - preferences [0 to 5]
  - origin location / station
  - destination location / station
- Each passenger takes multiple journeys

Simulation details (1)

- Random starting time step (0 to 99) for each passenger
- Each time step has weather [0 to 5; random]
- Train, boat, bus and taxi [0 or 5; random]
- Passenger is at one node each time step
- In each time step 5% of links are out-of-action
- Interleaved... (see next 3 slides)

Simulation details (2)

Figure 3: Input list production, step 1 – order

Time Step	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15 …
Journey 1	S	S	S	S	S	S										
Journey 2		S	S	S	S	S	S	S	S							
Journey 3					S	S	S	S	S	S	S	S	S	S	S	S
Journey 4	S	S	S	S	S	S	S	S								
Journey 5			S	S	S	S	S									
Journey 6							S	S	S	S	S	S	S	S	S	
Journey 7	S	S	S	S	S											

S = ‘journey state’

Simulation details (3)

Figure 4: Input list production, step 2 – shuffle

Time Step	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15 …
Journey 3					S	S	S	S	S	S	S	S	S	S	S	S
Journey 1	S	S	S	S	S	S										
Journey 7	S	S	S	S	S											
Journey 6							S	S	S	S	S	S	S	S	S	
Journey 4	S	S	S	S	S	S	S	S								
Journey 2		S	S	S	S	S	S	S	S							
Journey 5			S	S	S	S	S									

S = ‘journey state’

Simulation details (4)

Figure 5: Input list production, step 3 – obtain input vectors

Time Step	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**…**
Journey 3					**S**	**S**	**S**	**S**	**S**	**S**	**S**	**S**	**S**	**S**	**S**	**S**	
Journey 1	**I1**	**I4**	**I8**	**S**	**S**	**S**											
Journey 7	**I2**	**I5**	**I9**	**S**	**S**												
Journey 6							**S**	**S**	**S**	**S**	**S**	**S**	**S**	**S**	**S**		
Journey 4	**I3**	**I6**	**…**	**S**	**S**	**S**	**S**	**S**									
Journey 2		**I7**	**S**	**S**	**S**	**S**	**S**	**S**	**S**								
Journey 5			**S**	**S**	**S**	**S**	**S**										

S = ‘journey state’

IX = ‘Input X’

Simulation details (5)

Input is checked against the 'real world' preferences of the simulated customers to get a correct input output pair...

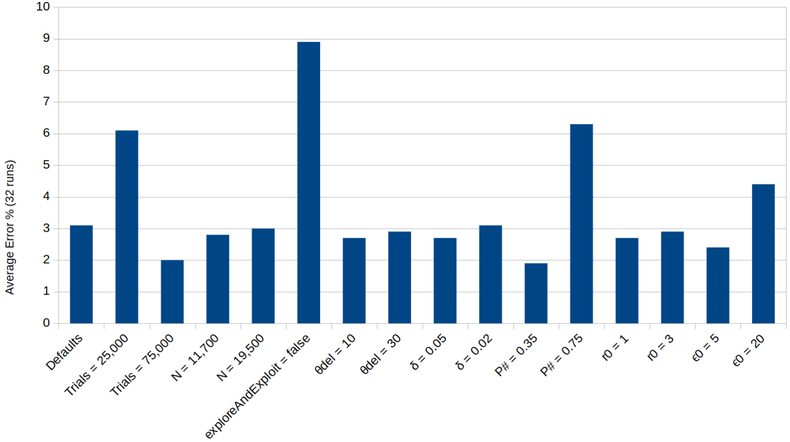
Condition	:	Action
(0)(0)(0)(5)(5)(3)(2)(5)(1)(0)(3)(4)(4)	:	?
[0,0] [0,0] [0,0] [5,5] [5,5] [#,#] [#,#] [5,5] [2,3] [4,5] [#,#] [#,#] [#,#]	:	1
[0,0] [0,0] [0,0] [5,5] [5,5] [#,#] [#,#] [5,5] [0,1] [0,1] [2,3] [#,#] [#,#]	:	2
[0,0] [0,0] [0,0] [5,5] [5,5] [#,#] [#,#] [5,5] [4,5] [0,1] [0,1] [#,#] [#,#]	:	3
...	:	...

Correct input–output pair in this example: `0005532510344:2`

We therefore assemble a list of inputs and answers ($>$ 51,000) for training and testing XCSI.

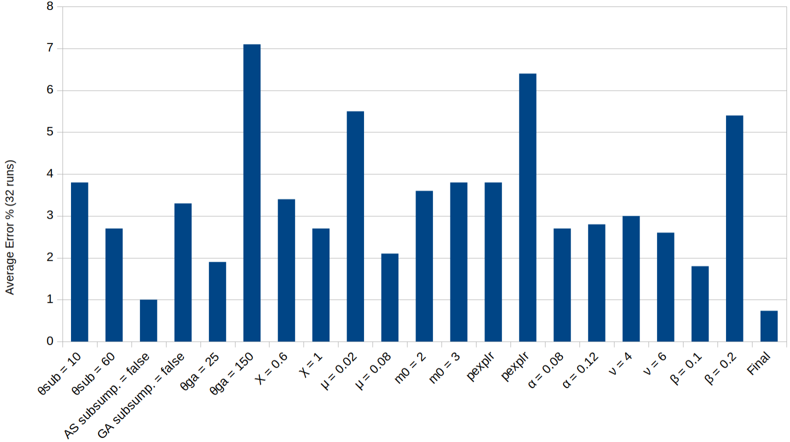
Results (1)

Figure 6: Error % for different parameter settings (1 of 2)



Results (2)

Figure 7: Error % for different parameter settings (2 of 2)



Final Parameters (Adjusted Only)

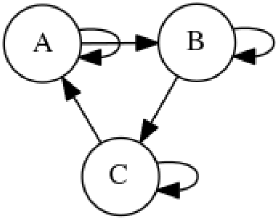
Parameter	Value	Brief Description
N	11700	Rule population size
$P_{\#}$	0.35	Probability of hash
ϵ_0	5	Error threshold
θ_{ga}	25	Genetic algorithm frequency
θ_{del}	10	Deletion threshold
β	0.1	Affects update of p,ϵ, and action set size for classifiers
α	0.08	Affects fitness updates
ν	6	Affects fitness updates
χ	1	Likelihood of GA crossover operation
μ	0.08	Likelihood of GA mutation operation
δ	0.05	Modifies the effect of fitness on classifier 'deletion vote'
θ_{sub}	60	Subsumption threshold
AS subsumpt.	false	Perform subsumption in the action set?

Final Performance

- Minimum trial error: 0.1%
- Average error: 0.734%
- Maximum trial error: 2%
- In concrete terms, over 99% of passengers would get the correct suggestion.

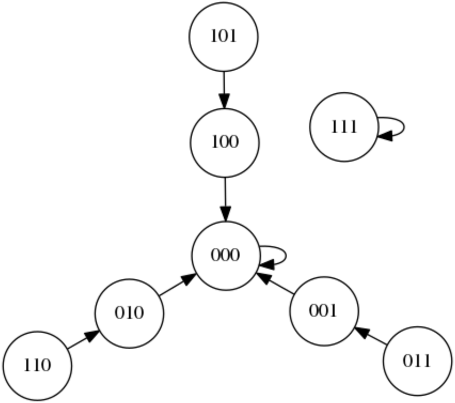
RBNs

Figure 8: A Random Boolean Network (RBN) with N=3, K=2



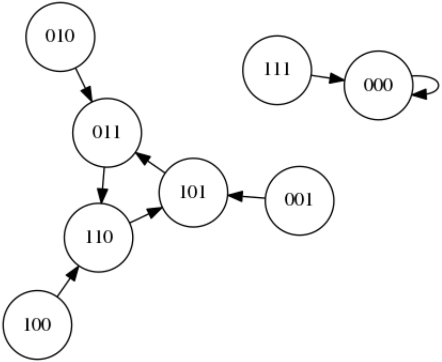
Controllability in RBNs (1/3)

Figure 9: State space of RBN of Fig. 8 (all AND); two attractors



Controllability in RBNs (2/3)

Figure 10: State space of RBN of Fig. 8 (all XOR); two atrractors

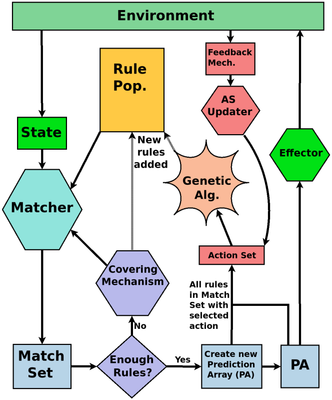


Controllability in RBNs (3/3)

- Each rule represents a *condition : action* expression that links specific states of the RBN (conditions) to bit flips (actions)
- To shift from single state to the state cycle attractor, apply one of `###:1; ###:2; ###:3`
- To shift from the state cycle to the single state attractor, apply one of `110:3; 011:1; 101:2; 001:3; 010:2; 100:1`
  - where # denotes "don't care" and the action represents the index of the bit to flip
- The objective is to

XCS – overview

Figure 11: XCS - condition is a bit string



XCS details in: Butz, Martin V., and Stewart W. Wilson. "An algorithmic description of XCS." International Workshop on Learning Classifier Systems. Springer, Berlin, Heidelberg, 2000.

Applying XCS to control RBNs

Applying XCS to control RBNs

Figure 12: Control graph for a N=5,K=2 RBN using XCS

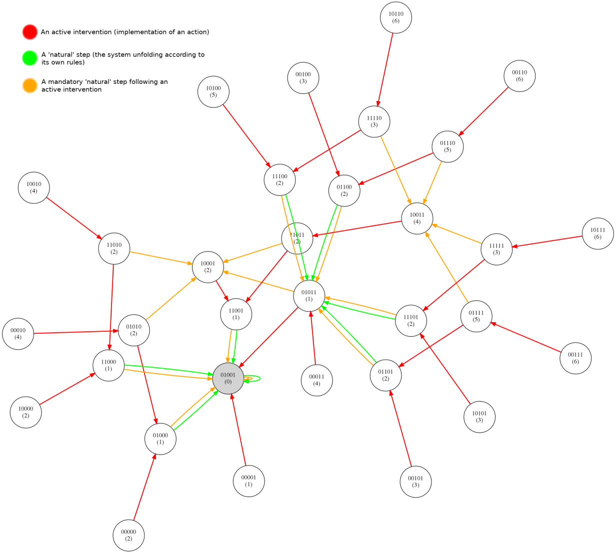


Figure for OCR vs XCS

System Dynamics Model

Population growth model

Controlling this model

Lotka-Volterra predator – prey model

XCSR

Conclusions

- XCSI is applicable to providing *personalised* travel recommendations
- As configured here, over 99% of passengers get the correct suggestion
- The technique is based on rules, hence human readable
  - why XCSI suggested a given action
  - how XCSI arrived at suggesting that action

Future Work

- Work has already begun. Version 2 includes:
  - relative journey times in QoS measures
  - more conditions (e.g. assistance required)
  - multi-modal options
  - ranked suggestions (not just 1)

Thanks/Acknowledgements

- Thank you for your attention

- This research was partly funded by the Department for Transport, via Innovate UK and the *Accelerating Innovation in Rail* (AIR) Round 4 programme, under the *Onward Journey Planning Assistant (OJPA)* project.

Covering mechanism

1. if match set has $< \theta_{mna}$ actions:

    1.1 randomly select action not in match set

    1.2 initialise new classifier with selected action and maximally specific condition components ([1,1][3,3] etc...)

    1.3 for each condition component, if random double in 0–1 range $< P_{\#}$, broaden the condition range (possibly to $\#$)*

    1.4 add classifier to population and match set

    1.5 re-start at 1

*See [Wilson, Stewart W. "Mining oblique data with XCS." International Workshop on Learning Classifier Systems. Springer, Berlin, Heidelberg, 2000] for details.

Action set updater (1)

For each classifier in action set (based on Butz and Wilson (2000)):

1. increment classifier experience
2. if (experience $< 1/\beta$):
   2.1 predicted payoff $+=$ ($\rho -$ predicted payoff) / experience
   2.2 error $+=$ ($|\ \rho -$ predicted payoff $| -$ error) / experience
   2.3 AS size $+=$ ($\sum_{c \in [A]}$ *numerosity* $-$ AS size) / experience
3. else:
   3.1 predicted payoff $+= \beta$ * ($\rho -$ predicted payoff)
   3.2 error $+= \beta$ * ($|\ \rho -$ predicted payoff $| -$ error)
   3.3 AS size $+= \beta$ * ($\sum_{c \in [A]}$ *numerosity* $-$ AS size)
4. update classifier fitness...

Action set updater (2)

Update classifier fitness (based on Butz and Wilson (2000)):

1. accuracySum $= 0$
2. create accuracy vector κ
3. for each classifier in action set:
    3.1 if(classifier error $< \epsilon_0$): $\kappa(\text{cl}) = 1$
    3.2 else: $\kappa(\text{cl}) = \alpha * (\text{cl. error}/\epsilon_0)^{-\nu}$
    3.3 accuracySum $+= \kappa(\text{cl}) * \text{cl. numerosity}$
4. for each classifier in action set:
    4.1 fitness $+= \beta * (\kappa(\text{cl}) * \text{cl. num./accuracySum - fitness})$

Genetic algorithm

1. select two parents via roulette wheel selection
2. create c1 and c2 (exact copies of parent 1 and parent 2)
3. for c1 and c2, set experience $= 0$ and numerosity $= 1$
4. if random double $< \chi$ then perform crossover
5. for c1 and c2, for each condition component, if random double $< \mu$ then mutate component*
6. add c1 and c2 to population
7. delete (less-fit) rules from population if required

*See [Wilson, Stewart W. "Mining oblique data with XCS." International Workshop on Learning Classifier Systems. Springer, Berlin, Heidelberg, 2000] for details.

Subsumption

- GA subsumption:
  1. if parent covers all inputs that offspring covers and action matches...
  2. ...and parent exp $> \theta_{\text{sub}}$ with parent error $< \epsilon_0$...
  3. then do not add offspring (increment parent numerosity instead)
- Action Set subsumption:
  1. find most general classifier in action set (covers most inputs)
  2. for each classifier in action set, check as with GA subsumption (but with general classifier, not parent)...
  3. does most general classifier subsume each other classifier?

XCS details in: Butz, Martin V., and Stewart W. Wilson. "An algorithmic description of XCS." International Workshop on Learning Classifier Systems. Springer, Berlin, Heidelberg, 2000.