Comput Math Organ Theory (2007) 13: 355–377
DOI 10.1007/s10588-007-9016-9

Simulating complex social behaviour with the genetic action tree kernel

Thorsten Chmura · Johannes Kaiser · Thomas Pitz

Published online: 30 May 2007

Abstract The concept of genetic action trees combines action trees with genetic algorithms. In this paper, we create a multi-agent simulation on the base of this concept and provide the interested reader with a software package to apply genetic action trees in a multi-agent simulation to simulate complex social behaviour. An example model is introduced to conduct a feasibility study with the described method. We find that our library can be used to simulate the behaviour of agents in a complex setting and observe a convergence to a global optimum in spite of the absence of stable states.

Keywords Multi-agent system · Genetic algorithm · Action trees · Social simulation

1 Introduction

Computer simulations open up a new way of analysing complex social and economic systems. The simulation stands out against the classical differentiation of deductive and inductive methods in economic theory: In simulations, explicit conditions are formulated in a formal language. So it is done in deductive procedures. However, simulation results are not interpreted through proofs but through the inductive evaluation of the data (Axelrod 1997). The modularity of a computer programme allows

T. Chmura · T. Pitz
Shanghai Jiao Tong University, Department of Economics, Fa Hua Zhen Road No. 535, Shanghai 200052, People's Republic of China

T. Chmura
e-mail: chmura@uni-bonn.de

T. Pitz
e-mail: tpitz@uni-bonn.de

J. Kaiser (✉)
University of Bonn, Laboratory for Experimental Economics, Adenauerallee 24-42, 53113 Bonn, Germany
e-mail: johannes.kaiser@uni-bonn.de

data to be harvested under controlled conditions and precise presuppositions, just like in an experiment.

An example of social simulations that has become classical in recent times is the finite cellular automaton which was developed by von Neumann and Ulam (von Neumann 1996). At first, it was used for an attempt to model the biological process of self-reproduction (Conway's game of life (Gardner 1970, 1971)). Another field of application of the cellular automaton is the analysis of complex technical and physical processes.

In social sciences, cellular automata have been used to create computer models as examples of social systems. They have often been implemented in order to demonstrate social interdependencies such as segregation effects in populations (Schelling 1969, 1971). Such emergent phenomena evolve through the fact that through their individual actions, agents generate circumstances which were not intended by them. Other examples for that are the origin of traffic jams (Nagel and Schreckenberg 1992) and the collapse of financial markets. Such relationships between the individual actions on the micro-level and the collective effects on the macro level describe a fundamental class of problems in social sciences which can very well be modelled through computer simulations.

Applications of computer simulations in economic sciences can often be found in the context of game theory. An important example is the theory of reinforcement learning developed by the biologist Harley (1981), which was intensely analysed by (Roth and Erev 1995; Erev and Roth 1998). Recent learning models such as Experience Weighted Attraction by Camerer also belong in this field (Camerer and Teck-Hua 1999). In simple game situations, experimental data could well be prognosed with the aid of such learning algorithms (Roth and Erev 1995; Selten et al. 2003). Obviously, there is also a reinforcement structure in the GAT algorithm presented by us in this work. However, most applications of reinforcement algorithms in economic literature use an atomic representation of strategies. This is the main difference to our approach. In the present paper, the representation of strategies—therefore called actions—is motivated by concepts of analytic language and action philosophy (Bratman 1987; Davidson and Harman 1973; Mele 1997; Searle 1983). According to some of these concepts, we used common language to consider the complexity and fussiness of actions.

However, in the approaches of learning algorithms based on game theory, many characteristics of social agents are not taken into account. Thus, many intentional attitudes (beliefs, desires, intentions), the alteration of preferences or the social context of agents is only rudimentary represented. Communication, adaptive abilities or a representation of the agents' knowledge are nearly completely missing. Apart from that, agents with heterogeneous behaviour patterns depending on the specific situation are important for the adequate depiction of complex social systems. These cannot be recorded by simple learning algorithms. Due to this criticism, multi-agent systems have developed out of the field of distributed artificial intelligence.

Software entities which differ from the usual learning algorithms in the way that they autonomously develop their own strategies for the coping with tasks are called agents. Agents do not own a merely trivial control of their actions. They use and administer knowledge, interact with other agents and possess the ability to cooperate and communicate. Agents perceive their environment and react to changes in their

surroundings. They own adaptive abilities and learn from experience and they do not only react to changes in their environment but take the initiative and affect their environment through their actions.

Among the first and probably best analysed architectures of multi-agent systems are the BDI systems (belief–desire–intention) whose development was started by Bratman (1987) in the middle of the 1980s. BDI systems are examples of top-down models. This means that the agents' intentional states, beliefs, desires, and intentions, are explicitly represented through the BDI structure. Epistemic, modal, or temporal logics are used for the manipulation of the BDI structure. Nevertheless, the implementation of these systems is usually very laborious due to the complex logical calculus.

A possible solution is offered by bottom-up modelling on the basis of neural networks or, as in this paper, by genetic algorithms (Holland 1975). This means that the entities relevant to the model, such as the characteristics of actions or intentional attitudes, are implicitly represented by a codification like binary sequences. On the basis of the codification, fundamental mathematical operations substitute the logical calculi.

Although widely unnoticed by social and economic scientists, machine learning algorithms have primarily become established as solutions for problems in engineering and information sciences, for instance in process management, traffic sciences and logistics, as well as on the Internet as a virtual service provider and in computer games (Weiss 1999). Thus, a new approach for the modelling of actions in multi-agent systems with a machine learning algorithm shall now be exemplary analysed in a social context in this paper. In contrast to this, game theory normally deals with the ontological structure of actions in a reductionist way. Through strategies, actions are mainly represented as atomic mathematical entities in economic models without any operational semantics. At best, characteristics of actions manifest themselves in their evaluation through the abstract preference relations or utility functions. We take the view that a linguistically motivated analysis of actions offers a deeper insight in the formal structure of actions in order to model those operational and procedural qualities of actions which are important for computer simulations. The terminology of analytical philosophy, especially analytical theories of action (Austin 1975; Searle 1971; Chisholm 1964; Davidson 1980), serves as a basis of the concept formation.

Multi-agent based simulation is a branch of distributed artificial intelligence that builds the base for computer simulations which connect the micro- and macro-level of social and economic phenomena like cooperation, competition, markets and social networks dynamics. The dynamics of social and economic systems manifest themselves in the formation, change, and disappearance of actions. For this reason, actions of many diverse forms appear in the application of computer simulations on economic and social processes. That creates the question how actions can be represented in computer simulations. In this paper, a general and uniform approach of modelling an as extensive class of actions as possible is analysed. On the one hand, this concept is supposed to take into consideration the diversity of actions, on the other to be operationally manageable nevertheless for the modelling of simulation models such the multi-agents system.

Searching for an integral model of actions, one inevitably encounters the question which qualities of actions separate these from different entities. This can be expressed

the most forcefully with the ontological question "What are actions?" Due to the heterogeneity and fundamental fuzziness of actions, one can give only incomplete answers to this. Therefore, we will attempt to approximate the ontological question through an epistemic and an evolutionary translation:

1. How can actions be described through a general model that is suitable for multi-agent simulations?
2. How can the formation and disappearance of actions be described in this model algorithmically?

Our approach follows the method of Pitz (2000). In the following, we briefly sum up the facts of this methodology. The interested reader can find another introduction with a simple example to this topic in Pitz and Chmura (2005). We extend their work by introducing a software capable of doing multi-agent based simulations with genetic action trees. It is not the aim of this paper to compare quantitative results with empirical data. Rather, the statistical evaluation of the simulations serves to prove endogenous qualitative changes in the behaviour of the agents in the simulations. The main aim is to exemplify the method of genetic action trees with a complex examples and to conduct a feasibility study with it.

The remainder of this article is structured as follows. Firstly, we draw up a generic model for genetic action trees. The next section deals with technical issues of the use of the software platform. After that, we introduce a simple example model and let our computer platform simulate it. The section closes with a short discussion of some details of the results of the simulation. In the final section, we summarise our results and conclude.

2 Genetic action trees

In this section, we introduce the basic functionality of genetic action trees. The details are left out, but the interested reader may find it in (Pitz and Chmura 2005).

2.1 The epistemic question

In this section, we answer the question how actions can be described through a general model that is suitable for multi-agent simulations.

2.1.1 Action types

In order to find answers to the epistemic question concerning a uniform system of describing actions, it is helpful to make use of some of the concepts used in analytical theories of action. For a definition of these fundamental notions in theories of action, it is worthwhile to examine the way in which actions are represented in common speech. Descriptions of actions such as "Last week Peter drank 2 bottles of wine with his two colleagues in his favourite pub." First of all suggest understanding actions as concrete singular objects to which a context of limited expansion in space and time can be attached. But if we consider actions to be singular entities in this way, we are

confronted with the problem of their repeatability and countability. It is therefore advisable to take a further step of abstraction and to subsume similar concrete actions as equivalence classes, so-called action types. Hence, the repetition of an action should be understood as the carrying out of an equivalent but indeed different action. It is exactly the analysis of iterations of similar actions which plays an essential role in computer simulations. A certain quantity of action types is called action space.

2.1.2 Action attributes

Just as the ontological question of the nature of actions cannot be solved generally, likewise the question of the characterisability of the equivalence relation cannot finally be answered but only in regard to the respective model. Instead of a formal definition of the equivalence relation for action types, we use quasi-deictic denotations of action types taken from common speech: action types can be denoted by using the infinitive form of verbs "to walk, to buy, to produce, to kill", etc. and a set of free variables. The free variables serve as a parameter for references of space and time as well as for the actors and objects affected by the processes when the action is performed. Likewise, we will treat quantitative and qualitative action attributes, such as "5 m/s" for "to walk" and "maliciously" for "to kill", etc. as free variables. In the following, we will call these free variables action attributes. Finally, we will understand certain preferences agents might have with regard to actions, such as "with pleasure" for "to travel" and "reluctantly" for "to drive a car", as action attributes in our model.

Talking about actions, it is naturally impossible to specify all free variables through an explicit allocation. Therefore, we will restrict ourselves in the notation to those variables which are essential, i.e. relevant for the model. The number and sort of free variables of an action type thus depends on the model. The more detailed the model, the higher the number of parameters and the more differentiated the range of the variables. The execution of an action belonging to an action type formally means the instantiation of all free variables.

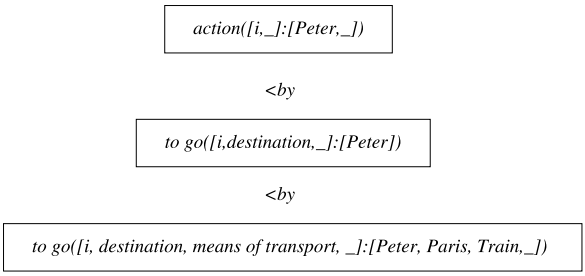
Notation

- If $H([i_1, \ldots, i_n])$ is an action type with a list of free variables, let $H([i_1, \ldots, i_n]$: $[k_1, \ldots, k_n])$ or in short $H(k_1, \ldots, k_n)$ be the execution of an action of type H with the allocation $[k_1, \ldots, k_n]$ of the free variables $[i_1, \ldots, i_n]$. Due to better legibility, the square brackets [ ] are sometimes left out.
- We use the meta-variable "_" for a set of variables which is not completely specified.

Example 1 (Action types)

- *to eat*([i]: [*Peter*]); Possible semantics: Peter is eating.
- *to drink*([i, w]:[*Peter*,*wine*]); Possible semantics: Peter is drinking some wine.
- *to buy from*([i, j, b, g]:[*Peter*,*Paula*,10 *Euro*,*book*]); Possible semantics: Peter buys a book from Berta for 10 Euro.
- *to offer*([i, k, J]:[*Paula*,*wine*,*consumer*]); Possible semantics: Paula offers the product wine to a set J of consumers.

Fig. 1 Unbranched action tree



2.1.3 Action trees

On certain action spaces, the preposition "by" induces a semi-order ($<_{\text{by}}$) (Goldman 1971). Figure 1 shows an example for an unbranched action tree.

Such action spaces can be arranged as a tree diagram using the most general action type $action(i, \_)$ and a set of free variables "_" (see Fig. 1). Here the semantics of $action(i, \_)$ be that "an agent i carries out any kind of action." This action is specified from the root to a leaf by the allocation of more and more of the free variables.

2.1.4 Binary decision-making actions—intentional degree

Decisions can be understood as special forms of action. We thus suggestively call them decision-making actions. In an action tree, decision-making actions possess at least two followers with regard to the $<$ by-relation.

In the simulations described in this paper, we restrict ourselves to binary decisions, i.e. decisions with two alternatives (see Fig. 2). H is a binary decision-making action with the nodes H_1 and H_2 which can be carried out by the agent in principle. In order to depict the agent's preference regarding the two alternatives, we use an intentional degree of H_1. The intentional degree of H_1 is represented by a real number $d_{H_1} \in [0, 1] \cup \{2\}$. The selection algorithm at the node H can be described as follows: One chooses a random number c from a finite subset $C \subseteq [0, 1]$. Then H_1 is chosen exactly then if $c < d_{H_1}$ is valid. Thus, changes in behaviour result from a modification of the set C. With $d_{H_1} = 0$ ($d_{H_1} = 2$), the node H_1 (H_2) can be eliminated in the model. The intentional degree d_{H_1} implicitly also determines the intensity of the wish to carry out H_2. The intentional degree describes what the agent *wants* to do. We understand d_{H_1} as the action attribute belonging to the decision-making action H.

In Fig. 2, let d_{H_1} denote the intentional degree of the left node, i.e. the intensity of the wish to carry out the left node of the tree.

2.1.5 The selection of an action type

The selection of an action type by an agent can be seen as the walk through the action tree from the root to a leaf. In doing so, the agent chooses one of the two subsequent nodes depending on the intentional degree, as described in Sect. 2.1.3. Occasionally, the agents get instructions from other agents which they are forced to follow. For instance, Peter can have an order to go to Paris. The free variable "destination" is then already allocated with "Paris." In this case, Peter's selection process starts at

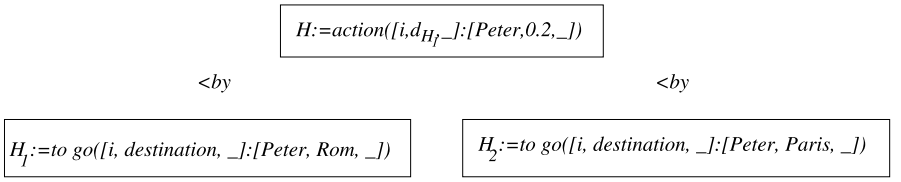


Fig. 2 Action tree with binary decision-making action

the node where free variables appear for the first time. In the example of Fig. 2, no decisions remain to be taken by the agent in this case. In the following, we will restrict ourselves to action trees with the following qualities:

1. Let the action tree be completely defined with regard to the actual model, i.e. each node exhaustively describes one action regarding to the given model. If a leaf of an action tree is reached, all variables which are necessary for the performance of the action are allocated.
2. Due to better legibility, we will only analyse action trees with two nodes at maximum in this introduction. Later on in Sect. 3, a more complex action tree is drawn up and discussed.
3. If an agent is unable to carry out any node, no action is conducted. The process of selecting an action will be restarted later on.

2.2 The evolutionary question—an algorithm for the modification of action attributes

In this section, we answer the question how the formation and disappearance of actions can be described algorithmically in this model.

2.2.1 Genetic action trees

The dynamics of action trees can be described through the alteration of the allocation of action attributes. In order to conduct these alterations as methodically coherently as possible, one can represent allocations of action attributes as sets of binary series. In the following, the allocation of an action attribute coded through the binary series will also be called the gene of an action attribute. A set of genes is called a gene pool. The alteration of the allocations results from the genetic algorithms operating on the gene pool of the action attribute (see Sect. 2.2.5).

Definition 1 (Genetic action trees) A genetic action tree $G(T)$ for a set of agents I has the following structure:

- T is an action tree.
- $\forall i \in I \wedge \forall H[A] \in T \wedge \forall a \in A$ let $C(i, A) \subseteq \{0, 1\}^n$ be the gene pool of an action attribute a of the action type H and of the agent i.
- $\forall H \in T$ of a decision-making action and the subsequent nodes H_1 and H_2, there is an intentional degree $d \in [0, 1] \cup \{2\}$ and a potentially empty set of exogenous conditions Δ_{H_1} and Δ_{H_2}, which have to be fulfilled for the specific agent i so that H_1 or H_2 can be chosen by i.

- $\forall c(i,a) \in C(i,a)$ let $\phi(c(i,a)) \in R$ be the fitness of $c(i,a)$. After the execution of an action, the fitness of $c(i,a)$ is modified with regard to the results of the action.
- $\forall c(i,a) \in C(i,a)$ let $\delta(c(i,a))$ be the semantics of $c(i,a)$.

In the next section, the meaning of the terms introduced just now shall be exemplified on the basis of how they are used. First, we will describe the choice algorithm of the actions through the agents.

2.2.2 The choice of the actions through the agents

After having been activated, each agent $i \in I$ checks if the list of instructions for actions the agent might have received contains any elements. In the later on described model, all instructions for actions are carried out successively first in first out. If no instructions for actions have been received by the agent i, i determines which action has to be carried out by following a path $[H_1, \ldots, H_n]$ of an action tree from the root H_1 to a leaf H_n. While doing so, all attributes of the action types $[H_1, \ldots, H_n]$ are successively randomly allocated with coded values $c(i,a) \in C(i,a)$ on the basis of an equal distribution. Let $H_\otimes(i, d_{H_1})$ be a decision-making action and $d_{H_1} \in [0,1] \cup \{2\}$ the intentional degree for the left node. If the agent is confronted with a decision-making action $H_\otimes(i, d_{H_1})$, the subsequent nodes H_1 and H_2, for which Δ_{H_1} or rather Δ_{H_2} have been violated, are eliminated. Δ_{H_1} and Δ_{H_2} are exogenous conditions specific to the model which guarantee the consistency and coherence regarding the agent i's abilities in its environment. Thus, the exogenous conditions define which actions *can* be carried out by i.

If it is the case that two nodes remain at the node $H_\otimes(i, d_{H_1})$ for i, the agent is confronted with two alternative actions. We describe the semantics of the decision-making algorithm:

Let $\beta : \{0, \ldots, 2^n - 1\} \to \{0,1\}$, $n \in N$, be the natural bijection which assigns each binary series $[x_1, \ldots, x_n] \in \{0,1\}^n$ with

$$\beta(k) = [x_1, \ldots, x_n] \quad \leftrightarrow \quad k = \sum_{i=1}^{n} x_i 2^{i-1}.$$

Let $\delta : \{0, \ldots, 2^n - 1\} \to \{0,1\}$ be defined by

$$\delta(k) = \frac{k}{2^n - 1}$$

for each $k \in \{0, \ldots, 2^n - 1\}$. Let $C(i, d_{H_1}) \subseteq \{0,1\}^n$ be the gene pool of the agent i with regard to the intentional degree d_{H_1}. The left node is chosen exactly then if $\delta(c(i, d_{H_1})) < d_{H_1}$ holds, while $c(i, d_{H_1})$ is chosen randomly from the set $C(i, d_{H_1})$. Thus, the genome specifies the threshold for a decision.

It is conceivable that an agent receives instructions for actions from another agent. That means precisely that action attributes are determined through another agent's instruction. The decision-making process then starts at a node H_1, $1 < t \leq n$, at which free variables appear for the first time, if all the conditions Δ_H of all preceding nodes H have been fulfilled. If a condition is violated, the instruction will not be carried out.

When the agent i has reached the root H_n of the action tree, all attributes are allocated with coded values. The action type is carried out in accordance with theses allocations and each allocation $c(i,a) \in C(i,a)$ with which the action type was carried out is judged through the change in its fitness value $\phi(c(i,a))$ due to the results of the action.

2.2.3 The evaluation of an agent's action

After the agent's run through an action tree, all attributes which are necessary for the specification of an action within the context of the model are clearly defined. Each gene has been assigned a concrete specification through the semantics δ which belongs to the actual model. After the determination of the necessary parameters, the action can now be carried out in the model according to the semantics δ. Actions change the agent's environment. Those changes are called the results of an action. The results of actions form the basis of the evaluation of actions or more precisely the evaluation of the genes of action attributes through the alteration of the fitness value ϕ.

2.2.4 Changes in the characteristic of action attributes

Genetic algorithms are used for the simulation of changes in action spaces. In the following, specific genetic algorithms which have proved useful in the case of the action spaces will be discussed. If a gene pool $C(i,a)$ of an agent i and an action attribute a is changed, this can evoke changes in the behaviour of agent i. Let $J \subset I$ be a subset of agents and a an action attribute. Let

$$C_j(a) := \bigcup_{i \in J} C(i, A)$$

be the common gene pool of the agents $i \in J$ of a. If J contains more than one agent, the intersubjective changes in behaviour can thus be simulated. For the application example analysed in this paper, the gene pool $C_I(a)$ was used. Here is valid: for all agents i and action attributes $C(i,a)$ it contained exactly one element.

2.2.5 The genetic algorithms for attributes of the action types

The genetic algorithms consist of mutation, selection and crossover. In the model described later on, the genetic algorithms were used on the sets $C_J(a)$ when the actions belonging to the attribute a had been carried out 10 times.

- Selection and mutation. Each gene $c(i,a) \in C_J(a)$ is changed with the probability $p_{\text{mut}}(c(i,a))$ at n pairwise different places which have been chosen randomly. If the gene $c(i,a)$ of the agent i is changed through mutation, the original gene is substituted with its mutant. $p_{\text{mut}}(c(i,a))$ is here proportional to the fitness $\phi(c(i,a))$. Thus, the substitution of a gene with its mutant is the more likely the smaller the fitness of the gene is.

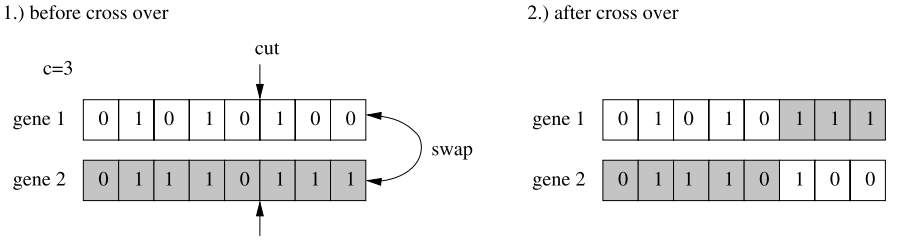


Fig. 3 Schematic display of the crossover

- Cross over. On the sets $C_J(a)$ a crossover was carried out on 5% of the elements of $C_J(a)$. A crossover is presented as an example in the following figure. The cut S is determined randomly on the basis of uniform distribution. This is shown in Fig. 3.

The model described here is limited to these three genetic operators. It should not be a problem to extend the algorithm to contain more operators like blocking, unblocking, techniques like simulated annealing, and the like.

There exist a variety of different approaches to utilise genetic algorithms in multi-agent systems. Other well-discussed applications of genetic algorithms are genetic programming and classifier systems. The basic idea of genetic programming is well described in (Koza 1990). The genes are represented by syntax trees of simple computer programs. The mutation has to respect the syntactical correctness of the genes. The crossover could be interpreted as exchange of subprograms. The fitness function is updated after running the programs. The key difference from genetic programming to our approach is that in algorithms using genetic programming, new types of action can arise. This is not always desired. In genetic action trees, all possibilities of action are pre-specified in a tree structure. Actions can be eliminated, but it is not possible to dynamically create new ones which haven't been defined in before. A different approach are learning classifier systems, which where described by (Holland 1975) as adaptive systems—like autonomous agents—that are able to categorise their environment. The kernel of classifier systems are "condition–action-rules" like "IF RECEIVE(Input) THEN Create(Output)" coded as binary sequences. The rules could be modified by genetic algorithms. In the genetic action trees, there are analogous mechanisms to that: a tree structure replaces the "IF...THEN" structure.

2.3 The GATKernel software package

The beforehand described algorithms have already been implemented as a computer programme. The software can be obtained free of charge in the Internet.[1] It is written in Java and thus should be runnable on any operating system. Commented source code for the exemplary model described in the next section is included in this package.

[1] http://www.bonneconlab.uni-bonn.de/econlab/individual.php?id=57. Note that the reference implementation makes use of Andy Khan's Java Excel API, available at http://jexcelapi.sourceforge.net/.

3 An exemplary application of the genetic action tree kernel

So far, we explained how the genetic action tree approach works and how one can implement a multi-agent system with the GATKernel. In this section, we draw up a not too demure example model to illustrate a way to apply the kernel to a specific research question. It is important to view the model described hereafter not as a serious research device but as a feasibility study for a complex global organisation problem. In this application, we deal with a alcoholic party. Note that this is not the only investigation of the ability of people to socially organise themselves in the context of a get-together. For example, Arthur (1994) introduced the so-called El Farol bar problem to game theory. This problem deals with a finite population of agents which want to visit the El Farol bar in Santa Fe, New Mexico. If more than 60% of the population go to the bar, it is too crowded to have fun for each agent and they all have less pleasure than if they stayed at home. On the other hand, if less than 60% of the population attend this bar, they all have a better time than at home. Another example would be the NetLogo party model by (Wilensky 1997). He investigates the grouping behaviour of male and female cocktail party guests. We create a setting in which a finite number of agents attend a party. Each one of the agents has to meet conflicting targets.

3.1 Our world

Consider an anonymous human being visiting a party in a not so distant world. Let's furthermore assume that this party is a rather boring place to be, so that the only joy of our person is to gain and maintain an optimum level of alcoholic intoxication. Several implications of this fact must be taken into account:

- There are only finite stocks of alcoholic beverages (which are limited to beer and a high spirit liquor, referred to as *schnaps*) and coffee. Thanks to the favourable position of the civilisation our individual lives and celebrates in, water supply is not limited.
- As a natural protection measure, the party visitor will fall asleep at once if the blood alcohol level exceeds a certain border. To counter such dire consequences he could simply suspend to consume beer and schnaps until the blood alcohol level decreases to a more acceptable extent. An even better alternative to this behaviour would be to drink coffee—in our individual's world, the black gold has this almost magic feature to lower the blood alcohol level.
- Even in this fabulous world, alcohol drinking entities will experience a hangover the day after. The quality of this hangover is a measure of consumed raw spirits and consumed liquids. Unfortunately, coffee dehydrates the consumer's body.
- If the considered individual neither consumes beer nor schnaps, the blood alcohol level drops by a small amount.

The objectives of our friend are now to stay as close as possible to the optimum blood alcohol level, not to fall asleep (as this would waste precious time he could spend celebrating), and to minimise the after day's hangover.

3.2 The model

The situation described in the previous section is an ideal testing vehicle for genetic action trees. Let one party visitor of all guests be one agent in the sum of all agents. To investigate the drinking behaviour of all agents and to find concluding hints for an optimum drinking strategy, we could easily utilise genetic algorithms.

There are several decisions our individual can take. Figure 4 shows a decision tree which was used to derive a genetic action tree. The nodes have only one gene each, the decision gene. The actual decisions, actions, and their implications are as follows:

- Node H_0: The agent has to make a decision whether to consume alcohol or not. If the agent is asleep ($S_i = 1$) or if there is neither beer nor schnaps left ($\sigma_i^b = 0 \wedge \sigma_i^s = 0$), the child node H_2 is not valid and the next node is H_1 by default. To calculate the fitness of a decision in a given period, several factors must be taken into account:

  1. The failure to eliminate the negative delta between actual (B_i) and optimum (B^*) blood alcohol level will be punished if $B_i < B^*$. In more formal terms, we need to reduce the fitness by $\min\{0; B_i - B^*\}$. Since an evaluation of the fitness is done only in every mth period, we will have to sum up this delta over the last m rounds.
  2. The fitness will be dramatically reduced if the agent falls asleep—this is the case if $S_i = 1$.
  3. Furthermore the agent's ability to reduce the expected hangover will be rewarded. Hangover is defined as the relation of total consumed raw spirits (A_i) to total consumed liquids (L_i): A_i/L_i. To judge the hangover adjustment in the

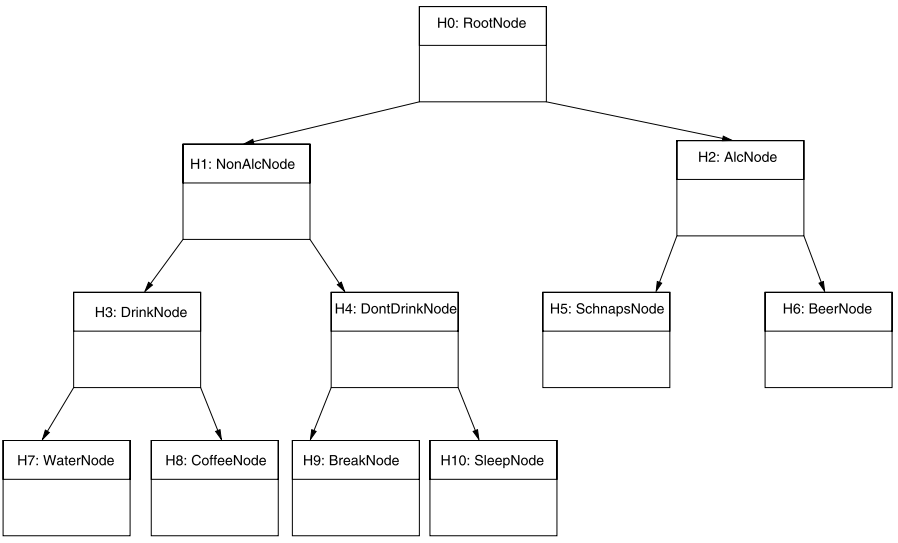


Fig. 4 The tree representing the choices which lead to certain actions

last m periods, we will add

$$\left(\frac{A_{i-m}}{L_{i-m}} - \frac{A_i}{L_i}\right)$$

to the fitness of the decision gene. The denominators in this term will never equal zero, because we assume that every party guest enjoys one unit of water in the beginning.

The fitness function for the decision gene is a sum of the terms mentioned above weighed by coefficients $\mu_j \ \forall j \in \{1, 2, 3\}$:

$$\phi(c(i, d_{H_0})) = \mu_1 \sum_{j=i-m}^{i} \min\{0; B_j - B^*\} - \mu_2 \sum_{j=i-m}^{i} S_j + \mu_3 \left(\frac{A_{i-m}}{L_{i-m}} - \frac{A_i}{L_i}\right). \tag{1}$$

- Node H_1: If the agent decided not to consume any alcohol there are several choices left. The child H_3 gets eliminated if the agent is asleep; otherwise he has to determine whether he wants to drink non-alcoholic beverages or not. This also has effect on the progression of the blood alcohol level because coffee will reduce it by a small amount, so we simply use (1) as the fitness function for the decision gene of H_1:

$$\phi(c(i, d_{H_1})) = \mu_1 \sum_{j=i-m}^{i} \min\{0; B_j - B^*\} - \mu_2 \sum_{j=i-m}^{i} S_j + \mu_3 \left(\frac{A_{i-m}}{L_{i-m}} - \frac{A_i}{L_i}\right). \tag{2}$$

- Node H_2: Now the agent has to choose his preferred kind of drink. He will be forced into having one special kind of beverage if there's nothing more left of the other. The fitness of this gene is affected by the expected hangover reduction and the improvement of the negative delta between actual and optimum blood alcohol level:

$$\phi(c(i, d_{H_2})) = \mu_1 \sum_{j=i-m}^{i} \min\{0; B_j - B^*\} + \mu_3 \left(\frac{A_{i-m}}{L_{i-m}} - \frac{A_i}{L_i}\right). \tag{3}$$

- Node H_3: This node corresponds to the decision on the type of non-alcoholic beverage the agent is about to drink. Since the coffee stock is limited it might occur that no more coffee is left. In this case, the child node H_8 is not a valid successor of H_3. The lack of an appropriate amount of consumed liquids will invalidate H_8, too. The influences of the decision in H_3 are limited to hangover reduction and decrease of the blood alcohol level. Thus, the fitness function is similar to (3):

$$\phi(c(i, d_{H_3})) = \mu_1 \sum_{j=i-m}^{i} \min\{0; B_j - B^*\} + \mu_3 \left(\frac{A_{i-m}}{L_{i-m}} - \frac{A_i}{L_i}\right). \tag{4}$$

- Node H_4: There is a special issue concerning this node: Though it is not a leaf of the tree, there is usually no decision to take because an agent would never voluntarily fall asleep. This fact is modeled by setting the intentional degree to $d_{H_9} := 2$. By default, node H_{10} is never reached. If the agent is asleep ($S_i = 1$), node H_9 gets invalidated. In this case H_{10} is the only successor of this node, this is the only way to reach node H_{10}. A fitness function is not necessary since there is no decision to make at H_4.
- Node H_5: At this leaf the agent will drink one unit of schnaps after ensuring—in the course of the nodes previously executed in the course of the action tree—that he is not asleep and that there is enough schnaps left. This will have some consequences. Next period's blood alcohol level will be increased by the schnaps (5), the schnaps stock will be decreased by the amount of one unit of schnaps (6), the variables for consumed liquids (7) and consumed raw spirits (8) will be adjusted, and the agent will fall asleep if the maximum level of intoxication Λ is exceeded (9). Note that if any of next period's variables is not defined in the leaf nodes, it defaults to the values of the current values.

$$B_{i+1} := B_i + \beta^s, \tag{5}$$

$$\sigma^s_{i+1} := \sigma^s_i - l^s, \tag{6}$$

$$L_{i+1} := L_i + l^s, \tag{7}$$

$$A_{i+1} := A_i + a^s, \tag{8}$$

$$S_{i+1} := \begin{cases} 0 & \text{if } B_{i+1} \geq \Lambda, \\ 1 & \text{if } B_{i+1} < \Lambda\,. \end{cases} \tag{9}$$

- Node H_6: The agent decided to drink beer. In our model, drinking beer is very similar to drinking schnaps—only some parameters differ in height. This action is carried out in this leaf node:

$$B_{i+1} := B_i + \beta^b, \tag{10}$$

$$\sigma^b_{i+1} := \sigma^b_i - l^b, \tag{11}$$

$$L_{i+1} := L_i + l^b, \tag{12}$$

$$A_{i+1} := A_i + a^b, \tag{13}$$

$$S_{i+1} := \begin{cases} 0 & \text{if } B_{i+1} \geq \Lambda, \\ 1 & \text{if } B_{i+1} < \Lambda. \end{cases} \tag{14}$$

- Node H_7: Our party guest has decided to drink water. This will not have influence on anything but the hangover quality and the blood alcohol level:

$$B_{i+1} := B_i + \beta^n, \tag{15}$$

$$L_{i+1} := L_i + l^n. \tag{16}$$

- Node H_8: This leaf represents the action of drinking coffee. Coffee reduces the blood alcohol level by $-\beta^c$. Besides this unusual consequence, the consume of

coffee dehydrates occurs and the amount of consumed liquids by $-s^c$. Note that β^c and s^c are negative constants. Due to quantity restrictions we will also have to decrease the amount of available coffee:

$$B_{i+1} := B_i + \beta^c, \tag{17}$$

$$\sigma^c_{i+1} := \sigma^c_i - l^c, \tag{18}$$

$$L_{i+1} := L_i + s^c. \tag{19}$$

- Node H_9: For some reason, the agent decided to do nothing. This behaviour will affect only the blood alcohol level because of alcohol decomposition:

$$B_{i+1} := B_i + \beta^n. \tag{20}$$

- Node H_{10}: The sleep action is never executed on purpose. If the alcohol consumption is about to wreak havoc on the brain, an internal biological self protection mechanism forces the agent to sleep. He will wake up only when the blood alcohol level falls under a certain limit called the minimum blood alcohol level V (22). Since liver and kidneys need time for alcohol decomposition, an agent never sleeps for only one period.

$$B_{i+1} := B_i + \beta^n, \tag{21}$$

$$S_{i+1} := \begin{cases} 0 & \text{if } B_{i+1} \leq V, \\ 1 & \text{if } B_{i+1} > V. \end{cases} \tag{22}$$

3.3 Calibration

1000 cycles were used to simulate the duration of the social gathering. The genetic optimisation occurred every tenth cycle, and 100 agents were competing about the optimal behaviour strategy. Implications for our model are:

- The party lasted ten hours represented by 1000 cycles, leaving one cycle to 36 seconds.
- The genetic optimisation was performed every five minutes.
- One hundred party guests had to share the available beverage resources.

Everyone who has ever organised a social happening of this size has been confronted with this puzzler: How much beverages do we have to buy? Contrary to real-life parties one resource should be scarce in our model, because a special interest arises in how the genetic algorithm tries to compensate the lack of one specific drink, be it coffee, schnaps, or beer. We assume that one healthy individual is capable of finishing half a litre of beer in 25 minutes, one hundred millilitres of coffee or water in five minutes, and 2 centilitres of schnaps in 5 minutes. Furthermore, the blood alcohol level of individuals in our model world decreases by 0.2 per mill each hour. In our model, the schnaps has been chosen to be the scarce resource.

After constantly modifying the parameters and evaluating the corresponding results, the initialisation of the parameters as displayed in Table 1 was used to accomplish the results described later. The intentional degrees were set up according to Table 2. They represent the consumption and behaviour preferences of the agents.

Table 1 Initialisation of the parameters

$\mu_1 := 10.0$	$s^c := -0.01$	$a^b := 0.0005$	$\sigma_0^b := 250$
$\mu_2 := 5.0$	$l^s := 0.002$	$a^s := 0.0008$	$\sigma_0^c := 50$
$\mu_3 := 10.0$	$l^b := 0.01$	$\beta^n := -0.001$	$\sigma_0^s := 20$
$B^* := 2.0$	$l^w := 0.01$	$\beta^b := 0.01$	
$\Lambda := 2.5$	$l^c := 0.01$	$\beta^s := 0.016$	
$V := 1.5$	$m := 10$	$\beta^c := -0.01$	

Table 2 Intentional degrees

d_{H_1}	d_{H_3}	d_{H_5}	d_{H_7}	d_{H_9}
0.3	0.7	0.4	0.3	2.0

3.4 Results

Though dealing with a problem which may not easily be solved by linear optimisation algorithms, the genetic action tree approach makes the fitness values first converge to and then fluctuate around a considerably high value by steadily adjusting each agent's behaviour strategy. As will be shown, this is even the case in a changing prevailing context. Since the algorithms used for optimisation in this example are not deterministic, the shown results must be regarded as a sample and not as a final or optimal solution to the problem. The investigation of the results of a multitude of simulations suggests that the samples do not differ significantly.

3.4.1 Overall performance

In the simulations, 100 agents had to find a decision in 1000 periods or rounds. This leaves the total number of actions taken to 100000. Figure 5 shows the distribution of taken actions. Since schnaps has been chosen to be a scarce resource (see Sect. 3.3) there have been only $\sigma_0^s = 20$ litres in stock. With one unit of schnaps being $l^s = 0.002$ litres, the action of drinking one unit of schnaps totals to 10000. The action of drinking one unit of beer was executed about twice as often, which may be due to the fact that the intentional degree of the corresponding tree node d_{H_5} was set to 0.4, that the schnaps stock was limited, and that the hangover quality increases less through the consumption of beer. The sleep action was performed the most. A detailed reasoning for this behaviour can be found in Sect. 3.4.4. The other actions were executed less often. This is a logical consequence of the high amount of periods spent sleeping and the low intentional degrees for not drinking alcohol.

3.4.2 Fitness and mutation

In this paragraph we focus on fitness and the mutation count of only the root node because its fitness function $\phi(c(i, d_{H_0}))$ considers all of the objectives one agent has. Figure 6 shows the development of the average, maximum, and minimum of the fitness functions of all agents. One can observe a small dent in the average and maximum values of the fitness functions. This is due to the fact that most of the agents

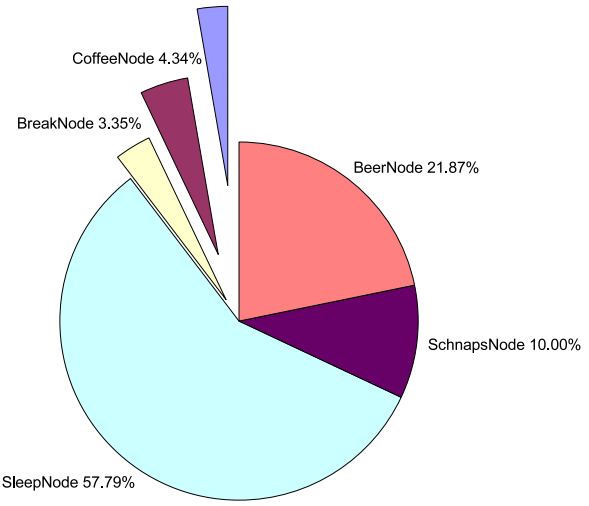


Fig. 5 The number of reached leaf nodes as totals

have been asleep in this range (see also Sect. 3.4.3). Apart from that, the average fitness values seem to stabilise on a considerably high level.

Nevertheless, mutation is not coming to a halt (see Fig. 7). Because a stable equilibrium cannot be reached in our setting, the decision genes have to be adjusted constantly. Consequently, the mutation count defined as the number of agents whose decision gene have mutated in a given period will not converge to a minimum or even to zero.

3.4.3 The development of the blood alcohol level

Figure 9 shows average and maximum blood alcohol levels of all agents in each round. In the used simulation sample the schnaps stock was empty by round 732 (see Fig. 8). Obviously, this had no effect on the average and maximum blood alcohol level of the agents. The dent in the average curve can be explained by the fact that 99 of the 100 agents have been fallen asleep by round 625. After period 661, the number of agents asleep was declining again.

Note that the minimum blood alcohol level equals 0 up to round 171. This is somewhat surprising because our model does not take into account designated drivers, invalids, former alcoholics, and people who do not consume alcohol for various other reasons. On the other hand this indicates that the effects of mutation prevent genes from "freezing" forever. One can see that the average blood alcohol level of all agents fluctuates around the optimum alcohol level after an initial warm-up phase. This is a satisfying result because it shows that the genetic algorithm is even capable of dealing with a changed prevailing context once the schnaps stock is empty.

3.4.4 Hangover quality and sleep

Hangover quality is defined as the total amount of raw spirits consumed divided by the total amount of liquids consumed by a specific agent. We take a closer look at the

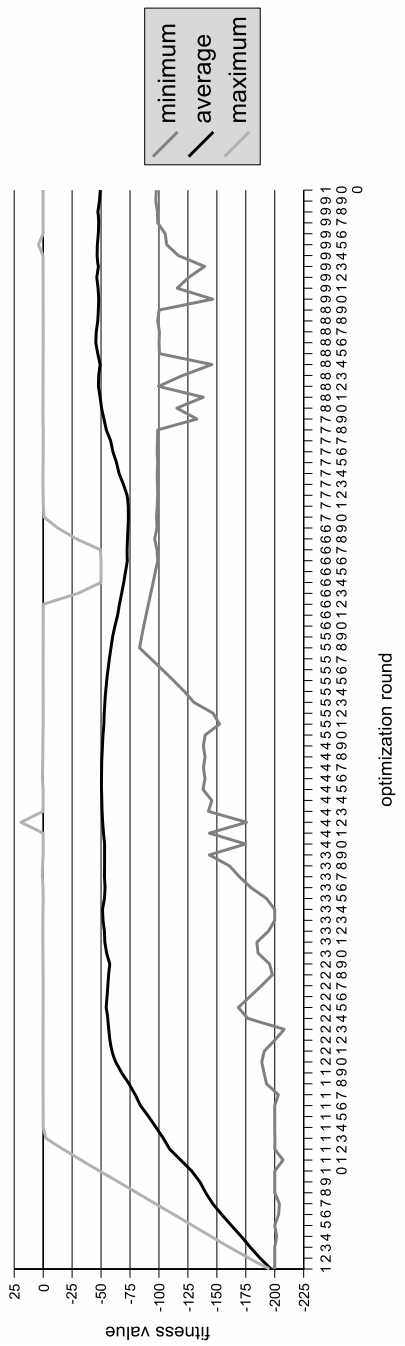


Fig. 6 Fitness values per genetic optimisation round

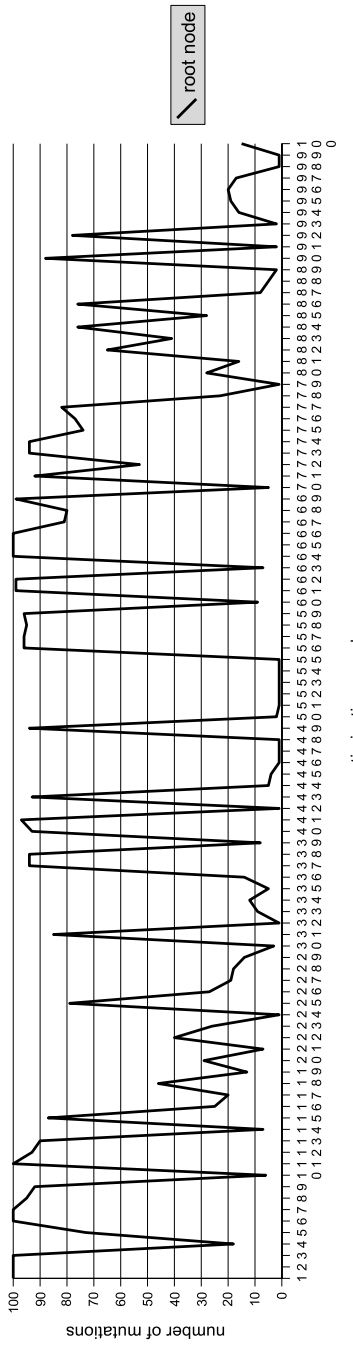


Fig. 7 The number of agents whose root node's decision gene was mutated

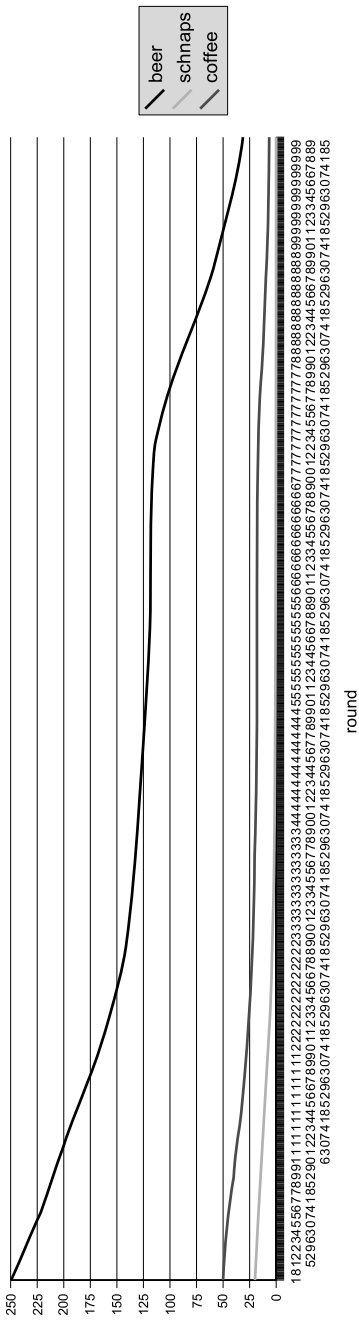


Fig. 8 The beverage stock in litres

Fig. 9 The blood alcohol level of all agents level in per mill

hangover quality part of the fitness functions (see equation 1) and find that

$$\mu_3 \left(\frac{A_{i-m}}{L_{i-m}} - \frac{A_i}{L_i} \right)$$

gets higher the lower the increase of hangover quality has been during the last ten ($=m$) periods. Keeping this in mind it makes perfect sense that the average hangover quality of all agents only increases by small steps (see Fig. 10). In spite of that, the genetic algorithm was able to quickly increase the blood alcohol level.

The great share of the "sleep" action can be easily explained. Firstly, the agents do not know about the negative effects of falling asleep until they fall asleep. So almost every agent falls asleep at least once. Secondly, alcohol decomposition takes time. In one period of sleep, the blood alcohol level decreases by 0.002 per mill. An agent falls asleep once the blood alcohol level exceeds 2.5 per mill and wakes up again as soon as it is lower than 1.5 per mill. This means that a sleeping agent sleeps at least for 500 periods. Figure 11 shows the number of sleeping agents per period.

4 Concluding remarks

In this paper, a general concept of the modelling of action spaces has been presented. The epistemic question "How can actions be described through a general model suitable for computer simulations?" was answered by a concept used in analytic theories of action, the action trees. To answer the evolutionary question "How can the emergence and disappearance of actions be described through a uniform algorithm within this model?", we made use of genetic algorithms. The synthesis of these two methods, the genetic action trees, has been implemented in an extendable software package called GATKernel. To illustrate the use of this software, we applied it to a simple model of social gatherings. We conclude that by looking at this feasibility study it seems plausible that the genetic action trees approach is useful for simulations of that kind.

To our knowledge, this is the first approach to publish a reusable multi-agent system which makes use of the methodology of genetic action trees. The method is very promising and can be expanded and applied in many different ways to a great variety of problems in social, behavioural, economic, and life sciences. We hope to provide the scientific community with a valuable tool to enrich the pool of state-of-the-art multi-agent systems. The GATKernel library should nevertheless only be seen as a starting point for more research in this direction. It could be enhanced by improving the interface to make the library more accessible to researches and scientists not capable of programming in Java. Furthermore, even more genetic operators could be implemented. For instance, blocking and unblocking operations could be introduced. It would also be possible to introduce simulated annealing techniques to the kernel: each step of a simulated annealing algorithm replaces a partition of the genes with a random "nearby" gene, chosen with a probability that depends on the difference between the corresponding fitness function values and on a global temperature parameter T, which is gradually decreased during the process.

To make the state-of-the-art multi-agent systems comparable, a benchmark study with related software is yet to create. All this can be a starting point for future work.

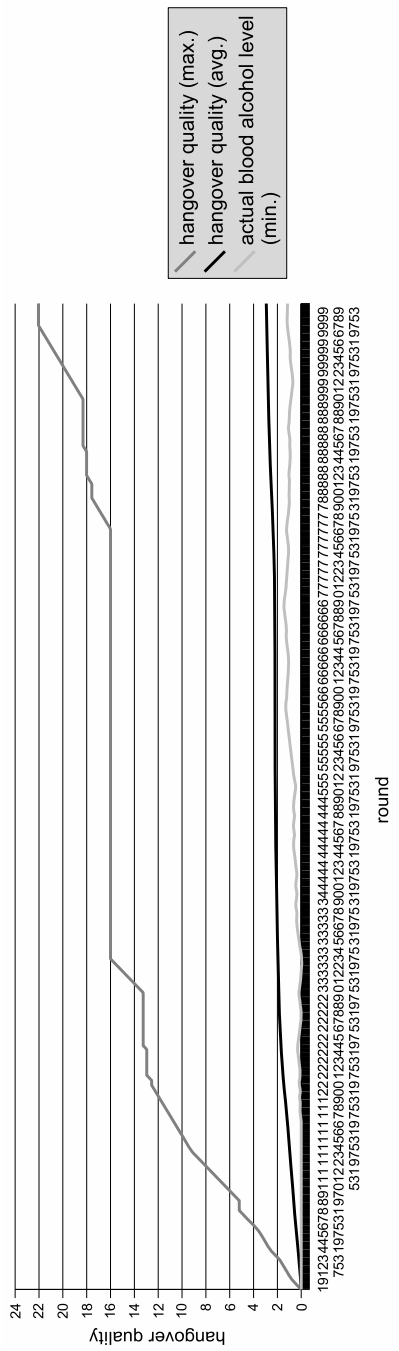


Fig. 10 The hangover quality of all agents defined as $\frac{A}{L}$

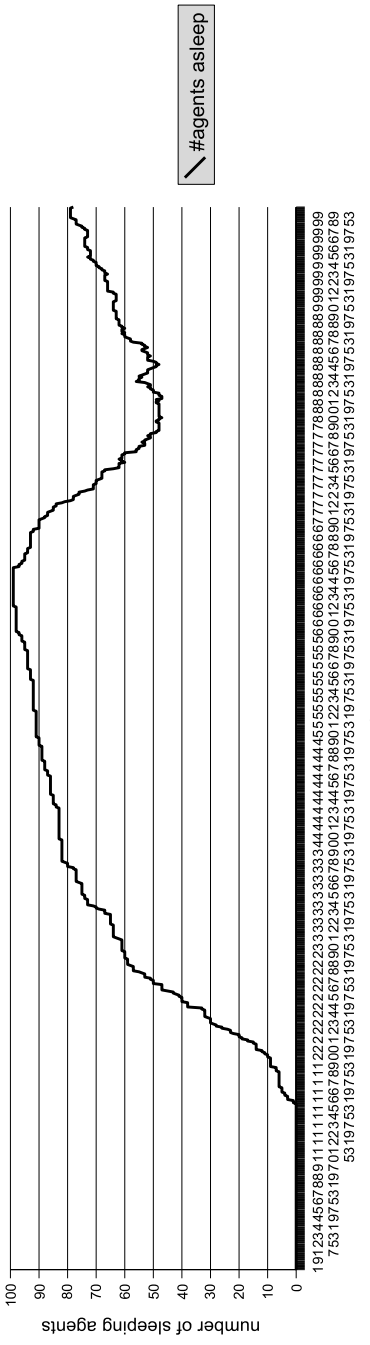


Fig. 11 The number of agents asleep per period

References

Arthur WB (1994) Inductive reasoning and bounded rationality. Am Econ Rev 84:406–411 (Papers and Proceedings)
Austin JL (1975) How to do things with words. Harward University Press, Cambridge
Axelrod R (1997) The complexity of cooperation. Princeton University Press, Chichester
Bratman ME (1987) Intentions, plans and reason. Harvard University Press, Cambridge
Chisholm R (1964) The descriptive element in the concept of action. J Philos 61(20):613–625
Camerer C Teck-Hua H (1999) Experience weighted attraction learning in normal-form games. Econometrica 67(4):827–874
Davidson D (1980) Essays on action and events. Oxford University Press, Oxford
Davidson D, Harman G (eds) (1973) Semantics of natural languages, 2nd edn. Springer, Berlin
Erev I, Roth AE (1998) Predicting how people play games: reinforcement learning in experimental games with unique, mixed strategy equilibria. Am Econ Rev 88(4):848–881
Gardner M (1970) The fantastic combinations of John Conway's new solitaire game life. Sci Am 223:120–123
Gardner M (1971) On cellular automata, self-reproduction, the Garden of Eden and the game life. Sci Am 224:112–117
Goldman A (1971) The individuation of action. J Philos 68(21):761–774
Harley CB (1981) Learning in evolutionary stable strategy. J Theor Biol 89:611–633
Holland JH (1975) Adaptation in natural and artificial systems. University of Michigan Press, Ann Arbor. Reprinted (1992) MIT, Cambridge
Koza JR (1990) Genetic programming: a paradigm for genetically breeding populations of computer programs to solve problems. Technical report, Computer Science Department, Stanford University
Mele A (ed) (1997) The philosophy of action. Oxford University Press, Oxford
Nagel K, Schreckenberg M (1992) A cellular automaton model for freeway traffic. J Phys I 2:2221–2229 France
Pitz T (2000) Anwendung genetischer Algorithmen auf Handlungsbaume in Multiagentensystemen zur Simulation sozialen Handelns, Frankfurt am Main
Pitz T, Chmura T (2005) Genetic action trees. Working paper, EconWPA
Roth AE, Erev I (1995) Learning in extensive games: experimental data and simple dynamic models in the intermediate term. Games Econ Behav 8:164–212
Searle JR (1971) The philosophy of language. Oxford University Press, London
Searle JR (1983) Intentionality. Cambridge University Press, Cambridge
Schelling TC (1969) Models of segregation. Am Econ Rev 59:488–493
Schelling TC (1971) Dynamic models of segregation. J Math Sociol 1:143–186
Selten R, Pitz T, Chmura T, Schreckenberg M, Wahle J (2003) Experiments on route choice behaviour. In: Emmerich H, Nestler B (eds) Schreckenberg interface and transport dynamics. Lecture notes in computer science, vol 32. Springer, Heidelberg, pp 136–155
von Neumann J (1996) The theory of self-reproducing automata, Burks A W. (ed). University of Illinois, Urbana
Weiss G (1999) Multi-agent systems. MIT, Cambridge
Wilensky U (1997) NetLogo party model. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL. http://ccl.northwestern.edu/netlogo/models/Party

Thorsten Chmura (Prof. Dr.) is assistant professor at the department of economics at the Shanghai Jiao Tong University in the P.R. of China. He did his Ph.D. in theoretical physics and worked 5 years together with Reinhard Selten at the laboratory for experimental economics. His research fields are experimental economics, behavioural finance, traffic behaviour and learning. He studied geology and geophysics at the University of Bonn in Germany and the Iowa State University, USA.

Johannes Kaiser is a research assistant and Ph.D. student of Reinhard Selten. He works at the Laboratory for Experimental Economics, University of Bonn, Germany, since September 2004. Johannes holds a master's degree in business and engineering from the University of Karlsruhe. His research interests include experimental economics, behavioural finance, and computational simulation techniques.

Thomas Pitz (Prof. Dr.) is assistant professor at the department of economics at the Shanghai Jiao-Tong University in the P.R. of China. He did his Ph.D. in logic and philosophy of science and worked several years together with Reinhard Selten at the laboratory for experimental economics. His research fields are experimental economics, game theory, traffic behaviour and learning. He studied mathematics at the TH Darmstadt, Germany.