Project 2b

Submitted by: Denes Csala

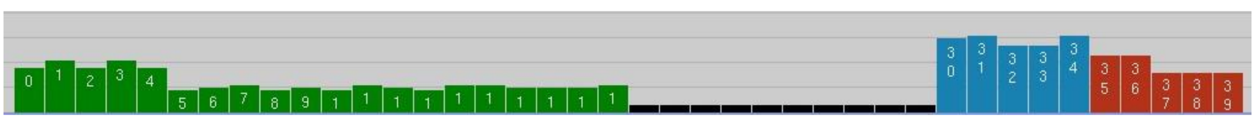
Team: Xala

I have coded up and ran the ***ε-greedy*** and ***satisficing*** algorithms in a few scenarios. In my experiments I have used a group of ~40 agents. The decision algorithms that they used other than the above two were ***random*** and ***follow on particular cycle*** throughout the simulation.

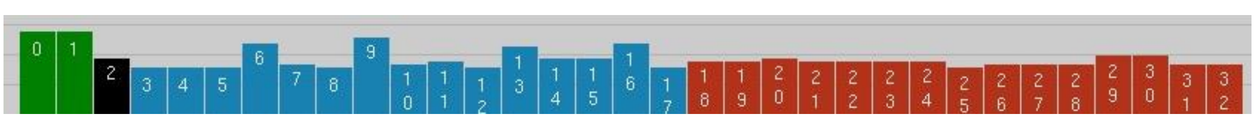
For the first set of experiments, I have used ~20 *random* agents, ~20 agents following the *cycle a-b-d*, one *ε-greedy* agent and one *satisficing* agent. In all scenarios, after the payoffs has stabilized, the *ε-greedy* agent won, followed closely by the *satisficing* one. An illustrative example, using the color coding define above is presented below.

The setup for this experiment was with the default payoff scheme (node C highest, node B 2nd highest), with ε default for the greedy algorithm and N=20…30 learning rounds for the *satisficing* algorithm and a λ=0.7…0.9.

However, as I started to introduce more (~5 agents) of the “more intelligent” algorithms, and steer the gameplay towards self-play, their edge started to diminish, as the more preferred cycles, such as *b-c* and *a-c-b-d* became more crowded. The performance of *random* was comparable to that of *satisficing*, in some cases.



In fact, if I set the majority of the agent population (~ 15+15) to use *ε-greedy* and *satisficing* algorithms, a small number of *random* agents will outperform them and even a static algorithm such as *a-b-d* would do so, in some cases.



If I changed the payoff structure and set node A significantly higher payoff than C, the *ε-greedy* and *satisficing* algorithms would still score high, and *satisficing* would win in some cases, the winning depending on the game length, but I found no clear pattern and alternating wins between the two was the dominant pattern. The competitive edge of these algorithms compared to *random* and static choice is significantly lower. It is important to note that our static algorithm used is *a-b-d*, which would in all rounds visit the high-payoff node A and, despite being high in numbers, would score decently in our tournament.

It is worthwhile to note the experimentation I had with the initial values for the *u* and *v* vectors of the *ε-greedy* algorithm. The algorithms first starts by taking the cycles one by one and lowering their *u* value significantly, until a stabilization occurs. These initial values greatly affect two algorithm characteristics: the lock-in cycle and the lock-in time. A high initial value (50 <) guarantees an almost certain lock-in on the cycle with the highest payoff, but this lock-in might occur too late and till then the *ε-greedy* practically behaves as random. A low initial value (< 5) guarantees a quick lock-in on a cycle, which might lead to a quick lead in the payoffs, but it might turn out to be the wrong one and our algorithm might fall behind at later time steps. Self-correction is possible in both cases, but it takes a significant amount of time (100+ cycles). I have observed, that with the given payoffs, a close to optimal initial value for *u* and *v* is 10.

For the case of the *satisficing* algorithm, a λ set too high – or even to 1, in the extreme case – might not lead to a chosen cycle and the algorithm might end up performing on par with *random*. A λ set too low will produce a lock-in into a cycle which does not yield the highest payoffs and, though *random* might be outperformed, other intelligent algorithms, such as the *ε-greedy* are not. Similarly, the number learning periods set too high (50+) would essentially make the *satisficing* algorithm into a *random* one, with little marginal gain – aspired payoff only increases very slowly after a while due to payoff limitations imposed by the capacity of the links of the world and the presence of other agents.

As a general observed behavior, the *ε-greedy* algorithm cycles through the cycle choices until it locks in into one (which can change later, if the route becomes too crowded – i.e. other learn too – and the payoffs are diminishing) whereas the *satisficing* algorithm changes its choice more frequently, it essentially stays on cycle until it becomes too crowded and the payoff diminish, then it randomly switches to another one until it meets its payoff expectation again and then stays there – and it repeats the cycle again.

As a final comment I would like to mention that I have run a special version of the *ε-greedy* algorithm, which only used the payoff information of the last cycle as opposed to the entire history. This is equivalent to changing the relation $v(a_t) = v(a_t) + \frac{r_t}{T_1 - T_0}$ into $v(a_t) = \frac{r_t}{T_1 - T_0}$. In the subsequent simulations, this has done comparably to the original *ε-greedy* version, with full history.