Annales Univ. Sci. Budapest., Sect. Comp. 52 (2021) 349–368

THE AUTONOMOUS AGENT AND MULTI-AGENT PARADIGM IN SOFTWARE ENGINEERING

László Z. Varga (Budapest, Hungary)

Communicated by András Benczúr

(Received April 30, 2021; accepted August 1, 2021)

Abstract. Research on distributed artificial intelligence became a large field of research which is nowadays called autonomous agent and multi-agent system research. The autonomous agent and multi-agent paradigm helps a lot to understand, model and develop software systems for applications where the whole system, or the different parts of the system cannot be under the direct control of a single "authority". This paper overviews the main characteristics of autonomous agents and multi-agent systems, as well as their relation to the basic theoretical aspects of software engineering. The paper discusses the problems of complex multi-agent problem solving, and in particular the online routing game model.

1. Introduction

The mainstream research on the methods of creating computer systems was hallmarked by names like Edsger W. Dijkstra, Niklaus Wirth, Donald Knuth and Sir Antony Hoare in the 1980s. Their research focused on the formal and theoretical foundations of the professional discipline of software engineering. The main tools to handle software complexity included the abstract data types [12] for the modularisation of programs, the algebraic semantics [7] for the

Key words and phrases: Autonomous agent, multi-agent system, specification, problem solving, online routing game.

2010 Mathematics Subject Classification: 68T42.

Supported by project Application Domain Specific Highly Reliable IT Solutions project which has been implemented with the support provided from the National Research, Development and Innovation Fund of Hungary, financed under the Thematic Excellence Programme TKP2020-NKA-06 (National Challenges Subprogramme) funding scheme.

specification and the meaning of software, the Hoare logic [14] for verifying program correctness, and the formal language communicating sequential processes (CSP) [15] for specifying the interactions of concurrent processes. Sir Antony Hoare was knighted by Elizabeth II in 2000 for his contributions to computer science.

The Faculty of Science of the Eötvös Loránd University, and in particular the Division of the Informatics Departments within the faculty, was not only following these trends, but the key professors Imre Kátai, Ferenc Schipp and László Varga, among others, were continuously monitoring the emerging leading edge research as well. One of these emerging leading edge research trends of that time was distributed artificial intelligence [1]. Tibor Gyires from the University of North Carolina at Charlotte was invited to one of the regular workshops of the Division of the Informatics Departments at Visegrád to talk about this topic. The author of this paper had the chance to assist him, and this event determined his research interest throughout his career path.

Distributed artificial intelligence became a large field of research which is nowadays called multi-agent systems [48, 35, 45] research. The first big multi-agent project in Europe was the Architecture for Cooperative Heterogeneous On-line Systems (ARCHON) project [46] which implemented the first real-world industrial applications of multi-agent systems [16]. Later, many other applications have been developed in different fields including the healthcare domain [18] in combination with data provenance [22]. Nowadays commercial products like the digital voice agents (Apple's Siri, Microsoft's Cortana, Amazon's Alexa, and the Google Assistant) use autonomous agent technologies. Multi-agent technologies are used to control the fleets of drones and the fleets of autonomous vehicles [39]. Rescue operations are supported by human-agent collectives [27, 17]. Fleet of robots deliver products to customers at the warehouse of the world's biggest e-commerce company [49]. Google Ads uses multi-agent agreement technology [6]. The Autonomous Agents and Multi-Agent Systems conferences became big world-wide events managed by the International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS)*. Regius Professorships have royal patronage in the United Kingdom and Ireland, and traditionally they existed only in classic scientific fields. The first Regius Chair of Computer Science was created in 2014 and the Regius Professorship was bestowed to Nick Jennings, an internationally-recognised authority in the areas of agent-based computing and multi-agent systems.

This paper overviews the main characteristics of autonomous agents and multi-agent systems, as well as their relation to the basic theoretical aspects of software engineering. Section 2 introduces the agent concept. Section 3 discusses the single agent problem solving. Section 4 introduces multi-agent

*http://www.ifaamas.org/

systems. Section 5 discusses the multi-agent problem solving. Section 6 discusses the problems of complex multi-agent problem solving, and in particular the online routing game model. Section 7 concludes the paper.

2. Autonomous agents

An agent is an active component that behaves intelligently in a complex environment to achieve some kind of goal. From software technology point of view agent technology promises to enable system designers to handle more complex systems than before. As systems become more and more complex, software development processes need higher and higher abstractions. In the beginning functional and modular programming techniques provided enough level of abstraction, then object oriented systems became the most commonly used technique to model complex systems. Agent technology promises to handle systems that object oriented techniques cannot adequately model, like large distributed organizations with incomplete information and distributed responsibility where individual components must dynamically adapt to unforeseen changes.

The most important characteristics of agents are those which are defined by Wooldridge and Jennings in [47]. An agent is a computer system situated in some environment. The agent is reactive which means that it is capable of sensing its environment and acting on it. The agent can autonomously act in its environment and make decisions itself. The agent has design objectives and can decide itself how to achieve them. While taking the decisions the agent is not just passive, but can take initiatives towards its goals. The agent has social abilities and can interact with the actors in its environment.

For a programmer, the agent concept seems very similar to object-oriented programming. Objects are computational entities that encapsulate some state, they are able to perform actions on this state through methods, and they can communicate through message passing. The object seems to have autonomy over its state, because its state is encapsulated and can be manipulated only through method invocations. However, the object does not have control over its behaviour. Any other object can invoke a public method of an object, and if the method is invoked, then it is directly executed. This approach is normal if we build a system in which all objects are directly controlled by a single "authority".

There are applications where the whole system, or the different parts of the system cannot be under the direct control of a single "authority". For example a rover on the planet Mars cannot be controlled directly, because the travel time of the control signal is too long. In this case, the control centre "delegates" the task to the rover, and the rover autonomously tries to achieve the delegated

goal in the actual circumstances which are not known exactly in advance. If the rover receives another request, then the request is not directly executed, but the rover decides whether and how it makes sense to execute the request. There are other applications where we could directly control our agent, but we delegate the task execution to an agent, because we do not want to be involved in every details of the task, and we trust that the agent can handle the changes in the environment. These are the single autonomous agent applications.

The concept of autonomy seems to be a strange concept for software engineers at the first sight. A software engineer would expect that the invocation of a program code would entail the direct execution of the program code, as it was the intention of the programmer. From the traditional software engineering point of view, if the program code is not directly executed, then it is a sign of unreliability. Of course, the parts of the code of the autonomous software agent are reliably executed, as it is expected by a software engineer. At this level the software agent is not autonomous, because it does exactly what is programmed into it. The autonomy concept is on the agent level. When a task is delegated to a software agent, then the task is executed by autonomous agent mechanisms which will be discussed in Section 3. At this level the software agent is autonomous, because it independently executes the delegated task and takes into account the current circumstances. The autonomy refers to the behaviour of the software agent.

There are applications where there is no single authority, and the different parts of the system represent the interests of different organisations or individuals. In these systems no common goal can be assumed. If an action is requested from an agent (its method is invoked) from another agent, then the requested agent may or may not execute the action. Many of the applications mentioned in the introduction section fall in this category. The autonomy concept in such multi-agent systems is discussed in Section 4.

3. Autonomous problem solving

Computer programs are written to solve given problems. According to the classical approach, the problem is specified in a formal language. If model-based specification is used, then the specification is a relation between two sets of states. The software program is regarded as a finite or infinite sequence of state transformations. The finite sequence of state transformations take the program from the possible initial set of states to the possible final set of states. This way the partial and total correctness of programs with respect to a specification can be defined. Hoare logic [14] can be used to reason rigorously about the correctness of computer programs. The autonomous agent paradigm does not fit into this approach. The autonomous agent is embedded in its

environment, and the autonomous agent is expected to perform "well" if there are unforeseen changes in the environment. The rest of this section presents the approach in [48] to formally define autonomous problem solving.

The environment of an autonomous agent may be in any of a finite set E of discrete, instantaneous states (3.1). A continuous environment can be modelled by a discrete environment to any degree of accuracy. The agents have a set of possible actions which transform the state of the environment (3.2).

$$E = \{e_i \mid i \in [0..n]\} \tag{3.1}$$

$$Ac = \{\alpha_i \mid i \in [0..m]\} \tag{3.2}$$

A run r of an agent in an environment is a sequence of interleaved environment states and actions (3.3). The environment starts in state e_0, and the agent chooses an action to perform on that state. The environment goes to one of a number of possible states, state e_1. The agent again chooses an action to perform, and the environment responds with a possible state, and so on. Let $\mathcal{R}$ be the set of all such possible finite sequences over E and Ac; R^{Ac} be the subset of $\mathcal{R}$ that end with an action; and R^E be the subset of $\mathcal{R}$ that end with an environment state.

$$r : e_0 \xrightarrow{\alpha_0} e_1 \xrightarrow{\alpha_1} e_2 \xrightarrow{\alpha_2} e_3 \ \cdots \xrightarrow{\alpha_{u-1}} e_u \ \ldots \tag{3.3}$$

The behaviour of the environment is represented by a state transformer function [8] which maps a run ending with the action of an agent to a set of possible environment states (3.4)[†]. The next state of an environment is determined by the action performed by the agent and the history of the current run. There is non-determinism in the state transition of the environment. All runs must terminate finally, either because the agent does not do any action, or because there are no possible successor states, i.e. $\tau(r) = \emptyset$. An environment Env is a triple $Env = \langle E, e_0, \tau \rangle$, where E is a set of environment states, $e_0 \in E$ is an initial state, and τ is a state transformer function.

$$\tau : \mathcal{R}^{Ac} \to \wp(E) \tag{3.4}$$

Agents are modelled [32] as functions which map runs ending with an environment state to actions (3.5). An agent selects its next action based on the history of the run. Agents are deterministic. Let $\mathcal{AG}$ be the set of all agents.

$$Ag : \mathcal{R}^E \to Ac \tag{3.5}$$

[†]The $\wp$ symbol is the usual notation for the subset of the environment states that contains the possible outcome states of the run.

A system is a pair containing an agent and an environment: $\langle Ag, Env \rangle$. The set of terminated runs of agent Ag in environment Env is $\mathcal{R}(Ag, Env)$.

Two agents Ag_1 and Ag_2 are behaviourally equivalent in environment Env if and only if $\mathcal{R}(Ag_1, Env) = \mathcal{R}(Ag_2, Env)$. Two agents Ag_1 and Ag_2 are behaviourally equivalent if and only if they are behaviourally equivalent in all environments.

A relation between two sets of states is not very convenient to specify the problem to be solved by an agent, because agents are autonomous, and they are embedded in their environment. Often, the outcome of the actions of the agent cannot be described with a binary value (acceptable if it is in the result set and not acceptable if it is not in the result set), because the agent is expected to produce the "best" result in the given circumstances. This can be specified with a preference ordering of the possible outcomes.

One way of specifying the problem to be solved by an agent is a utility function (3.6) which assigns a real value to every environment state. The higher the utility, the better. The task of the agent is to bring about states that maximize utility. This simple specification method assigns utilities to current state only, and it does not take into account how the agent achieved the state and what the side effects were. To avoid this problem, the task of the agent can be specified with a function which assigns a utility to runs themselves (3.7).

$$u : E \to \mathbb{R} \tag{3.6}$$

$$u : \mathcal{R} \to \mathbb{R} \tag{3.7}$$

If the specification of the problem to be solved by an agent is a preference ordering, then the behaviour of an agent cannot be described by a binary value (i.e. we cannot tell if the autonomous agent correctly solves the problem or not). Instead, there are agents which solve the problem "better" than other agents, and there are optimal agents. $P(r|Ag, Env)$ denotes the probability that run r occurs when agent Ag is placed in environment Env. The sum of the probabilities of all possible runs is 1 (3.8). The optimal agent Ag_{opt} in an environment Env is the one that maximizes expected utility (3.9).

$$\sum_{r \in \mathcal{R}(Ag, Env)} P(r|Ag, Env) = 1 \tag{3.8}$$

$$Ag_{opt} = arg \max_{Ag \in \mathcal{AG}} \sum_{r \in \mathcal{R}(Ag, Env)} u(r) P(r|Ag, Env) \tag{3.9}$$

Equation (3.9) defines the properties of the optimal agent, but the agent is defined with a function (3.5) which may not be implemented on a real computer. Russell and Subramanian [32] introduced the notion of bounded optimal agents to address this issue. Let $\mathcal{AG}_m$ denote the subset of $\mathcal{AG}$ that can be implemented on a machine m (3.10). The optimal agent Ag_{opt} in an environment Env among those agents that can be implemented on a machine m is the one that maximizes expected utility (3.11).

$$\mathcal{AG}_m = \{Ag \mid Ag \in \mathcal{AG} \text{ and } Ag \text{ can be implemented on } m\} \tag{3.10}$$

$$Ag_{opt} = arg \max_{Ag \in \mathcal{AG}_m} \sum_{r \in \mathcal{R}(Ag, Env)} u(r) P(r|Ag, Env) \tag{3.11}$$

There are autonomous agent applications, where it is more convenient to talk about tasks in terms of "goals to be achieved" rather than utilities. In these cases the specification of the problem to be solved by the agent is a predicate task specification. In a predicate task specification the utility function is a predicate over runs, i.e. the range of the utility function $u : \mathcal{R} \to \mathbb{R}$ is the set $\{0, 1\}$. A run $r \in \mathcal{R}$ satisfies the specification u if $u(r) = 1$, and fails to satisfy the specification otherwise. Ψ denotes a predicate specification, and $\Psi(r)$ indicates that run $r \in \mathcal{R}$ satisfies Ψ.

A task environment is a pair $\langle Env, \Psi \rangle$, where Env is an environment, and $\Psi : \mathcal{R} \to \{0, 1\}$ is a predicate over runs. $\mathcal{TE}$ is the set of all task environments. $\mathcal{R}_\Psi(Ag, Env)$ denotes the set of all runs of an agent Ag in an environment Env that satisfy Ψ in the task environment $\langle Env, \Psi \rangle$ (3.12).

$$\mathcal{R}_\Psi(Ag, Env) = \{r \mid r \in \mathcal{R}(Ag, Env) \text{ and } \Psi(r)\} \tag{3.12}$$

An agent Ag succeeds in the task environment $\langle Env, \Psi \rangle$ if equation (3.13) is satisfied, i.e. Ag succeeds in $\langle Env, \Psi \rangle$ if every run of Ag in Env satisfies specification Ψ. In this case the agent is strictly successful.

$$\mathcal{R}_\Psi(Ag, Env) = \mathcal{R}(Ag, Env) \tag{3.13}$$

An alternative definition of success is that the agent succeeds if at least one run of the agent satisfies Ψ (3.14). In this case the agent is potentially successful.

$$\exists r \in \mathcal{R}(Ag, Env) \text{ such that } \Psi(r) \tag{3.14}$$

Another definition of success that takes into account the non-deterministic nature of the state transformer function τ is the probability $P(\Psi|Ag, Env)$ that

Ψ is satisfied by Ag in Env (3.15) The success of an agent is the probability that the specification Ψ is satisfied by the agent. In this case the agent is successful with a probability.

$$P(\Psi|Ag, Env) = \sum_{r \in \mathcal{R}_\Psi(Ag, Env)} P(r|Ag, Env) \tag{3.15}$$

The two most common types of predicate task specifications are achievement task specifications and maintenance task specifications. Achievement tasks 'achieve state of affairs φ' and maintenance tasks 'maintain state of affairs ψ'.

In an achievement task, the agent is required to bring the environment into one state of a set of goal states $\mathcal{G}$. A task specified by a predicate Ψ is an achievement task if $\Psi(r)$ is true just in case one or more of $\mathcal{G}$ occur in r. An agent is successful if every run of the agent in the environment results in one of the states $\mathcal{G}$. Formally, the task environment $\langle Env, \Psi \rangle$ specifies an achievement task if and only if there is some set $\mathcal{G} \subseteq E$ such that for all $r \in \mathcal{R}(Ag, Env)$, the predicate $\Psi(r)$ is true if and only if there exists some $e \in \mathcal{G}$ such that $e \in r$.

A task environment with specification Ψ is a maintenance task environment if we can identify some subset $\mathcal{B}$ of environment states, such that $\Psi(r)$ is false if any member of $\mathcal{B}$ occurs in r, and true otherwise. Formally, $\langle Env, \Psi \rangle$ is a maintenance task environment if there is some $\mathcal{B} \subseteq E$ such that $\Psi(r)$ if and only if for all $e \in \mathcal{B}$, we have $e \notin r$ for all $r \in \mathcal{R}(Ag, Env)$.

More complex tasks might be specified by combinations of achievement and maintenance tasks, like e.g. "achieve any one of states $\mathcal{G}$ while avoiding all states $\mathcal{B}$".

4. Multi-agent systems

If there are more than one agent in the same environment and they act on the real world, then sometimes the real world might impose restrictions on their activities, for example because two agents want to use the same resource at the same time. In this case, the agents have to coordinate their plans. Even if there is no conflict in the real world, the agents might want to distribute the task allocations among themselves, and they must interact with each other. The interactions are even more complex when there is conflict in the real world between the activities of two groups of agents. In this case the groups of agents have to coordinate among themselves how to interact with the other group.

When several agents act on the environment, then their actions may depend on the actions of the other agents. If one agent makes a choice, then the other agent is already restricted and has to make a choice depending on the choice of

the other agent. In an ideal situation, the different agents have preference (3.6) for the same state and all other states are less preferable for all of the agents. A somewhat less ideal, but still very good situation is when agents can still find a state which is most preferable for all the agents, but there are other states which give the same utility value for all the agents. In this case agents can select one of these preferable states, but they must agree which one, because if an agent deviates from this state towards another most preferable state, then none of the agents achieve the most preferred state. It is also possible that there are more than one states with which agents are all satisfied and do not want to deviate from it if the others do not deviate, however one of these preferable states may be better that the other one. The multi-agent system is in a Nash equilibrium, if no agent has the incentive to unilaterally deviate from the preferable state. The concept of agent preferences (represented with utility functions) is in line with game theory, which is an accepted foundation for multi-agent systems [29].

The efficiency of the agent system can be measured as a combination of the utility functions of all of the agents. A simple efficiency measure is the sum of the utilities of all the agents and according to this measure an agent system is in sum optimal state if the sum of the utilities is maximal. An agent system is in a Hicks optimal state, if the utility is maximized for all of the agents in the agent system. An agent system is in Pareto optimal state, if it satisfies more or less all of the agents and in all other states at least one agent's utility function gives smaller value if at least another agent's utility function gives higher value. The Hicks optimal state cannot always be achieved. The sum optimal and Pareto optimal state may not be equilibrium state, if at least one agent might achieve better utility by deviating from the optimal state.

If the agents are maximising their utilities individually, then it may not lead to an optimal situation. This is known as the "price of anarchy" [30]. The price of anarchy is the ratio between the sum of the utilities of an equilibrium situation and the sum of the utilities of the optimal situation. An example of this is the prisoner's dilemma [28] in which the equilibrium is not optimal.

In order to get to the desired state, the agents have to coordinate their actions by exchanging messages. The messages are exchanged similarly to usual network communication protocols which are governed by protocol rules so that the participating partners can get to some useful result and are not locked in for example a deadlock. Agent interaction protocols build on communication protocols and strive to ensure for example community level results [33]. It is expected that an agent interaction protocol guarantees that agents eventually get to some agreement and this agreement leads to either sum, Hicks or Pareto optimal state. Participating in agent interaction protocols must be Nash equilibrium behaviour for the participating agents, i.e. all of the agents must be interested in keeping to the protocol rules which must be simple enough so

that agents can easily determine the optimal strategy. Multi agent systems are usually distributed, and usually there is no centralized node, and this must be the case for agent interaction protocols as well.

5. Multi-agent problem solving

Multi-agent systems offer a promising software engineering approach for developing applications in complex domains. In these domains the "globally optimal solution" cannot be provided by a single software entity. There may be different reasons for this. If the problem is computationally hard, then the optimal solution cannot be computed in time. For example the complete and optimal algorithm for multi-agent path finding (MAPF) [34] can provide solution for about eight robots in a real-world warehouse, but big package delivery warehouses of today work with hundreds of robots [49]. Sometimes the users would not accept a single central solution. For example if a single central entity could compute and assign the "globally" optimal routes for the vehicles in a city, then passengers would feel like in an utopia town, because they would prefer to optimise their own routes instead of the global optimum. In these complex domains we can specify the subtasks of the global problem to be solved, we can assign the subtasks to the agents, and we expect that the community of the agents provide the global solution.

The set of the utility functions of the agents of a multi-agent system can be regarded as the specification of the subtasks of the global task to be completed by the multi-agent system. The individual agents provide their solutions as described in Section 3. The solution provided by the multi-agent system is the aggregation of the solutions provided by the individual agents. It is difficult to tell what the solution provided by the multi-agent system will be.

Game theory describes the behaviour of multi-agent systems well in most of the cases. The game theory model involves a solution concept [13] which forecasts the solution (or the possible solutions) of the problem if the agents are rational. The most common solution concept is the equilibrium concept. The system is in equilibrium, if none of the agents can change to another behaviour to achieve better results from its own point of view, assuming that the other agents do not change their behaviour. According to the game theory model, the solution provided by a multi-agent system will be an equilibrium, if it exists. We would like that the multi-agent system achieves the optimal solution. The individually optimising agents get to the equilibrium which is worse than the optimum in most of the cases. This is expressed by the price of anarchy concept as seen in Section 4.

The above reasoning of the game theory solution concept predicts that the rational behaviour of the selfishly optimising agents leads to the equilibrium.

However classic game theory models assume an idealistic situation: all the agents know what the equilibrium is, all the agents know what other agents are doing, and all the agents know what their role is in the equilibrium. In addition, classic game theory does not investigate how this idealistic situation emerges. In accordance with the basic theory of multi-agent systems [48], the agent behaviour goes in cycles: the agents perceive their environment (possibly communicating with other agents), decide what action to perform, and then perform the action. This means that the model of multi-agent systems has to investigate the dynamic behaviour of the system over time. We cannot be sure that multi-agent systems get to the equilibrium through the feedback cycles, and they stay in the equilibrium. Moreover, the assumption that an agent can only reason about its own behaviour, and the other agents do not change their behaviour, is too strong an assumption.

Let us take the example of the Stackelberg game of [36] shown in *Figure* 1.

	Left	Right
Up	1 , 0	3 , 2
Down	2 , 1	4 , 0

Figure 1. An example game.

The utility of the row player is the first figure, and the utility of the column player is the second figure in each cell. According to classic game theory reasoning, the row player must always select Down, because whatever the selection of the column player, the row player is better off with Down than with Up. There is no clear strategy for the column player, but its utility will be at least 0 whatever action it selects. If it selects Left, then the best outcome might be 1. If it selects Right, then the best outcome might be 2. So the column player is likely to select Right. This classic rational game theory reasoning results in action pair (Down,Right) of the agents with $4 + 0 = 4$ global utility.

If the game in *Figure* 1 is played repeatedly, then the column player might notice that the row player keeps playing Down, and the column player might be better off with Left which is a better action for him, and unfortunately worse for the row player. If, in response to the action of the column player, the row player starts repeatedly playing Up, then the column player might notice that Right would be a better action. The two players will end up with a repeated (Up,Right) play, and they would not be willing to deviate from this, because after a while they would end up here again. This rational reasoning of the repeated game results in (Up,Right) with $3 + 2 = 5$ global utility, which is actually happens to be the global optimum.

The above example shows that rational behaviour in dynamic games might result in other solution than the outcome from the equilibrium solution concept of the one shot game.

6. Complex multi-agent system

We study the problem solving of complex multi-agent systems in the traffic routing application where autonomous vehicles want to find their way in a city. This domain is interesting for the main message of this paper, because it demonstrates that the existence of the static game theory equilibrium may not guarantee a stable solution in a real-world setting. We model the dynamic real-world setting with the online routing game model discussed later in this section. From computer science point of view, the road traffic is a large-scale and open multi-agent system that tries to solve the *routing problem*. The routing problem is a network with traffic flows going from a source node to a destination node. The vehicles of the traffic flows continuously enter the network at the sources, they choose their routes to their destination, and quit the network at the destination. The traffic is routed in a congestion sensitive manner. If more vehicles enter a road, then their travel time will be longer on that road. The participants of the traffic want to minimise their travel time.

Complex multi-agent systems may not stay in a static equilibrium, however in many cases they converge to the static equilibrium. The ϵ-approximate equilibrium captures the notion of convergence to a static equilibrium [9]. The system converges to the ϵ-approximate equilibrium, if the more times the game is repeated, then the more agents experience travel times not more than $1 + \epsilon$ times the travel time in the static equilibrium. Many dynamical systems do not necessarily converge to a fixed point like the equilibrium. In this case, the dynamic solution concept may be based on a set of system states, called sink states [10] or attraction states which are similar to the attractors of dynamic systems [21]. Sooner or later the system enters one of the sink states, and then it never leaves the set of sink states. This is also called attracting equilibrium, if the sink set attracts the system. This kind of dynamic solution concept is in the focus of current research [24, 25, 23]. Although an equilibrium is the outcome of rational behaviour, agents may sometimes come to another solution. As [19] write: "limiting ourselves to equilibria as a reference point could lead us to qualitatively incorrect conclusions about system behaviour".

The online routing game model [37, 38] investigates the traffic routing application of autonomous vehicles, and in particular how the traffic flows of autonomous vehicles evolve over time. The model contains elements of the routing game model [31], the queuing model [5] and the concept of online mechanisms [26].

The *online routing game model* is the sextuple $< t, T, G, c, r, k >$, where $t = \{1, 2, ...\}$ is a sequence of time steps, T time steps give one time unit (e.g. one minute), G is a directed multi-graph representing the road network, c is the cost function of G with c_e for each edge e of G, r is a vector of flows, and $k = (k^1, k^2, ...)$ is a sequence of decision vectors $k^t = (k_1^t, k_2^t, ...)$ made in time step t.

Edges have FIFO property, and there is a minimum following distance gap_e on the edges which corresponds to the maximum capacity in the queuing model. The cost function maps the flow $f_e(\tau)$ (that enters the edge e at time τ) to the travel time on the edge. The cost c_e for the agent entering the edge e at time step t is never less than the remaining cost of any other agent already on edge e at time step t increased with the time gap gap_e, which is specific to edge e. This ensures the FIFO policy. The value of the flow $f_e(\tau)$ is the number of agents that entered the edge e between $\tau - T$ (inclusive) and τ (non-inclusive). If two agents enter an edge exactly at the same time τ, then one of them (randomly selected) suffers a delay gap_e, which is part of its cost on edge e, and its remaining cost is determined at the delayed time, so its cost on edge e will be $gap_e + c_e(f_e(\tau + gap_e))$. If several agents enter an edge at the same time, then they are randomly ordered with a delay gap_e. The cost functions are nonnegative, continuous, and nondecreasing. The cost functions have a constant part which does not depend on the inflow to the edge, and a variable part which varies with the inflow to the edge. The variable part is not known to any agent of the model. The agents learn the variable cost only when an agent exits an edge, and broadcasts its cost to the other agents.

The decision made by the agent of the flow r_i in time period t is k_i^t. The decision k_i^t is how the agent is routed on a single path of the paths leading from s_i to t_i. The actual cost of a path $p = (e_1, e_2, e_3, ...)$ for a flow starting at time τ is

$$c_p(\tau) = c_{e_1}(f_{e_1}(\tau)) + c_{e_2}(f_{e_2}(\tau + c_{e_1}(f_{e_1}(\tau)))) + c_{e_3}(f_{e_3}(\tau + c_{e_1}(f_{e_1}(\tau)) + \\ + c_{e_2}(f_{e_2}(\tau + c_{e_1}(f_{e_1}(\tau)))))) + ...,$$

because the actual cost of an edge is determined at the time when the flow enters the edge.

Intention-aware Online Routing Games [38] are a special type of routing games, where the agents can perceive their environment as described above, and in addition they also receive aggregated information about the intentions of other agents in the system. In order to facilitate the agents to make predictions, and to include the future state of the traffic in their decisions, intention-aware prediction methods were proposed. In the intention-aware prediction methods, the agents communicate their intentions to a service. The role of the service is to support stigmergic communication among the agents, and it

is often implemented with bio-inspired techniques. The service aggregates the data about the agent collective, and it sends a feedback to the agents [4]. The *intention-aware* [44] and the *intention propagation* [3] approaches are based on this scheme. The coordination mechanism provided by these schemes can scale with the complexity of real-world application domains.

It is proved [37] that if the agents of the online routing game selfishly try to minimise their cost computed from the currently observable cost (real-time data), then equilibrium is not guaranteed, although a static equilibrium exists. "Single flow intensification" may also happen: agents subsequently entering the online routing game select alternative faster routes, and they catch up with the agents already on route, and this way they cause congestion. As a result, sometimes some online routing games may produce strange behaviour, and the agents may be worse off by exploiting real-time information than without exploiting real-time information.

Although intention-aware prediction improves system behaviour, it is proved [38] that the above problems of online routing games are possible even if intention-aware prediction is applied. However, it is proved [40] that in a small but complex enough network of the Braess paradox [2], where there is only one source–destination pair, the agents might just slightly be worse off in the worst case with real-time data and prediction. It is also proved [42] that in the Braess network of [40], the system converges to the static equilibrium within a relatively small threshold. The conjecture in [41] says that the system converges to the static equilibrium in bigger networks as well, if simultaneous decision making is prevented. This conjecture neither has been proved nor refuted analytically.

Investigations of more complex networks indicate that the prediction method applied by the prediction service has a great impact on the behaviour of the system. The formal description of the algorithms of two intention-aware prediction methods were presented in [43]: the detailed prediction method and the simple prediction method.

The *detailed prediction method* takes into account all the intentions already submitted to the service, then it computes what will happen in the future if the agents execute the plans assigned by these intentions, and then it computes for each route in the network the predicted travel time by taking into account the predicted future travel times for each road of the route. The prediction algorithm used in [44] is close to this detailed prediction method, but the main difference is that the prediction algorithm of [44] uses probabilistic values, while the detailed prediction method is deterministic.

The *simple prediction method* also takes into account all the intentions already submitted to the service, and then it computes what will happen in the future if the agents execute the plans assigned by these intentions. However, when the simple prediction method computes for each route in the network the

predicted travel time, then it takes into account only that travel time prediction for each road which was computed at the last intention submission. This way, the simple prediction method needs a little bit less computation. The simple prediction method is a kind of approximation and it does not try to be an exact prediction of the future. As time goes by, if no new prediction is generated for a road, then the simple prediction method "evaporates" the last prediction for that road, like the bio-inspired technique of [3].

It turned out in [43] that the detailed prediction method did not lead to a better behaviour in the experiments than the simple prediction method. It seems that the more detailed knowledge of the future from the current situation and intentions may not be better for the whole system.

Unfortunately, the results of the online routing game model are mainly negative: exploiting real-time information, in some networks, may sometimes result in unwanted system behaviour. This is observed in real traffic situations as well, so the online routing game model points out a real problem that needs to be solved.

The intention-aware online routing model has been used in empirical simulation investigations, and the analytic proofs are still needed. The model is rather new and the attention of theoretical researchers has to be attracted. Nevertheless, the empirical investigations pointed out that if the autonomous vehicles do not use prediction, or if they use an improper prediction method, then it may create worse traffic than the static equilibrium would indicate.

7. Conclusion

The autonomous agent and multi-agent paradigm is perceived as one of the best ways to understand, design and develop software systems that solve complex problems. The classical theory of programming defines how we specify a problem to be solved, and how we can determine if a software program solved the specified problem or not. On this basis, the correctness of software programs can be determined.

In this paper we have discussed how the problem to be solved by autonomous agents and multi-agent systems can be specified at the abstract level. The task of a single agent can be specified with a utility function which assigns a real value to every environment state. Multi-agent systems are expected to be applied in complex domains where we can specify the subtasks of the global problem to be solved, and we can assign the subtasks to agents. The set of the utility functions of the agents of a multi-agent system can be regarded as the specification of the subtasks of the global task to be completed by the multi-agent system.

We have discussed how we can determine if autonomous agents and multi-agents systems solve the specified problem. The autonomous agent is embedded in its environment, and the autonomous agent is expected to perform "well" if there are unforeseen changes in the environment. The solution provided by the multi-agent system is the aggregation of the solutions provided by the individual agents. According to the game theory model, the solution provided by a multi-agent system will be an equilibrium, if it exists. Complex multi-agent systems may not stay in a static equilibrium, and many dynamical systems do not necessarily converge to a fixed point like the equilibrium. In this case, the dynamic solution concept may be based on a set of system states, and the multi-agent system may fluctuate around an equilibrium.

We have studied the online routing game model to show that the classical solution concepts of static games may not be applied in dynamic real-world settings, and complex solution concepts are needed. Selfish optimisations of autonomous agents may produce unwanted global behaviour, and the system may not produce the expected solution. In order to improve the quality of the solution provided by multi-agent systems, the system needs to be extended with additional constructs. The online routing game model shows that agreement technologies, like intention-awareness, are needed to facilitate the problem solving of multi-agent systems. Novel multi-agent system engineering methods, which include the environment engineering as well, need to be developed [20] to improve the behaviour of multi-agent systems.

Proving the correctness of autonomous agents and multi-agent systems is still an open and ongoing research topic. For example, the rational verification approach of [11] checks the temporal logic properties that will hold in a system when agents of a multi-agent system behave rationally. However, agents may not have enough information to make rational decisions, and as we have seen, in some complex domains the rational behaviour may not induce a clear solution concept.

References

[1] **Bond, A.H. and L. Gasser,** (Eds.) *Readings in Distributed Artificial Intelligence*, Morgan Kaufmann Publishers, Inc., San Mateo, California, 1988.

[2] **Braess, D.,** Über ein Paradoxon der Verkehrsplanung, *Unternehmensforschung*, **12** (1968), 258–268.

[3] **Claes, R., T. Holvoet and D. Weyns,** A decentralized approach for anticipatory vehicle routing using delegate multi-agent systems, *IEEE Transactions on Intelligent Transportation Systems*, **12** (2011), 364–373.

[4] **Claes, R. and T. Holvoet,** Traffic coordination using aggregation-based traffic predictions, *IEEE Intelligent Systems*, **29** (2014), 96–100.

[5] **Cominetti, R., J. Correa and N. Olver,** Long term behavior of dynamic equilibria in fluid queuing networks, in: Eisenbrand, F., and Koenemann, J., (Eds.) *Proc. of Integer Programming and Combinatorial Optimization (IPCO 2017)* (Waterloo, ON, Canada, 2017), Springer International Publishing, *Lecture Notes in Computer Science, volume 10328*, 2017, 161–172.

[6] **Edelman, B., M. Ostrovsky and M. Schwarz,** Internet advertising and the generalized second-price auction: Selling billions of dollars worth of keywords, *American Economic Review*, **97(1)** (2007), 242–259.

[7] **Ehrig, H. and B. Mahr,** *Fundamentals of Algebraic Specification 1: Equations and Initial Semantics*, Springer-Verlag, Berlin, 1985.

[8] **Fagin, R., J.Y. Halpern, Y. Moses and M. Vardi,** *Reasoning About Knowledge*, MIT Press, Cambridge, MA, 1995.

[9] **Fischer, S., and B. Vöcking,** On the evolution of selfish routing, in: Albers S., Radzik T., (Eds.) *Proc. of the 12th European Symposium on Algorithms* (*ESA '04*) (Bergen, Norway, 2004), Springer, Berlin, Heidelberg, *Lecture Notes in Computer Science, volume 3221*, 2004, 323–334.

[10] **Goemans, M., V. Mirrokni, and A. Vetta,** Sink equilibria and convergence, in: Tardos, É., (Ed.) *Proc. of the 46th Annual IEEE Symposium on Foundations of Computer Science* (*FOCS 05*) (Pittsburgh, PA, SA, 2005), IEEE Computer Society, 0-7695-2468-0/05, 2005, 142–151.

[11] **Gutierrez, J., M. Najib, G. Perelli and M. Wooldridge,** Automated temporal equilibrium analysis: Verification and synthesis of multi-player games, *Artificial Intelligence*, **287** (2020), 103353.

[12] **Guttag, J., E. Horowitz and D. Musser,** Abstract data types and software validation, *Communications of the ACM*, **21(12)** (1978), 1048–1064.

[13] **Halpern, J.Y. and Y. Moses,** A procedural characterization of solution concepts in games, *Journal of Artificial Intelligence Research*, **49** (2014), 143–170.

[14] **Hoare, C.A.R.,** An axiomatic basis for computer programming, *Communications of the ACM*, **12(10)** (1969), 576–580.

[15] **Hoare, C.A.R.,** Communicating sequential processes, *Communications of the ACM*, **21(8)** (1978), 666–677.

[16] **Jennings, N.R., E.H. Mamdani, J.M. Corera, I. Laresgoiti, F. Perriolat, P. Skarek and L.Z. Varga,** Using Archon to develop real-world DAI applications, *IEEE Expert*, **11(6)** (1996), 64–70.

[17] **Jennings, N.R., L. Moreau, D. Nicholson, S.D. Ramchurn, S. Roberts, T. Rodden and A. Rogers,** Human-agent collectives, *Communications of the ACM*, **57(12)** (2014), 80–88.

[18] **Kifor, T., L.Z. Varga, J. Vázquez-Salceda, S. Álvarez, S. Willmott and S. Miles,** Provenance in agent-mediated healthcare systems, *IEEE Intelligent Systems*, **21(6)** (2006), 38–46.

[19] **Kleinberg, R.D., K. Ligett, G. Piliouras and É. Tardos,** Beyond the Nash equilibrium barrier, in: *Proc. of Innovations in Computer Science – ICS 2010* (Tsinghua University, Beijing, China, 2011), Tsinghua University Press, 2011, 125–140.

[20] **Mascardi, V., D. Weyns, A. Ricci, C.B. Earle, A. Casals, M. Challenger, A. Chopra, A. Ciortea, L.A. Dennis, Á.F. Díaz, A. El Fallah-Seghrouchni, A. Ferrando, A., L. Fredlund, E. Giunchiglia, Z. Guessoum, Z., A. Günay, K. Hindriks, C.A. Iglesias, B. Logan, T. Kampik, G. Kardas, V.J. Koeman, J.B. Larsen, S. Mayer, T. Méndez, T., J.C. Nieves, V. Seidita, B.T. Teze, L.Z. Varga and M. Winikoff,** Engineering multi-agent systems: State of affairs and the road ahead, *ACM SIGSOFT Software Engineering Notes*, **44(1)** (2019), 18–28.

[21] **Milnor, J.,** On the concept of attractor, *Commun. Math. Phys.*, **99** (1985), 177–195.

[22] **Moreau, L., P. Groth, S. Miles, J. Vázquez-Salceda, J. Ibbotson, S. Jiang, S.J. Munroe, O.F. Rana, A. Schreiber, V. Tan and L.Z. Varga,** The provenance of electronic data, *Communications of the ACM*, **51(4)** (2008), 52–58.

[23] **Omidshafiei, S., C. Papadimitriou, G. Piliouras, K. Tuyls, M. Rowland, J.-B. Lespiau, W.M. Czarnecki, M. Lanctot, J. Perolat and R. Munos,** α-rank: Multi-agent evaluation by evolution, *Scientific Reports*, **9**, Springer Science and Business Media LLC, (2019).

[24] **Papadimitriou, C. and G. Piliouras,** From Nash equilibria to chain recurrent sets, in: Sudan, M., (Ed.) *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science - ITCS 16* (Cambridge, Massachusetts, USA, 2016), Association for Computing Machinery, New York, NY, United States, 2016, 227–235.

[25] **Papadimitriou, C. and G. Piliouras,** Game dynamics as the meaning of a game, *ACM SIGecom Exchanges*, **16(2)**, Association for Computing Machinery (ACM), (2018), 53–63.

[26] **Parkes, D.C.,** Online mechanisms, in: Nisan, N., Roughgarden, T., Tardos, É., and Vazirani, V. V., (Eds.) *Algorithmic Game Theory*, Cambridge University Press, 2007, 411–439.

[27] **Ramchurn, S.D., T.D. Huynh, F. Wu, Y. Ikuno, J. Flann, L. Moreau, J.E. Fischer, W.C. Jiang, T. Rodden, E. Simpson, S. Reece, S. Roberts and N.R. Jennings,** A disaster response system based on human-agent collectives, *Journal of Artificial Intelligence Research*, **57** (2016), 661–708.

[28] **Rapoport, A. and A.M. Chammah,** *Prisoner's Dilemma: A Study in Conflict and Cooperation*, The University of Michigan Press, Ann Arbor, 1965.

[29] **Rosenschein, J.S.,** Multiagent systems, and the search for appropriate foundations, in: Ito, Jonker, Gini, and Shehory (Eds.) *Proc. of the 12th International Conference on Autonomous Agents and Multiagent Systems* (Saint Paul, Minnesota, USA, 2013), International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS), ACM Digital Library, 2013, 5–6.

[30] **Roughgarden, T. and É. Tardos,** How bad is selfish routing?, *Journal of the ACM*, **49(2)** (2002), 236–259.

[31] **Parkes, D.C.,** Routing games, in: Nisan, N., Roughgarden, T., Tardos, É., and Vazirani, V. V., (Eds.) *Algorithmic Game Theory*, Cambridge University Press, 2007, 461–486.

[32] **Russell, S. and D. Subramanian,** Provably bounded-optimal agents, *Journal of Artificial Intelligence Research*, **2** (1994), 575–609.

[33] **Sandholm, T.W.,** Distributed Rational Decision Making, in: Weiss, G., (Ed.) *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, Cambridge, MA, USA, MIT Press, 1999, 201–258.

[34] **Sharon, G., R. Stern, A. Felner and N.R. Sturtevant,** Conflict-based search for optimal multi-agent pathfinding, *Artificial Intelligence*, **2019** (2015), 40–66.

[35] **Shoham, Y. and K. Leyton-Brown,** *Multiagent Systems – Algorithmic, Game-Theoretic, and Logical Foundations*, Cambridge University Press, New York, 2009.

[36] **Shoham, Y., R. Powers, and T. Grenager,** If multi-agent learning is the answer, what is the question? *Artificial Intelligence*, **171(7)** (2007), 365–377.

[37] **Varga, L.Z.,** Online routing games and the benefit of online data, in: Klügl, F., Vizzari, G., and Vokřínek, J. (Eds.) *Proc. of the 8th International Workshop on Agents in Traffic and Transportation* (*ATT 2014*), Paris, France, 2014, 2014, 88–95.

[38] **Varga, L.,** On intention-propagation-based prediction in autonomously self-adapting navigation, *Scalable Computing: Practice and Experience*, **16** (2015), 221–232.

[39] **Varga, L.Z.,** A game theory model for self-adapting traffic flows with autonomous navigation, in: McCluskey, T.L., Kotsialos, A., Müller, J.P., Klügl, F., Rana, O., Schumann, R., (Eds.) *Autonomic Road Transport Support Systems*, Autonomic Systems, Bâle, Basel, Switzerland, Birkhäuser Verlag, 2016, 13–28.

[40] **Varga, L.Z.,** Benefit of online real-time data in the Braess Paradox with anticipatory routing, in: Kounev, S., Giese, H., and Liu, J. (Eds.) *Proc. of the 2016 IEEE International Conference on Autonomic Computing* (*ICAC 2016*), Würzburg, Germany, 2016, IEEE Computer Society, 2016, 245–250.

[41] **Varga, L.Z.,** How good is predictive routing in the online version of the Braess Paradox? in: Kaminka, G. A., Fox, M., Bouquet, P., Hüllermeier, E., Dignum, V., Dignum, F. and van Harmelen, F. (Eds.) *Proc. of the 22nd European Conference on Artificial Intelligence* (*ECAI 2016*), The Hague, The Netherlands, 2016, IOS Press, 2016, 1696–1697.

[42] **Varga, L.Z.,** Equilibrium with predictive routeing in the online version of the Braess paradox, *IET Software*, **11** (2017), 165–170.

[43] **Varga, L.Z.,** Two prediction methods for intention-aware online routing games, in: Belardinelli, F., and Argente, E. (Eds.) *Multi-Agent Systems and Agreement Technologies. EUMAS 2017*, Springer International Publishing, **LNCS 10767** 2018, 431–445.

[44] **de Weerdt, M.M., S. Stein, E.H. Gerding, V. Robu and N.R. Jennings,** Intention-aware routing of electric vehicles, *IEEE Transactions on Intelligent Transportation Systems*, **17** (2016), 1472–1482.

[45] **Weiss, G.,** (Ed.) *Multiagent Systems, Second Edition*, MIT Press, Cambridge, MA, 2013.

[46] **Wittig, T.,** (Ed.) *ARCHON : an architecture for multi-agent systems*, Ellis Horwood, New York, 1992.

[47] **Wooldridge, M. and N.R. Jennings,** Intelligent agents: Theory and practice, *The Knowledge Engineering Review*, **10(2)** (1995), 115–152.

[48] **Wooldridge, M.,** *An Introduction to MultiAgent Systems – Second Edition*, John Wiley & Sons, Chichester, United Kingdom, 2009.

[49] **Wurman, P.R., R. D'Andrea and M. Mountz,** Coordinating hundreds of cooperative, autonomous vehicles in warehouses, *AI Magazine*, **29(1)** (2008), 9.

L. Z. Varga
ORCID ID: 0000-0001-8088-4528
Faculty of Informatics
Eötvös Loránd University
Budapest
Hungary
lzvarga@inf.elte.hu