BRA - An Algorithm for Simulating Bounded Rational Agents *

Stephan Schuster

June 1, 2010

Abstract

This paper describes a simulation approach for modelling decision-making processes under incomplete and imperfect information in Agent-based Computational Economics (ACE). The main idea is to represent decision-making in a model-free framework that can be applied to a larger set of simulation problems, not just the domain modelled. The method translates some basic sociopsychological concepts from the bounded rationality and learning literature into an executable algorithm. In a simple example, the algorithm is applied in the domain of behavioural game theory, illustrating how the algorithm can be used to reproduce observed patterns of human behaviour.

JEL Classification C63

1 Introduction

In the economic literature, several approaches for modelling human decision making under uncertainty and imperfect information exist. These approaches, often summarised under a broad notion of bounded rationality, modify the orthodox definition of perfectly informed and hyperrational agents with simpler, psychologically more plausible behaviour. Thus, for example, subjective expected utility theory acknowledged that individuals are not fully informed and replaced objective probabilities with subjective. Prospect Theory (Kahnemann and Tversky 1979) modified perfect rationality by stating that human decisions are biased by the anticipation of future losses rather than gains, which explains deviations from the maximising principle. While such theories are modifications of rational choice, Simon (Simon 1956a) introduced with the satisficing principle a psychological perspective into decision theory. Satisficing agents deviate from rational choice behaviour by not maximising, but rather searching for satisfactory alternatives.

*This is a reprint of the publication in Computational Economics. The official version available at: www.springerlink.com

Decision theory formally treats decisions under various degrees of certainty; learning theory on the other hand is interested how actors learn about their environment and alternatives and to make better choices over time. Also here, behavioural approaches have been introduced as an alternative to rational (Bayesian) updating of beliefs. A popular approach to modelling learning in Game Theory is Reinforcement Learning (RL). For example, Erev and Roth tested the predictive capabilities of simple RL models by comparing theoretical results with data from behavioural game theory experiments (Erev and Roth 1998; Roth and Erev 1995). Some models add also simple belief components: Camerer and Ho's 'Experience Weighted Attraction' learning model combines reinforcement learning and fictitious play in games (Camerer and Ho 1999).

ACE models typically model agents with imperfect and incomplete information and local interactions. Learning and decision-making is typically represented ad hoc, and introduced as the need arises. Often, RL approaches have been applied. Only few models use more generic approaches, for example Learning Classifier Systems (e.g. LeBaron et al 1999; Kirman and Vriend 2001; LCS). LCS represent decisions in form of simple if-then rules, of which initially several actions are proposed in a certain situation. A learning algorithm is responsible for searching and selecting the best rules. An alternative representation comes from psychology, where the cognitive architecture CLARION has been applied to social simulation (Sun and Naveh 2007). In CLARION, agents learn by a combination of rule extraction and RL. For an overview of how learning is represented in ACE models, see Brenner (2006).

The algorithm presented here proposes a simple generic decision model for boundedly rational, adaptive artificial agents in ACE models. Instead of asking how learning can be represented in a concrete model, the idea here is to build on a framework for a wider range of applications. Due to its general nature, the approach is therefore quite pragmatic - operationalisation and its application as computer program are central design issues. The contribution of this paper is of a technical nature and lies in the combination of machine learning techniques with simple sociopsychological concepts of learning. Technically, it is similar to LCS or CLARION, but designed more specifically for economic decision problems.

The paper is structured as follows: First, the simple conceptual model of the Bounded Rationality Algorithm (BRA) is presented, before being specified formally as algorithm, and compared with similar approaches. An example demonstrates how the algorithm works in practice and to what types of problems it may be applied.

2 Concept

The algorithm assumes agents that start with very limited information about the world, and possess no causal model of how their actions affect themselves or their environment. The goal is to build an internal representation of the environment and increase its knowledge about the world in order to guide its decisions. The following paragraphs outline the prototypical (human) mental model that forms the basis of the approach.

A mental model is an internal representation of an external reality. It is built using experience, perception, and existing problem-solving strategies. A mental model contains minimal information, is unstable and subject to change and used to make decisions in novel circumstances. A mental model must be 'runnable' and able to provide feedback about the results. Humans must be able to evaluate the results of actions or the consequences of a change of state (Markham 1999). Assuming in a simplified manner that humans are mainly interested in own welfare, their goal is to find suitable behaviour strategies that optimise utility under different conditions. Information processing and memory is costly, so that the internal model being built has to be minimal and efficient. Three main phases can be distinguished here:

- *Evaluate cognitive cues:* In any state of the environment, the agent must be able to choose an action. If low or even negative rewards are experienced, the agent can attempt to apply a different action. If this fails to improve the agent's welfare, this is a hint to pay attention to more cues from the environment and distinguish better between different situations.
- *Decide what to know:* Paying attention to all cues from the environment is computationally too expensive and memory too limited; humans must filter out certain aspects of their perception in order to decide and act effectively. An agent has to 'decide what to know' (Rubinstein 1998). Since the measurement of useful information is the agent's welfare generated by its actions, this decision procedure can be represented a search over all possible state-action mappings. If the agent is satisfied with a mental model containing a subset of these mappings, it might stop searching for a better model or decrease its search intensity.
- *Update the cognitive model:* If the environment changes, some aspects of the internal model might become obsolete. The agent will then experience a change in utility. In certain states, learning new actions might be sufficient. However, it might also be that the representation of the state is not accurate anymore (e.g. a new type of agent appears). In this case the representation has to be changed, e.g. by removing old representations and start the search process anew for certain parts of the model.

Simon (Simon 1956b) already lined out a concept of how an organism can learn these relationships. His framework is based on the following components:

- The set of behaviour alternatives A
- The set choice alternatives A' for bounded rational or computationally less powerful individuals; this set may be only a subset of A.
- Possible future states S
- Payoffs connected with S, represented as a function of S, $V(s)$.

- Probabilities for S. There is uncertainty which sate occurs after a particular behaviour.

Bounded rational individuals do not typically know the mapping from behaviour alternatives A to future welfare $V(s)$. A possible strategy to learn about the occurrence and the desirability of these future states is according to Simon: Start with a mapping of each action alternative $a \in A$ to the whole set of S. Using a utility function such as $V(s) \in \{-1, 0, +1\}$, find $S' \subset S$ such that (expectedly) $V(s) = 1$. Then gather information to refine the mapping $A \to S'$ (i.e., which actions lead to which result under certain conditions) and search for feasible actions $A' \in A$ that map to S' (Simon 1956b). This is a simple way to represent (dynamic) knowledge about the environment with an action-centered, experience-based approach.

As an example, consider an employer who faces the decision whether to hire or not to hire a worker. A worker might be more or less highly skilled, which is not directly observable without screening for additional attributes. At a very coarse level, the possible current states are: 'worker is skilled' and 'worker is not skilled'. To the decision maker, these states are hidden (information is incomplete). After deciding whether to hire or not to hire, the possible future states are: 'hired and worker skilled'; 'not hired and worker not skilled'; 'not hired and worker skilled'; 'hired and worker not skilled' - probably in this preference order. Without taking into account any additional information, skilled and unskilled workers will be hired with the same probability, using, for example, a blind or random strategy as no clues from the environment exist. Taking a test, and adding the results to the current state description allows a more accurate mapping. The initial states would be transformed into 'worker skilled and test was good', 'worker not skilled and test was good', 'worker skilled and test was bad', 'worker not skilled and test result was bad'. Such additional information enables the agent to estimate, after gaining some experience, the chance a worker with a good or bad test result belongs to the group of skilled workers or not.

The BRA algorithm formalises a similar idea as Simon, but simplifies it further. The important difference is that state and transition probabilities are not represented explicitly. The agents considered here are simpler, act more subconsciously and choose their actions probabilistically based on the average reward. The average reward of an action reflects the desirability of that action in a given state, and can therefore be interpreted as a preference for an expected outcome. In the hiring example, this means that the decision maker does not calculate a transition probability matrix based on his alternatives, but learns this relationship implicitly by experiencing a higher reward for hiring workers with a good test result and translating this into a higher selection probability of the hiring action.

3 The Algorithm

The basic idea of BRA is to build an internal, flexible model of the environment the agent lives in. The environment is accessible by the input state $s \in S$

defining the current 'situation' the agent is in, where S denotes the input dimensions. The input state is matched with an internal symbolic representation $C_i \subseteq C : \{c_1 \dots c_n\}$ of that state. The agent then chooses an action according to the general form $r_i : C_i \rightarrow A_{r_i} \in D_{C_i}$. A_{r_i} is the action set available in the specified condition, C is the set of all possible conditions that can be generated from S, and C_i is a collection of conditions derived from C. The action set is fixed, closed, and defined within a certain domain of actions D_{C_i} applicable in the state C_i. The alternatives are assumed to be always comparable and complete.

Two main components are distinguished: RL (section 3.1) is responsible for experience-based learning, state space partitioning (section 3.2) for rule extraction and the building of the internal model structure. Section 3.3 summarises the algorithm as pseudo-code.

3.1 Reinforcement Learning

RL (e.g. Sutton and Barto 1998) is used to implement the dynamic aspect of knowledge generation in the model. Agents learn by trial and error which action to apply in a given state. Successful actions are rewarded. Actions which yield a higher reward are selected with a high probability in the future, whereas bad actions, receiving a lower reward, are selected less often. The history of these reinforcements is summarised as action strength q. Whenever an action a has been applied, the strength is updated with the reward $p(t)$ observed for that action by the following equation:

$$q(a_t) = q(a_{t-1}) + \gamma(p(t) - q(a_{t-1})) \tag{1}$$

This action-value function updates the strength of the current action based on the weight γ of previous experiences and the current reward p. For a value $\gamma = 0.5$, for example, and reasonably large t this function approximates the average payoff generated with action a. The smaller γ, the stronger the impact of past experiences; conversely, for $\gamma = 1$ only the reward of the last action is considered, and all previous experiences discarded.

In the next step, the action probability is calculated according to the selection function:

$$pr(a_{i,t+1}) = \frac{e^{q(a_i)\alpha}}{\sum_{j,j\neq i} e^{q(a_j)\alpha}} \tag{2}$$

This selection function, also known softmax policy in RL, determines each action's selection probability depending on its own strength relative to the strengths of the alternative actions. The parameter α, $0 < \alpha < 1$, is a learning parameter that determines the rate of exploration. The larger α the less the influence of the action strength on the selection probability.

3.2 State space partitioning

Learning by doing as described above happens for a given state s. This section describes how states are represented and perceived in the agent's internal world model.

Representation and search paths The state s is represented internally as a collection of attributes $\{att_1 \dots att_n\}$. Each attribute can have a number of possible values, for example nominal values such as 'low' or 'high', or numerical ranges, e.g. 0-1000. Attributes are connected by simple predicate logic. For example the predicate '(profit = low *or* profit = medium *or* profit = high) *and* (sales $0 <$ sales < 1000)' could describe the situation of a firm in the dimensions profit and sales. This representation is called a 'state descriptor', and denoted with C_i. To each state descriptor a set of actions is bound from which the action policy for this state can be learnt. This binding constitutes formally the mapping $r_i : C_i \rightarrow A_{r_i}$.

The agent starts with a model covering all possible states. This initial state entails all attributes with their value spaces, thus the coarsest representation possible. In consecutive time steps, specialisations are developed stepwise by the application of a heuristic search method. For this, the space of state descriptions is represented as a tree, where nodes at higher levels contain coarser, and nodes at deeper level of the tree finer mappings. Finer grained descriptions are 'expanded' from the predicates at higher levels. Which descriptions are expanded depends on a heuristic evaluation function, which here is the agent's utility. The task of the search process is thus to find the level of detail that describes the environment in such a way that generates the highest welfare for the agent.

State expansion mechanism Before the internal model is updated, the agent acts in its environment over a period μ. During this period, the value of existing mappings $R = \{r_1 \dots r_n\}$ is updated using feedback from the environment. After each μ steps, the state expansion mechanism is applied: First the node r_{expand} with the highest value on the search path is selected. If the search path is empty, the root node is selected. From there, the next level of the tree is expanded by partitioning the value spaces of the attributes constituting the conditions of r_{expand}. For attributes having discrete values, one value is picked randomly. Attribute values representing numeric ranges are split in half. For each partitioned attribute a new condition is created containing the partitioned attribute values or value range, and the remaining original attribute values (i.e. the number of successor nodes equals the number of attributes $\times$ 2 in the original condition). The conjunction of the predicates of the resulting level (after reduction) is equivalent to the expression of the parent node. By mapping A to each newly created condition set the new mappings R' are generated. The path from each $r' \in R'$ up to the root node is added to the search path (without duplicates). The conjunction of state descriptions with no children in the search path is then equivalent to the initial state description. The RL mechanism selects actions only from the matching descriptor in the search path, so that there is always exactly one state description activated and one action selected at a time.

For example, going back to the firm example above, of the initial, exhaustive description $C_{initial}$: (profit = low *or* profit = medium *or* profit = high) *and* ($0 <$ sales < 1000) the attribute profit is selected, and of its value range the value 'high'. The value space of the attribute is then divided into the expression 'profit=low or profit=medium' and 'profit=high', respectively. The resulting specialised state descriptions are C_1 : (profit =low *or* profit = medium) *and* ($0 <$ sales < 1000) and C_2 : (profit = high) *and* ($0 <$ sales < 1000), respectively. Analogously, the sales attribute is split in two intervals and two successor descriptors generated, so that four successor descriptors are created.

Model specialisation and generalisation With the state expansion mechanism it is possible to specialise the conditions in the state-action space in many ways. A heuristic evaluation function determines the direction of this process. The process proceeds as follows: First, the value of a state at time t is calculated as

$$v(r,t) = v(r,t-1) + \frac{1}{2}(q(a_t) - v(r,t-1)) \tag{3}$$

where $q(a_t)$ is the reward of the executed action in the state described by r. The function approximates an average of the state description value.

Next, before an expansion happens, some constraints have to be satisfied: A parameter χ limits the maximum number of nodes the tree can have, i.e. the maximum number of situations the agent can differentiate. New states can only be evolved at the cost of 'forgetting' other state descriptions (see below for deletion). Due to the possible deletion of nodes it is possible that state descriptions that were deleted are expanded again and endless cycles of generalisation and specialisation occur. The right balance has to be found depending on the stability of the environment; preventing many visits of identical descriptions too early can be harmful if the environment changes; on the other hand it binds valuable resources in the agent's mental processing. To tune this balance, a function with a cost parameter $\zeta, 0 < \zeta \leq 1$ is used to compute a value determining whether the successor description should be developed or not: The better a state descriptor compared with the average performance (measured by the average reward g) and the smaller ζ, the more recurrent expansions beginning from that state descriptor are allowed (equation 4).

$$expand(r) = \begin{cases} true, & \text{if } expansions(r) = 0 \text{ or} \\ & \zeta \times expansions(r) \times g < v(r,t) \\ false, & \text{otherwise} \end{cases} \tag{4}$$

A state description might lead to a good solution strategy, but if only rarely visited is only of limited value (they only use up scarce memory space and processing capabilities). Therefore the heuristic h accounts for the frequency of state activations:

$$h(r,t) = v(r,t)\frac{activations(r)}{t} \tag{5}$$

The search process selects the node with the maximal heuristic $h(r, t)$ in the search path, if the *expand* condition is satisfied.

Before new state descriptions are developed after μ steps, the descriptions of the current level may be deleted if they did not outperform the value of their parent states (performance could be, e.g., the average of the state description values). The parent node becomes thereby the most specialised descriptor again. Analogously to rule specialisation, this generalisation process sets in after a certain time $\nu, \nu < \mu$.

The process stops if all possible descriptions have been expanded (e.g. because χ has been reached) and the environment is reasonably stable so that deletions of once developed specialised state descriptions do not occur frequently.

In principle, this mechanism tests the usefulness of different representations of the state space. Useful partitions are kept and possibly refined further, useless partitions are discarded and might be forgotten over time. The agent generates and tests cognitive alternatives, and depending on experience, decides what to know in the sense of (Rubinstein 1998) mentioned above.

Avoiding local search optima The mechanism may end up after a number of expansions at a level of the tree with a particular configuration of descriptors in the search path. There is no back-propagation of values, e.g. an update of the successor states with a discounted value of the current state, so that more general descriptions higher up in the tree or in other partially developed paths can have higher, although outdated values. If such higher historical values exist, they are used as a hypothesis that the current search path has become suboptimal due to a changed environment, and that different paths should be explored. The process may therefore switch with probability $\rho, 0 \leq \rho \leq 1$ to a different, higher valued node in the tree and continue the expansion from there. The path to this node becomes thereby the search path.

3.3 The complete algorithm

Table 1 summarises the main parameters and notation, then the algorithm is listed in pseudo-code.

Table 1: Summary of notation

Name	Description	Value range
γ	discount parameter for reward	$0 \dots 1$
ν	interval at which underperforming rules can be deleted	$0 \dots \mu$
μ	interval at which new rules can be generated	$0 \dots \infty$
ζ	cost parameter determining the frequency of re-exploring already visited paths	$0 \dots 1$
χ	maximum number of nodes	$1 \dots \infty$
p	payoff (reward)	$0 \dots \infty$
g_t	average payoff (reward) until time t	$0 \dots \infty$
A	action set of actions a	
q_{a_t}	strength of action $a \in A$	$0 \dots \infty$
pr_{a_i}	action selection probability of action a_i	$0 \dots 1$
C_i	conditions that can be generated from input dimensions S	
r_i	A state-action mapping $C_i \rightarrow A$	
$v(r,t)$	The state value function	
$h(r,t) = f(v(r,t))$	The heuristic selection function	

```
{Setup and Initialisation}

Define the time discount for action updates $\gamma$
Define the update-cycle $\mu$
Define the delete-cycle $\nu, \nu < \mu$
Define the cost of expansion $\zeta$
Define the maximum number of state descriptions $\chi$
Define the probability of switching the search path $\rho$
Define the search path $search\_path$ as a subset of $R$
Define $expansions(r)$ as a function counting the number of expansions from $r$
Define $activations(r)$ as a function counting the times $r$ matched a state
Define $parent(n)$ as the parent of a node $n$ in the state-tree $T(R)$
Define $children(n)$ as a function returning all children of a node $n$ in $T(R)$
Define $uniform(x \dots y)$ as a uniform random distribution in the interval $x \dots y$

$q(a) = 0, \forall a \in A$
$C_1 \leftarrow S$
$search\_path \leftarrow r_1 : \{C_1 \rightarrow A\}$

repeat

  {Reinforcement learning}

  observe reward $p(t-1)$ received after executing $a_{t-1}$
```

```
$g_t = g_{t-1} + \frac{1}{2}(p(t) - g_{t-1})$
$q(a_t) \leftarrow q(a_{t-1}) + \gamma(p(t) - q(a_{t-1}))$

$v(r,t) \leftarrow v(r,t-1) + \frac{1}{2}(q(a_t) - v(r,t-1))$

$activations(r) \leftarrow activations(r) + 1$

compute situation $s \hookleftarrow S$

find the most specific mapping $r_a \in search\_path$ matching $s$

$pr_{a_{i,t+1}} \leftarrow \frac{e^{q_{a_i} * \alpha}}{\sum_{j,j \neq i} e^{q_{a_j} * \alpha}}, \forall a \in A_{r_a}$

select action $a_t$ from the resulting distribution and execute $a_t$

{State space partitioning}

{Expand}
**if** rest($\frac{t}{\mu}$) $= 0$ and $|R| < \chi$ **then**
  $r_{expand} \leftarrow \max h(r,t), \forall r \in search\_path$
  **if** $\zeta \times g_t \times expansions(r_{expand}) < v(r_{expand}))$ **then**
    partition $r_{expand}$ according to expansion mechanism into $R' \leftarrow \{r'_0 \ldots r'_n\}$
    initialise the value of the new states with $v(r_{expand,t})$
    append $R'$ as children of $r_{expand}$
    add $R'$ to $search\_path$
    $expansions(r_{expand}) \leftarrow expansions(r_{expand}) + 1$
  **end if**
**end if**

{Delete obsolete mappings}
**if** rest($\frac{t}{\nu}$) $= 0$ **then**
  {determine the most recent expanded mapping $r_{expanded}$ and its children $CH$}
  $CH \leftarrow \{ch_1, \ldots ch_n\} \subset search\_path$, $children(ch_i) = \emptyset$
  $r_{expanded} \leftarrow parent(ch_i)$
  **if** $v(r_{expanded}, t) > \frac{1}{|CH|} \sum_{i=0}^{|CH|} v(ch_i)$ **then**
    delete $CH$
  **end if**
**end if**

{Avoid local search optima}
**if** $uniform(0,1) > \rho$ **then**
  clear $search\_path$
```

```
        $r_{max} \leftarrow \max v(r,t), \forall r \in R$, $children(r = \emptyset)$
        add the path from $r_1$ to $r_{max}$ to $search\_path$
    **end if**

**until** end of simulation
```

4 Implications

The algorithm helps artificial agents to structure perceptional information from the environment into useful knowledge bits, comparable to state-space partitioning approaches from machine learning (e.g. Uther and Veloso 1998; Lau and Lee 2004). Its main function is to distinguish between important and unimportant features of the environment.

To keep computational complexity within bounds, BRA builds on certain assumptions and limitations. Thus, the action set is given; no action sequences or complicated plans can be computed; it is abstracted from any domain-relevant knowledge. The algorithm as such combines knowledge and action elements in a systematic, heuristic search process and finds strategies, it does not generate them.

A major difference of the BRA approach from decision theoretic approaches is its learning component. It allows to start from a state of ignorance, where the desirability of actions is unknown, to a state of uncertainty where the action weights indicate the likely result of an actions with respect to own utility, to a state of certainty if the environment is deterministic. The last case is given if action selection probabilities become close to 1. Only under such conditions of absolute certainty and provided large enough differences in rewards, rational maximising behaviour can be approximated.

The state space partitioning procedure has been specified to operate on crisp sets. As a consequence, more realistic or vague representations are not possible, although this might be desirable in certain circumstances. For example, consider the partitioning rule that splits numerical intervals evenly into two exclusive intervals (section 3.2). A fictive profit interval of a firm, $0 < profit < 1000$, could thus be split into low and high ranges by creating two new intervals $0 < profit \leq 500$ and $500 < profit < 1000$. For low profit ranges probably a different strategy will learnt as in the high range (for example successive lowering of prices until higher turnover is achieved). For certain raw input variables, e.g. $300 < input(profit) < 700$, it is however difficult to judge whether it is better to the apply the low or high profit strategy. Fuzzy set theory and fuzzy logic (e.g. Klir and Folger 1988) treats such uncertainty relations formally. Fuzzy logic provides some measures that specify the degree to which set or expression an element actually belongs, thus avoiding arbitrary matching if the boundaries of a concept are unclear. In the engineering literature about fuzzy controllers (e.g. Passino and Yurkovich 1998), approaches have been developed that combine such fuzzy inference systems with rule-based action systems (e.g. Takagi and Sugeno 1985) and Q-Learning (e.g. Jouffe 1998; Fuzzy Q-Learning). In fuzzy controllers the conditions of production rules are represented as fuzzy concepts. The input state is matched with a candidate set of rules and a distribution of membership functions calculated that describe the degree of applicability of these rules. In Fuzzy Q-Learning, the rule to be executed is typically determined in a two-stage RL procedure in which first the action within a rule and then across candidate rules is determined. Fuzzy concepts are obviously a natural extension of the state representation in BRA. However, BRA cannot handle intersecting sets of conditions, because in the case of two matching conditions it is undefined which rule is activated. Integrating Fuzzy Q-Learning requires some extensions: First additional state-expansion rules that generate predicates with intersecting value ranges must be defined; then conflict resolution strategies have to be defined that choose a rule with respect to their applicability according to some fuzzy truth value.

5 Related approaches

To put the BRA approach into context, this section describes two characteristic similar algorithmic approaches to learning, and compares them with BRA.

CLARION The cognitive architecture CLARION (e.g. Sun and Slusarz 2005) was designed to capture implicit and explicit learning processes in humans. The main assumption is that there are two different levels of learning: A subsymbolic level and a more explicit, declarative level. The subsymbolic, or 'bottom' level represents low-skill, often repetitive tasks where learning proceeds in a trial-and-error fashion. Knowledge on this level is typically not accessible and difficult to express with language. On the symbolic, or 'top' level, knowledge is directly accessible and can be expressed with language. This level typically represents more complex knowledge. It can be acquired by experience, but also by means of explicit teaching.

The input state is made up of a number of dimensions, and each dimension specifies a number of possible value or value ranges. Action selection takes place using RL in the bottom level, or by firing production rules on the top level. Which level is used is determined stochastically. After the action was performed, top and bottom levels are updated with the feedback received from the environment.

At the bottom level, the RL mechanism is implemented with a neural net. The input layer is constituted of the values of the input state. Three intermediate layers are used to compute Q-values (allowing memory of action sequences), while the fourth layer chooses an action according to standard RL.

At the top level, the rules conditions are constructed out of the input dimensions, their consequents from actions available to the agent. The rules are, for compliance with the bottom level, implemented as network. Rule extraction, specialisation and generalisation is determined by feedback from the bottom level: If there is no rule matching the current state and the action was successful according to some performance criterion, a new rule is created with the current state as the condition, and the performed bottom level action as consequent. Existing successful rules matching the current input state are generalised by adding another element of the input values to the condition. More specialised rules that are now covered are deactivated. Unsuccessful rules are likely to be too general rules and are specialised by removing elements from the condition. Deactivated rules are reactivated if the specialised rule does not cover them any more. An information gain measure that estimates the performance of rules under different conditions serves as the success criterion.

Learning Classifier Systems Also Learning Classifier Systems (LCS) aim at the extraction of rules. The basic idea is to start with a set of initial rules (classifiers) and to evolve this set over time by application of mechanisms for modification, deletion and addition of new rules. Whereas earlier LCS (e.g. Holland 1975) relied mostly on the Genetic Algorithms paradigm newer versions have more in common with RL approaches and therefore have also been described as generalised RL (Sigaud and Wilson 2007).

An LCS consists of a population of classifiers. A classifier contains a condition part, an action part, and an estimation of the expected reward. Typically, the condition part consists of the three basic tests 0 (property does not exist), 1 (property exists) and #. # represents a generalisation and stands for both 0 or 1. A classifier has one action as a consequent, but typically several classifiers match a condition in the environment and hence compete with each other. The action to be executed is the selected according to some RL mechanism (e.g. the ϵ-greedy policy).

Many LCS use a genetic algorithm to create new rules by selecting and recombine the fittest classifiers from the population (where fitness is, e.g., the expected reward received from the environment). A covering operator is called whenever the set of matching classifiers is empty. The operator adds a classifier matching the current situation with a randomly chosen action to the population. Sophisticated systems may limit the population size, and add corresponding eviction and generalisation procedures.

Newer approaches have more in common with RL and even symbolic approaches to learning. The FOXCS system (Mellor 2008), e.g., uses a symbolic representation of rules. Rule mutation, covering and deletion is based on logical operations using the Prolog logic programming language. Anticipation-based classifier systems (e.g. Butz 2002; ACS) extend the classifier representation with the description of the next state and build a model of transitions. A specialisation mechanism is applied when the classifier oscillates between correct and incorrect predictions, indicating that a splitting of the condition might improve the match. Generalisation is based on complex algorithms that estimate whether generalisation will result in an improvement (see also Sigaud and Wilson (2007) for an overview of LCS).

Comparison RL is the most important aspect for generating action-centred knowledge in the related approaches as well as for the Bounded Rationality

Algorithm. Differences exist in the way such knowledge is used to build internal models of the environment:

- BRA does not start with a psychological model of skill acquisition as CLARION or no explicit model at all as machine learning, but a sociopsychological model of bounded rationality.
- The BRA uses a pure symbolic representation of conditions with simple first order predicate logic. CLARION has to transform them in a network structure, many LCS use binary strings.
- CLARION modifies rules only after evaluation of bottom level actions; ACS compares prediction errors. BRA is much less sophisticated here, using a simple heuristic generate-and-test procedure to decide whether a rule should be specialised or generalised. If the test phase fails (possibly only after a long time when the environment changes), the generated rule is deleted again. CLARION and more sophisticated LCS keep detailed statistics and perform complex estimations to decide about generalisation and specialisation of specific rules.
- BRA starts with a state description covering all possible states and builds a model by searching heuristically through the space of possible descriptions that can be expanded logically from the initial descriptor. In CLARION and most LCS, new state descriptions are generated when they are encountered. it is not necessary to describe the state space fully. The exploration of the state space in BRA is more 'structured'. But BRA is also more likely to not discover useful descriptions - if the best differentiations are very fine-grained, the chance that the process stops developing new states is much higher because a path might (wrongly) appear as not improving to the agent. That is, in BRA a conflict between efficiency and the cost of cognitive capabilities exists (intended by design - see section 2) that may produce different and, compared to pure machine learning, suboptimal results.

6 An example - statistical discrimination

The concept of statistical discrimination is based on the principal-agent decision problem and was first introduced by Arrow (Arrow 1973). In the example simulation experiment the principal-agent problem is represented by the relationship between employers and workers. Workers decide whether to invest or not invest into skills, but these investments are not fully observable by the employers. Employers have to rely on signals like an employment test result or an interview to estimate productivity. In the absence of clear results, they might however use physical markers like gender or race as an indicator. Statistical discrimination exists when an employer is reluctant to hire a certain type of worker because he expects a low productivity of that type. This equilibrium can be self-reinforcing if the workers of the discriminated type come to the conclusion that it is not worth investing because they do not expect to get hired.

The simulation is based on a classroom experiment reported by Fryer et al (2005). They find emergence and persistence of statistical discrimination in some experiments they conducted, whilst in others discrimination did not evolve. The hypothesis of this experiment is that if BRA is a *valid* model of decision making, some simulations must exhibit similar patterns of statistical discrimination as observed by experimental game theorists.

In the model there are 10 worker agents and 5 employer agents. Workers are assigned the colours green and purple randomly with equal probability. Each round, workers and employers are paired randomly, and employers must decide to hire or not to hire a worker depending on the result of the employment test and the colour of the worker. Skill investment costs are drawn randomly and reported to the workers, i.e. the investment costs in the long run are the same for each colour. The investment cost varies between 0 and 0.1. If no investment is made, no cost is incurred. The test outcome might be either good (R) or bad (B). The probability of a good test result is 0.2 if no investment was made, and 0.5 if a investment was made. Two draws are made, so that if the result is RR or BB employers can infer with a relatively high certainty the productivity of the worker: Because the probability that a worker did not invest is more than twice as high if the outcome is BB, it is very likely that the worker most likely did not invest. Conversely, an RR result is a strong hint that the worker invested; however, some uncertainty remains. In the event of RB (or BR), there is the highest uncertainty. It is therefore expected that in this condition colour might be used as a cue to aid decision making.

The payoffs are as follows: If a worker is hired he receives of 0.3, if he was no hired 0.15. His payoff is 0.3 minus the investment cost (if there was any), and 0.15 minus the investment cost, respectively. Employers receive a payoff of 0.2 if the worker was not hired, 0.4 if a worker who invested was hired, and 0 if a worker who did not invest was hired.

This model setup is with minor variations identical to the original setup in Fryer et al (2005). One difference is the magnitude in rewards, which was divided by 10 for this experiment, and the distribution of players. Furthermore, in the original game there were as many employers as workers, and workers were split exactly half in green, and half in purple. In the simulation, workers' groups are partitioned randomly, so that one worker group is often larger than the other. Also, there are only half as much employers as workers. The reason is to support learning: The smaller the worker group, the more likely similar behaviour simply by chance and thus it is 'easier' to discriminate. Similarly, with fewer employers variation may decrease by chance, and feed back into worker decisions.

The agents are implemented as follows: Workers have a simple mapping with an empty state description and actions invest/not-invest as action set. Thus the choice is solely based on the payoff received (simple RL). Employers on the other hand may use the different test outcomes and colours of the agents explicitly to construct rules according to BRA. The action set consists of hire/not-hire. Rules may then be generated that differentiate between colour only, between colour and test result, or only between test results. If the majority of generated rules is based on colour only, statistical discrimination is clearly observable; if rules are only based mainly on test outcomes there is meritocracy. More precisely, three different sets of state-action mappings are specified. The first set contains only one initial rule $r_1 : C_1 \to A_{r_1}$ with C_1 : (test-result = BB) *and* (colour = purple *or* colour = green), the second set contains $r_2 : C_2 \to A_{r_2}$ with C_2 : (test-result = RR) *and* (colour = purple *or* colour = green), and the third set is given by $r_3 : C_3 \to A_{r_3}$ with the C_3 : (test-result = BR) *and* (colour = purple *or* colour = green). This specification in principle pre-wires some knowledge about the relationship between test-result and productivity by restricting the possible combinations. Per mapping, only two rules can be generated, limiting the maximum possible rules to six. A different possibility would be to not specify exclusive sets and let agents learn from an initial rule $r_0 : C_0 \to A_{r_0}$ with C_0 : (test-result = BB or test-result=BR or test-result=RR) *and* (colour = purple *or* colour = green). This would increase the possible combinations to twelve and make learning much more difficult. Experimenting with such a specification confirmed this. Furthermore, such a setup would mean that agents have to learn facts and relationships that were common knowledge in the classroom experiment (e.g. the higher probability that a worker invested when the test-outcome was RR).

The expansion parameters are fixed with $\zeta = 0.05$, $\rho = 0.3$, $\mu = 100$, $\nu = 75$, $\chi = 100$. The learning parameters α and γ, i.e. learning rate and reward discount are varied both at the employer and worker side. Note that ζ is set to a small value to allow employer agents to re-evaluate their rules frequently. The hypothesis here is that due to the reciprocal behaviour of agents revaluations of extracted rules are necessary to adapt to the simulation dynamics easier and more often.

Fryer et al (2005) report that many experiments had very similar hiring rates, while there were some where strong discrimination was observed. The result shown in their paper exhibits a pattern where discrimination develops quickly and persists over the number of time steps played (about 20). For example, the hiring rate of green players quickly jumped and remained at 0.8, even after they slightly decreased investment activity; purple workers however would invest less, and consequently get hired less often at a level of about 0.4. The classroom discussion showed that employers started with a belief that purple workers did not invest. When they were not sure whether the purple player invested or not, they would rather not hire them. Because chances for purple workers to get hired decreased, they stopped investing, by this reinforcing employers' beliefs.

Using experimentation and a genetic algorithm optimal values for the α and γ values were searched for. As a simple fitness indicator, the average difference between hiring levels of green and purple workers over the whole simulation is computed. Using the Fryer et al (2005) experiment as a benchmark, those simulations were closer looked at that developed a similarly high, persistent discrimination. Figure 1 gives an impression how different parameter settings affect the outcome. In general, it seems that settings with low learning param-

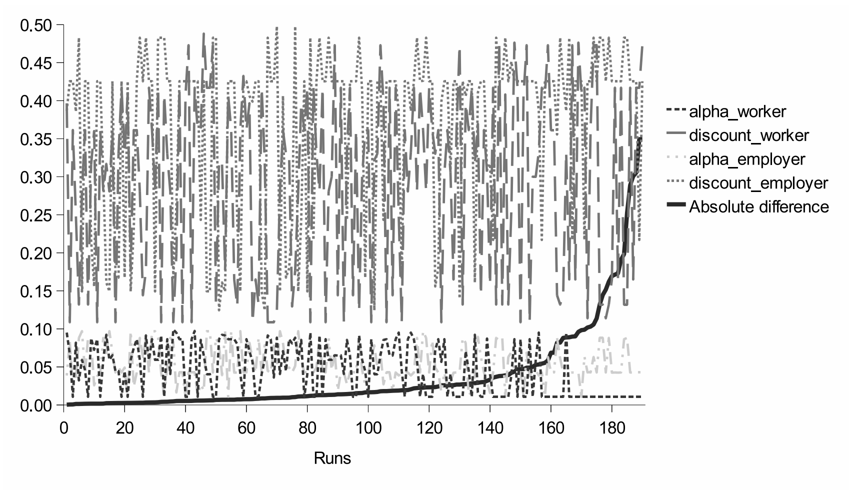

Figure 1: Difference in hiring hiring rates for various parameter settings, sorted by absolute difference α_{worker}, γ_{worker}, $\alpha_{employer,r_1}$, $\alpha_{employer,r_2}$, $\alpha_{employer,r_3}$, $\gamma_{employer,r_1}$, $\gamma_{employer,r_2}$, $\gamma_{employer,r_3}$.

eters for workers (early lock-in into investment/non-investment behaviour) had the best chances to produce the largest differences between groups. The main influence on fit is workers' α. The smaller this value, the more likely that discrimination occurs. The likely reason for this behaviour is that with relatively stable worker behaviour, employers have enough time to adjust their strategies. The more unpredictable worker behaviour the harder to respond, and thus less likely that an equilibrium emerges.

Figures 2 and 3 show one of the simulations where discrimination emerged clearly. Hiring and investment rates were measured every 10 timesteps, and are given as moving averages over 10 measurement points, i.e. over 100 steps. As figure 2 shows, the employment level is very high for purple workers and low for green workers. This pattern is very similar to the classroom experiment. Moreover, it seems that this equilibrium state can collapse very quickly for no or only very little changes in investment behaviour (see figure 3). The latter result only occurs in the simulation, neither did it in the classroom experiment, nor would one expect discrimination to change so drastically in reality.

It can be concluded that r_1, r_2 and their successors reinforce colour-based decisions even though a BB test-outcome points to a higher probability of a non-investing worker than an RR outcome. For example, C_1 might have been expanded into C_{11} : (colour = purple) *and* (test-result = BB) choosing action 'not-hire' with small probability, and C_{12} : (colour = green) *and* (test-result = BB), choosing 'not-hire' with much larger probability. A likely explanation is that hiring decisions in the event of the more ambiguous test-result BR favoured purple workers but not green workers. This reinforces the purple workers' investment behaviour which in turn supports discrimination even when test-results

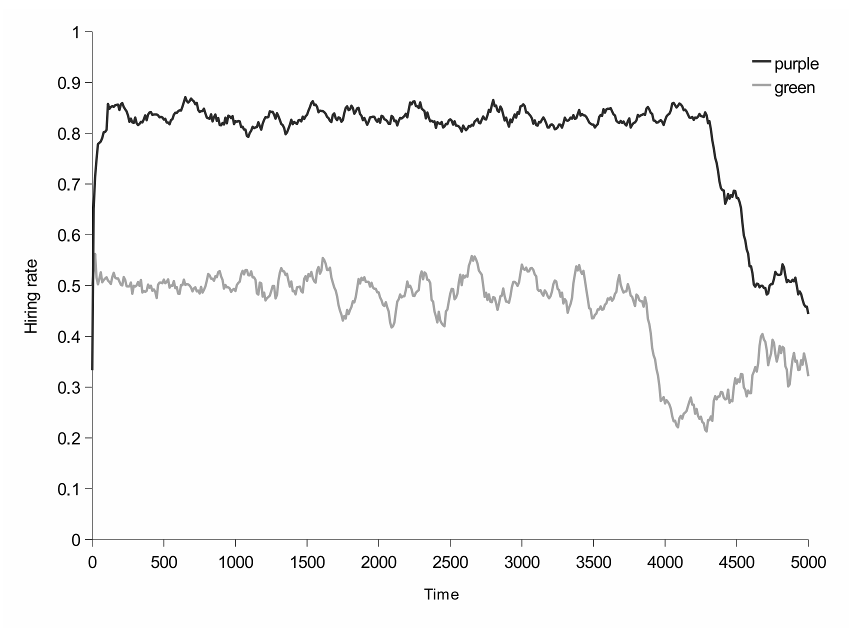

Figure 2: Hiring rates (6 purple workers, 4 green workers). The relevant parameters are $\alpha_{worker} = 0.015$, $\gamma_{worker} = 0.3$, $\alpha_{employer,r_1} = 0.05$, $\alpha_{employer,r_2} = 0.05$, $\alpha_{employer,r_3} = 0.08$, $\gamma_{employer,r_1} = 0.1$, $\gamma_{employer,r_2} = 0.1$, $\gamma_{employer,r_3} = 0.21$

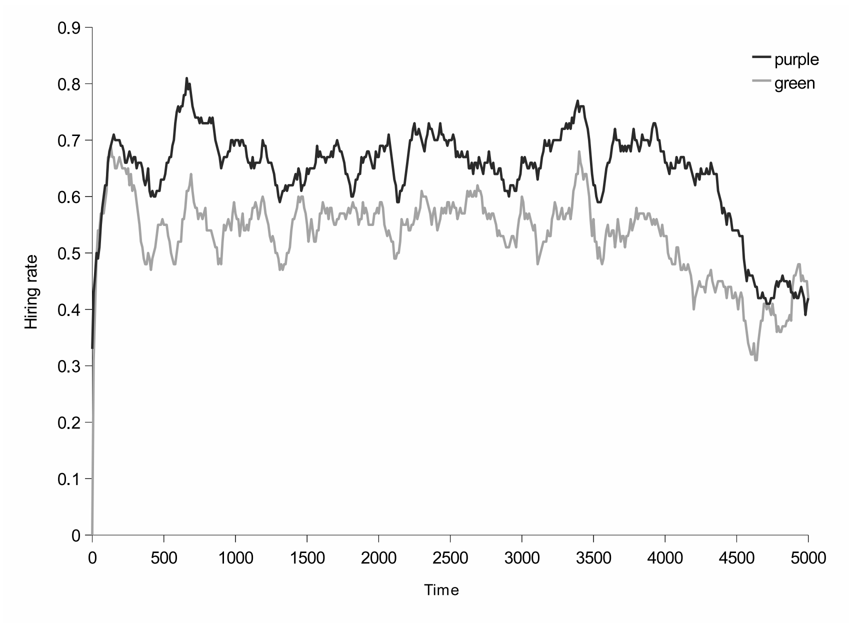

Figure 3: Skill investment rates (6 purple workers, 4 green workers). The relevant parameters are $\alpha_{worker} = 0.015$, $\gamma_{worker} = 0.3$, $\alpha_{employer,r_0} = 0.05$, $\alpha_{employer,r_1} = 0.05$, $\alpha_{employer,r_2} = 0.08$, $\gamma_{employer,r_0} = 0.1$, $\gamma_{employer,r_1} = 0.1$, $\gamma_{employer,r_2} = 0.21$

are less ambiguous. An investment that is only slightly higher may thus lead to a very high employment rate for purple workers (about 0.7 averaged over the whole simulation), and conversely to low employment rates for green workers. This confirms the hypothesis that discrimination can be a very persistent feature, even if there are no or very limited facts supporting this behaviour. Also, the author conducted experiments where learning only happened in the BR case (in case of RR agents always hire, in case of BB never hire). The main result is that the differences between colours was typically lower, supporting the hypothesis that discrimination does not alone happen because of uncertainty, but because of the beliefs that emerged during the simulation. However, to fully comply with the empirical example, one would expect much lower investment rates for green players because of the negative feedback from the employer side. This is obviously also the reason why discrimination can disappear quickly in the simulation: Whereas the drop in investment rates after time-step 3500 occurs for both worker groups, employers react in the beginning only by hiring less green workers which they are already reluctant to hire. Since also the purple workers invest less, this perturbation in employers' rewards finally causes employers to re-evaluate their strategies and treat both colours more similar, roughly corresponding to the actual difference in investment rates.

7 Conclusion

In this paper, an algorithm that replicates simple decision process of simply structured, bounded rational actors has been described, formalised and demonstrated. The approach is of a pragmatic nature and motivated by the application of machine learning techniques in ACE models. As such it adds to RL based systems in the agent-based modelling field and combines elements already used by learning architectures such as CLARION and LCS. BRA is different from these approaches as it is less general than a cognitive architecture and explicitly built upon a simple sociopsychological approach to learning.

The motivation of the example simulations was to assess the usability of the algorithm from the perspective of verification as well as validation. A model of statistical discrimination was chosen for this purpose. The results showed that BRA agents

- learn to differentiate between different behaviours
- are able to this in quite dynamic environments
- are thereby able to replicate to a large - but not full - extent patterns that have been observed in behavioural game theory

Only if agents have enough pre-wired knowledge about the game, similar results as in reality can emerge. If BRA is used as a black box to learn every aspect of the game from scratch it will fail, because not particular attention was paid to the problem of combinatorial explosion (a weakness of many reinforcement learning approaches in general).

Several limitations of the approach have been indicated, and will be explored in future work. Important topics include: The fuzzification of state descriptions to prevent wrong classifications due to crisp set boundaries; the dynamic addition and deletion of new input dimensions to allow for a more dynamic evolution of the state space; better handling of combinatorial explosion making simulations results less dependent on the initial setup of the rule system.

References

Arrow K (1973) The theory of discrimination. In: Ashenfelter O, Rees A (eds) Discrimination in Labor markets, Princeton University Press, Princeton, NJ

Brenner T (2006) Agent learning representation: Advice on modelling economic learning. In: Tesfatsion L, Judd K (eds) Handbook of Computational Economics, 2, Elsevier, chap 18, pp 896–942

Butz M (2002) An algorithmic description of acs2. In: Lanzi P, Stolzmann W, Wilson S (eds) Advances in learning classifier systems, Lecture Notes in Artificial Intelligence, Springer, Berlin, vol 2321, pp 211–229

Camerer C, Ho T (1999) Experienced-weighted attraction learning in normal form games. Econometrica 67(4)

Erev I, Roth A (1998) Predicting how people play games: reinforcement learning in experimental games with unique mixed-strategy equilibria. American Economic Review (88)

Fryer R, Goeree J, Holt C (2005) Experience-based discrimination: Classroom games. Journal of Economic Education 36(2):160–170

Holland J (1975) Adaptation in natural and artificial systems: An introductory analysis with application to biology, control, artificial intelligence. University of Michigan Press, Ann Arbor

Jouffe L (1998) Fuzzy inference systems learning by reinforcement methods. IEEE Transactions On Systems, Man and Cybernetics-Part C, Applications and Reviews 28(3):338–355

Kahnemann D, Tversky A (1979) Prospect theory: An analysis of decision under risk. Econometrica 27

Kirman A, Vriend N (2001) Evolving market structure: An ace model of price dispersion and loyalty. Journal of Economic Dynamics & Control 25:459–502

Klir G, Folger T (1988) Fuzzy Sets, Uncertainty, and Information. Prentice Hall, Englewood Cliffs, N.J.

Lau HY, Lee IS (2004) Adaptive state space partitioning for reinforcement learning. Engineering Applications of Artificial Intelligence 17:577–588

LeBaron B, Arthur W, Palmer R (1999) Time series properties of an artificial stock market. Journal of Economic Dynamics & Control 23:1487–1516

Markham A (1999) Knowledge Representation. Lawrence Erlbaum Associates, Mawah NJ

Mellor D (2008) A learning classifier system approach to relational reinforcement learning. In: Bacardit J, Bernad-Mansilla, E, Butz M, Kovacs T, Llor X, Takadama K (eds) Learning Classifier Systems (LNAI), Springer, vol 4998, pp 169–188

Passino KM, Yurkovich S (1998) Fuzzy Control. Addison Wesley

Roth A, Erev I (1995) Learning in extensive form games: Experimental data and simple dynamic models in the intermediate run. Games and Economic Behaviour 6

Rubinstein A (1998) Modeling Bounded Rationality. MIT Press

Sigaud O, Wilson S (2007) Learning classifier systems: A survey. Soft Computing 11:1065–1078

Simon H (1956a) A behavioural model of rational choice. In: Simon H (ed) Models of man, social and rational: mathematical essays on rational human behavior in a social setting, Wiley, New York

Simon H (1956b) Rational choice and the structure of the environment. In: Simon H (ed) Models of man, social and rational: mathematical essays on rational human behavior in a social setting, Wiley, New York

Sun R, Naveh I (2007) Social institution, cognition, and survival: A cognitive-social simulation. Mind and Society 6(2):15–42

Sun R, Slusarz P (2005) The interaction of the explicit and the implicit in skill learning: A dual-process approach. Psychological Review 112(1):159–192

Sutton R, Barto A (1998) Reinforcement Learning: An Introduction. MIT Press, Cambridge, MA

Takagi T, Sugeno M (1985) Fuzzy identification of systems and its applications to modeling and control. IEEE Transactions on Systems, Man, and Cybernetics 15:116–132

Uther WTB, Veloso MM (1998) Tree based discretization for continuous state space reinforcement learning. In: Proceedings of the fifteenth national conference on Artificial intelligence, American Association for Artificial Intelligence, pp 769–774