Deutscher Luft- und Raumfahrtkongress 2021
DocumentID: 550022

AUGMENTATION OF THE SELF-GUIDED LEARNING CAPABILITIES OF A REINFORCEMENT LEARNING-BASED DESIGN OPTIMIZATION ALGORITHM FOR ELECTRIC FIXED-WING VTOL UAV PROPELLERS

M. Thiele, M. Hornung
Technical University of Munich, Chair of Aircraft Design, Boltzmannstr 15, 85748 Garching, Germany

Abstract

A Monte-Carlo Tree Search Optimization Algorithm that is trained using a reinforcement learning approach is enhanced with increased self-learning capabilities. These enable a dynamic and adaptive transition between exploration and exploitation on gained knowledge and a parameter selection process based on probability distribution functions. The algorithm is able to converge to the minimum of commonly used ontimization test functions and can devise an optimal parameter set of a propeller geometry for a fixed-wing VTOL UAV or other design geometric tasks.

Keywords

Reinforcement Learning; Monte-Carlo Tree Search; VTOL UAV; Propeller Design

1. INTRODUCTION

The development of high performance fixed-wing VTOL UAV for remote sensing or transport applications is more and more driven by complex, mission-specific requirements. In an effort to enhance overall system performance, VTOL UAV tend to merge hover- and cruise propulsion systems into one hybrid propulsive powertrain which requires propellers tailored to multiple flow states. The aerodynamic design of these highly efficient propellers poses challenges for traditional calculation and optimization algorithms due to the high number of interdependent parameters. To tackle this task, new approaches have to be employed. The concept of reinforcement learning, traditionally used to develop decision strategies for robots or games, enables a fast and efficient way to train a geometrical optimization algorithm.

Thus, a novel optimization algorithm has been developed in [1] which exhibits very good scaling effects with the number of parameters considered. It defines an optimal propeller geometry while still maintaining a very detailed geometrical parameter set of the propeller with up to $20-40$ parameters which poses significant difficulties for a wide range of traditional optimization strategies. The task of assigning values to all optimization parameters is performed based on an iterative Monte-Carlo Tree Search algorithm similar to the concepts proposed in [2] that is trained using a reinforcement learning approach. It enables the seamless integration of multiple design goals and constraints using a custom defined target function that is generating the reward.

The algorithm proposed in [1], which is inspired by work from [3] and [4], demonstrates the successful application of reinforcement learning procedures on a physics-based design task. It is especially suited to optimize design parameters when the design space is large and the target function can be evaluated rapidly. The previously presented results [1] show that the original algorithmic architecture can outperform commonly used traditional optimization procedures by converging in a rather short computational time to a good solution while the performance is comparable to other more sophisticated optimization procedures.

This paper will focus on changes to the algorithm's architecture that have been implemented to tackle the fixed division between exploration and exploitation that has been identified as a shortcoming in the preceding paper. These changes lead to an enhancement of the algorithm's optimization capabilities and transform the algorithm to be more stable as well as adaptive to individual design tasks.

2. BASIC APPROACH

The baseline approach used in this paper remains as described in [1] and will be summarized in the following sections in order to better characterize the changes that are implemented for this paper.

A basic Markov Decision Process (MDP) as shown in Fig. 1 with an agent in a state s_t is performing an action a_t in an environment. It will place the agent in a new state s_{t+1} and return a reward r_{t+1}. This reward is used as a feedback on the quality of the new state and serves as an indication on the quality of the most recent action. [1, 4]

The probability for the selection of an action is controlled by the value-function V which depicts an expected reward assigned to each possible action when in a given state. The shape and magnitude of this value-function over each parameter's design space needs to be learned with reinforcement learning. [1]

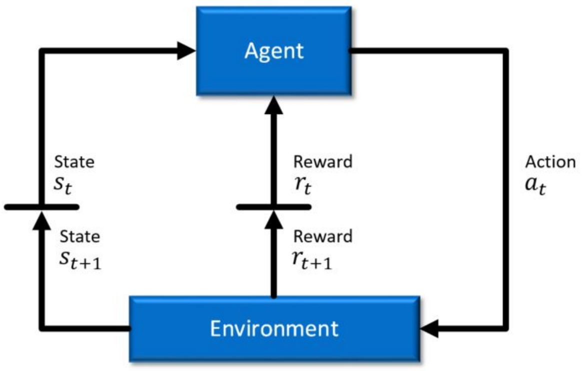


FIG 1. Schematic structure of a reinforcement learning procedure [1, Fig. 1]

Applying these principles to the design algorithm of this paper defines an action as the process of selecting exactly one value for each geometric design parameter within its design space. The evaluation of a complete geometry with an appropriate aerodynamic model delivers the reward value.

2.1. Parameterization

Following [1], this paper will also focus on the optimization of a VTOL UAV propeller geometry as its use case. A VTOL UAV propeller as sketched in Fig. 2 is usually a fixed-pitch propeller with two or more blades. The geometric set of parameters to be optimized for the test cases of Section 4 is defined exactly the same as the parameter set used in [1]. Global parameters are defined, such as the rotational speed Ω and the propeller radius R as well as local parameters defined radially along the blades such as the radial distributions of the chordlength $c(r)$ and local pitch $\beta(r)$. Each parameter is limited by a lower and upper boundary enclosing discrete or continuous values in between.
The pitch and chordlength distributions of a propeller cannot be chosen as arbitrary values along the blades as the rate of change along the radial coordinate is limited. A random distribution of chordlength and pitch values will not produce a valid geometry and will not converge in the aerodynamic model. Thus, these parameters are not considered directly at distinct radial positions but are rather derived from an envelope distribution that is shaped using several design parameters.
These envelope design parameters are equal to the definition presented in [1] and consist of 12 design parameters for the chordlength and the pitch distribution each.
One critical design parameter that is not included in the geometry definition is the local airfoil shape. This algorithm requires all parameters to change the propeller behavior in a sufficiently deterministic way when sweeping over the values between the lower and upper limits. A list of airfoil shapes will change the propeller behavior in a non-deterministic manner. This

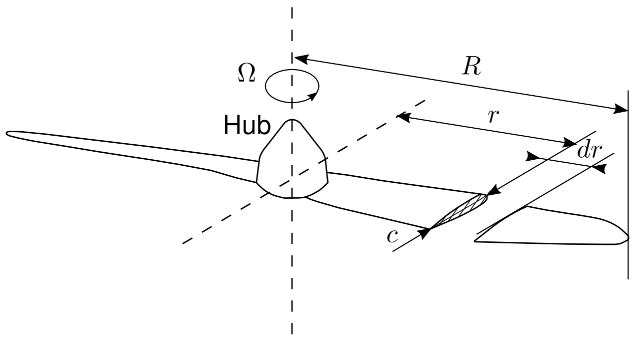


FIG 2. Sketch of the propeller definition [1, Fig. 2]

impact on the design cannot be predicted by the current optimization algorithm without further improvements or intermediate steps. A solution for this would be the prior usage of clustering algorithms or a principal component analysis for the airfoil behavior over a given set of airfoils. This has been investigated in [5] and partly in [6] with algorithms described in [7, 8]. In order to focus on the functionality of the design algorithm, the calculations presented here use a default airfoil that is scaled with the local chordlength.
A two bladed propeller will be optimized as this is the most commonly used for electrically driven UAVs. Furthermore, the Blade Element Momentum (BEMT) Algorithm that is used to predict the reward is incapable of estimating the mutual influence between multiple blades which would lead to a tendency of using as many blades as possible during the design process. Another characteristic that is not represented in these sample calculations is the local blade sweep which is assumed to be zero. Including the sweep characteristic in the design optimization process is, however, easily possible.
This parameter set can be evaluated rapidly within a BEMT model that is described in [9] which is derived from the baseline model by [10]. This model is embedded within the SARF propeller framework that is developed in [11] and further expanded in [9]. The propeller efficiency can be calculated with the BEMT model and is used as the reward value for this study.

2.2. Search Tree Structure

The parameters are structured in a Monte-Carlo Search Tree as described in [1] and shown in Fig. 3. It is adapted from Monte-Carlo Tree structures that are used in [4, 12]. Each row of the tree is created by one design parameter P_j and each node n_i depicts a value that this parameter can obtain. Each parameter is defined by a lower and upper boundary. Depending on whether a parameter is of discrete or continuous nature, there is a different amount of nodes per parameter. This structure is only able to handle discrete nodes for the optimization process. Therefore, a continuous parameter needs to be pseudo-discretized internally with a finite number of nodes.
The number of nodes per parameter can be arbitrary, however, a large number of nodes will negatively af-

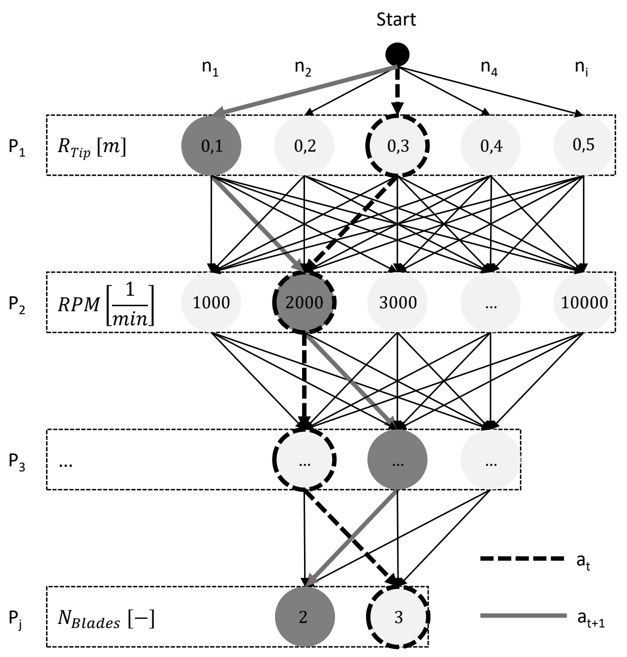


FIG 3. Search tree with optimization parameters and two paths [1, Fig. 4]

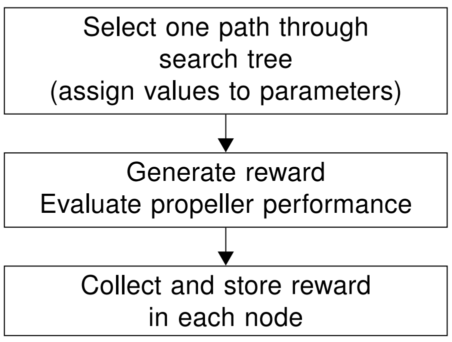

Select one path through search tree (assign values to parameters)
Generate reward Evaluate propeller performance
Collect and store reward in each node

FIG 4. Event flowchart [1, Fig. 5]

fect the convergence speed of the algorithm. More than 15 to 20 nodes per parameter should be avoided. On the other hand, it is a fixed requirement that continuous parameters can converge within a continuous parameter scope to an arbitrary value. This is enabled by virtual refinement of the nodal resolution which makes these parameters quasi-continuous during the parameter fixation procedure that will be described below. [1]

A path through the tree from the top to the bottom defines one set of parameter values for all design parameters, which enables the objective evaluation and reward generation. The node selected in each layer is not dependent on the nodes selected in other layers. Therefore the number of possible paths through the tree can be calculated by the product of the number of nodes for each layer, which creates a very large design space for a higher number of parameters. When searching for an optimal parameter set, the algorithm is assigned the task of searching the path that provides the best reward. This search effort is grouped in an "Event" procedure as shown in Fig. 4. The received reward is then stored individually in each visited node. New nodes can be generated within the layers during the parameter assessment process in order to account for continuous design parameters.

2.3. Knowledge Incorporation

Selecting individual nodes successively forms an action a_t characterized by the path that is traveled through the layers of the search tree. A reinforcement learning algorithm has two basic options for the node selection process: Exploration or Exploitation

For the exploratory action, a random node will be selected which results in an arbitrary path as is highlighted by a dotted line in Fig. 3. Contrary to common reinforcement learning problems, the target of the optimization is not a sequence of actions, but rather the convergence to exactly one optimal parameter set, which translates to the discovery of one optimal action. The exploitation of gained knowledge has to be incorporated in the actions itself rather than in the action sequence. A path with exploitation is therefore generated by performing a tree search for the highest expected reward value in all paths through the decision tree.

The training process of a reinforcement learning algorithm has to perform a trade-off between design space exploration and knowledge exploitation. The previous implementation in [1] was not capable to adapt the algorithmic behavior to different optimization tasks and constrained environments.

Convergence was enforced by iteratively selecting the best nodes for each parameter. After assessing a sufficiently large number of paths while exploring, all individual value-functions V and their predicted peak locations were compared and the most promising and influential parameter with its peak node was chosen for exploitation. The iterative optimization process was then resumed on a new set of paths with the limitation that all paths run through this chosen node. This lead to a progressive definition of one value for all parameters until a final configuration was reached. Inherent to this approach was a hard and predefined split between exploration and exploitation of learned knowledge. An already fixed design parameter's value was not intended to be reassessed once it had been fixed even after more knowledge for this domain was available.

An independent and dynamically changing tendency to perform exploration or exploitation of the design space with the possibility to incorporate new knowledge for all parameters is one of the key changes to the original V1.0 algorithm that are implemented and demonstrated in this paper.

3. TECHNICAL IMPROVEMENTS

The complete procedure is summarized in Fig. 5 which shows the improved V2.0 algorithm flowchart. The iterative optimization process will generate a set of paths through the search tree. Those paths will

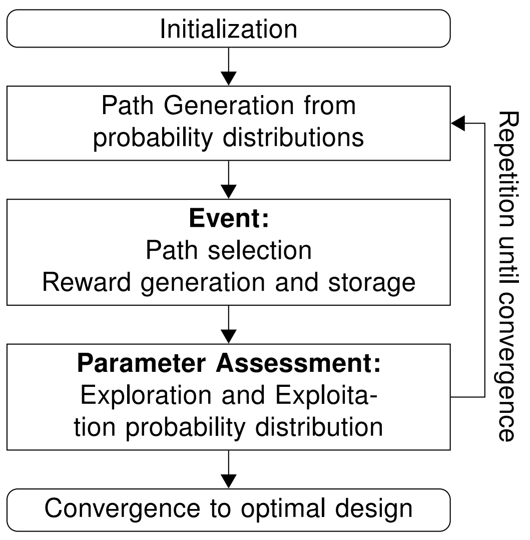

Initialization
Path Generation from probability distributions
Event: Path selection Reward generation and storage
Parameter Assessment: Exploration and Exploitation probability distribution
Convergence to optimal design
Repetition until convergence

FIG 5. Algorithm flowchart

be evaluated in events (Fig. 4). These events can be evaluated independently and can therefore easily be distributed to a multi-core hardware. Knowledge gained with the current set of paths is then assembled for the parameter assessment process which defines criteria for the generation of new paths. This is repeated until convergence. As becomes evident in Section 4, the number of iterations is not part of the convergence criterion as it has been for the V1.0 algorithm.

Both, the path generation and the parameter assessment process, are affected by changes implemented for this improved V2.0 algorithm.

3.1. Parameter Assessment Process

The collection procedure of the reward values for each path assigns the received reward r_t to each visited node n_i within the current path. Unchanged to the procedure in [1], the reward value function V_j for a parameter P_j is calculated from all rewards stored in each node. This is carried out separately for each parameter. The sample plot of Fig. 6 shows the procedure for one parameter. Each of the 4 nodes has been visited multiple times. During each visit, the reward generated by the corresponding path is stored (orange dots). The weighted average of these orange dots is then calculated (red dots). These red dots represent the observed reward for each node. The same regression algorithm already used in [1] is then applied on the red dots to yield the continuous value function V_j, representing the predicted reward (shown in blue). As a next step, peaks in this value function are determined as well as the approximation quality. Furthermore, peak clarity indicators are deducted. The metrics to calculate these indicators is described in [1].

In order to incorporate the newly developed dynamic exploitation, two probability density functions (PDF) are assigned to each parameter and are evaluated for each node selection during the path generation

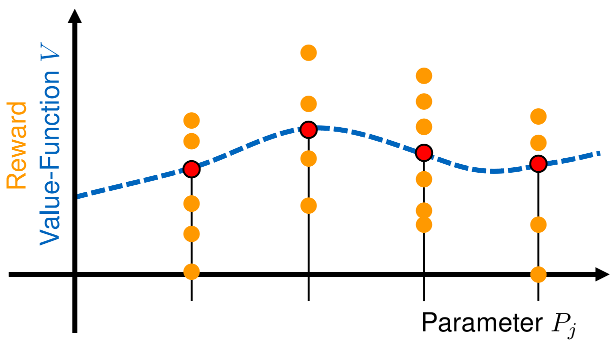


FIG 6. Reward regression process

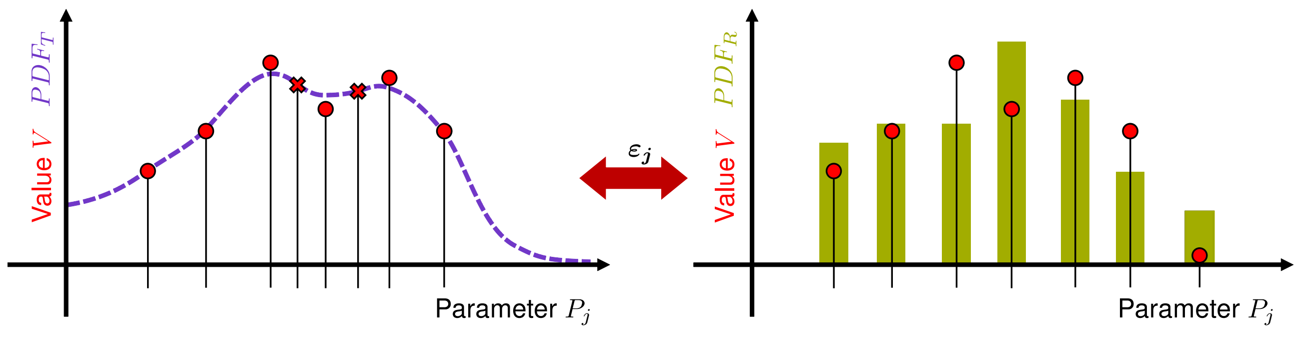
step. An artificial schematic of both $PDFs$ is depicted in Fig. 7. Technically, the green probability function PDF_R represents a probability mass function for a discrete probability distribution [13]. As the handling of both functions within the algorithm is equal, both will hereafter be called PDF. The two $PDFs$ of Fig. 7 represent the exploration and exploitation attractiveness, respectively. Their use as a base for the node selection for a path through the search tree will be explained in Section 3.2.2. Their shape must be adapted (for each parameter separately) during each parameter assessment process with the newly generated knowledge.

The vertical lines with a red dot on top represent the optimization value function V shown in Fig. 6 and indicate the magnitude of the expected reward for a propeller when using a certain node of this parameter. These data points do not belong to the PDF but are rather used as input values for their calculation and adaption. Additional data points indicated by a red cross are placed at certain positions. These will be explained below.

The PDF_R of Fig. 7b is responsible for the exploration of the design space. It is a discrete PDF as exploration is only allowed to happen at the predefined discrete nodes of a certain parameter. Subsequently, it consists of one probability bar per node.

Selecting a random action is a common strategy for a reinforcement learning algorithm when exploring the domain. To replicate this, the PDF_R would need to be kept uniform throughout the calculations. However, various studies have shown that an unguided exploration is not suited for all reinforcement learning problems [14]. Therefore, the concept of intrinsic motivation proposed in [15] is adapted and incorporated into the exploration process. The algorithm is thereby extended with *curiosity* adapted from [16] and *novelty-seeking* which is introduced in [17].

Nearly all of the green bars displayed in Fig. 7b differ from the position of equal probability. They show examples of effects that are considered in this PDF. The rightmost red data point shows a very low expected reward value. Reasons for this can include either a generally low reward with this parameter setting or an increased likelihood of a boundary condition violation resulting in a reward of zero for the responsible path. After a reasonable number of iterations it



(a) Exploitation $\mathbf{PDF_T}$ **(b) Exploration $\mathbf{PDF_R}$**

FIG 7. Schematic of both $\mathbf{PDF_T}$ and $\mathbf{PDF_R}$ for one parameter $\mathbf{P_j}$

is to be expected that further exploration for this node is not desired because there will not be a benefit in increasing the certainty of this assumption. The middle data point is located between two maxima. This pattern of two maxima that are separated between a local minimum can either represent the true target function behavior, or can indicate insufficient information at these nodes which leads to a false prediction of one neighboring maximum. Lastly it can also imply a hidden true optimum at the position of the middle node. Curiosity dictates the exploration attractiveness at these nodes being especially high in order to steer the exploration to these regions of the parameter space. Other regions of interest are characterized by an increased uncertainty of the prediction.
The calculation of this PDF_R is achieved with several indicators that are evaluated over the design space on each node of interest, which are:

- Nodes with a high prediction error and their neighboring nodes - expressed in a high discrepancy between the observed and the predicted reward
- Nodes with a high quartile spacing of the orange dots shown in Fig. 6
- Nodes with a low validity rating and their neighboring nodes, characterized by an increased likelihood for a reward evaluation to fail due to non-convergence or boundary condition violation
- Nodes which are under-explored compared to their peers, triggering the novelty-seeking aspect of the implementation
- Nodes in between peaks of the predicted reward as, e.g., the middle node in Fig. 7b

A histogram of nodes will be calculated from nodes that are selected by the combination of all these indicators. The exploration attractiveness of the nodes is increased according to the histogram count of each node. Nodes not selected by these indicators will have their exploration attractiveness reduced.

The PDF_T of Fig. 7a on the other hand is responsible for knowledge exploitation. It is important to note, that the value function V drawn in blue in Fig. 6 is not equal to the purple PDF_T in Fig. 7a, even though the data points indicated by red dots are equivalent. The value function V is generated after a collection of paths has been evaluated. This new knowledge is incorporated in the PDF_T with the learning rate $\alpha \in [0,1]$ [18]. Similar to other machine learning algorithms, this parameter indicates how large the influence of the most recent knowledge is on the PDF_T. After normalizing both the PDF_T and V, it is calculated as:

$$PDF_{T_{New}} = (1-\alpha)\, PDF_{T_{Old}} + \alpha\, V$$

The learning rate α is not fixed during the optimization process. Instead it will decrease with each episode in order to prevent outliers detected late in the optimization process to negatively affect the overall result.
Parameters with a continuous parameter space are also discretized in a small amount of test nodes [1]. These are shown by red dots. The exploitation however can happen at an arbitrary position within a continuous parameter space. Therefore it is possible to change the discrete sampling points of this PDF_T. The right most node shown in Fig. 7b has been deactivated from Fig. 7a because this node will result in the smallest reward and is therefore not suitable for exploitation. On the other hand, two additional nodes (red cross) have been placed in regions with a high expected reward which helps converging to an optimum solution within the continuous parameter space. These additional nodes are available in addition to the not deactivated nodes during the next path generation process.

3.2. Path Generation

The path generation routine follows a 2-tiered approach. The assessment whether to enact exploration or exploitation for each parameter is followed by the node selection procedure for each path.

3.2.1. Exploration-Exploitation Trade-Off

The exploration-exploitation trade-off denotes the probability of the algorithm to explore the design space and perform a random subset of all possible actions versus the exploitation of gained knowledge. As described by [19], it is an important contribution to the convergence of a reinforcement learning algo-

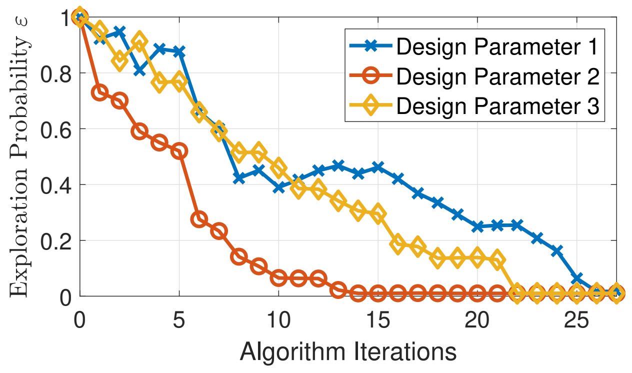


FIG 8. Evolution of exploration probability factors ε_j over algorithm iterations

rithm. This trade-off is evaluated by the probability factor $\varepsilon \in [0, 1]$. One such factor ε_j is assigned to each design parameter and determines the probability, whether the previously gathered knowledge will be exploited or a targeted local exploration of the design space is carried out for a certain parameter on the path through the search tree. This makes the implemented algorithm an epsilon-greedy Q-learning process [4].

The factor ε_j is dynamically adapted for each design parameter during the parameter assessment process according to the tendencies exhibited in the previous and current optimization steps. Knowledge gained during later stages of the optimization process is thereby used to reassess design parameters and is thus enabling the learning process to dynamically switch between exploration and exploitation of the current learned knowledge in a self determined way. This enables the parameter selection process to be inherently dynamically adaptive over the whole design space and enhances the capabilities compared to the V1.0 algorithm to move away from a previously found local optimum and to converge towards a global optimum solution of the design task.

In the beginning, no previous knowledge is available. As evident in Fig. 8 for three exemplary design parameters, ε_j is close to one. Each path through these parameters will therefore perform exploration, thus run through a random or insufficiently explored node. By applying simulated annealing [20], this factor is slowly decreased as more knowledge is gathered, which raises the probability of exploitation being performed (exploitation results in the selection of a promising node of a parameter). As the design converges to an optimal solution, ε_j approaches zero, even though it is capped at $\varepsilon_j = 0.01$. A value of 0 indicates exploitation being chosen for each path and is desired if the knowledge gained about a certain parameter's influence on the overall design is sufficient. The cap at $\varepsilon_j = 0.01$ ensures a remaining tendency to explore other areas in the design space, which helps moving out of a local optimum towards a global one. The probability factor ε will decrease independently and with a different pace for each parameter depending on the robustness of the current parameter space approximation. Design parameter 2 of Fig. 8 shows a rapid decline as this parameter's influence on the overall solution is more pronounced than the influence of the others. This means that the certainty of the prediction is higher for this parameter. It is worth noting that it is possible for ε_j to increase temporarily during the convergence process, as evident in the trend shown by design parameter 1. This may happen if the peaks found in this parameter's value-function V become less prominent and therefore the location of an optimum in this parameter's design space becomes less clear due to newly gained knowledge.

The variation of ε_j takes the peak clarity and regression quality indicators that are defined in [1] into account. A change factor $\Delta_{\varepsilon j}$ is calculated and later multiplied with the original ε.

$$\varepsilon_{j(t+1)} = \varepsilon_{j(t)} * \Delta_{\varepsilon j}$$

The peak clarity indicator of all parameters is ordered in a descending order. A linear rate of change is then imposed on the sorted parameter list, where Δ_ε of the highest parameter on the list is decreased by the highest amount, parameters on the middle position remain unchanged and parameters on the lower end are increased. Additionally, the peak separation factor described in [1] is also directly multiplied with Δ_ε in order to take the predictive value of an observed peak into account. The regression quality indicator is considered similarly to the peak clarity indicator by imposing a linear rate of increase or decrease on the list of parameters sorted by a descending regression quality indicator.

This baseline variation of ε where the individual parameters are weighted against each other would not lead to a sustained reduction of ε for all parameters which is required for eventual algorithm convergence. Therefore, additional aspects are considered for each parameters exploitation probability factor. The legacy parameter assessment process of [1] features a 2-tiered system to select the most promising parameter for node fixation. In a first step, a subset of all parameters is selected for further evaluation. These parameters' probability factor ε will be reduced by 10%. The parameter that would have been selected in the second step as the fixation parameter of the parameter assessment process will receive an additional reduction of ε by 10%.

3.2.2. Node Selection Procedure

The node selection procedure as the second step in the path generation process is equivalent for exploration as well as exploitation. The two probability density functions (PDF) shown in Fig. 7 that are assigned to each parameter are evaluated for each node selection for each generated path. The shape of these PDF is adjusted during the parameter selection process as described above.

The parameter ε influences which PDF is used for the node selection. A random parameter value is then

drawn using this PDF by transforming it to a cumulative distribution function and selecting its value using a random number between 0 and 1. Exploration may only happen at the pre-computed discrete locations of a parameter. Therefore, the PDF_R of Fig. 7b depicted in green is a discrete PDF. Exploitation (for continuous parameters) however, can select an arbitrary value within the parameter limits, which is why this PDF_T of Fig. 7a is drawn as a continuous PDF. Internally, however this PDF_T is also used as a discrete PDF where additional discrete locations can be placed at arbitrary positions as explained above.
With this, the path generation process, which is the heart of the reinforcement learning procedure, can be performed efficiently using a simple random number generator. A batch set of paths will be generated simultaneously by creating two random numbers $R_1, R_2 \in [0,1]$ for each path. The first random number R_1 is compared to the probability factor ε and determines whether to explore knowledge or to rather exploit knowledge.

$$R_1 > \varepsilon \longrightarrow \text{Exploitation, } PDF_T$$
$$R_1 \leq \varepsilon \longrightarrow \text{Exploration, } PDF_R$$

The second random number R_2 is then used to pick a suitable node within the parameter space using the corresponding cumulative distribution function, that is calculated from the PDF.
This collection of paths can then be evaluated independently in a parallel computing environment. Afterwards, the results are again processed in the parameter assessment process as shown in Fig. 5.

4. COMPUTATIONAL RESULTS AND APPLICATION

Overall algorithm convergence is achieved by the probability factor ε converging towards zero for all parameters. This trend can be seen in Fig. 8. The overall most promising solution for the optimization problem is then determined by evaluating the value function V for all parameters and selecting the highest peaks as final design parameter values.
However, like with any other stochastic optimization method it cannot be guaranteed that the found solution is the absolute global optimum. The selection of nodes based on random samples will inherently lead to a probabilistic behavior. Starting the algorithm multiple times will therefore converge to different solutions. The newly introduced methodologies will, however ensure that a more consistent optimization result is achieved over multiple optimization procedures compared to the V1.0 algorithm

4.1. Algorithm Tests on Benchmark Functions

Various benchmark studies are conducted for the improved V2.0 algorithm. The estimation of the performance on well established optimization benchmark functions provides a showcase of the new capabilities. Three test functions are evaluated, each with $d = 10$ design parameters.

- The Ackley function, proposed by Ackley in [21]
- The Levy function No.3, proposed by Levy in [22]
- The Michalewicz Function, introduced by Michalewicz in [23]

The internal pseudo-discretization of each design parameter with 10 intermediate steps yields 1×10^{10} possible combinations with an even greater true variability when design parameters can be selected from a continuous parameter space.

4.1.1. Ackley Function

The Ackley function [21] is shown in Fig. 9 in two dimensions. It is defined by the following function:

$$f(x) = -\,a\;exp\left(-b\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}\right) - exp\left(\frac{1}{d}\sum_{i=1}^{d} cos(c\,x_i)\right) + a + e$$

The three constants are used with their most common values: $a = 20$, $b = 0.2$ and $c = 2\,\pi$. The function is usually evaluated in the domain $x_i \in [-32.768, 32.768]\ \forall\ x_i = 1 \cdots d$ and the global minimum of this function resides at

$$f(x^*) = 0, \text{ at } x^* = (0, \cdots, 0)$$

The function shape is characterized by a plateau at $f(x) = 20$ with a steep circular hole in the middle. As evident in the contour plot below the function in Fig. 9, the upper surface is not flat, but rather consists of many small peaks and local minima which need to be avoided by the optimization algorithm.

The convergence plot of the optimization over the Ackley function is shown in Fig. 10. The value of the Ackley function which represents the reward is drawn over the number of algorithm iterations. Each entry in the graph represents one event where the

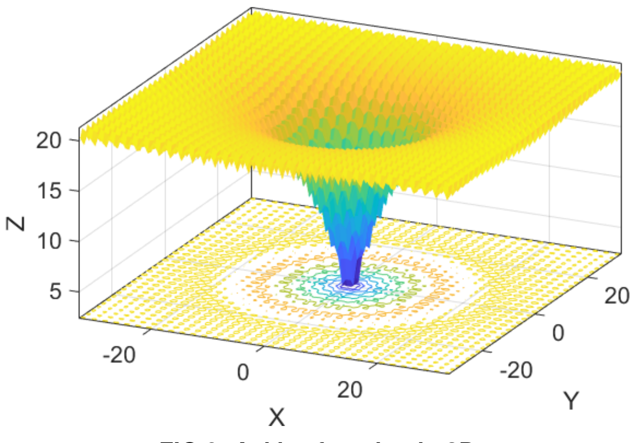


FIG 9. Ackley function in 2D

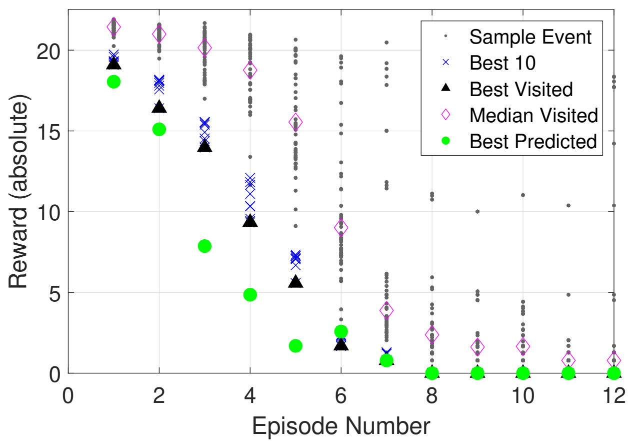


FIG 10. Ackley function convergence

optimization algorithm evaluated the target function. The y-axis displays the absolute value instead of the internally used value-function in order to better interpret the results. Each tick of the x-axis shows an episode, where a batch of paths is evaluated with a subsequent parameter assessment step that leads to the adaption of the PDF with the newly found knowledge.

The gray dots show 50 randomly selected sample events out of roughly 2000 events per episode in order to show the general tendency of the parameter selection. The pink diamond indicates the median position of all events in the current episode. These two groups show a clear tendency of the algorithm to converge to the minimum. Initially, most events lead to a result somewhere on the top plateau, while later on most events fall somewhere within the vicinity of the minimum. A residual share of events can be seen with a few gray dots scattered in the upper right corner of the figure. These datapoints represent events where exploration in other areas of the design space is carried out and form an important contribution to the algorithm extending the search beyond a local minimum.

The 10 best events of each episode are marked by a blue cross. The spacing between those markers is shrinking with each iteration which indicates a concentration of the best search results close to the optimum. The black triangle on the other hand is a global metric that displays the best observed event over the whole optimization procedure. As the results improve with every iteration for this problem, it is always located at the best position of a blue cross. Lastly, the selected optimal design is shown by the green dots. The design parameter values for this selected design are chosen individually for each parameter as explained above from the highest peaks found in the value-functions of each design parameter. As such, the precise combination of these design parameter choices has not necessarily been visited during the first few iterations. Therefore, the green dots can show a better prediction than the best visited state so far, which illustrates the predictive capabilities of this algorithm. It is of note that the design parameter value associated with the best predicted design will receive an increase in its corresponding PDF_T value. However, due to the paths through the search tree individually selecting design parameter values according to PDF_R and PDF_T, it is not guaranteed that the exact combination needed for one iteration's best predicted result will be visited in the subsequent iteration. This intended behavior is evident between episodes 3 and 4 as well as between episodes 4 and 5. Simply enforcing the visitation of the design parameter combination with the best predicted result will lead to overfitting of the learning process and is therefore not implemented.

The overall trend of the graph shows a rapid convergence to the global minimum of $f(x^*) = 0$. After the predicted result and the best observations reach the minimum, more and more of the remaining events also move in this direction. This is due to both PDF reducing the probability of visits in other regions of the design space.

4.1.2. Levy Function

The Levy function is shown in Fig. 11 - also with only two dimensions. More specifically, it is the third example function presented in [22] and therefore often called Levy No.3. It is defined by:

$$\begin{aligned} f(x) = {} & \sin^2(\pi w_1) \\ & + \sum_{i=1}^{d-1} (w_i - 1)^2 \left[1 + 10\sin^2(\pi w_i + 1)\right] \\ & + (w_d - 1)^2 \left[1 + \sin^2(2\pi w_d)\right] \\ \text{where } & w_i = 1 + \frac{x_i - 1}{4} \quad \forall i = 1, \dots, d \end{aligned}$$

It is usually evaluated on the hypercube $x_i \in [-10, 10]\ \forall\ x_i = 1...d$ and the global minimum of the function is located at

$$f(x^*) = 0, \text{ at } x^* = (1, \cdots, 1)$$

The function exhibits several long valleys with high peaks in between. The global minimum on the other hand resides within a very large, shallow basin.

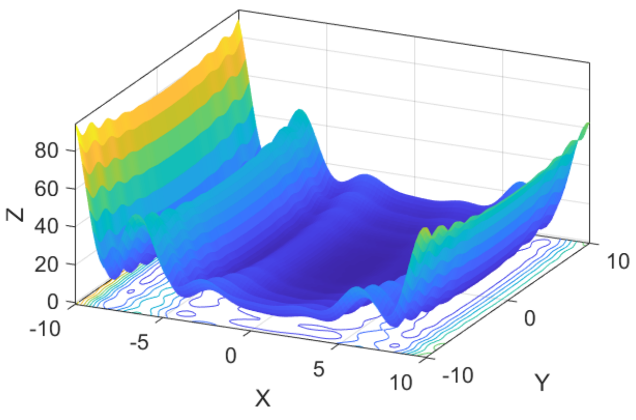


FIG 11. Levy function No. 3 in 2D

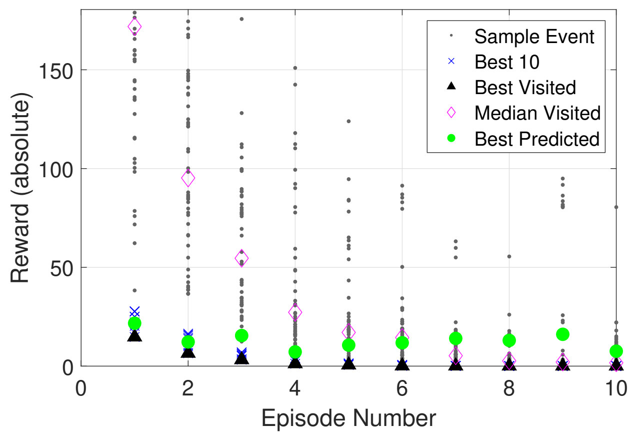


FIG 12. Levy function convergence

The convergence plot for the optimization of the Levy function is shown in Fig. 12. This function has no clear upper bound such as the one present for the Ackley function. Consequently, the variability of the gray dots in the first two iterations is much greater than that of the previous function in Fig. 10. The median value is also at a much higher position. The best events of the first iteration on the other hand are relatively seen much closer to the optimum. This is due to the much wider area of low function values present in the Levy function. Randomly selecting design parameter values will inevitably lead to results close to the minimum function value. Subsequent iterations lead to the majority of events selecting parameter values which evaluate to a small function value with the minimum observed events (drawn with a black triangle) falling very close to the global minimum. However, due to the gentle slopes of areas near the minimum, the green circles showing the best predicted values fail to reach the minimum reliably. This is caused by the individual parameters' value-function V not developing a clear peak that can be selected for the best predicted value. This test case shows that an only slightly pronounced minimum present in the design space will not be found with certainty.

4.1.3. Michalewicz Function

The last test function has a very complex shape. It is shown in Fig. 13 in two dimensions. The appearance of this function changes with the number of dimensions that are used for evaluation. The so-called Michalewicz function is one of the most commonly known test functions that Michalewicz developed for the validation of optimization algorithms. It is defined by:

$$f(x) = -\sum_{i=1}^{d} \sin(x_i) \sin^{2m}\left(\frac{ix_i^2}{\pi}\right)$$

and is usually evaluated on the hypercube $x_i \in [0, pi] \;\; \forall \;\; x_i = 1..d$. The coordinate of the global minimum is different for each dimension. According to [24], when evaluated in $d = 10$ dimensions, the global minimum with the value $f(x^*) = -9.6601517$ is located at:

$$\begin{aligned} x_1 &= 2.202906 & x_2 &= 1.570796 \\ x_3 &= 1.284992 & x_4 &= 1.923058 \\ x_5 &= 1.720470 & x_6 &= 1.570796 \\ x_7 &= 1.454414 & x_8 &= 1.756087 \\ x_9 &= 1.655717 & x_{10} &= 1.570796 \end{aligned}$$

Michalewicz function exhibits a high plateau that is separated by steep valleys. The parameter m is responsible for the steepness of these valleys. It is most commonly and also for these tests set to $m = 10$. A depression is present at each intersection of two valleys with one of the depressions being significantly deeper than the others. The number of valleys and depressions is increased when the function space is extended to more than two dimensions. Furthermore, the global minimum value $f(x^*)$ is dependent on the number of design parameters.

The convergence plot for the optimization of the Michalewicz function is shown in Fig. 14. Similar to the Ackley function, the best predicted reward of the first few iterations is better than the best observed reward, which means a minimum within the range

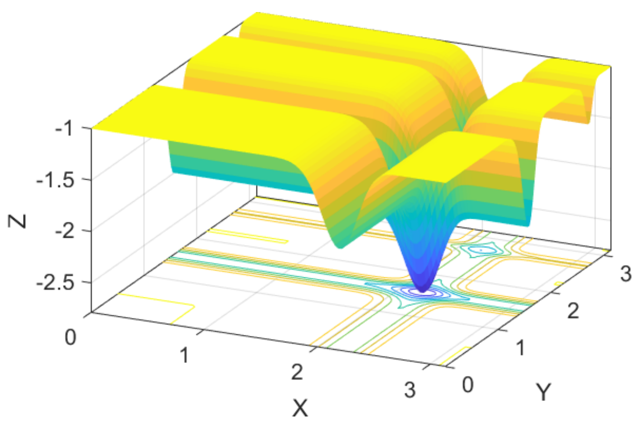


FIG 13. Michalewicz function in 2D

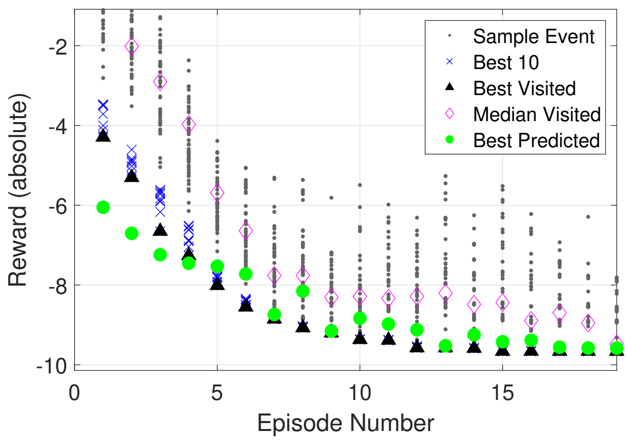
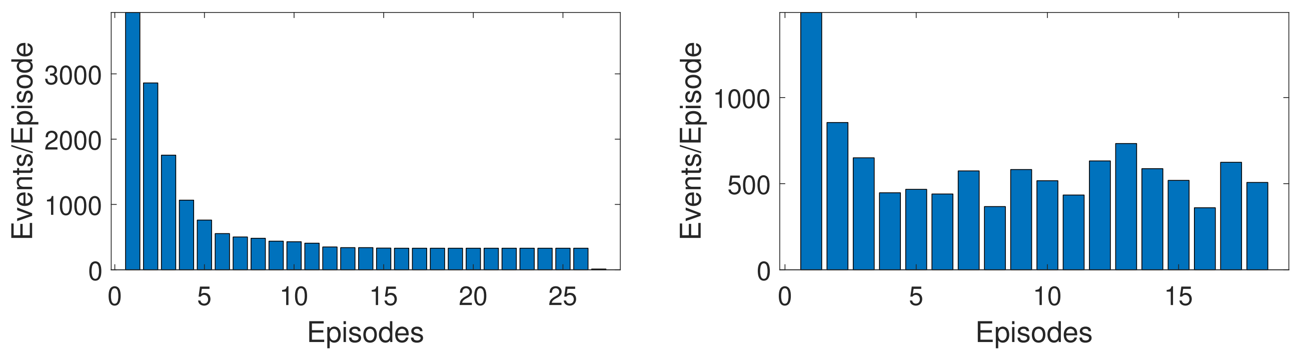


FIG 14. Michalewicz function convergence



(a) Algorithm version V1.0

(b) Algorithm version V2.0

FIG 15. Target function evaluations

of certain design parameters is already predicted prior to the full exploration of the design space in this region. Due to this function having a very complex shape, the rate of improvement towards reaching the global minimum value of $f(x^*) \approx -9.6601517$ is slower than for the previous functions. Final convergence towards the global minimum is achieved after roughly 17 iterations with ≈ 2000 target function evaluations each, which translates to the exploration of a very small amount of the whole design space. A slight oscillation of the best predicted value can be observed between episodes 5 and 10, which is caused by large changes in the value-function V in between those iterations, which leads to the local peak values changing locations for some parameters.

4.2. Comparison to Original Version V1.0

In order to assess the effect of the improvements presented in this paper, the new algorithm (V2.0) is compared to the original version (V1.0) that was described in [1]. The benchmark problem setup utilizes the parameterization of a propeller introduced in Section 2.1. The comparison will be carried out on two main performance criteria: computational cost and robustness of the result.

First, the computational cost of performing an optimization with equal starting criteria is assessed. It is measured by the number of target function evaluations including paths resulting in an invalid design. One event (Fig. 4) entails one evaluation of the target function in order to generate a reward. Therefore, the number of events that are required for convergence provide the first comparison for the computational cost which is independent of the computational power a machine can provide. Both algorithm variants are able to calculate a set of events on parallel machines.

A total of 10 optimization calculations has been performed with both algorithm versions. Two sample graphs detailing the number of events until convergence are shown in Fig. 15. The most striking difference between the two figures is the number of episodes and therefore algorithm iterations that is required for convergence. The V1.0 was only capable of fixing one design parameter after each iteration. This requires the number of iterations and the number of parameters to be equal. The V2.0, however is able to converge to a final prediction for all design parameters at the same time. This results in fewer iterations for a high amount of parameters. On the other hand, the number of iterations can now also exceed the number of design parameters as evident from Fig. 14 and may even vary when multiple optimizations are carried out with the same starting criteria due to the nature of an optimization problem. Nevertheless, this functionality leads to a more stable result and fewer overall iterations when the number of design parameters is large.

Another notable difference is the number of events for each iteration. As the V1.0 algorithm did not reassess results that have been fixed, it was required to generate more knowledge prior to the conclusion of the first few episodes. The number of events needed for the proper assessment of the design space after the first episode is roughly cut in half. Later episodes show a more variable number of episodes in Fig. 15b. The original algorithm would only perform exploitation on all design parameters that are fixed. While this leads to fewer invalid design choices being evaluated, it will also prevent the algorithm to move out of a local minimum. The ever present small fraction of exploration in the improved path generation results in more invalid evaluations, which adds some random small number of evaluations to the later episodes as can be seen in Fig. 15b. It therefore leads to better results with a very small additional computational cost per episode. The number of events for the last episode is also very different. As everything except the last parameter is fixed in the last episode for the V1.0 algorithm, it is only required to perform as many events as discrete locations are present in the last design parameter. In contrast, the V2.0 algorithm does not use the number of episodes as a convergence criterion and thus does not exhibit this effect. This is also the reason that the bar chart of Fig. 15b does not seem to indicate convergence of the algorithm, as the exploration probability of Fig. 8 is used as the convergence variable instead. Fig. 16 details the mean number of evaluated events over all episodes until final convergence during the 10 optimization procedures of the V1.0 being

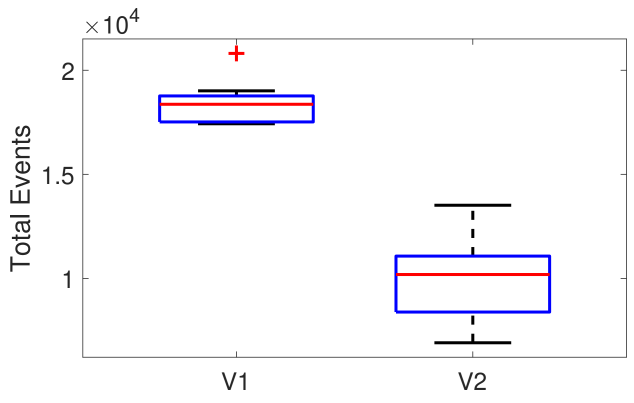


FIG 16. Number of events during 10 design optimizations

$\overline{N}_{events,V1.0} \approx 18400$ while the mean number of events for the V2.0 is $\overline{N}_{events,V2.0} \approx 9900$. This large difference has two main origins: The reduced number of episodes required for convergence and the reduced number of events within the first few iterations. The baseline number of required events in later episodes is slightly in favor of the V1.0 as this version produces slightly less invalid search paths during later episodes. Furthermore, as evident in Figs. 15 and 16, the V2.0 algorithm produces a much greater variety of the number of events needed for convergence.

In addition to the pure event count, the computational costs associated with the parameter assessment processes have to be considered. The parameter assessment is carried out after a certain number of events have produced a valid result and concludes one episode. It can in general not be carried out in parallel on multiple processor cores. In absolute terms, this metric is roughly doubled for the V2.0 algorithm. On the conventional processor, that has been used for this comparative study, the mean processing time for the parameter assessment was $t_{proc_{V1.0}} = 1.05\,\text{sec}$ for the V1.0 vs $t_{proc_{V2.0}} = 2.12\,\text{sec}$ for the V2.0. Furthermore, the relative percentage of the total time taken for the optimization is also increased significantly for the V2.0 algorithm. While the parameter assessment of the V1.0 algorithm accounted for $\approx 2\%$ of one episode's total computational cost, this number increased to $\approx 13\%$ for the V2.0. This relatively and absolutely greater computational cost is, however in total more than recovered due to the faster overall convergence rate of the algorithm.

The second performance metric measures the robustness and overall quality of the result. This is evaluated by assessing the results in terms of the resulting performance from the same 10 optimization calculations that have been used above. The results of 10 V1.0 optimizations deliver a mean final reward value (propeller efficiency) of $\eta_{V1.0} = 0.53$. The V2.0 algorithm delivers a mean final efficiency of $\eta_{V2.0} = 0.70$. Both values fall short of a theoretical maximum of $\eta_{opt} \approx 0.75 - 0.8$ for a propeller of this size class. However, the V2.0 version greatly enhances the quality of the overall optimization results. Furthermore, it delivers a more consistent result where the standard deviation of all results for the V2.0 is $s_{V2.0} = 2.78$ efficiency-percentage points while the standard deviation of V1.0 results is $s_{V1.0} = 3.13$ percentage points

5. CONCLUSION

The objective of this paper was to enhance the design capabilities of the reinforcement-learning-based optimization algorithm that was developed in [1]. The main goal to remove the fixed division between exploration and exploitation and replace the behavior by a more adaptive and dynamic one has been achieved. A large volume of information about the design space and the influence of individual design parameters on the overall result is evaluated and used dynamically during the optimization process which leads to a global algorithm convergence at an improved rate to better results. Overall algorithmic stability is increased as outliers discovered in early episodes will subsequently be re-investigated. This leads to the algorithm reliably moving away from locally optimal solutions towards a global optimum. With this, it is especially well suited for optimization problems with many design parameters that have a mixed discrete and continuous design space.

However, some caveats remain as these improvements still do not guarantee that a global optimum will be found for all optimization problems. The algorithm has proven to find good final parameter values for a variety of optimization problems with relatively few iterations, but did not reliably find the best results for all test cases. Shallow global optima remain hard to find as shown in the test case of the Levy function. Design parameters with many prominent local optima furthermore hinder the rapid convergence and can lead to oscillations. This can be tackeled by finding more general and robust evaluation criteria to update the PDF in the future.

The main exploration and exploitation behavior is embedded in deterministic manipulations of the PDF that are based on all knowledge available at a certain time during the optimization. Nevertheless, a remaining non-deterministic behavior remains within the algorithm due to the inherent use of random samples to select values from these PDF.

Overall, this new version of the algorithm performs reasonably well on several commonly used test functions and compared to the first version greatly improves the results obtained for the case of a propeller optimization.

Contact address:

moritz.thiele@tum.de

References

[1] Moritz Thiele, Adrian Staudenmaier, Swapan Madabhushi Venkata, and Mirko Hornung. De-

velopment of a reinforcement learning inspired monte carlo tree search design optimization algorithm for fixed-wing vtol uav propellers. In AIAA, editor, *AIAA Scitech 2020 Forum*, Reston, Virginia, 2020. American Institute of Aeronautics and Astronautics. DOI: 10.2514/6.2020-1639.

[2] Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 2012. DOI: 10.1109/TCIAIG.2012.2186810.

[3] Adrian Staudenmaier. Konzept- und machbarkeitsstudie zu reinforcement learning basiertem propeller-design. Mastersthesis, Technical University of Munich, Garching, 2018.

[4] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction*. A Bradford book. MIT Press, Cambridge, Mass., [reprint] edition, 2010. ISBN: 9780262193986.

[5] M. Frank. *Geometriebasierte Abschätzung von unbekannten Profilbeiwerten*. Bachelors thesis, Technical University of Munich, Garching, 2018.

[6] Alexander Guffler. *Multi-Objective Optimization of Slotted Low Reynolds Number Airfoils with Metaheuristic Algorithms*. Masters thesis, Technical University of Munich, 2017.

[7] Thomas S. Huang, Teuvo Kohonen, and Manfred R. Schroeder, editors. *Self-Organizing Maps*, volume 30. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001. DOI: 10.1007/978-3-642-56927-2.

[8] Shai Ben-David Shai Shalev-Shwartz. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, New York, NY, 2014. ISBN: 978-1-107-05713-5.

[9] Moritz Thiele, Martin Obster, and Mirko Hornung. Aerodynamic modeling of coaxial counter-rotating uav propellers. In Vertical Flight Society, editor, *8th Biennial Autonomous VTOL Technical Meeting & 6th Annual Electric VTOL Symposium*, 2019.

[10] Eli B. Giovanetti and KennethC. Hall. Minimum loss load, twist, and chord distributions for coaxial helicopters in hover. *Journal of the American Helicopter Society*, 62(1):1–9, 2017. DOI: 10.4050/JAHS.62.012001.

[11] Moritz Thiele. *Erweiterung und Validierung eines Rotortools mit Vorbereitung einer Konfigurationsstudie*. Masterthesis, Technical University Munich- TUM, München, 2016.

[12] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016. DOI: 10.1038/nature16961.

[13] Michael J. Evans and Jeffrey S. Rosenthal. *Probability and Statistics: The Science of Uncertainty: Perth, Australia, 5-8 December 2016*. W. H. Freeman, 2009. ISBN: 9781429281270.

[14] Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. Go-explore: a new approach for hard-exploration problems. *Nature*, 590(7847):580–586, 2021. DOI: 10.1038/s41586-020-03157-9.

[15] Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc, 2016.

[16] Jürgen Schmidhuber. Developmental robotics, optimal artificial curiosity, creativity, music, and the fine arts. *Connection Science*, 18(2):173–187, 2006. DOI: 10.1080/09540090600768658.

[17] Joel Lehman and Kenneth Stanley. Novelty search and the problem with objectives. pages 37–56. 2011. DOI: 10.1007/978-1-4614-1770-5_3.

[18] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer New York, New York, NY, 2017. ISBN: 978-0-387-84857-0. DOI: 10.1007/b94608.

[19] Wei Xue, Haihong Zhang, Xueyu Wei, Tao Tao, and Xue Li. Optimizing exploration-exploitation trade-off in continuous action spaces via q-ensemble. In Sankal Khanna, Jian Cao, Quan Bai, and Guandong Xu, editors, *PRICAI 2022: Trends in Artificial Intelligence*, pages 148–160, Cham, 2022. Springer Nature Switzerland.

[20] Preetum Nakkiran. Opt2020: 12th annual workshop on optimization for machine learning: Learning rate annealing can provably help generalization, even for convex problems.

[21] D. Ackley. *A Connectionist Machine for Genetic Hillclimbing*. The Springer International Series in Engineering and Computer Science. Springer US, 2012. ISBN: 9781461319979.

[22] A. V. Levy, A. Montalvo, S. Gomez, and A. Calderon. Topics in global optimization. In J. P. Hennart, editor, *Numerical Analysis*, Lecture Notes in Mathematics Ser, pages 18–33. Springer Berlin / Heidelberg, Berlin, Heidelberg, 1982.

[23] Zbigniew Michalewicz. *Genetic algorithms + data structures: = evolution programs ; with 36 tables*. Springer, Berlin and Heidelberg, 3., rev. and ex-

tended ed., 1. corrected printing edition, 1999. ISBN: 978-3-540-60676-5.

[24] Charlie Vanaret, Jean-Baptiste Gotteland, Nicolas Durand, and Jean-Marc Alliot. Certified global minima for a benchmark of difficult optimization problems, 2014.