The Xala Algorithm in the JiaoTong World

Submitted for CIS603 – Project 3

Denes Csala

Masdar Institute of Science and Technology, Abu Dhabi, United Arab Emirates

Keywords: Markov Decision Process; expert algorithm; system dynamics; adaptive learning.

1 Introduction

The purpose of this project was to create and simulate an efficient algorithm which would navigate a number of agents in the *Jiao Tong* (JT) world presented in class and attempt to maximize its payoff. We have created an expert algorithm (*Xala*) mainly driven by adaptive greedy-learning and system dynamics-style [1] feedback processes.

In the first section of the report, we present the *JT* game, followed by Section 2 with the modeling possibilities and our chosen modeling types. In *Section 3* we present the logic of the game, in the chosen modelling context, including the general algorithms that will play the *JT* game, besides *Xala*, presented in detail in *Section 4*. In order to observe the dynamics of the *JT* game and monitor the performance of the *Xala* algorithm, we have designed a set of experiments, presented in detail in *Section 5*, with the simulation results following in *Section 6*, before closing with conclusion in *Section 7*.

[1] John D. Sterman, Business Dynamics: Systems Thinking and Modeling for a Complex World, 2000, Wiley

The *JT* world, where the *JT* game is set can be modeled by a graph with the following nodes and directed edges, also graphically represented in Figure 1.

Nodes: A, B, C, D: Edges: AB, AC, BC, BD, CB, CD, DA

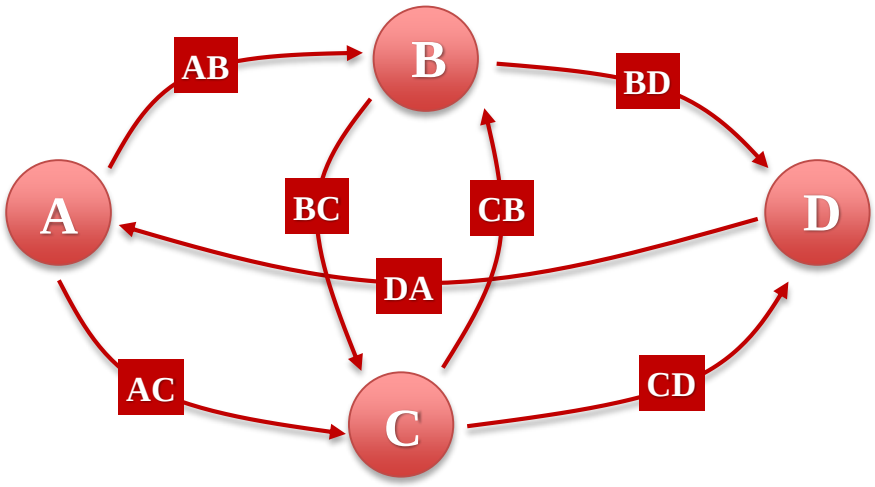

Figure 1 – The Jiao Tong world graph

2 Game design

Modeling the *JT* world can be approached from a multitude of perspectives. In this project we have used a combination of *Repetitive Cycle* (RC) modeling (equivalent to conversion to a normal-form game and widely discussed in class) and *Markov Chain* (MKC) modeling. In the former, the agent's decisions are taken in the nodes, whereas in the latter on the edges. This structure has been chose because it offers the benefits of both approaches, resulting in significant algorithmic flexibility and freedom. Also, empirically we have observed in our conducted tests that our algorithms based on both modeling approaches outperformed those employing a simpler approach, as presented in *Section 6*.

2.1 Repetitive Cycle Model

As it was presented in class, the following possible movement cycles can be identified in *JT* world, modeled as sequences of visiting certain nodes. Each of these cycles represents an action that an agent can take. Likewise the *JT* game can be converted into an (infinitely) repeated normal form multiplayer game. These cycles and their corresponding sequences of nodes are the following:

Cycle 0: B → C → …

Cycle 1: A → B → D → …

Cycle 2: A → C → D → …

Cycle 3: A → B → C → D → …

Cycle 4: A → C → B → D → …

Then any agent would choose a succession of these cycles as part of its strategy and an algorithm would return at the end of each cycle which cycle to play next, based on some rules – i.e. best expected payoff. We use this modeling approach for the *JT* game in the first agent of the *Xala* expert algorithm, dubbed *Greedy 0.1*.

2.2 Markov Chain Model

However, it is important to observe that at every node of the *JT* world the agents playing the game are required to take a decision regarding which link to continue on. This represents a *Markov Decision Process* (MDP). Since essentially the decision can be considered not to be taken in the node, but rather on the edge when traveling towards that node, the *MDP* is selecting an edge after another. At any given point in time, an agent will be moving along some edge. This can be considered a *state*. Then the action of selecting another edge to continue onto represents the *MDP* of which state transition next into for the agent. The sequence of these decisions gives us a sequence of *transitions* between different states and it is known *Markov Chain* (MKC)[2]. Taking the *dual* of the graph (Figure 2) that represents the *JT* world, the

[2] J. R. Norris, Markov Chains, 1998, Cambridge University Press

graph of this *MKC* (Figure 3) emerges This can be explained by the fact that, in contrast to the *RC* model, where the decisions are taken in nodes, in the *MKC* model the decisions are taken on edges – a duality.

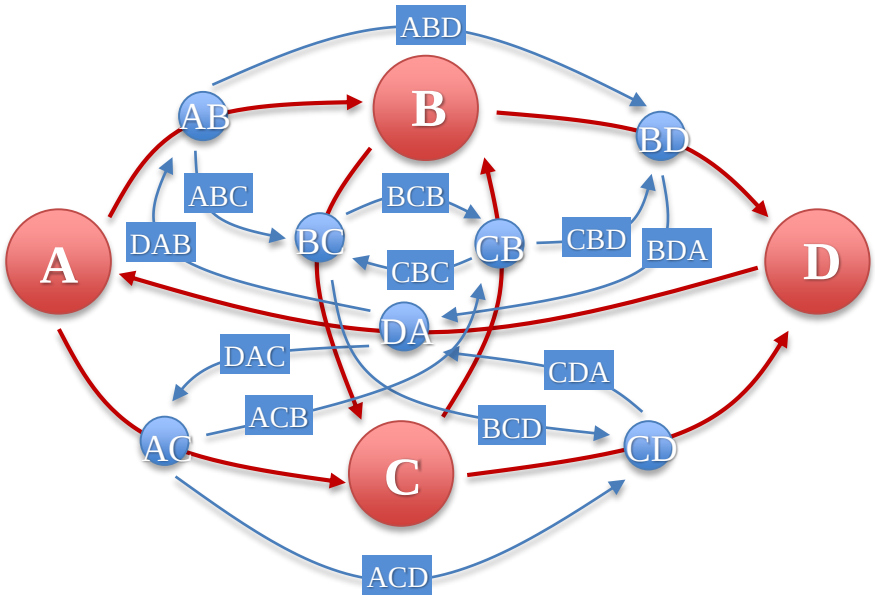

Figure 2 – The Jiao Tong world graph dual

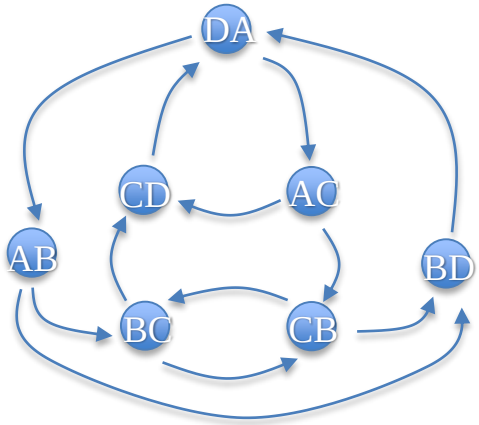

Figure 3 – Simplified Jiao Tong world graph dual, equivalent to MKC graph

The *MKC* graph, equivalent to the *dual* of the *RC* graph with have the following nodes and their incoming and outgoing edges (links):

AB:	in – DAB,	out – ABC, ABD
AC:	in – DAB,	out – ACB, ACD
BC:	in – ABC, CBC,	out – BCB, BCD
BD:	in – ABD, CBD,	out –BDA
CB:	in – ACB, BCB,	out – CBC, CBD
CD:	in – BCD, ACD,	out – CDA
DA:	in – CDA, BDA,	out – DAC, DAB

3 Game logic

In order to help with the design of a winning algorithm for the *JT* game, we have conducted a set of multi-agent simulations, where the *Xala* algorithm was tested against a variety of others.

The reason behind choosing the algorithms below was multifold: We expected a lack of edge-based decision-making algorithms and a higher percentage of algorithms based on the algorithms already discussed and presented in class and the previous projects. While our overall purpose in this project was to create a robust and effective learning algorithm, it was also part of the objective to win the *JT* tournament, where we expected that mostly algorithms based on the above selection would enter. Although, using the same logic, we could argue that we will be encountering algorithms based on satisficing in the same proportion, it is widely reported[3] that for the *MKC* type of problems follower

[3] Richard S. Sutton, On the Significance of Markov Decision Processes, 1997, University of Massachusetts

Alberto Reyes et al., Learning Qualitative Markov Decision Processes, 2005, Conference on Neural Information Processing Systems

learning algorithms perform the best. Also in multi-agent environments q-learning is a good performer[4]. Likewise we use a selection of greedy-based and q-learning algorithms, among a few more generic ones. A few of these have already been previously described in class or presented in earlier versions of this project. The algorithms used in our analysis and their setup parameters are the following:

- Predefined cycle
    - Plays Cycle 1: A → B → D → … throughout the entire game. We discuss later that the payoff scenarios have been setup in such a way that the nodes of this cycle are mostly on the highest payoff routes.
- ε – greedy
    - Plays a greedy learning algorithm, as presented for *Project 2*
        - $\varepsilon = 1/(2+t/3)$
        - $u_{initial} = \{10,10,10,10,10\}$
        - $v_{initial} = \{10,10,10,10,10\}$
- Satisficing
    - Plays a satisficing learning algorithm, as presented for *Project 2*
        - $N = 50$
        - $\lambda = 0.9$

[4] Gerald Tesauro, Extending Q-Learning to General Adaptive Multi-Agent Systems, 2003, IBM Thomas J. Watson Research Center

Adrian K. Agogino and Kagan Tumer, Quicker Q-Learning in Multi-Agent Systems, 2005, NASA Ames Research Center

Eduardo Rodrigues Gomes and Ryszard Kowalczyk, Dynamic Analysis of Multiagent Q-learning with ε-greedyExploration, 2009

- Q-learning
    - Plays a greedy learning algorithm, but only uses information learnt in the last cycle
        - $\varepsilon = 1/(2+t/6)$
        - $u_{initial} = \{10,10,10,10,10\}$
        - $v_{initial} = \{10,10,10,10,10\}$
- Linear decay Q-learning
    - Plays a greedy learning algorithm, but the value of the information learnt each cycle loses significance linearly with time
        - $\varepsilon = 1/(2+t/6)$
        - $u_{initial} = \{10,10,10,10,10\}$
        - $v_{initial} = \{10,10,10,10,10\}$
        - $v_{t+1} = (v_t + v_{t+1}) /2$
- Exponential decay q-learning
    - Plays a greedy learning algorithm, but the value of the information learnt each cycle loses significance exponentially with time
        - $\varepsilon = 1/(2+t/6)$
        - $u_{initial} = \{10,10,10,10,10\}$
        - $v_{initial} = \{10,10,10,10,10\}$
        - $v_{t+1} = sqrt\ (v_t * v_{t+1})$

- Greedy 0
    - Plays a greedy learning algorithm, with initial values set to 0 to simulate "instant learning".
        - $\varepsilon = 1/(2+t/3)$
        - $u_{initial} = \{0,0,0,0,0\}$
        - $v_{initial} = \{0,0,0,0,0\}$
- Greedy trained
    - Plays random for the first N cycles, then a greedy learning algorithm, with initial values set to average payoffs obtained in each cycle during the first N cycles.
        - $N = 50$
        - $\varepsilon = 1/(2+t/3)$
        - $u_{initial} = \{u_N, u_N, u_N, u_N, u_N\}$
        - $v_{initial} = \{v_N, v_N, v_N, v_N, v_N\}$
- Highest Payoff Lock-in
    - Plays random for the first N cycles, then repeats the cycle with the highest average payoff obtained in the first N cycles infinitely.
        - $N = 50$
- Random
    - Plays an edge (not a cycle!) at random, at every node.

4 Xala algorithm

While simulating games with agents using the algorithms presented above it has we have found out that while using *RC* modeling and node-based decisions is really efficient in the early stages of the game, in the later part all of the algorithms become less efficient, after having locked in on the same cycles and some edges being left completely unused at times. Likewise we tried to create an algorithm with a higher resolution than *RC* logic, turning therefore to *MKC* modeling. The system dynamics modeling technique is easy to implement for *MKC* modeling[5] and having had extensive experience with system dynamics, we have decided to design a system dynamics-based algorithm for this project.

However, with experience of later simulations, a system-dynamics only algorithm proved to be inefficient in the early stages of the game and therefore its learning process was affected and it failed to perform well later.

This is when we thought of creating an to expert algorithm that combines the benefits of greedy techniques in the early stages of the game, while at the same time gathering data used for training the system dynamics model and then gradually shift to using the latter. As for the algorithm based on greedy we chose as basis the best performing algorithm from the above presented, which turned out to be *Greedy 0* with a slowly declining learning rate ε. We will refer to this algorithm used in the initial stages of the game as *Greedy 0.1* and to the system dynamics-based algorithm as *SD*. The resulting algorithm from the combinations of these will be referred to as the *Xala* algorithm.

Formally, *Xala* is an expert algorithm that utilizes two experts: *Greedy 0.1* and *SD*. The choosing of experts is done with a relatively simple process by choosing *Greedy 0.1* with probability $\gamma = 1$ for the first τ_1 time steps and gradually shifting to *SD*, with $\gamma \to 0$ with a diffusion rate δ. Later, at times τ_2, τ_3, …, the *Greedy 0.1* is played again with a probability $\gamma = 1$, used to "retrain" *SD*, a process which we will present in detail in Section 4.3.

[5] C. E. Riddalls et al., Modelling the dynamics of supply chains, 2000, International Journal of Systems Science

4.1 Greedy 0.1

Greedy 0.1 is the first expert used in the *Xala* algorithm and it is used to provide a good performance in the initial stages of the *JT* game, while gathering data and training the *SD* algorithm. It is a greedy learning algorithm, as presented for *Project 2* with the following parameter values:

- $\varepsilon = 1$, if $t <$ *nr of cycles* (5) + 1 $\qquad\qquad \varepsilon = 1/(1.5+t/6)$, if $t >$ *nr of cycles* (5)
- $u_{initial} = \{0, 0, 0, 0, 0\}$
- $v_{initial} = \{0, 0, 0, 0, 0\}$
- $v = \{u_{average0}, u_{average1}, u_{average2}, u_{average3}, u_{average4}\}$
- $u_{averagei}$ is the average payoff of the first k iterations of cycle i ($i = 0\ldots4$, $k = 3$)

It is important to remark that the decay of ε has been slowed down a bit compared to the previous case (constant in denominator changed to 1.5 from 2), for the purpose of providing better learning for *SD*. Also, the initial utilities are dynamically set based on the average utility encountered in the first k iterations.

4.2 SD

For the *SD* algorithm we have built a system dynamics model of the *JT* world. The model is identical to the dual graph of the world, due to similarities between modeling *MK*C*s* and system dynamics. The nodes of the dual graph (the edges of the original) correspond to *Markov states* and they are modeled as *stocks*, whereas the transitions between them (nodes of the original, edges of the dual) are modeled as *flows*. The concept of this model is presented in Figure 4.

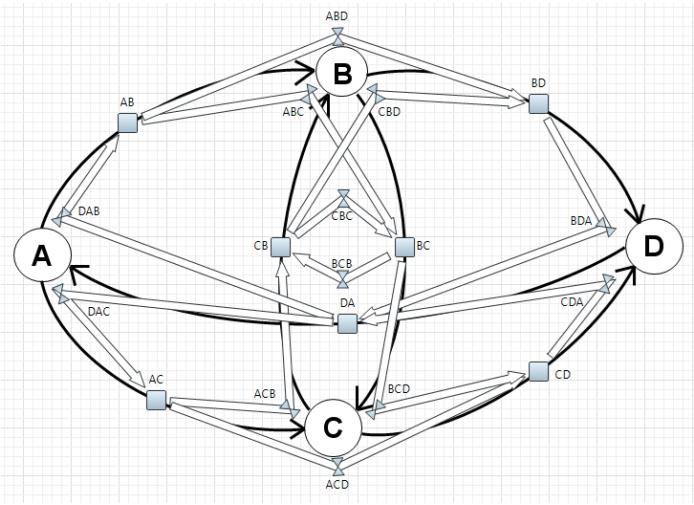

Figure 4 – Jiao Tong world system dynamics model concept, equivalent to model JT graph dual

First this model was simulated in dedicated system dynamics software (*AnyLogic*) in order to gain understanding of the main driving dynamics behind processes, before being converted into programming code (Figure 5).

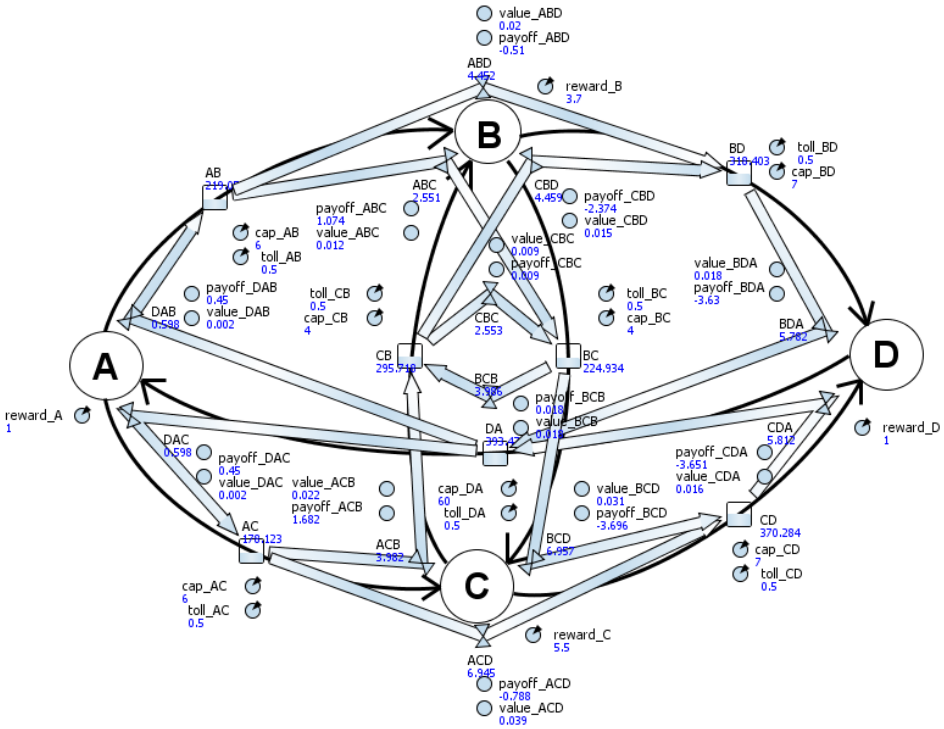

Figure 5 – Working Jiao Tong world system dynamics model

After simulation, the driving dynamics of the game emerges and hence a good algorithm logic can be deduced, which is the following.

From here on *node* and *edge* refers to the properties of the *dual* graph, i.e. the *system dynamics model*.

The starting information is taken from the *Greedy 0.1* algorithm, having run already. We monitor average traffic, average costs, average travel time, changes in traffic, changes in costs and travel time. Then we assume that for every node (stock) E a certain number (p) of edges are incoming and a certain number (s) are outgoing. Then, the incoming traffic (T_{in}) is the sum of all traffic on incoming edges and the outgoing traffic (T_{out}) is the sum of all traffic on outgoing edges. Since no agent stays in the nodes, these values should be equal. Then, using the data gathered previously by *Greedy 0.1* we estimate that of all incoming traffic into node E, a certain percentage (α_1) will take the first edge, a certain percentage α_2 will take the second edge, and so on. Since the *JT* world dual only contains nodes with maximum two incoming and maximum two outgoing edges, the percentage traffic choosing the first edge will be noted with α, while the percentage of choosing the second will be then $1 - \alpha$.

Afterwards, having monitored what is this percentage and knowing the average (historical) travel times, one can estimate the expected traffic on every edge, at a certain time in the future, with decaying accuracy, of course. This information is then used to create a dynamic model about the state of the *JT* world and make a beneficial decision for the agent. The algorithm logic is as follows:

- in the first N cycles, *Greedy 0.1* gathers information about:
    - The current traffic on each node
    - The percentage of choosing the first outgoing edge on each node
    - The starting time, finishing time and hence the average time spent on every node (equivalent to traversing an edge in the JT world) – normalized to the average time of the world

- then the first estimate of the traffic (*expectedTraffic*) on every edge is formed using
    - $expectedTraffic_i = T_{out}^t = \alpha \cdot \sum_{edges} T_{in}^{t-1}(E)$, for every edge (i = 0...6)
- The expected traffic on the just traversed edge is updated with the actual traffic encountered.
- A similar process is followed to find out the expected travel time on every edge and updated on the just traveled edge.
- The expected traffic values are weighted then by the average time values to create the "*expectedTrafficSmart"* indicator
    - $expectedTrafficSmart_i = expectedTraffic_i \cdot avgTravelTime_i$, for every edge (i = 0...6)
- Using this indicator and information about the capacity of the links, the expected congestion (c) is calculated, which a positive or zero value, with zero meaning that there is no congestion on the node, i.e. the number of vehicles is lower than the link capacity.
    - $c_i = \max(0, (expectedTrafficSmart_i - capacity_i) / capacity_i)$, for every edge ($i$ = 0…6)
- Then the expected cost of traversing every edge and consecutively the expected utility of taking route is calculated using the payoff values provided minus tolls and the calculated expected congestion data multiplied by a congestion cost η_i, for every edge (i = 0...6)
    - $u_i = payoff_i - toll_i - \eta_i \cdot c_i$, for every edge ($i$ = 0...6)
- When the agent crosses and edge, the algorithm updates the corresponding traffic, cost and time values.

The sub-problem of the estimation of the congestion cost η

While we can just easily assume a fixed number for congestion cost, it has been clearly seen in the *JT* world that payoffs do not change linearly neither with congestion, or time. This means there is a more complex equation driving time – and congestion-based costs and it needs to be estimated. We have created a linear and an exponential approximation model for the congestion cost, resulting in the algorithms *SD linear* and *SD exponential*.

The linear congestion cost approximation model assumes that the cost of crossing an edge in the *JT* world (equivalent to staying in one node of the *JT* world dual) increases linearly with congestion. Therefore:

- $\eta_i = \eta_{base} \cdot c_i$, for every edge ($i = 0...6$) ($\eta_{base}$ = payoff of maximum payoff node of the world)

It has been found that proves ineffective when dealing with payoff scenarios where the payoff are very close to each other, and hence the exponential model was created, essentially putting a stronger emphasis on congestion. For example if we have a scenario in which we can choose between routes, but one has a significantly higher payoff than the other, we prefer to take that route, despite the congestion we might face (up to a certain level, of course). But, if we have the opposite scenario, in which we have two routes which both yield the same payoff, then the least congested route is preferred. Likewise, an exponential congestion model was designed to model this phenomenon.

The exponential congestion cost approximation model assumes that the cost of crossing an edge in the *JT* world (equivalent to staying in one node of the *JT* world dual) increases exponentially with congestion and it is calculated locally, in each node, based on obtainable payoffs of the target nodes. Therefore:

- $\eta_i = [\eta_{base} \cdot 1/(1+\sigma \cdot |payoff_{i1} - payoff_{i2}|)]^{c_i}$, for every edge ($i = 0...6$)
  - where: σ is congestion exponent ($\sigma = 0.5$)
  - $|payoff_{i1} - payoff_{i2}|$ is the diameter of payoffs between taking edge 1 and 2. If there are more than two outgoing edges, the difference between the smallest and the largest value is taken.

4.3 Xala

Xala is the expert algorithm that chooses between playing *Greedy 0.1* and *SD,* playing *Greedy 0.1* with probability γ. All parameters of both *Greedy 0.1* and *SD* are calculated and adjusted at every time step τ. After the first time steps τ_1, γ gradually declines and *SD* gets chosen more frequently with a probability diffusion of δ. At certain time steps τ_2, τ_3,… *Greedy 0.1* is played again with probability γ=1 in order to "retrain" *SD.*

- γ =1 for the first τ_1 time steps (τ_1 is decided accordingly to $t = 50$ of *Greedy 0.1*)
- γ =1 if $\tau = \tau_2$ or $\tau = \tau_3$ or … ($\tau_2 = 100$; τ_3, τ_4, … = 0)

- $\gamma = \gamma \cdot \delta \quad (\gamma_0 = 0.5, \delta = 0.2)$

Both the *Greedy 0.1* and *SD* learning processes are run the background, regardless of which expert is chosen to be played by *Xala*, since the architecture of the algorithms permits this, hence all parameters (*t, u, v, ε* for *Greedy 0.1*; *expectedTraffic*, *c*, *η* for *SD*) are updated every time step.

5 Experiment Setup

In order to gain a good understanding about the *JT* game dynamics and test the *Xala* algorithm for robustness, 5 characteristic payoff scenarios have been created. 3 games of 300 minutes each have been run for each of these scenarios and the average performance (cumulative or average payoff) of the algorithms over these 3 scenarios was taken as indicator.

Table 1 – Experiment setup

Scenario	Payoff A	Payoff B	Payoff C	Payoff D
1	1.0	3.7	5.5	1.0
2	1.0	7.0	3.5	1.0
3	4.0	4.0	4.0	4.0
4	5.0	0.0	– 5.0	5.0
5	1.0	1.0	20.0	1.0

- *Scenario 1*: base payoff scenario, as used in the previous projects. Nodes have payoffs close to each other (small payoff diameter) and the *B*→*C*→... cycle yields a significantly higher payoff than other, hence it is expected to get congested fast. The low capacity of links *BC* and *CB* thus makes this scenario interesting.
- *Scenario 2*: A scenario with flipped payoff order between *B* and *C*, the presence of *Predefined Cycle* agents, which were set up to play Cycle 1: A → B → D → …regardless of the payoffs makes this interesting.
- *Scenario 3*: A completely symmetric scenario, with every node yielding the same payoff.
- *Scenario 4*: A scenario with the high payoffs transferred into nodes *A* and *D*, also contains a zero and a negative payoff.
- *Scenario 5*: Asymmetric payoff scenario, where one node yields significantly higher payoff than the others.

6 Simulation Results

In the following we represent the simulation results using a set of graphs that depicts the performance of the algorithms in the *JT* world over time. The unit of the *X* axis (time) is 16.67*ms*, whereas on the *Y* axis, the average nominal payoff of each algorithm is represented (it changes game by game). Along with the algorithms presented in section 2, we use two versions of the *Xala* algorithm in our experiments, one with linear time cost adjustment (*Xala linear*) and one with exponential time cost adjustment (*Xala Expo*).

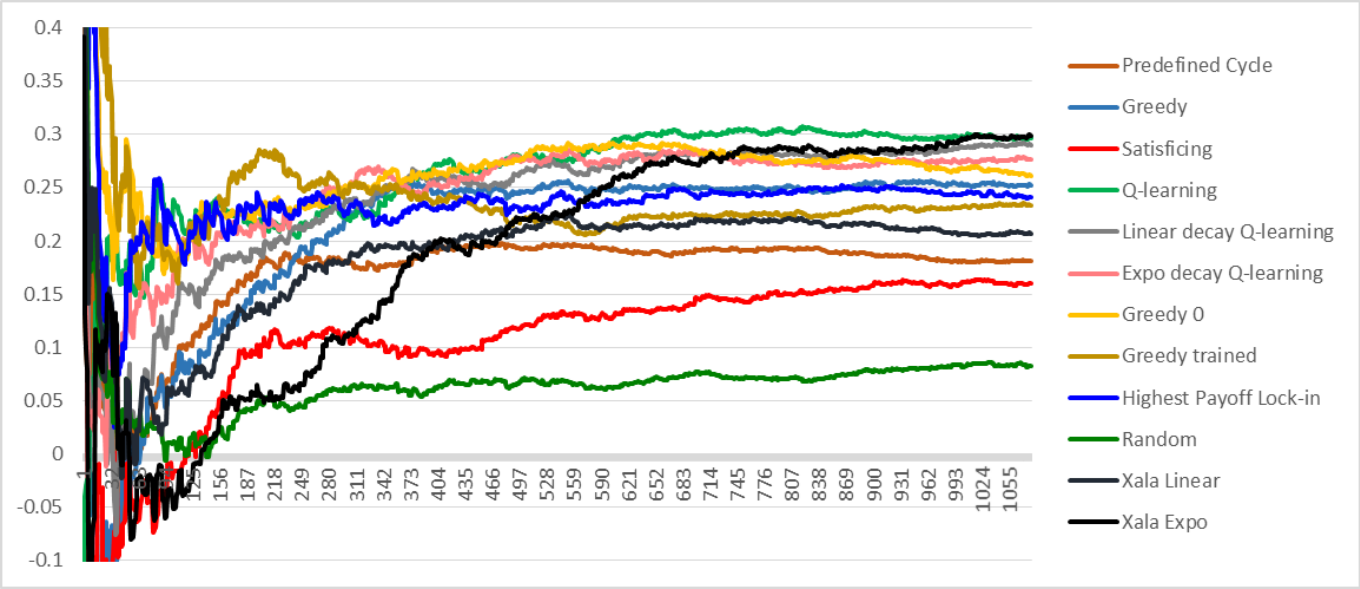

Figure 6 – Scenario 1 simulation results

In *Scenario 1*, *Q-learning* and *Xala Expo* finish on the top. We can notice the tow training periods of *Xala Expo* and also that it has a slower learning rate than the others – the accurate estimation of expected congestion *c* requires a lot of data, but eventually it outperforms them. *Greedy*-based algorithms do well, with *Random* trailing, as expected.

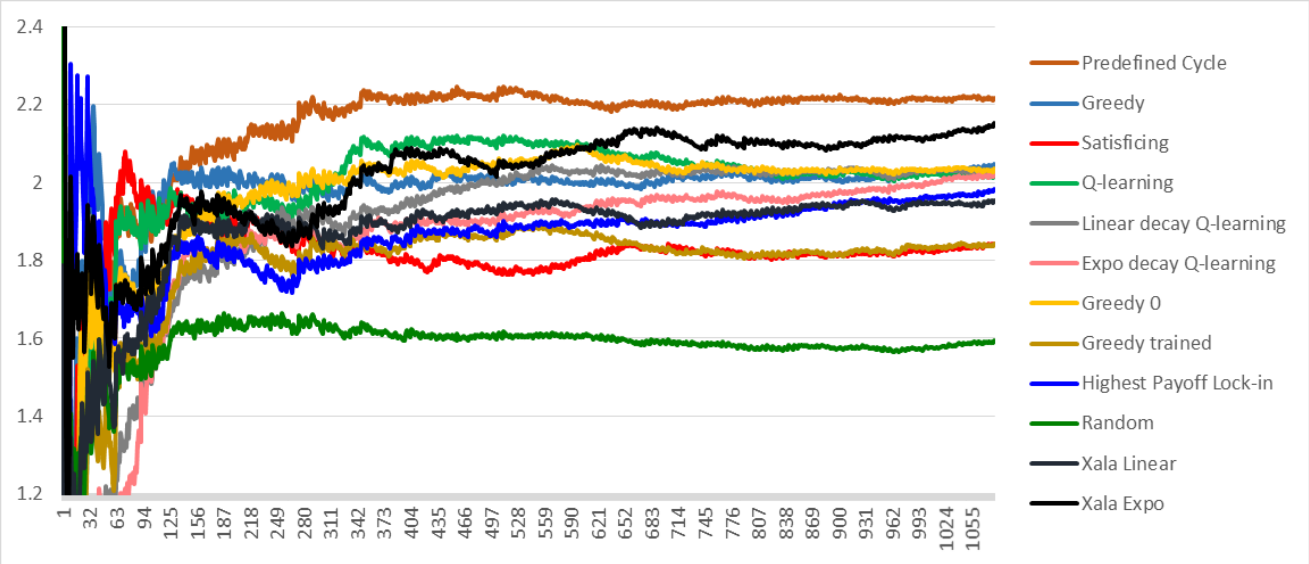

Figure 7 – Scenario 2 simulation results

In *Scenario 2*, the *Predefined Cycle* algorithm, which follows the high-payoff cycle finishes on the top, which somewhat expected, followed by *Xala Expo*. It is important to note that the difference between *Xala Expo* and its followers is much larger than in *Scenario 1*, meaning that *Xala Expo* managed to avoid the congestion created on the A→B→D→… cycle at the right times.

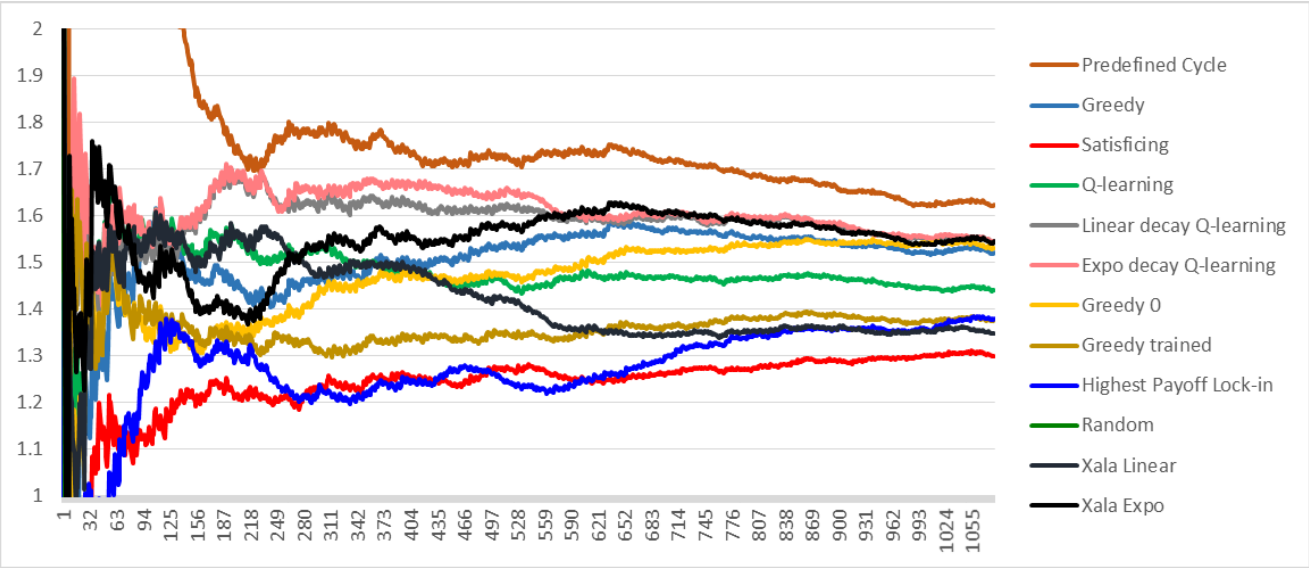

Figure 8 – Scenario 3 simulation results

In *Scenario 3* is again won by the *Predefined Cycle* algorithm, followed by *Xala Expo* and *Exponential decay Q-learning.* With this being a perfectly symmetric world, with equal payoffs in every node, the edge-based decision logic meant "higher resolution" for *Xala Expo* and managed to take the lesser congested routes, leading to higher payoffs.

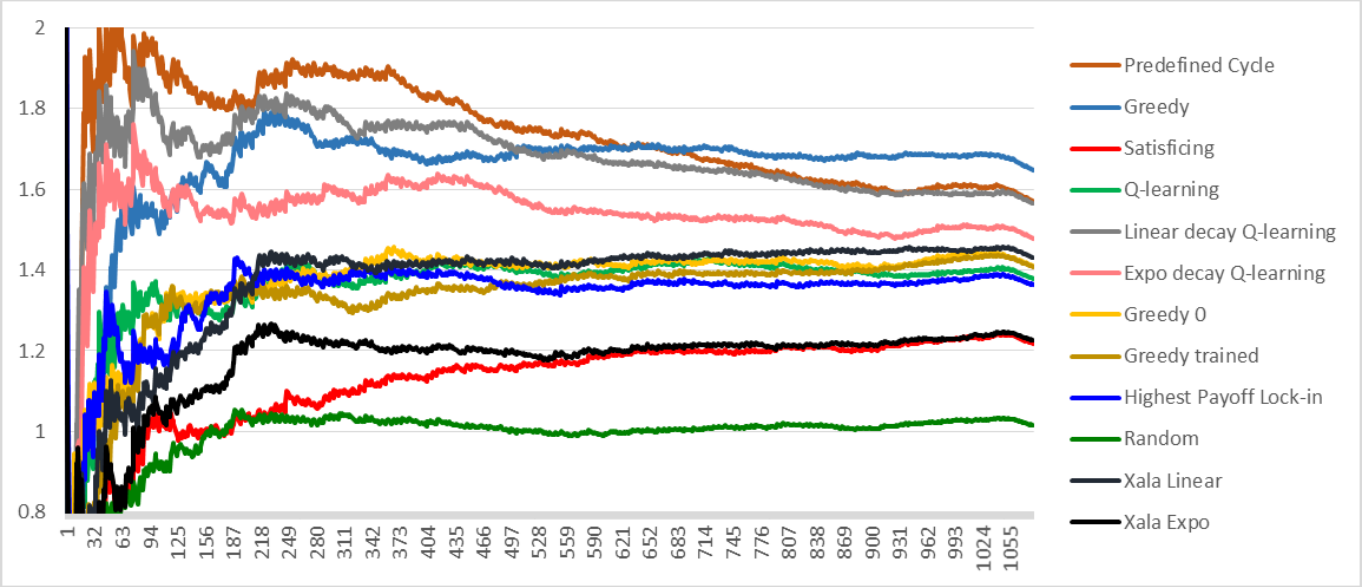

Figure 9 – Scenario 4 simulation results

In *Scenario 4*, *Xala Expo* finishes low, and *Xala Linear* in the middle. The scenario is won by *Greedy*. We have observed that this is due to the self-inflicted congestion of *Xala Expo*, which was not significant in other scenarios, now it became so, leading to lower finishes. This is possibly due to the fact the required learning period for this payoff scenario for *Greedy 0.1* is longer than 50 cycles, a fact that can be confirmed by looking at the performance of other *Greedy*-based algorithms. This resulted in an efficient learning and thus an erroneous estimation of congestion by *SD* and consecutively a bad performance for *Xala Expo*.

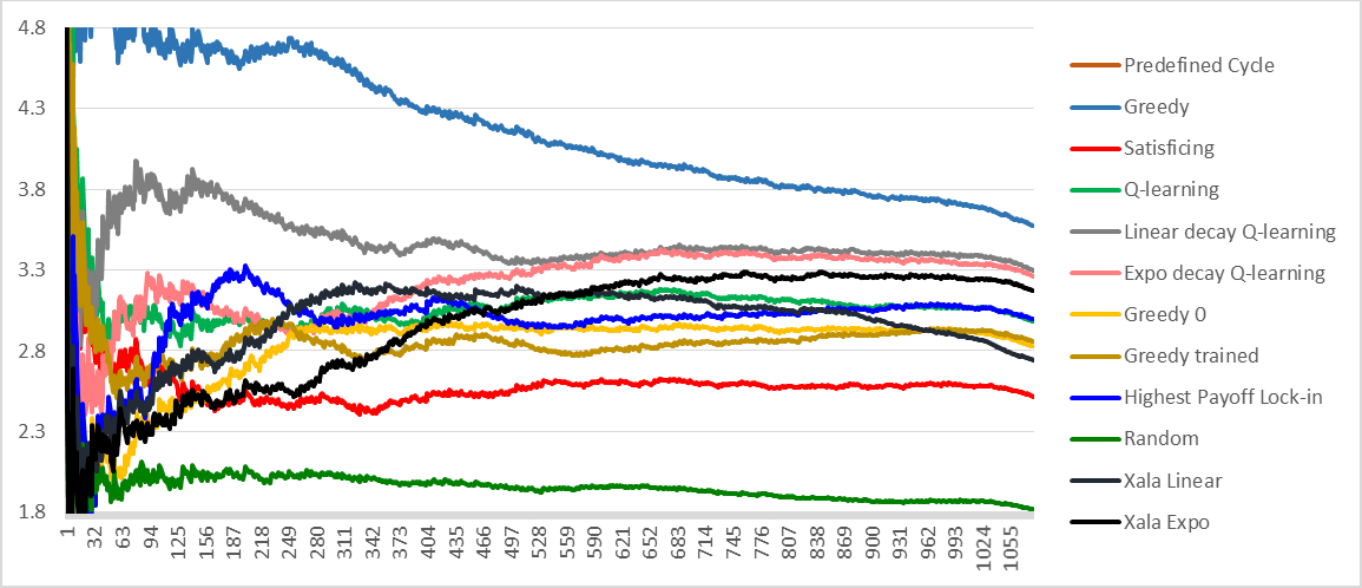

Figure 10 – Scenario 5 simulation results

Scenario 5 is the "outlier problem", where one of the nodes has a much higher payoff than the others. *Xala Expo* performs relatively well in this Scenario, won by *Greedy*, followed by the two adaptive *Q-learning* algorithms. It is possible that the initial utilities values for the *Greedy* algorithm allowed a faster learning for it and hence it obtains a significant advantage in the first part of the simulation. The initial utilities of *the Greedy 0.1* played by the the two *Xala* algorithms are estimated based on the *JT* world diameter, a technique which was deduced empirically, but might fall under revision and need adjustment, based on this last performance.

7 Conclusions

Figure 11 presents the overall simulation results, after averaging and normalizing the payoffs of each scenario. We can observe that the *Xala Expo* algorithm wins by a large margin (not taking into account the *Predefined Cycle*, which is practically a dummy algorithm, since it uses information not available to other agents, and it is deliberately placed on a high payoff route, predefined by the game designer). *Greedy* and *Q-learning*-based algorithms finish in the middle, with *Satisficing* and *Random* trailing. This results is as expected by literature, predicting that *Q-learning*-based

algorithms perform well in MKC problems. The *Xala Linear* performs slightly worse than much simpler *Greedy*-based algorithms, but this has been predicted, as the relationships between the payoffs and congestion was expected to not to be linear. Then, a bad estimation of the congestion leads to the *SD* expert suggesting congested routes, leading to low payoffs for *Xala Linear*.

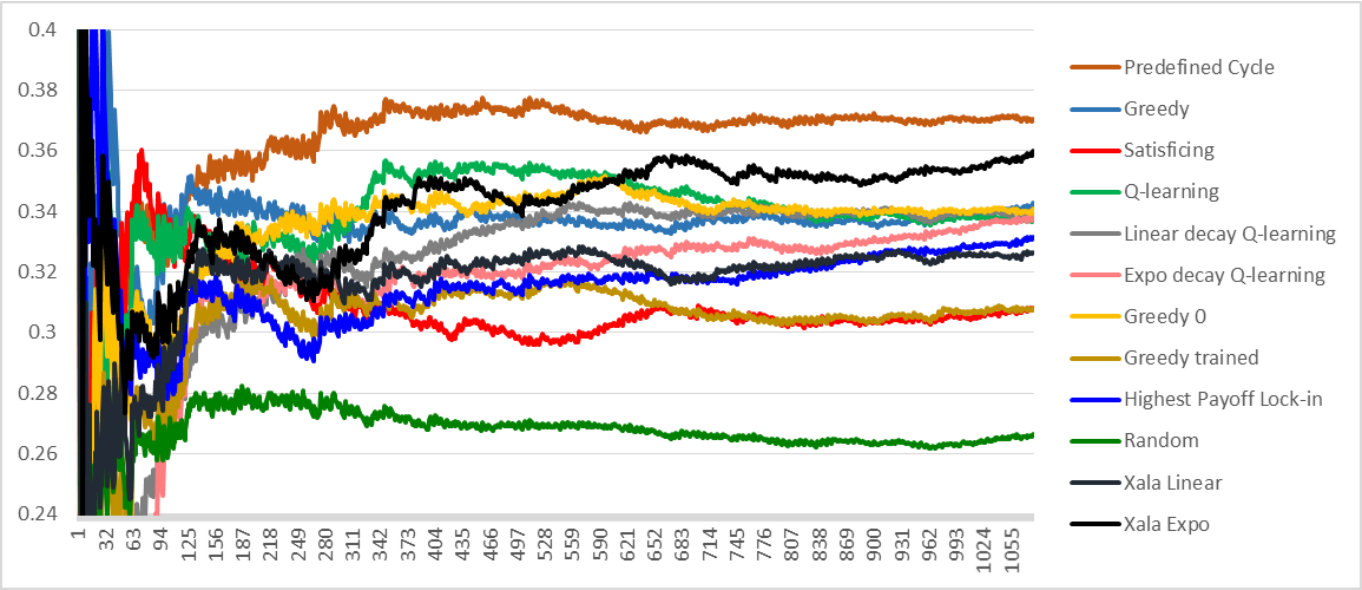

Figure 11 – Overall simulation results

We believe that Xala Expo algorithm has great potential in MKC type decision-making problems and it worth studying further, as the estimation of a lot of parameter was impossible due to the time constraints of this project.

An optimization of the Greedy 0.1 expert could be carried out, especially focusing on learning times, learning rate ε and the values of the initial utilizes, since we suspect this might have cause the bad performance of both *Xala* algorithms in *Scenario 4*.

Arguably, the estimation of the relationship between congestion and payoff is the Achilles-heel of the *SD* expert, likewise a more complicated mechanism could be used for a more accurate estimation, such as a regression model or even a neural network. We believe that with a good travel-time and congestion estimator, very accurate predictions of

future behavior can be achieved, leading to an easy selection of the least congested and/or the highest payoff routes and behavior optimization in general.

The choosing of experts of the *Xala* algorithm can also be tweaked and more experts can be included for potentially better results. The decision of whether to implement re-training cycles or not and when is also an important aspect.

It is also important to note that we believe that the 3+3 *Xala* algorithms had a negative effect on each other, scoring higher on average against the other algorithms on their own. Also, the parameter values and settings were developed with the requirements of this project in mind, but they have been designed to allow for easy generalization.

The Xala Expo expert algorithm using two agents based on adaptive ε-greedy learning and system dynamics performed very well in our simulations of the Jiao Tong game and we believe it has great potential in MKC type decision-making problems and it is worth investing more time and effort into its study and research.

Nomenclature:

JT: Jiao Tong

RC: Repetitive cycle

MDP: Markov Decision Process

MKC: Markov Chain