Robotic Path Planning by Q Learning and a Performance Comparison with Classical Path Finding Algorithms

Phalgun Chintala, Rolf Dornberger, and Thomas Hanne
University of Applied Sciences and Arts Northwestern Switzerland, Basel/Olten, Switzerland
Email: thomas.hanne@fhnw.ch

***Abstract*—Q Learning is a form of reinforcement learning for path finding problems that does not require a model of the environment. It allows the agent to explore the given environment and the learning is achieved by maximizing the rewards for the set of actions it takes. In the recent times, Q Learning approaches have proven to be successful in various applications ranging from navigation systems to video games. This paper proposes a Q learning based method that supports path planning for robots. The paper also discusses the choice of parameter values and suggests optimized parameters when using such a method. The performance of the most popular path finding algorithms such as A* and Dijkstra algorithm have been compared to the Q learning approach and were able to outperform Q learning with respect to computation time and resulting path length.**

***Index Terms*—reinforcement learning, Q learning, robot navigation, path planning, path finding, shortest path**

I. Introduction

Path planning is a fundamental and critical task of moving robots. The main objective of path finding is to find a safe trajectory for the mobile robot to move it from the starting point to the ending point without any collision with obstacles. Finding paths with minimal time and energy consumption is always desired [1].

With emerging needs and growth of technology, robots have been employed for a large variety of applications ranging from aerial photography, bomb disposal, mining, nuclear applications to performing medical interventions. Depending on the environment, path finding can be categorized into 1) Global path planning; in a static and structured environment where the robot already knows the location of obstacles and a model of the environment; 2) Local path planning; in dynamic environments, where the robot explores the given environment by taking actions and then uses the information for path planning [2]. Dijkstra algorithm and A* algorithm are the two most popular algorithms for path planning in static environments which compute an optimal global path since the position of the obstacles is pre-known [1]. While the supervised learning algorithms and sequential search-based navigation approaches are relatively easy to implement and are effective to find an optimal path in static environments, most of the approaches fail in real world applications. Studies indicate that external influences such as noise or closely spacing of obstacles would also have a negative impact on the performance of such algorithms [1].

To tackle complexities and the unpredictable nature of the environments, heuristic path planning methods have emerged which emulate human like behavior-based characteristics. Literature indicates that approaches based on Artificial Neural Network (ANN), Genetic Algorithms (GA), Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), Wavelet, Fuzzy Logic (FL) have been frequently used to tackle such scenarios [3].

Reinforcement Learning (RL) is one such heuristic approach developed by mimicking the learning behavior of animals and humans – by interactions. It is about learning what to do and how to proceed to obtain maximum rewards from the environment. Another important aspect of RL is that it considers the whole problem of the environment and thus addresses how the local paths planned at every step could fit into the larger picture of the path planning. So the actions at every step may not only influence the rewards at that step, but also a set of subsequent rewards that would be obtained in future iterations by taking that step [4]. In its simplest form, RL can be treated as a sequence of Markov decision processes which capture the key aspects of a learning agent, i.e., reward, action, and goal. The performance of a RL algorithm hugely depends on two factors used in the strategy of the algorithm known as 1) exploration and 2) exploitation. Exploration refers to selecting an action with a probability value greater than zero in every state to learn about the environment. Exploitation is using the current knowledge of the agent to achieve good performance by selecting particular (e.g., greedy) actions [5].

Q learning is a promising off-policy variant of RL, where the value functions can be updated by hypothetical actions. This is the major difference when compared to RL, where the value functions can only be updated based on experience [6]. In recent times, algorithms based on Q learning have been successfully used for short-term and long-term planning and decision-making processes in autonomous navigation tasks with minimal assumptions.

Manuscript received January 14, 2022; revised April 1, 2022.
Corresponding author *: Thomas Hanne

doi: 10.18178/ijmerr.11.6.373-378

The algorithm in this paper uses a greedy policy, which means that the expected highest reward for a given action is chosen at every step.

The paper is structured as follows: A description of the problem and the suggested optimization method including chosen parameters will be given in Section II. Section III presents the results and a related discussion. Conclusions are presented in Section IV.

II. Description of the Optimization Problem

A. Problem Model

The problem model using the conventional algorithm-based approaches involves decomposing the given environment to a grid graph. The starting and the ending point need to be defined. The static locations of the obstacles need to be specified on the grid. The distances such as Manhattan distance or Chebyshev distance are used to compute the path of the robot on the grid. The objective function is based on minimizing distance and maximizing a reward value at each step [1][7]. The implementation and results of the classical path finding approaches are discussed in Section III of the paper.

For evaluating the Q learning in this paper, the environment has been simulated as discrete, non-overlapping grid, each grid position representing a state that the agent could take. The robot can move either in a straight line or diagonally to left, right, up, or down, a grid position can hence have eight possible neighbors which a robot can transit to. Fig. 1 shows a sample 20x20 grid, the yellow positions indicate the spaces into which the robot can navigate, and the dark positions represent the obstacles in the environment. In general, from the starting position SP to the ending position EP, several valid paths (at least one) are possible.

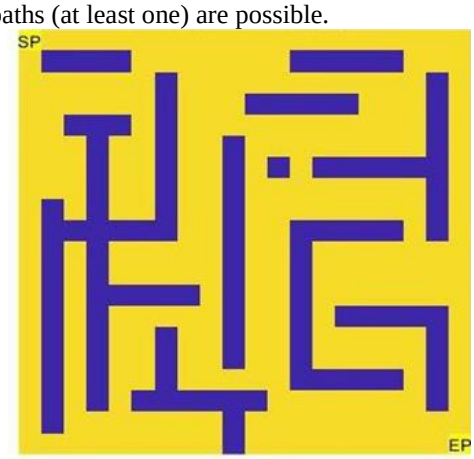

Figure 1. Robot environment.

Following [3][8][9][10][11] used for solving similar path planning problems, each decision step is modelled as a Markov Decision Process (MDP) and the resulting combination of all steps behave as a Markov Chain [8][10].

The MDP at every step can be represented by the quintuple set of {S, A, P_a, R_a, γ, π}, where:

- S denotes the set of all possible states s,
- A denotes the set of all possible actions a,
- P_a (s, s') is the probability that action a in state s at time t, will lead to a state s' at time t+1,
- R_a (s, s') is the immediate reward after the agent moves from state s to state s',
- γ is the discount factor with $0 < \gamma < 1$,
- π (a_t | s_t) denotes the policy of the agent depending on the action a and state s at time t,

The state and action spaces are finite and belong to the set of real numbers. The policy function π is a mapping from the state space to the action space. The set of these state-action pairs are stored in a table, so that the algorithm can learn from the consequence of previous actions in the future (exploitation) [5].

The robot interacts with the environment: From a given state s and a time t it chooses an action a from the possible set of actions based on a policy π to move to a state s' and receives a reward. Since the robot is limited at every time t by the set of actions and states from which it can select, these are two main constraints in the optimization problem, which will be further discussed in the coming sections.

B. Optimization Method

The main optimization objective is to choose a policy π that will maximize the rewards at the current step and also the expected sum of rewards from the future steps. The state value action function $Q^{\pi}(s, a)$ measures the sum of the rewards from state *s* after taking an action *a* following a policy π. The common idea in [3][8][9][10][11] involves the robot acting autonomously in a given environment and updating the policy in a way that $Q^{\pi}(s, a) \rightarrow Q^{*}(s, a)$, where $Q^{*}(s, a)$ represents the maximal reward that can be obtained by following a policy at that given state.

$$Q^{*}(s, a) = \max Q^{\pi}(s, a)$$

The update equation for Q learning at every time step *t* can then be formulated as:

$$Q^{t}(s, a) = Q^{t-1}(s, a) + \alpha[R(s, a) + \gamma \max Q^{'} (s^{'}, a^{'}) - Q^{t-1}(s, a)]$$

where:

- Q^{t} (s, a) denotes the updated new Q value in the table,
- Q^{t-1}(s, a) denotes the previously recorded Q value,
- α denotes the learning rate,
- R(s, a) denotes the immediate reward obtained,
- γ denotes the discount factor,
- max $Q^{'}$ ($s^{'}$, $a^{'}$) denotes the maximum expected reward,
- $Q^{'}$ ($s^{'}$, $a^{'}$) = E[$R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \gamma^4 R_{t+5} + \ldots\ldots$]

The algorithm gives the importance to the actions which can yield a maximum immediate reward, as the future rewards are discounted exponentially by a factor of γ ($0 < \gamma < 1$).

Algorithm: Q Learning

Choose parameters $\alpha \in (0,1]$, $\gamma \in (0,1)$;

Initialize Q(s, a), for all $s \in S$, $a \in A(s)$ arbitrarily; Initialize the current state *s* (starting position); Loop for every iteration:

Choose an action *a*, by using the greedy policy ε; Observe the reward $r \in R(s,a)$;

Observe the current transitioned state s';

Update the Q value in the table using the equation

$$Q^t(s, a) = Q^{t-1}(s, a) + \alpha[R(s, a) + \gamma \max Q'(s', a') - Q^{t-1}(s, a)]$$

Repeat until the *s* is the end position

The robot can use extensive exploration or a greedy strategy to maximize the reward values at every step. However, too much exploration could have a negative impact on the learning speed and time taken for planning the successive steps. Strategically, these could be used as the regularization parameters to prevent excessive exploration [12].

C. Parameter Selection

Learning Rate (α): The learning rate parameter α needs to be carefully selected to avoid a too quick convergence and thus to allow optimal learning rates in the environment. Selecting the α close to zero ($\alpha > 0$ and $\alpha <<1$) causes the learning to occur very slowly, which means that the algorithm will tend to be inefficient in navigation tasks where quick response rates are desired. Choosing α close to 1 causes the learning process to be very quick and the paths may not be optimal [4].

Discount factor (γ): The discount factor γ determines the importance of the consequences of the future actions and rewards. Choosing γ close to zero will make the algorithm short-sighted, allowing only local path planning with less regard to how these paths would fit into the overall solution. Choosing γ close to 1 makes the algorithm strive for higher long-term rewards and the chosen local paths can be far from optimal. γ values above 1 are excluded, as the agent can never converge to a solution as future rewards become infinite [13].

Q value initialization: The initial Q values can have a significant impact on the convergence of the algorithm. The initialized value (Q_i) when compared to a Q value in the future (Q_∞) will make the states already visited either more or less attractive. If $Q_i < Q_\infty$ it would make the agent explore less in the beginning as the unvisited states are less unattractive than the current state, which slows down the learning process. If $Q_i > Q_\infty$, the robot exhibits a systemic exploration behavior, as the unvisited states are more attractive than the visited ones [14].

Policy (ε): The policy selection is crucial in balancing the trade-off between exploration and exploitation. Choosing a greedy policy ε will enable the robot to always select the action with the highest estimated reward. This policy ensures that sufficient iterations are carried out. As a consequence, each action will be tried out multiple times, thus arriving at an optimal action. A disadvantage of choosing a greedy ε policy is that the algorithm favors the random and known actions equally. To counteract this, a softmax based ε policy has been used recently. A softmax policy assigns a weight to each of the actions based on their action-value estimate; hence, bad random actions are unfavorable and will be avoided [15] [16].

III. Results & Discussion

A. Q Learning

Two environments of different grid sizes 20x20 and 10x10 with different sets of obstacles have been setup in MATLAB to test the performance of the Q learning algorithm. For a given (x, y) on the grid the state no. can be expressed as:

$$\text{state no.} = (x-1) * \text{rowsize} + y$$

where x is the grid position in the x direction; y is the grid position in the y direction; rowsize is the length of the row.

All the obstacles have the dimension of one grid cell and hence can be arranged in walls, corners, or complex labyrinths. These simulations were only performed with static obstacles. On a 10x10 grid with obstacles not spaced very closely, as shown in Fig. 2, the algorithm converges in the optimal case to a final path within 32 iterations across 10 instances of the same simulation. For the next simulation, a labyrinth styled 20x20 grid as shown in Fig. 3, has been used, the algorithm converged after 62 iterations with a final step size of 26.

On a 20x20 grid, with the values of the Q initialized to zero ($Q_i < Q_\infty$) the average number of explored steps in the first five iterations were 302, compared to 335 when Q values were initialized with values from a normal distribution over 10 instances. The behavior remained consistent across different grid sizes, with the algorithm taking an average of 26 steps in the first five iterations, compared to 41 steps when initialized with non-zero random values.

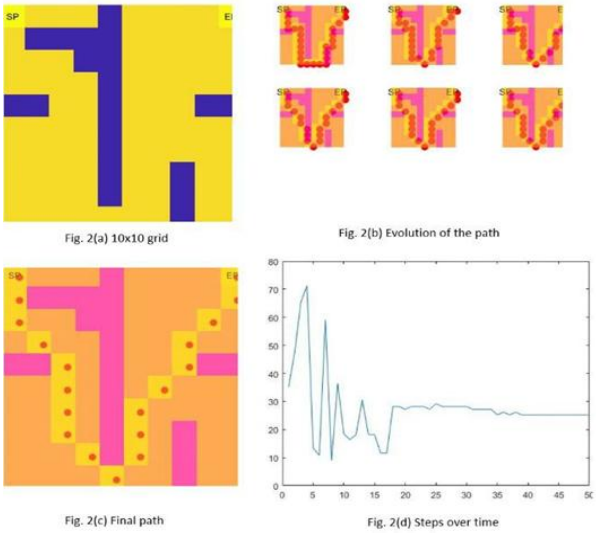

Figure 2. 10x10 grid simulation results.

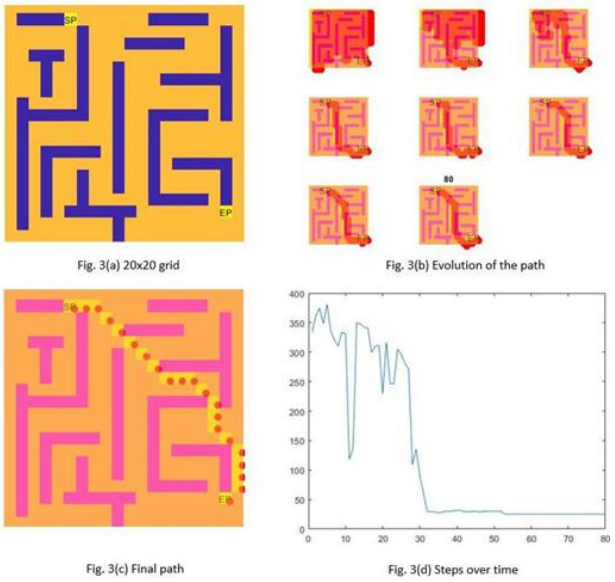

Figure 3. 20x20 grid simulation results.

As described in Section 2, choosing a reward discount factor ($\gamma > 1$) resulted in an infinite expected reward and the algorithm did not converge to a solution. The simulation has been performed over 10 instances and at every single instance the algorithm exhibited non-convergence as shown in Fig. 4(a). Fig. 4(b) shows the final path attained for a discount factor of $\gamma = 0.1$. Since the future rewards are far more discounted, the final path is far less optimal compared to that in Fig. 3(c).

The average time of convergence over 10 iterations with $\gamma = 0.9$ was evaluated with different learning rates over the two different grid sizes. In both cases, as shown in Table I, the time of convergence decreases with an increase in α. In the simulations with $\alpha = 0.1$, the algorithm has not yielded to an optimal global path.

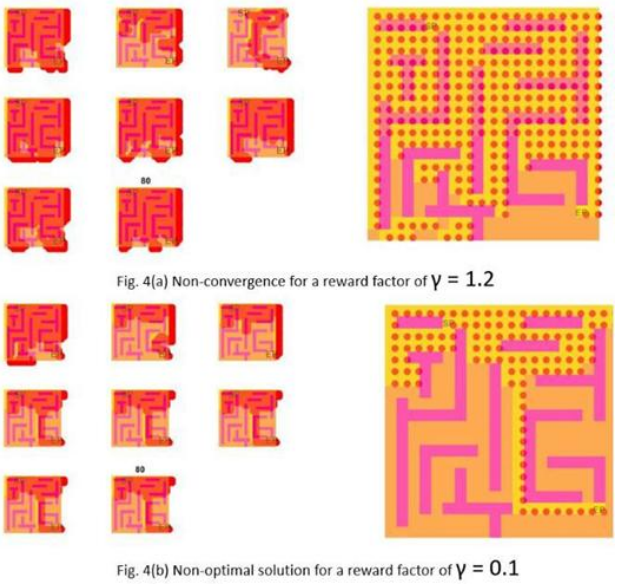

Figure 4. Results for different γ values.

TABLE I. TIME OF CONVERGENCE FOR DIFFERENT LEARNING RATES

Grid Size	($\alpha = 0.1$)	($\alpha = 0.5$)	($\alpha = 0.9$)
10x10	14.9 sec	10.8 sec	7.4 sec
20x20	97.1 sec	52.3 sec	23.2 sec

B. *Path Finding with A* and Dijkstra Algorithms*

In the following, the classical path finding approaches A* and Dijkstra are compared on the same environments (example grids with static obstacles) with the previously explained Q learning algorithms. The average convergence time and the final number of steps (path length) over 10 instances were considered the primary evaluation criteria in the comparison.

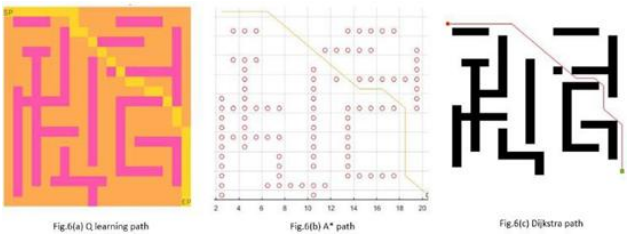

Figure 5. Comparison of paths resulting from Q learning, A* algorithm, and Dijkstra algorithm.

TABLE II. PERFORMANCE EVALUATION ON A 10X10 GRID

Grid Size(10x10)	Q - learning	A* algorithm	Dijkstra algorithm
Time	9.1 sec	1.85 sec	1.21 sec
Steps	18	7	9

TABLE III. TIME OF CONVERGENCE FOR DIFFERENT LEARNING RATES

Grid Size(20x20)	Q - learning	A* algorithm	Dijkstra algorithm
Time	14.8 sec	2.46 sec	1.71 sec
Steps	35	6	7

In a static environment, with different grid sizes and placement of obstacles, the performances of the A* and Dijkstra algorithms are superior to the Q learning approach proposed in the paper. A* algorithm, which is an expansion of Dijkstra's algorithm, presented the shortest path (7 steps on a 10x10 grid and 6 steps on a 20x20 grid) in every instance. A* is successful in decreasing the total number of states by presenting a heuristic estimation of the cost from the current state (including start position) to the end goal state (end position). Dijkstra's algorithm on the other hand, due to the static nature of the environment in which the simulations were conducted, needs the shortest times for computing the path (1.21 sec on a 10x10 grid and 1.71 sec on a 20x20 grid). The algorithm implementation was relatively simple and memory efficient compared to that of Q learning.

IV. Conclusions

The Q learning algorithm proposed in the paper solves a Markov Decision Process at every step by learning the optimal state-action value function or Q-function for solving a static path planning problem. The learning process is close to that of living beings, as it is based on the rewards or the consequence of actions that it takes at every decision step. The algorithm is adaptive and does not need large labeled datasets or the models of the environment to be pre-known and can make intelligent decisions in uncertain conditions. The algorithm also allows off-policy learning and is convergent. However, the algorithm is not memory efficient and is computationally intensive compared to the classical path finding approaches such as Dijkstra's and A*. Given a static and a known environment, it is preferable to use a classical path finding approach as it is faster and performs effectively. Such classical path planning algorithms either converge to a solution or confirm that a solution is unachievable. Hybrid based approaches such as D*, short for dynamic A*, can cope with such shortcomings of classical path planners and the unpredictability of the dynamic environments.

During this study, the following outlook has emerged: Several variations in Q learning such as hyper Q learning, Bayesian Q learning, relative Q learning have recently been proposed to improve the convergence time, maximize the performance, and reduce the number of steps to reach the optimal Q-value. The algorithm is also limited in its application to discrete spaces, and powerful function approximators need to be applied to extend its use to continuous environments. Here, Deep Q learning approaches can address the issue of the application of Q learning to continuous environments, which is an important step towards solving many real-world problems. Evaluating the performance of such algorithms in comparison to the hybrid classical path-based approaches is one of the future research scopes that will continue this research.

Conflict of Interest

The authors declare no conflict of interest.

Author Contributions

The research topic has been specified and supervised by Rolf Dornberger and Thomas Hanne. Phalgun Chintala conducted the research project including implementation and numerical experiments. and provided a first draft of the paper. The document was further revised by Rolf Dornberger and Thomas Hanne. All authors had approved the final version.

References

[1] P. Mehta, H. Shah, S. Shukla, and S. Verma, "A review on algorithms for pathfinding in computer games," in *Proc. IEEE Sponsored 2nd International Conference on Innovations in Information Embedded and Communication Systems ICIIECS,* vol. 15, 2015.

[2] A. Muhammad, M. Ali, and I. Shanono, "Path planning methods for mobile robots: A systematic and bibliometric review," *ELEKTRIKA- Journal of Electrical Engineering*, vol. 19, pp. 14-34, 2020.

[3] M. W. Otte, "A survey of machine learning approaches to robotic path-planning," University of Colorado at Boulder, Boulder, 2008.

[4] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*, MIT Press, 2018.

[5] M. Coggan, Exploration and exploitation in reinforcement learning. Research supervised by Prof. Doina Precup, CRA-W DMP Project at McGill University, 2004.

[6] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3-4, 1992, pp. 279-292.

[7] I. Altaharwa, A. Sheta, and M. Alweshah, "A mobile robot path planning using genetic algorithm in static environment," *Journal of Computer Science*, vol. 4, no. 4, pp. 341-344, 2008.

[8] M. M. U. Chowdhury, F. Erden, and I. Guvenc, "RSS-Based Q-Learning for Indoor UAV Navigation," in *Proc. MILCOM 2019 - 2019 IEEE Military Communications Conference (MILCOM)*, 2019, pp. 121-126.

[9] J. Xin, H. Zhao, D. Liu, and M. Li, "Application of deep reinforcement learning in mobile robot path planning," in *Proc. 2017 Chinese Automation Congress (CAC)*, Jinan, China, 2017, pp. 7112-7116.

[10] X. Lei, Z. Zhang, and P. Dong, "Dynamic path planning of unknown environment based on deep reinforcement learning," *Journal of Robotics*, vol. 2018, 2018.

[11] H. Quan, Y. Li, and Y. Zhang, "A novel mobile robot navigation method based on deep reinforcement learning," *International Journal of Advanced Robotic Systems*, vol. 17, no. 3, May 2020.

[12] S. Dasgupta, "Analysis of a greedy active learning strategy," *Advances in Neural Information Processing Systems*, vol. 17, pp. 337-344, 2005.

[13] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd edition Prentice Hall/Pearson Education, Upper Saddle River, 2010.

[14] H. Shteingart, T. Neiman, and Y. Loewenstein, "The role of first impression in operant learning," *Journal of Experimental Psychology: General*, vol. 142, no. 2, p. 476, 2013.

[15] M. Tokic and G. Palm, "Value-difference based exploration: adaptive control between epsilon-greedy and softmax," in *Proc. Annual Conference on Artificial Intelligence*. Springer, Berlin, Heidelberg, 2011.

[16] O. Nachum, et al., "Bridging the gap between value and policy based reinforcement learning," *arXiv preprint* arXiv:1702.08892, 2017.

Phalgun Chintala is currently pursuing his master's degree in Business Information Systems at the School of Business of the University of Applied Sciences and Arts Northwestern Switzerland FHNW in Basel, Switzerland.

He worked as a machine learning engineer at Max Planck Institute of Psychiatry (Munich) and Think Ahoy (Wiesbaden) before cofounding an ecommerce startup Chicken Basket in India. His current research interests include artificial intelligence particularly related to business applications, robotics, image analysis and processing.

Rolf Dornberger, born in Germany, studied Air- and Aerospace Engineering (diploma 1994) at universities in Stuttgart (Germany), Barcelona (Spain) and Grenoble (France). He holds a PhD (1998) in Air- and Aerospace Engineering from the University of Stuttgart, Germany, in the field of numerical methods for modelling and simulation, and an international University Teaching Certificate from the University of Basel, Switzerland (2017).

After his PhD, he worked in industry in different management positions as a Consultant, IT Officer and Senior Researcher in different international engineering, technology, and IT companies in the field of energy, software, IT, and airline business in Switzerland. He taught part-time at universities in Zurich and Stuttgart. Today, he is Professor and Head of the Institute for Information Systems at the School of Business of the University of Applied Sciences and Arts Northwestern Switzerland FHNW in Basel, Switzerland. Additionally, he is a guest lecturer at the Faculty of Business and Economics of the University of Basel, Switzerland. He is the (co-)author of more than a hundred scientific publications. His current research interests include artificial intelligence, particularly the nature-inspired methods of computational intelligence, modelling, simulation and optimization, robotics, human-machine interaction, and innovation management.

Prof. Dr. Dornberger is a member of Swiss Informatics and Rotary, has been vice president and board member of two Swiss associations, and has/had various commitments as a member of international and technical program committees, book author and editor (Springer) and journal editor, honorary chairman, session leader, organizer of special sessions, reviewer, invited keynote speaker.

Thomas Hanne received master's degrees in Economics and Computer Science, and a PhD in Economics. From 1999 to 2007 he worked at the Fraunhofer Institute for Industrial Mathematics (ITWM) as senior scientist. Since then, he is Professor for Information Systems at the University of Applied Sciences and Arts Northwestern Switzerland and Head of the Competence Center Systems Engineering since 2012. Thomas Hanne is author of more than 160 journal articles, conference papers, and other publications and editor of several journals and special issues. His current research interests include computational intelligence, evolutionary algorithms, metaheuristics, optimization, simulation, multicriteria decision analysis, natural language processing, systems engineering, software development, logistics, and supply chain management.