Neural Computing and Applications (2020) 32:13885–13916
https://doi.org/10.1007/s00521-020-04794-x

ORIGINAL ARTICLE

Coping with opponents: multi-objective evolutionary neural networks for fighting games

Steven Künzel[1] · Silja Meyer-Nieberg[1]

Received: 17 October 2018 / Accepted: 17 February 2020 / Published online: 14 March 2020

Abstract
Fighting games represent a challenging problem for computer-controlled characters. Therefore, they have attracted considerable research interest. This paper investigates novel multi-objective neuroevolutionary approaches for fighting games focusing on the Fighting Game AI Competition. Considering several objectives shall improve the AI's generalization capabilities when confronted with new opponents. To this end, novel combinations of neuroevolution and multi-objective evolutionary algorithms are explored. Since the variants proposed employ the well-known $R2$ indicator, we derived a novel faster algorithm for determining the exact $R2$ contribution. An experimental comparison of the novel variants to existing multi-objective neuroevolutionary algorithms demonstrates clear performance benefits on the test case considered. The best performing algorithm is then used to evolve controllers for the fighting game. Comparing the results with state-of-the-art AI opponents shows very promising results; the novel bot is able to outperform several competitors.

Keywords Neuroevolution · Evolutionary algorithms · Multi-objective optimization · NEAT · Fighting games

1 Introduction

Modern digital games rely strongly on artificial intelligence (AI) techniques for creating interesting non-player characters (NPCs) which challenge or support the human player. Artificial intelligence for games [1] has therefore gained considerable interest—in game development as well as in research. Two main points come into play: First of all, it is the goal to create games which gain and retain the player's interest for a long time. The main contributing factors are convincing and appropriately challenging non-player characters. On the other hand, research interest in digital games arises from the intrinsic nature of these games themselves. Coming from the algorithmic point of view, they represent important and often difficult test cases for algorithms aiming for the control of agents in dynamic and non-deterministic environments. Non-player characters in digital games have to cope with human user input and with actions that are difficult to foresee and may appear random. Furthermore, the reaction of the NPC must be close to real time and be adapted, if possible, to the skill level of the player.

1.1 Fighting games as testbed for artificial intelligence

This paper focuses on neuroevolution for fighting games which are also called *'Beat 'em Up'* games [2]. These adversarial games typically consist of fights between two characters, each endowed with a certain number of life points [3]. The fight continues until the time limit is reached or until one of the characters does not have any points left.

Fighting games are of interest because they represent one of the major areas in commercial games. In addition, they are a special case of games of synchronized moves which remain difficult for AI players [4]. In recent years, there has been increased research interest expressed by several competitions at the major conferences in computational and artificial intelligence. Up to now, several techniques have been applied for fighting games, see Sect. 4 for more information.

✉ Steven Künzel
steven.kuenzel@unibw.de

Silja Meyer-Nieberg
silja.meyer-nieberg@unibw.de

[1] Department of Computer Science, University of German Federal Armed Forces Munich, Werner-Heisenberg-Weg 39, 85577 Neubiberg, Germany

1.2 Neuroevolution and multi-objective optimization

This paper focuses on reinforcement learning and evolutionary neural networks or neuroevolution. Both have been applied successfully to fighting games before [5]. Evolutionary neural networks represent hybrid approaches combining artificial neural networks and evolutionary algorithms. Two main areas can be distinguished. In the case of the first, the evolutionary algorithm substitutes the weight learning algorithm of traditional neural networks, see e.g. [6]. In the case of the second, the structure as well as the weights is adapted with the help of evolution [7, pp. 8–12]. Here, the Neuroevolution of Augmenting Topologies (NEAT) [7] represents the major algorithm class.

When designing a controller for a fighting agent several objectives need to be taken into account: A simple example consists of adapting an agent that strives for a strategy that maximizes the damage to the opponent and minimizes its own loss. Or on the other hand, one can look for an NPC which is capable of beating the player while behaving as human-like as possible. In both cases, it may not be possible to optimize all criteria simultaneously. Here, multi-objective approaches come into play. A further point is made by considering that an agent should be able to cope with several opponents that employ different strategies. Here, either a fast online adaptation is required [2] or general fighting strategies must be developed. In the case of the latter, the consideration of several criteria may support the process. The present paper provides a first investigation concerning this research question. While multi-objective formulations arise naturally in the area of fighting games, however, they are seldom considered.

The same holds for the general area of neuroevolution. So far, only a few multi-objective approaches can be found in the literature: The modular multi-objective NEAT (MM-NEAT) [8] which evolves modular ANNs based on NSGA-II [9] and NEAT. Van Willigen et al. proposed the NEAT-PS [10] that utilizes the Pareto strength approach of SPEA2 [11] to aggregate a vector of fitness values into a single scalar and make it applicable in standard NEAT. Furthermore, NEAT-MODS [12] is a multi-objective version of NEAT that promotes an elitist and diverse population by a modified selection procedure. Section 2.1 will provide a more comprehensive description of these algorithms.

1.3 Novel contributions

Recently, we proposed two novel multi-objective variants of NEAT [13] which use principles of modern multi-objective algorithms that allow operating with a larger number of objectives. Details are provided in Sect. 2. This paper extends the work of [13] by focusing on the important selection procedure in EMOAs. To improve the performance of the algorithms, it introduces novel sorting mechanisms, both based on the $R2$ indicator. In addition, this paper also contributes to the multi-objective optimization itself. The determination of the $R2$ indicator value requires considerable computational effort which typically increases with the number of objectives. For this reason, approximations are often used. This paper introduces a novel efficient approach of computing the exact $R2$ contribution of each solution of a μ-sized population, which is faster than the original method by factor μ.

Based on the multi-objective NEAT variant nNEAT, a novel ANN-controlled agent for fighting games is developed and evaluated in this paper. To this end, it also introduces a new lightweight simulation framework based on the FTGAIC, which allows to evolve and especially evaluate AI controllers much faster than previously possible. This is essential not only for evolving ANNs efficiently but also for creating and testing other AI players that require game simulations.

1.4 Outline of the paper

The paper is structured as follows: Sect. 2 introduces our multi-objective algorithms mNEAT and nNEAT [13] and proposes two novel sorting mechanisms for those based on $R2$ contribution [14]. In Sect. 3, a benchmark problem for multi-objective neuroevolution is described and furthermore experiments to assess the quality of the novel sorting mechanisms are conducted. Additionally, our neuroevolutionary algorithms are compared to NEAT-PS and NEAT-MODS to investigate whether one of the algorithms performs superior. Section 4 provides details concerning the FTGAIC. In addition, the modifications to the FTGAIC software that have been necessary for evolving the corresponding controller are described. Section 5 characterizes the underlying artificial neural network of the ANNBot, including its input and output values and controller strategy. Furthermore, the details of the experiments, that is, how the ANNs are trained and the results of the fitness evaluation of the ANNBot against other, already established, AI opponents in the FTGAIC are provided. Finally, the paper is concluded with a summary of our experiments' results in Sect. 6. A conclusion and outlook on future work is given in Sect. 7. Additionally, the proof of the novel computing procedure for the $R2$ contribution is provided in "Appendix 1".

2 Neuroevolution and multi-objective optimization

This section provides the methodological background of our approach. First, a brief overview concerning neuroevolution and multi-objective optimization is given. Afterwards, the two novel algorithms developed are described. Both can be combined with different sorting procedures for the selection. Two new sorting mechanisms for our algorithms are introduced. One of these focuses mainly on maximizing quality, while the second promotes diversity as well as quality.

2.1 NEAT for single- and multi-objective applications

Neuroevolution also referred to as evolutionary neural networks represents a hybrid approach which combines evolutionary algorithms with artificial neural networks. This paper focuses on topology and weight evolving artificial neural networks (TWEANNs) and more specifically on the Neuroevolution of Augmenting Topologies (NEAT). This method has been introduced by [7] and [15] and has emerged as one of the standard techniques in this area. It uses a genetic algorithm to adapt the structure as well as the weights to the task at hand. To achieve this, it operates with two gene types: node genes which code the neurons of the network and link genes which represent the links between the neurons. NEAT considers specially adapted crossover and mutation operators that work with the particular network structure. In general, feedforward and recurrent structures can be evolved. The NEAT approach is based on three main principles, see e.g. [15],

1. *Complexification* NEAT starts with simple structures which grow if necessary during the evolutionary process.
2. *Speciation* Whenever a topology change occurs, the weights of the structure need to be adapted accordingly. Therefore, even optimal structures require time until they perform well. To protect topology changes against a premature discarding, NEAT groups networks with similar structures into species and mainly only allows competition inside the groups.
3. *Historical marking* In order to enable crossover between nonlinear structures as networks, the degree of similarity between networks and network parts needs to be determined. Historical markings record when a particular change, i.e. a topological mutation occurred. Therefore, it can be used to determine similar parts of the networks.

The general version of NEAT considers scalar fitness functions to assess the quality of a population member. As stated previously, only a few approaches beside ours have been developed which account for several potentially conflicting objectives. The presence of several objectives necessitates a more general concept to compare candidate solutions: the Pareto dominance. In short, a population member dominates another population member if it surpasses it in one objective and is at least as good as the other concerning the remaining ones.

One main point, where evolutionary multi-objective algorithms differ from their single-objective counterparts, is the selection procedure. In general, three main classes of algorithms can be distinguished. One group operates with decomposition techniques. The next class first sorts the population with respect to the Pareto dominance into classes. Then, a secondary selection criterion is applied. Here, several concepts have been introduced. They range from density measures which shall result in a higher population diversity to so-called quality indicator functions as e.g. the Hypervolume indicator also called $\mathcal{S}$-metric or the $R2$ indicator. Well-known examples of the class include the NSGA-II [9], the SPEA2 [11], or the SMS-EMOA [16]. The latter considers the concept of a dominated Hypervolume. The third group of algorithms only uses the quality indicator to compare solutions as for example the IBEA [17]. Many of the recently introduced variants of multi-objective evolutionary algorithms utilize quality indicators at least as the secondary measure.

Neuroevolution and multi-objective optimization have already been combined. Our literature research resulted in three main variants: MM-NEAT, NEAT-PS, and NEAT-MODS which are described in the following.

MM-NEAT Schrum and Miikkulainen [8] introduced the modular multi-objective NEAT (MM-NEAT) which evolves modular ANNs using a combination of the NSGA-II [9], one of the major evolutionary multi-objective algorithms (EMOAs), and NEAT. Modular ANNs may consist of multiple modules, each resulting in different outputs and thereby control strategies, e.g. to handle different sub-tasks. Along with special module mutation operators, MM-NEAT employs NEAT's ANN representation and variation operators. MM-NEAT is based on NSGA-II with components borrowed from NEAT [8].

NEAT-PS Van Willigen et al. [10] proposed the NEAT-Pareto strength (NEAT-PS) which falls back to another well-known EMOA, the SPEA2 [11]. Here, the authors mainly utilize the Pareto strength approach which aggregates a fitness vector into a scalar fitness value for each solution in a given set. Using that scalar, any multi-objective optimization problem can be treated as a single-objective one in NEAT-PS. Thus, the only

addition to NEAT is the Pareto strength fitness determination in NEAT-PS [10].

NEAT-MODS Based on investigations of NEAT-PS that revealed that the algorithm does not guarantee a uniform evolution of all objectives due to its focus on elitism only, Abramovich and Moshaiov proposed the NEAT-multi-objective diversified species (NEAT-MODS) [12]. NEAT-MODS is built on NEAT and mainly differs in three components: (1) Population: While NEAT discards all solutions of the prior generation in favour of the offspring, NEAT-MODS takes all solutions from the parental and the offspring generation into account for survivor selection. (2) Sorting: These solutions are sorted based on the procedure of NSGA-II, namely combining non-dominated ranking and crowding distance. Thereby, a ranking of all solutions is established. (3) Survivor selection consists of two sub-steps: (3a) Species selection: The first $q = \mu/K$ species occurring in the sorted list of solutions are taken into account for member selection. Every species is only added once to that list. (3b) Member selection: Iteration over all selected species while adding the best member each to the next generation. In the next iteration, add the second best member of each species to the next generation. This is continued until no more slots are available in the next generation. If a species s does not have enough members, i.e. $|s| < n$ when the nth member is queried, it is skipped. On the account of selecting the best members of each species, NEAT-MODS achieves a worthwhile goal, namely maintaining a diverse population of elitist solutions [12].

Reference [13] introduced two novel algorithms: mNEAT and mNEAT-IB. Note that in order to distinguish more clearly between the two different approaches, this paper refers to mNEAT-IB as nNEAT, where the "n" stands for a next variant of a multi-objective NEAT. Both algorithms combine neuroevolution with multi-objective principles. However, they differ in the design and the degree of similarity by which they are either closer to the original NEAT procedures or to the respective underlying multi-objective algorithm. The first, mNEAT, is an extension of the original NEAT to evolve ANNs for multi-objective tasks. The latter, nNEAT, is a novel combination of SMS-EMOA [16] and NEAT. An overview of the described multi-objective variants of NEAT, their characteristics, and differences is given in Table 1.

First experiments in [13] have shown that the variation of using only the $R2$ indicator as quality indicator performs best. However, as experiments in Sect. 3 reveal, further promising combinations exist. Additionally, the $R2$ indicator's performance deteriorates with increasing population size. The procedure of both algorithms is briefly described below.

Multi-Objective NEAT The multi-objective NEAT (mNEAT) augments the original NEAT to cope with a multi-objective task. Therefore, mainly the quality assessment and speciation procedure of NEAT have been adopted, while the remaining features of NEAT remained unchanged. The current version of mNEAT does not adjust the speciation threshold automatically in order to match a user-defined number of species. This feature can easily be re-enabled by the user finally. The mNEAT starts with a population $\mathcal{P}$ of μ minimal random networks. The algorithm requires that all networks have already been evaluated at the beginning of each epoch. The networks in $\mathcal{P}$ are sorted depending on their quality contribution and then speciated, i.e. sorted into species. If there are already any existing network-to-species mappings, those are removed in advance. The different species are stored in a set of species S. Here, a difference to the initial mNEAT proposed in [13] which affects the species' fitness occurs. As mNEAT is capable to deal with multiple objectives, a more sophisticated approach is applied. All population members are copied into a temporary set F. Each solution in F has $K + 1$ fitness values, whereas the first K values are taken from the original solution and the last fitness value equals the number of members of the species the solution belongs to. The last objective, i.e. the species size, is applied as an additional helper objective here; this approach has, among others, been proposed in [18, 19]. In the $(K + 1)$-dimensional objective space, the quality contribution of each solution $f \in F$ is determined and added to the total contribution of the containing species. This procedure penalizes members of large species only in one objective. In contrast to the previous approach which summed up the values of all members in each objective, the remaining objectives are not affected. On the other hand, large species have an advantage compared to small species because there exist more members that can contribute to the species' contribution. This will only be the case if a large species s' contains many well-performing solutions in the context of F, thereby s' should be allowed to produce an appropriate amount of offspring anyway. A disadvantage that naturally occurs using this approach is that the dimension of the objective space is increased by one. The number of offspring a species s is allowed to spawn depends on its quality contribution in relation to the sum of the quality contributions of all species. Every epoch all solutions of $\mathcal{P}$, except the species' representatives (best member of a species), are discarded from the population. Thus $\mu - |S|$ new solutions need to be created using the variation operators of NEAT. To avoid discarding promising solutions, all non-dominated solutions are stored in an external archive. The population thereby explores

Table 1 Multi-objective variants of NEAT

	Framework	Sorting	Remarks	References
NEAT-PS	NEAT	PS	Original NEAT; PS aggregates fitness vector into scalar	[10]
NEAT-MODS		NDR + CD	Steady-state population model; two-staged survivor selection	[12]
mNEAT		Any multi-objective	Multi-objective speciation; archive of best solutions	[13]
MM-NEAT	NSGA-II	NDR + CD	Modular ANNs and corresponding variation operators	[8]
nNEAT	SMS-EMOA	Any multi-objective	–	[13]

PS Pareto Strength, *NDR* Non-dominated ranking, *CD* Crowding distance

The column *Framework* indicates on which algorithm the corresponding algorithm is based. All non-NEAT-based algorithms employ at least NEAT's representation and variation operators for neuroevolution. The term *Any multi-objective* means that an arbitrary combination of multi-objective quality measures can be applied

promising solutions, while the archives ensure that no progress gets lost. The whole procedure is repeated until predefined stopping criteria are fulfilled [13].

nNEAT The nNEAT (called mNEAT-IB in [13]) is a combination of SMS-EMOA [16] and NEAT. It retains the genetic encoding of ANNs, the innovation IDs, and the variation operators (crossover and mutation) from NEAT and combines these with the framework provided by the SMS-EMOA. First, an initial population $\mathcal{P}$ of μ minimal networks is created and evaluated. The individuals in $\mathcal{P}$ are sorted using an arbitrary sorting mechanism, e.g. utilizing a primary and optionally a secondary sorting criterion. Then, 2λ solutions are selected using stochastic universal sampling from $\mathcal{P}$ to create λ offspring by crossover and mutation. The population has a size of $\mu + \lambda$ now; before reduction, the λ new solutions need to be evaluated. After evaluation and sorting (at the beginning of the next epoch), the population is reduced by the worst λ solutions (steady-state population model [20, p. 80]). The procedure then continues with parent selection for the next generation [13]. Different combinations of sorting criteria are allowed. [13] conducted an experimental analysis using non-dominated sorting, the Hypervolume, and the $R2$ indicator. Our experimental analysis on the three-dimensional double pole balancing problem has shown that the nNEAT, especially when using the $R2$ indicator for sorting, was very promising compared to other variations and the previous version of mNEAT. It should be noted that both algorithms and all investigated variations of nNEAT were capable of finding promising ANN controllers for the cart within very few evaluations [13]. In Sect. 3, multi-objective double pole balancing problem is investigated once more in order to compare mNEAT and nNEAT to NEAT-PS and NEAT-MODS. Note that the experiment is carried out under different conditions, for example without a bias neuron that was part of each ANN in the previous experiment. This helped to reduce the number of necessary evaluations significantly. See [7, p. 50] for more details on the removed bias neuron.

2.2 Novel sorting mechanisms

Our experiments in [13] have shown that the performance of nNEAT strongly depends on the quality assessment of candidate solutions. As it was revealed, the combination of different measures may play an important role. As stated, it is common procedure in EMOAs to combine primary and secondary selection criteria. In general, these shall promote the creation of a set of candidate solutions of good quality together with a high diversity. In this section, two quality measures for our algorithms, both based on the $R2$ indicator, are introduced.

Sorting Solutions In order for these quality measures to work, a structure `Sorting-Mechanism` (short `SM`) that fulfils the following conditions is presupposed. A detailed textual description and examples are foregone here in order to keep the description as short as possible. Let there be a set $A \subset \Omega$ of at least two solutions, where Ω is the universe of all possible solutions, a `SM` s for which three functions are defined

$$\begin{aligned} q: &\quad \Omega \times \texttt{SM} \rightarrow \mathbb{R} \\ next: &\quad \texttt{SM} \rightarrow \texttt{SM} \\ sort: &\quad A \times \texttt{SM} \rightarrow A \end{aligned}$$

where q assigns a scalar utility value to a solution depending on s, the utility is to be maximized. *next* returns the subordinate `SM` of s or `null`, if it is not defined. The function *sort* takes a list of solutions and returns that list, sorted by the function q of s and its subordinate $next(s)$. Furthermore, an `SM` may be combined from multiple sub-`SM`s, which compute the q-values for all solutions of a list separately that are combined finally, e.g. by multiplication. The *sort*-function of the superordinate `SM` then has to ensure that the conditions defined in the following are

fulfilled in that case, too. The requirements to q are as follows:

1. $\forall a \in A : 0 \leq q(a,s) \leq 1$
2. $\exists a,b \in A : a \neq b \wedge q(a,s) = 0 \wedge q(b,s) = 1$

Furthermore, if there is a next SM t assigned to s

$$t = next(s) \neq \texttt{null}$$

and there exist one or multiple subsets of solutions with equal value c for q:

$$\exists B \subseteq A \wedge c \in [0,1] : \forall b \in B : q(b,s) = c.$$

Then, if $0 < c < 1$ there exist two bounding utility values q_{low} and q_{high} defined as follows:

$$\exists i,j \in A \backslash B \wedge i \neq j : q(i,s) = q_{\text{low}} < q_{\text{high}} = q(j,s) \wedge \nexists i',j' \in A \backslash B : q_{\text{low}} < q(i',s) \leq c \leq q(j',s) < q_{\text{high}}.$$

Otherwise, if $c = 0$ or $c = 1$, q_{low} is set to zero, respectively q_{high} to one. Furthermore, let there be

$$q_{\text{diff}} = q_{\text{high}} - q_{\text{low}}.$$

and a constant $0 < u < \frac{1}{2}$. To ensure that *sort* returns the list of solutions sorted at most possible extend, the value of q for all solutions of equal q-value is redefined by t's q in the borders of $[q_{\text{low}}, q_{\text{high}}]$, if t is defined:

$$\forall b \in B : q(b,s) = q_{\text{low}} + q_{\text{diff}}(u + q(b,t)(1-2u)).$$

The constant u maintains a certain distance to q_{low} and to q_{high}.

An implementation of a `SortingMechanism` fulfilling these conditions allows combining different sorting procedures and quality indicators in arbitrary order. Only the implementation of q has to be done properly for each different measure. For example the combination of non-dominated sorting as s and the $R2$ indicator as $next(s)$ allows first to sort a list of solutions by the means of Pareto dominance. All solutions belonging to the same front then will be sorted using their corresponding $R2$ contribution. The `SortingMechanism` has been implemented for our experiments.

2.2.1 Iterative *R*2 ranking

Díaz-Manríquez et al. [14] introduced a novel ranking method using the $R2$ indicator, which is based on the following idea: Whenever the $R2$ contribution of all μ solutions of a population $\mathcal{P}$ is computed ($\forall i \in \mathcal{P} : R_{2_c}(i) = R_2(\mathcal{P} \backslash \{i\}) - R_2(\mathcal{P})$), it is likely that only a part of $\mathcal{P}$'s solutions have a positive $R2$ contribution. The others remain with a contribution of zero; however. Díaz-Manríquez et al. created a ranking which is similar to the approach of non-dominated sorting: First, the $R2$ contribution for all $i \in \mathcal{P}$ is computed. Then, all solutions $i \in \mathcal{P} : R_{2_c}(i) > 0$ are removed from $\mathcal{P}$, sorted in decreasing order and added to the sorted population $\mathcal{P}'$. These solutions represent rank 1. For the remaining solutions $\mathcal{P} \backslash \mathcal{P}'$, this procedure is repeated until $|\mathcal{P}| = 0$ or $\forall i \in \mathcal{P} : R_{2_c}(i) = 0$. In the latter case, the individuals of worst rank remain unsorted, because each had a $R2$ contribution of zero. For further sorting, another quality measure may be applied. To continue, however, may not be worth the effort because it only affects the worst solutions in the population. Now most, instead of only the best, solutions in $\mathcal{P}'$ are ordered depending on their $R2$ contribution. In the worst case, this requires μ iterations. Because the computation of the $R2$ contribution of all solutions of a μ-sized population for K objectives requires time $O(K\mu^2|\Lambda|)$, Díaz-Manríquez et al. suggest an efficient way of computing an approximation of the $R2$ contribution: They assume that the $R2$ contribution of each solution is the sum of the contribution for all weight vectors, the solution contributes most [14]. This is a very efficient approximation, but does not take the contribution of the rest of the population into account. Therefore, additionally the second best contributing solution for each weight vector has to be considered. An approach for computing the exact $R2$ contribution for all solutions of a μ-sized population for K objectives in time $O(K\mu|\Lambda|)$ is described in "Appendix 1".

The runtime of iterative $R2$ ranking to sort a population, in combination with our exact method for computing the $R2$ contribution of each solution, is bounded between $O(K\mu|\Lambda|)$, if all solutions have a positive $R2$ contribution in the first iteration, and $O(K\mu^2|\Lambda|)$, if in every iteration only a single solution has a positive $R2$ contribution. For the computation of the $R2$ contribution of the solutions, a default amount of $|\Lambda| = 100$ weight vectors is used throughout this paper.

2.2.2 Quality and diversity: the QD measure

An important aspect for genetic algorithms is the diversity of the underlying population which can be defined in various ways. One meaning of diversity can be summarized informally as the number of different individuals a population consists of. This can be measured in different ways, for example as the difference in genotypes or phenotypes or as the difference in behaviour, i.e. fitness values, or statistical measures such as entropy [20, p. 31]. The speciation in mNEAT determines the similarity of ANNs by comparing their genotypes, whereas the approach described in this section defines the similarity of two ANNs or,

more generally spoken, solutions by the means of their fitness values. This will be referred to as the *behavioural diversity* in the context of this paper.

The maintenance of diversity plays also an important role in popular algorithms like NSGA-II and SMS-EMOA. While Deb et al. apply the Crowding Distance, which employs behavioural diversity as secondary sorting criterion in NSGA-II [9], Beume et al. utilize the Hypervolume indicator for that purpose. The Hypervolume contribution of a solution is equal to the objective space that is dominated exclusively by that solution; hence, a solution with greater Hypervolume contribution is located in a less dense settled area of the objective space than a solution with lower Hypervolume contribution [16]. Thereby, both algorithms rely on a quality measure that fosters behavioural diversity as secondary sorting criterion.

Falcón-Cardona et al. [21] recently proposed the CRI-EMOA which utilizes two different quality indicators IGD^+ [22] and the Riesz s-energy [23]. The CRI-EMOA combines the strengths of both indicators, the IGD+ in promoting convergence and the Riesz s-energy in fostering diversity in the population, in order to be applicable efficiently for different multi-objectives problems, independent from the shape of the Pareto front. The algorithm applies non-dominated sorting as primary sorting criterion and then switches between the two indicators, depending on the convergence behaviour over the last generations. If it thereby has been found that the evolutionary progress is stagnating, the Riesz s-energy is applied in order to increase the diversity in the population. Otherwise, the IGD^+ is applied [21].

While NSGA-II and SMS-EMOA clearly focus more strongly on the convergence (non-dominated ranking, first criterion) than on diversity (Crowding Distance respectively Hypervolume, second criterion), CRI-EMOA selects the priority of convergence and diversity dynamically based on the current situation. Therefore, additional effort has to be put into the analysis of the "situation". For our mNEAT and nNEAT, another quality measure is suggested that promotes diversity as well as convergence. In order to achieve this, always two quality values for each solution in the population are determined: One value describing the solution's quality with respect to convergence and another value that represents the solution's behavioural diversity. These values are normalized over the whole population between 0 and 1, as previously described. Finally, the total contribution each solution is the product of its quality and diversity value. Thereby a solution does only have a good total contribution if it can achieve considerable values in both, quality and diversity. Only being a well-performing solution, among many well-performing solutions will not be enough in order to compete with other solutions. The procedure of our approach is very straightforward in order to promote diversity and quality with equal priority. In future, more sophisticated variations are possible, for example, dynamically adjusting the priority of diversity and quality depending on the current population and previous generations, similar to [21].

A sample implementation for the *QD measure* is given by

- Quality = Non-dominated ranking + $R2$ indicator,
- Diversity = Crowding Distance.

The goal is to sort the population as fine-grained as possible with respect to quality and diversity. The crowding distance will serve as an appropriate measure for the behavioural diversity as it is likely to assign unique values to each solution that then can be sorted in decreasing order. Our implementation differs to the one applied in NSGA-II in so far that the solutions with the lowest and highest values in each objective will get assigned a value of 1 (instead of ∞). Please keep in mind that all objective values are normalized between 0 and 1, and thereby 1 is the maximal theoretically possible distance per objective. For sorting the population by quality, the combination of non-dominated ranking as first, and $R2$ indicator as secondary criterion were selected. Of course, this fits exactly into the application field of the iterative $R2$ contribution approach introduced in Sect. 2.2.1. On the other hand, this often requires a large number of iterations until a population is sorted completely. Therefore, this paper follows the sorting scheme proposed by R2-EMOA [24] in order to reduce the number of iterations for sorting the population depending on quality. The `SortingMechanism` introduced earlier in this section determines and normalizes the quality and diversity values for all solutions in the population and determines their final contribution values by multiplying the respective quality and diversity values with each other.

The runtime of the QD measure depends on the runtime of the single measures applied for quality and diversity determination. In our example, implementing the quality determination has a worst case runtime of $O(K\mu^2 + K\mu|\Lambda|)$. On the other hand, the runtime of the crowding distance computation is determined by the sorting of the population, which is necessary for all K objectives and therefore requires $O(K\mu \log \mu)$ [9].

3 Experimental analysis: a multi-objective double pole balancing problem

Reference [13] introduced a multi-objective version of the well-known double pole balancing problem, see e.g. [7]. The problem is an extension of the pole balancing problem [25], which describes the following task: Two poles are

mounted on a cart using a hinge. The cart is placed at the centre of a track of limited length. When the experiment starts, the cart has to be moved along the track in order to keep the poles balanced. Furthermore, the cart is not allowed to leave the track. If it does leave the track or at least one of the poles falls off to the left or right, the experiment has failed. The double pole balancing problem has also been applied by Stanley [7] in order to compare the performance of NEAT to other algorithms. Our multi-objective version (moDPB) extends the problem, which initially takes the time in balance (f_1) as fitness function, by two fitness functions: the number of directional changes of the cart (f_2) and the mean distance of the cart from the centre of the track (f_3). By adding two additional fitness functions, the task now is to find a controller that is not only able to balance the poles on the cart, but also keeps the cart at a safe distance from the edges of the track and consumes only a low amount of energy. The controller for the cart is represented by an ANN which gets the position of the cart, the angles of the poles, and the velocities of the cart and the poles as input.

This section provides a comparison of our novel algorithms to other multi-objective versions of NEAT. For a comprehensive analysis and comparison of the algorithms' performance, the following procedure is employed:

The experiment is repeated for 50 times using each algorithm (or variation of algorithm), and the default parameter configuration, shown in Table 2, is used. In addition, all networks use the same activation function, the sigmoid function[1] given by

$$f(x) = \frac{1}{1 + \mathrm{e}^{-4.9x}}.$$

Furthermore, different population sizes $\mu = \{50, 200, 500\}$ are investigated. The experiment is stopped after a predefined number of evaluations, i.e. created and evaluated solutions, have been reached. During each experiment, information about the current population is extracted at several points of the evolutionary process. In addition, information about the fitness values of the best solutions found during each run is stored.

The analysis of the algorithms' performance is based on the following measures which are commonly considered in literature:

Runtime Behaviour The runtime behaviour is determined from the Hypervolume dominated by a population after a certain number of evaluations. It gives information about the convergence behaviour of an algorithm. Is it converging fast within a small number of evaluations or does it require a large number of evaluations? To which Hypervolume level it converges at all?

Effectiveness The effectiveness of an algorithm can be assessed by evaluating its mean best fitness (MBF). This comprises the average fitness of the best solutions found during a series of experiments. Better average values indicate that an algorithm can find solutions of better quality than another algorithm with worse values, i.e. it is more effective. This measure plays an important role for experiments that might not be repeated for several times, i.e. being run for a single time and thereby "have to" find a good solution [20, p.151f].

Efficiency The efficiency of an algorithm can be determined by investigating the average number of evaluations to find a solution (AES) of at least *satisfying fitness*. The term satisfying fitness describes a minimum fitness threshold that an ANN x^* has to fulfil. For moDPB, it is defined as follows: $f_1(x^*) = 0 \wedge f_2(x^*) \leq \frac{1}{20} \wedge f_3(x^*) \leq \frac{1}{5}$. These fitness values were chosen since many controllers exist that easily can achieve good results in f_1 but require a large amount

Table 2 Parameters of mNEAT and nNEAT and their respective default values

Name	Value
Weight mutation range	2.5
Modify weight probability	0.2
Add neuron probability	0.03
Add link probability	0.05
Looped connection probability	0.2
Crossover probability	0.5
Mutation probability	0.5
Gene enabled on crossover probability	0.1
Mate by choosing probability	0.6
Interspecies mating rate	0.001
Survival threshold	0.2
Replacement rate	0.3
Speciation coefficient	0.25
Factor C1 excess	1
Factor C2 disjoint	1
Factor C3 weight difference	0.4
Age bonus malus	0.1
Maximum stagnation	15
Selection pressure	2
Age threshold young	10
Age threshold old	50

The parameters that do also occur in NEAT-PS and NEAT-MODS will get assigned the specified default values. An explanation of the parameters can be found in "Appendix 2"

[1] The value for $k = 4.9$ has been proposed by Stanley [7, p. 146] and is also applied here.

of energy. These values define an arbitrary border of minimum efficiency that might be shifted by the user. This will also very likely shift the average number of evaluations. An algorithm that requires least evaluations to find a solution x^* than another algorithm is stated to be more efficient [20, p. 152f].

Success Rate With the values extracted for AES and the maximum number of evaluations, one can easily define a measure for the overall success rate (SR) of an algorithm. In how many experiments of a series can the algorithm find a solution x^* before the evolutionary process is terminated? The success rate is a measure for effectiveness: An algorithm with higher success rate is stated to be more effective (on the given problem instance) than one with a lower success rate [20, p. 152].

Diversity The diversity in a population is a crucial element for exploring new regions of the search space [20, p. 87]. If the diversity decreases far enough, evolutionary methods might not produce any purposeful offspring by the means of exploration any more. In that case only exploitation, i.e. investigating promising regions of the search space in detail can be performed. The evolutionary process should be designed as a trade-off between exploration and exploitation [20, p. 42]. Therefore, it is worth and necessary to maintain the diversity in the population at a level as high as possible. Two different measures for diversity will be investigated: (1) Genotype diversity (Stanley's ANN distance measure [7, pp. 38–40]) and (2) behavioural diversity (Crowding Distance [9] with maximum value 1 per dimension). In this paper, the diversity of a population equals the average distance (which depends on the measure applied) of each pair of solutions in the population.

In order to assess whether the differences observed are statistically significant, statistical tests are carried out. First, a Friedman test [26] is conducted to determine if there is any significant difference within a set of samples. If there is a difference, a pairwise Friedman post hoc test [26] reveals between which samples the significant difference(s) exist. For the correction of the family-wise error rate, Shaffer's algorithm [27] is applied. Since only successful runs are taken into account in AES, there may be data missing. This does not allow to conduct a Friedman test. Instead, the Skillings-Mack test [28] that is designed to deal with incomplete data sets will replace the Friedman test.

In this section, the effect of the novel sorting mechanisms (Sect. 3.1) introduced in Sect. 2.2 will be investigated. Furthermore, our novel neuroevolutionary algorithms are compared to already existing multi-objective versions of NEAT (Sect. 3.2). Finally, a summary of our findings will be given. All tables in the following sections are emphasized, where the italic stands for the best value(s) and bold represents the worst value(s). All values in between will remain de-emphasized.

3.1 Influence of the sorting mechanism

This section investigates the influence of the new sorting mechanisms introduced in Sect. 2.2. The analysis is based on the nNEAT variant which uses the $R2$ indicator as the only sorting criterion. The reason of this choice is based on experiments previously carried out for the moDPB [13]. The nNEAT or mNEAT-IB has been referred to in [13] with only the $R2$ indicator emerged as the most promising combination. Therefore, it was selected for the present series of experiments. Please note that several differences exist between the experimental set-up in [13], and here, [13] only investigated a small population of $\mu = 50$ individuals, additionally the maximal runtime was set to 25,000 evaluations. Also all ANNs started with an additional bias neuron which is not part of the ANNs in this paper's experiments; this mainly explains the lower number of evaluations necessary on average.

The following abbreviations are used in the tables and figures in this section:

- *A1* = nNEAT, Sorting: $R2$ indicator
- *A2* = nNEAT, Sorting: iterative $R2$ indicator
- *A3* = nNEAT, Sorting: QD measure (Q: Non-dominated Ranking + $R2$ indicator, D: Crowding Distance)

Table 3 shows the success rate of the different sorting mechanisms. It is discernible that all sorting mechanisms result in near perfect success rates for a small population size (i.e. $\mu = 50$). While the A1 variant exhibits some runs without achieving a successful outcome, the loss appears negligible. On the other hand, when $\mu = 500$, the success rate of A1 deteriorates dramatically. This is caused by the problem in this case that only a small part of the population is assigned a positive $R2$ contribution. The majority of the population will have a contribution of zero and thereby remain unsorted. Thus, A1 is not capable to select adequate parents for offspring creation in many cases since it has no basis to differentiate. The parent selection in nNEAT is done using Stochastic Universal Sampling, where each individual has a chance to be selected as parent. The larger the population size, the more individuals will remain with a zero contribution and thereby the parent selection will happen mostly randomly. In contrast, A2 and A3 sort the population (nearly) completely and thus give better performing individuals and better chance for reproducing themselves in offspring creation.

Table 3 Success rates of the corresponding algorithms

μ	A1 (%)	A2 (%)	A3 (%)
SR			
50	*98*	**100**	**100**
200	*98*	**100**	**100**
500	*52*	**100**	**100**

A run of an algorithm is said to be successful if it found a solution of (at least) satisfying fitness within 15,000 evaluations

Table 4 Number of evaluations necessary to find a solution of (at least) satisfying fitness

μ	A1	A2	A3
Evaluations			
50	1914	*2013*	**1214**
	± 1785	± *1840*	± **592**
200	*5051*	**2344**	2424
	± *2741*	± **838**	± 763
500	*9317*	**3864**	4410
	± *3042*	± **1223**	± 1134

Note that only successful runs, i.e. runs that could find such a solution within 15,000 evaluations, are regarded

The average number of evaluations to a solution (AES) of satisfying fitness is shown in Table 4. When $\mu = 50$, A3 requires only 1, 200 evaluations with a comparably small standard deviation of 600 evaluations. It is discernible that the number of evaluations necessary increases for all combinations when the population size is increased. This happens quite naturally as a larger population means that there is more space for individuals that do not perform as well (while this holds vice versa, too). Furthermore, the number of evaluations for A2 and A3 increases only slightly. Only for A1, the number of evaluations grows strongly, compared to the other combinations. While A2 requires the least evaluations for $\mu \geq 200$, A3 achieves the most consistent results as the standard deviation indicates.

Table 5 shows the mean best fitness values obtained over 50 repetitions. In nearly all instances, A2 achieves significantly better fitness values than A1 and A3.

The only exceptions are f_1 (time in balance) for $\mu \geq 200$, where no significant difference between A2 and A3 exists. Furthermore, when $\mu \geq 200$ the fitness values of A3 are also significantly better than the values achieved by A1. Only for $\mu = 50$, A3 performs worse than A1; this might be explained by some outliers increasing the mean fitness. Summarizing, the fitness values achieved by A1–A3 are acceptable. The only exception is A1 for $\mu = 500$, where the achieved fitness is not able to compete with the other combinations' fitness.

The mean best fitness values are underlined by the runtime behaviour, which are shown in Fig. 1. The figure shows the average Hypervolume of the population at the corresponding number of evaluations. First and foremost, it is visible that all combinations converge the fastest with a small population size. The convergence speed slows down with increasing population size. In particular for A1, the deterioration is serious: While it can keep pace with A2 and A3 when $\mu = 50$, the convergence speed of A1 for $\mu = 200$ is already much slower than the slowest convergence speed of A2, respectively, A3. The situation even deteriorates when the population size is increased further. On the other hand, A1 also converges to the same level as A3; it only requires more evaluations. Since the experiment was stopped after 15,000 evaluations, a statement whether it would also reach that Hypervolume level for $\mu = 500$ is not possible. However, looking at the development during the first 15,000 evaluations, it appears to be likely. For $\mu \leq 200$, A3 increases the population's Hypervolume minimally faster than A2 but achieves a maximum level of 0.97, while A2 converges nearly to 1. For $\mu = 500$, A2 approaches its maximum Hypervolume level minimally faster than A3, and the final level is also higher than the one attained by A3. The runtime behaviour shows that A2 and A3 converge similarly fast but A2 always converges to a higher level than A3 does. A1 converges slower than the other combinations, and the difference becomes even more clear when the population size is increased. The difference of the maximum Hypervolume level achieved by A2 and A3 may exist since A3 sorts a population not only based on quality but also diversity. Thereby, it always gives priority to solutions that are as diverse as possible as well as good performing. Looking only for well-performing solutions without regarding diversity can be useful in "easy to solve" problems, like moDPB with velocities.

Finally, the genotype and behavioural diversity are shown in Fig. 2, respectively, in Fig. 3. All combinations for all population sizes start with (nearly) the same diversity. All percentage data occurring in the following paragraphs refer to this initial value, which represents 100% here.

Let us first consider the genotype diversity, e.g. Fig. 2. In general, the diversity decreases during the evolutionary process. However, the rate of diversity loss is more pronounced for small population sizes. This holds for all combinations. Considering the different variants, the diversity of A2 drops to a level of 10–15% of its initial value and thereby the lowest diversity. This behaviour can be observed for all population sizes. In contrast, A1 is affected strongly by the choice of μ. For $\mu \geq 200$, its genotype diversity is only slightly reduced during the 15,000 evaluations. The reason can be found, in combination with the other data evaluated concerning moDPB, that only very small progress has been made by A1 when

Table 5 Comparison of the average fitness of the best solutions found within 15,000 evaluations

μ	ID	f_1	f_2	f_3
Mean best fitness				
50	A1	0.019	**0.002**	*0.005*
		± 0.134	± **0.002**	± *0.023*
		A2, A3	**A2, A3**	*A2, A3*
	A2	**0.0**	**0.002**	**0.001**
		± **0.0**	± **0.002**	± **0.001**
		A1, A3	**A1, A3**	**A1, A3**
	A3	*0.06*	*0.113*	0.002
		± *0.24*	± *0.306*	± 0.001
		A1, A2	*A1, A2*	A1, A2
200	A1	*0.024*	*0.008*	*0.007*
		± *0.127*	± *0.012*	± *0.017*
		A2, A3	*A2, A3*	*A2, A3*
	A2	**0.0**	**0.002**	**0.001**
		± **0.0**	± **0.002**	± **0.001**
		A1	**A1, A3**	**A1, A3**
	A3	**0.0**	0.003	0.002
		± **0.0**	± 0.003	± 0.001
		A1	A1, A2	A1, A2
500	A1	*0.198*	*0.092*	*0.046*
		± *0.326*	± *0.213*	± *0.08*
		A2, A3	*A2, A3*	*A2, A3*
	A2	**0.0**	**0.002**	**0.001**
		± **0.0**	± **0.003**	± **0.001**
		A1	**A1, A3**	**A1, A3**
	A3	**0.0**	0.005	0.003
		± **0.0**	± 0.003	± 0.002
		A1	A1, A2	A1, A2

The third line for each algorithm indicates the algorithms to which statistically significant differences exist. Note that large values (compared to mean) in standard deviation are caused by a very small number of outliers

$\mu \geq 200$. For a small population of $\mu = 50$, A1 performed similar to A2 and A3, and the population's genotype diversity also approached a level of 25%. Furthermore the diversity of A3 only got reduced to a level of 35–40% and thereby the highest value for genotype diversity in combination with reasonable progress concerning quality.

The situation about the behavioural diversity is similar: All combinations with all population sizes start at a similar initial value and then enter a phase of diversity reduction. This phase is then followed by a phase of increasing diversity; the larger the population the later in evolutionary process this phase starts. This indicates that "after a few generations", the behavioural diversity starts to increase. This phase is either followed by a third phase of decreasing diversity that ends in stagnating diversity values (A1 and A2) or directly in stagnating diversity (A3). Whereas the diversity of A3's populations stagnates at a level of 130–150% and thereby the highest level, A2's populations stagnate at the lowest diversity level of 1–10%. For A1 with $\mu = 50$, the diversity also drops to a level of 48%. For $\mu = 200$, it even increases to 145%. Only for $\mu = 500$, the behavioural diversity of the population stagnates at its initial level. The diversity analysis shows that the diversity of the populations of A2 always attains the smallest values compared to the other combinations. On the other hand, the highest diversity values are achieved by A3. The diversity levels of A1 are dissimilar depending on the population size applied. This might be caused by the problem that when the population size is increased a larger absolute number of solutions remains unsorted every epoch, thereby the nNEAT is not able to drive the development of the population into a certain, purposeful direction.

The experiments conducted show that both quality measures, iterative $R2$ and QD, are able to fulfil the purpose they have been designed for. While iterative $R2$ achieves the best quality within a small number of evaluations, see Fig. 1, the QD is able to keep the genotype and behavioural diversity at a high level, see Figs. 2 and 3. Nevertheless, all combinations investigated here are able to solve the given task reliably within a small number of evaluations. Only A1 requires an increasing amount of time, proportional to population size.

3.2 Comparison to NEAT-PS and NEAT-MODS

As diversity and quality are both important, the QD measure is applied as the default sorting mechanism for mNEAT and nNEAT in the following experiment. Here, our algorithms mNEAT and nNEAT are compared to other multi-objective versions of NEAT, namely the NEAT-PS and NEAT-MODS. The configuration of the four algorithms remains the same as in Sect. 3.1, see Table 2. Note the following abbreviations for the algorithms:

- *A4* = nNEAT, Sorting: QD measure (Q: NDR + $R2$, D: CD)
- *A5* = NEAT-PS, Sorting: Pareto Strength
- *A6* = NEAT-MODS, Sorting: 1. Non-dominated Ranking + 2. Crowding Distance
- *A7* = mNEAT, Sorting: QD measure (Q: NDR + $R2$, D: CD)

Table 6 shows the success rates of the algorithms. It is discernible that A4 is successful in each experiment of the series, while A5–A7 are not able to find a solution of satisfying fitness in each repetition. While the success rates of A5 and A7 increase with the population size, the success rate of A6 seems to be independent from the choice of μ.

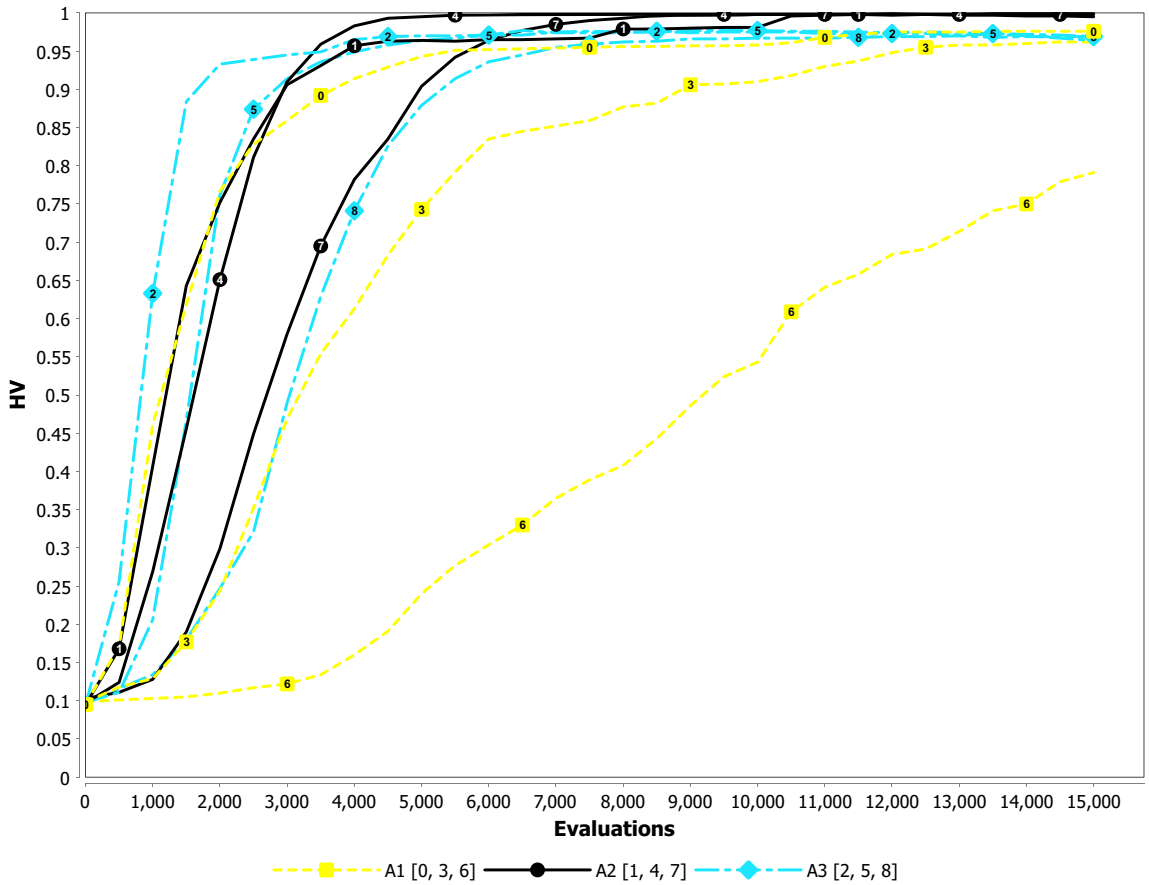

Fig. 1 Runtime behaviour based on the average Hypervolume value after a certain number of evaluations. μ: {0, 1, 2} = 50; {3, 4, 5} = 200; {6, 7, 8} = 500

Table 7 reveals that A4 requires the least number of evaluations to find a solution of satisfying fitness. The number of evaluations grows with increasing population size. In contrast, the number of evaluations required by A6 decreases with increasing population size. This leads to A6 being the best algorithm in the case of the largest population size. A5 and A7 are only successful in at most 50% of all repetitions. For A5, no clear trend in the number of evaluations could be identified. The standard deviation for A5 and A7 remains nearly constant. A4 shows only a small standard deviation which means that it is consistently able to find solutions of satisfying fitness within a small number of evaluations. It is an interesting question, whether A7 could find a solution within an even lower number of evaluations when the population size is increased further. While A5 and A7 are only successful in a comparably small number of repetitions, A4 and A6 appear promising with a preference of A4, nNEAT, due to its singular success rate of 100%.

The mean best fitness in Table 8 underlines this finding: A4 achieves the best mean fitness values for all objectives and population sizes. These are also significantly better than the fitness values of A5–A7. The second best fitness values were reached by A7 in f_1 and f_2, also with significant difference the values of A5 and A6. In particular, the fitness values for f_1, i.e. the time in balance, are important here. When the fitness for f_1 is inferior, the values of f_2 and f_3 are only of small practical relevance. Thereby, a certain threshold value for f_1 could also be defined as constraint. The mean best fitness values for f_1 for A5 and A6 for each population size are in the range of 0.698–0.967. Thereby, the best controllers found by A5 and A6 are not able to balance the cart for more than 30.2% of the experiment's duration. This is very likely caused by the default behaviour of NEAT to discard large parts of the population every epoch. Thereby, a well-performing solution might be found during the evolutionary process, but this in turn may be discarded before the evolutionary process stops. Our nNEAT (A4) follows a steady-state approach and thereby only discards the worst performing solutions each epoch. Furthermore, mNEAT (A7) does rely on an external archive that stores the best performing solutions found so far. The archive represents the population of mNEAT in the experiment evaluation. This explains why the best

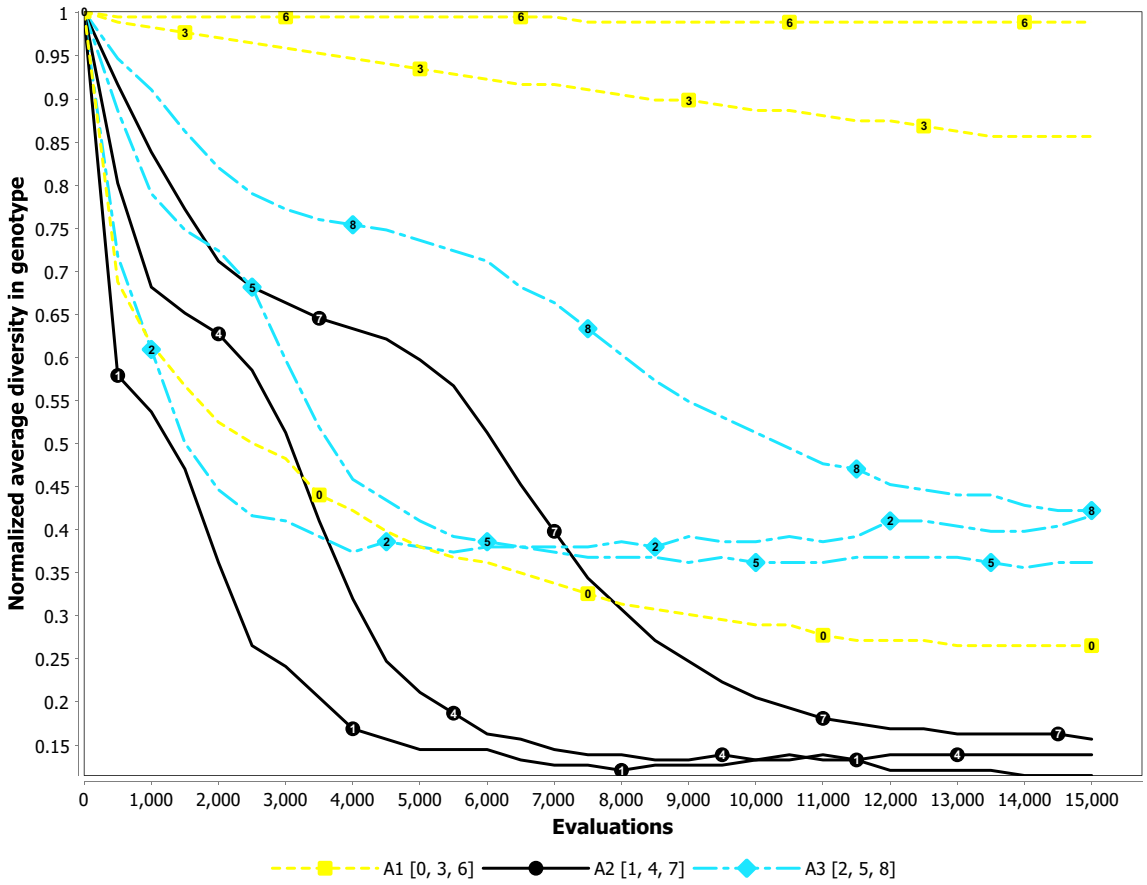

Fig. 2 Average diversity (Genotype) of all samples at the corresponding number of evaluations each. All values are in range $|0.019, 0.166| = 0.147$. μ: {0, 1, 2} = 50; {3, 4, 5} = 200; {6, 7, 8} = 500

fitness values attained by A7 are much better than the ones of the other "pure" NEAT-based algorithms A5 and A6. All in all, the best values are obtained by nNEAT and mNEAT for all population sizes, while with respect to the success rate or runtime behaviour, NEAT-PS and NEAT-MODS are also able to find solutions of promising fitness. The runtime behaviour is shown in Fig. 4. It is discernible that A4 and A7 initially converge faster for a smaller population size but achieve the best Hypervolume level for a medium population size of $\mu = 200$. The Hypervolume level achieved by A4 lies between 0.93 and 0.96, and A7 could reach 0.87–0.92. On the other hand, A6 attained the best Hypervolume level, 0.83, for a small population $\mu = 50$ and the worst, 0.76, for $\mu = 500$. Finally, A5 is not able to make considerable progress when $\mu \leq 200$. For a large population size of $\mu = 500$, A5 finally reaches a Hypervolume level of 0.5. It is an interesting question whether NEAT-PS will be able to converge to similar Hypervolume levels as A4, A6, and A7 when the population size is increased further. Summarizing the runtime behaviour shows that A4 reaches a higher Hypervolume level than A7, A7 a higher level than A6 and, finally, A6 a higher level than A5. This shows clear performance benefits of all algorithms over NEAT-PS.

The comparison of the genotype diversity in Fig. 5 reveals that all algorithms start at a similar initial level which is followed by two contrasting trends: Either the diversity is reduced during evolution or it increases. A diversity decrease is observed for A4 and A5. The only exception is the small population size $\mu = 50$ for A5, where the diversity is increased to 200%. The diversity increases for A6 and A7. While A4 and A5 stagnate at a similar level of diversity between 50 and 70%, A7 achieves a higher diversity (220–270%) than A6 (180–200%). The difference between the first and second group might be explained by the nature of NEAT-MODS (A6) and mNEAT (A7) to foster genotype diversity by speciation. Furthermore, NEAT-PS seems not to be able make any reasonable or directed progress and thereby achieves those dissimilar diversity levels. On the other hand, nNEAT does not focus on genotype but on behavioural diversity. This makes the loss of genotype diversity comprehensible in that case.

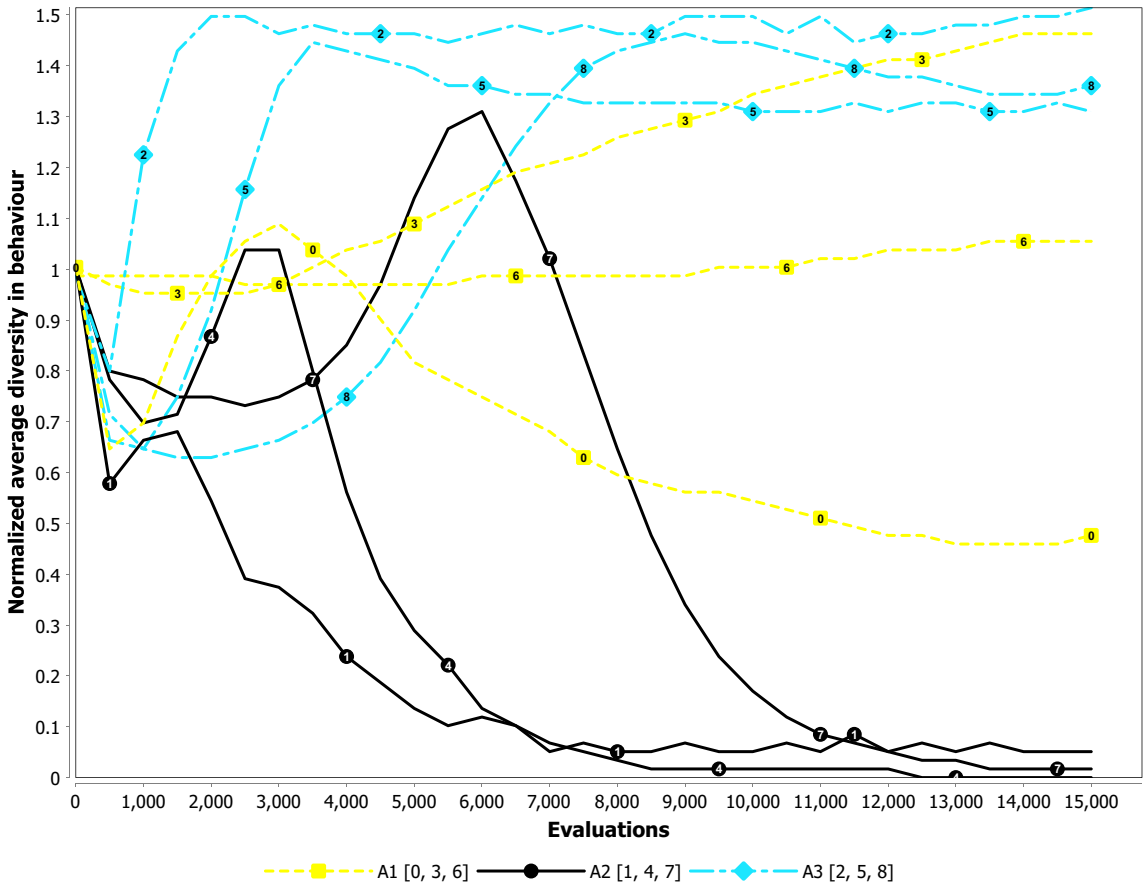

Fig. 3 Average diversity (Behaviour) of all samples at the corresponding number of evaluations each. All values are in range $|0.0, 0.089| = 0.089$. μ: {0, 1, 2} = 50; {3, 4, 5} = 200; {6, 7, 8} = 500

Table 6 Success rates of the corresponding algorithms

μ	A4 (%)	A5 (%)	A6 (%)	A7 (%)
SR				
50	**100**	*14*	74	26
200	**100**	*12*	60	50
500	**100**	*36*	66	50

A run of an algorithm is said to be successful if it found a solution of (at least) satisfying fitness within 15,000 evaluations

Table 7 Number of evaluations necessary to find a solution of (at least) satisfying fitness

Evaluations				
μ	A4	A5	A6	A7
50	**1961**	*7770*	6850	4192
	± **1152**	± *3239*	± 2837	± 3389
200	**3027**	*6038*	5607	5572
	± **958**	± *3574*	± 3825	± 3214
500	6064	7568	**4015**	*7730*
	± 1687	± 3341	± **2138**	± *3059*

Note that only successful runs, i.e. runs that could find such a solution within 15,000 evaluations, are regarded

The situation for the behavioural diversity draws a different image, see Fig. 6. All algorithms start at a similar level and first enter a phase of diversity reduction. While A5 ends in diversity stagnation between 60 and 70%,[2] A4 and A7 enter a phase of diversity enlargement. The smaller the population size is, the earlier in the evolutionary process this phase starts. While A4 and A7 both increase the diversity, their behaviour differs over the course of the evolution. A4's populations' diversity stagnates at levels between 145 and 165%. The diversity of A7's populations seems to increase continuing over the 15,000 evaluations and might grow further with more evaluations given. At the end of the evolutionary process, it achieves diversity levels between 120 and 145%. For both, A4 and A7, the diversity attains a higher level when a smaller population is applied. For A6, the impact of the initial reduction phase is

[2] Except for $\mu = 50$, then the diversity stagnates at the initial value.

Table 8 Comparison of the average fitness of the best solutions found within 15,000 evaluations

μ	ID	f_1	f_2	f_3
Mean best fitness				
50	A4	**0.0**	**0.048**	**0.003**
		± 0.0	**± 0.177**	**± 0.004**
		A5, A6, A7	**A5, A6, A7**	**A5, A6, A7**
	A5	*0.959*	0.33	0.011
		± 0.177	± 0.394	± 0.013
		A4, A7	A4, A6, A7	A4, A6, A7
	A6	0.794	*0.564*	0.067
		± 0.402	*± 0.486*	± 0.151
		A4, A7	*A4, A5, A7*	A4, A5, A7
	A7	0.047	0.231	*0.076*
		± 0.151	± 0.229	*± 0.093*
		A4, A5, A6	A4, A5, A6	*A4, A5, A6*
200	A4	**0.0**	**0.003**	**0.003**
		± 0.0	**± 0.003**	**± 0.002**
		A5, A6, A7	**A5, A6, A7**	**A5, A6, A7**
	A5	*0.967*	0.208	0.032
		± 0.15	± 0.325	± 0.055
		A4, A6, A7	A4	A4, A6, A7
	A6	0.698	*0.571*	*0.121*
		± 0.461	*± 0.493*	*± 0.21*
		A4, A5, A7	*A4*	*A4, A5, A7*
	A7	0.025	0.155	0.064
		± 0.084	± 0.234	± 0.104
		A4, A5, A6	A4	A4, A5, A6
500	A4	**0.0**	**0.007**	**0.004**
		± 0.0	**± 0.008**	**± 0.003**
		A5, A6, A7	**A5, A6, A7**	**A5, A6, A7**
	A5	*0.893*	0.242	0.025
		± 0.3	± 0.375	± 0.055
		A4, A6, A7	A4, A6, A7	A4, A6, A7
	A6	0.835	*0.638*	0.04
		± 0.359	*± 0.482*	± 0.091
		A4, A5, A7	*A4, A5, A7*	A4, A5, A7
	A7	0.045	0.142	*0.058*
		± 0.113	± 0.274	*± 0.101*
		A4, A5, A6	A4, A5, A6	*A4, A5, A6*

The third line for each algorithm indicates the algorithms to which statistically significant differences exist. Note that large values (compared to mean) in standard deviation are caused by a very small number of outliers

comparably small. On the other hand, it is not capable to increase its populations' diversity at all, except for $\mu = 50$ where it attains a level of 125%. The fact that nNEAT (A4) and mNEAT (A7) achieve higher diversity levels than the other algorithms might be caused by the quality measure QD that promotes quality as well as behavioural diversity. Furthermore NEAT-MODS (A6) also applies the Crowding Distance as secondary sorting criterion and thereby achieves a higher diversity than A5 does. Note that due to the diversity value of the population is being averaged over the population size, a large population is likely to contain more similar individuals on the whole than a small population does.[3]

3.3 Summary of the multi-objective double pole balancing experiments

This section compared different quality measures for nNEAT (Sect. 3.1) and the nNEAT and mNEAT to NEAT-PS and NEAT-MODS (Sect. 3.2). The obtained results show that the QD measure is very promising for driving the evolutionary process not only to find well-performing but also diverse solutions. Furthermore, nNEAT as well as mNEAT are able to outperform NEAT-PS and NEAT-MODS with respect to the given problem.

Note that all investigated algorithms used the same parameter configuration in the experiment. Future work will investigate the influence of additional effort being put into parameter adjustment to improve the performance of our algorithms even further. The population size also plays an important role. In particular, NEAT-PS seems to depend strongly on an adequately large population. One major problem occurring with NEAT-PS might be its difference to SPEA2: While SPEA2 relies on Pareto strength in combination with a diversity measure [11], NEAT-PS does only apply the Pareto strength approach without respect to diversity [10]. Future research will also investigate approaches for dynamically adjusting the population size. An example is represented by IPOP-CMA-ES [29] which increases its population size with every restart. Also using a large maximum number of evaluations might be beneficial as indicated in this section. Of course, the superiority of our algorithms over other multi-objective variants of NEAT does not allow to claim a general preference of our algorithms over the others. Therefore, further experiments, for example, the multi-objective double pole balancing without velocities, and experiments with more than three objectives will be conducted in future research.

[3] All ANNs initially have the same topology, the only difference are the weights of the links. As all weights are set randomly within range from `-Weight Mutation Rate` to `Weight Mutation Rate`, a large population will contain more similar solutions than a small does. The diversity of a small population naturally attains a higher level than a large population in many cases. Note that this also depends on the prevalent selection pressure and other factors, e.g. the variation operators.

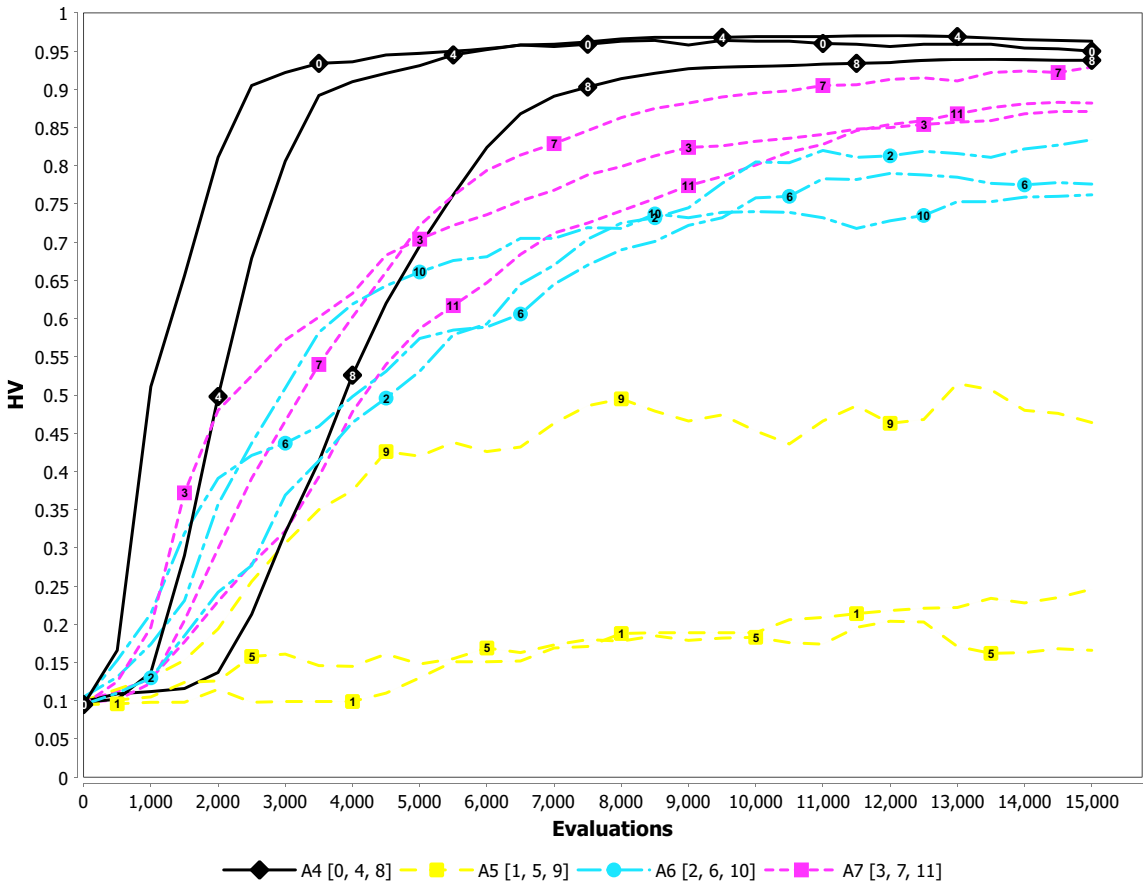

Fig. 4 Runtime behaviour based on the average Hypervolume value after a certain number of evaluations. μ: {0, 1, 2, 3} = 50; {4, 5, 6, 7} = 200; {8, 9, 10, 11} = 500

4 The fighting game AI competition

As stated before, games and digital games have experienced considerable research interest in the AI community. Aside from the quest for challenging non-player characters, digital games are an interesting testbed for AI algorithms. In order to assess and compare the performance of different techniques, several platforms and competitions have been introduced. Lu et al. [30] give an overview over different competitions that have been carried out back in 2013. A well-known example is based on TORCS (The Open Racing Car Simulator [31]) which aimed for developing AI-controlled drivers. However, not only racing car simulations have been considered. The earliest publications concerning AI and games can be found in the area of board games [1, p. 119]. Further examples include card games, arcade games as Pac-Man or the often used variant Ms Pac-Man, or strategy games as StarCraft, see the overview in [1, pp. 119–145]. Lu et al. [30] suggest another genre for comparing AI in competitions: fighting games, which have attracted research interest for several years, see e.g. [32].

Similar to well-known commercial games as Tekken[4] and Street Fighter[5], they introduced a novel Java-based fighting game framework, specialized on developing agents based on artificial intelligence. The FightingICE platform is a 2D fighting game simulation which provides the framework used in the **F**igh**T**ing **G**ame AI Competition (FTGAIC). It is developed by the Intelligent Computer Entertainment Lab. (ICE Lab.) at the Ritsumeikan University, Japan. The program is implemented in Java but also offers a wrapped Python platform. The framework allows researchers and programmers to develop their own AI players for the FTGAIC [33].

There have been a lot of interesting contributions throughout the annually competitions based on different approaches. These include for example hierarchical planners [34] or dynamic scripting [2]. A hierarchical task network (HTN) groups high-level tasks into subordinate compound-tasks and primitive tasks. Every task can depend on multiple pre-conditions. Selecting a path

[4] http://www.tekken.com, last accessed: 2019-09-12.

[5] https://streetfighter.com, last accessed: 2019-09-12.

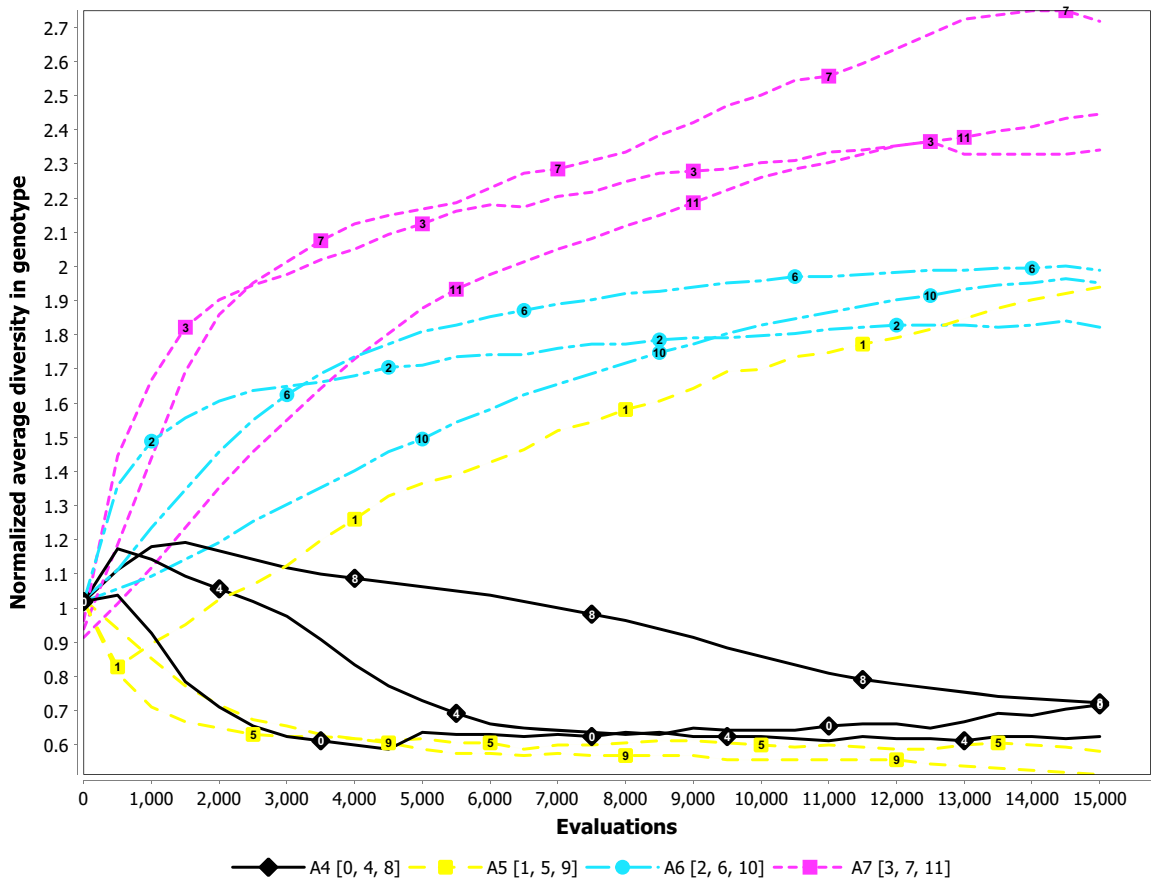

Fig. 5 Average diversity (genotype) of all samples at the corresponding number of evaluations each. All values are in range $|0.083, 0.445| = 0.362$. μ: {0, 1, 2, 3} = 50; {4, 5, 6, 7} = 200; {8, 9, 10, 11} = 500

through the network thus may happen from left to right (priority-based) or in any other way, e.g. using a heuristic [34]. Dynamic scripting combines the strengths of scripting with the capability of regarding even initially unforeseen situations. Therefore, a script is composed dynamically during runtime based on a set of predefined rules [2]. Note that hierarchical planners as well as dynamic scripting and other rule-based approaches require expert knowledge about the problem domain [2, 34].

On the other hand, approaches exist that do not require any domain-specific knowledge. Among these are genetic programming [35], neuroevolution [36], and other evolutionary approaches, and Monte Carlo tree search [3]. The first two are based on the biological process of evolution and aim at iteratively creating solutions of higher performance by mutating and combining existing ones. They mainly differ in the representation of their solutions: While genetic programming typically encodes candidate solutions as binary trees, where nodes are functions or logical operators and leaves represent the data being processed [35], neuroevolution evolves artificial neural networks as described in this paper. Note that Martínez-Arellano et al. [35] replaced the binary tree by a sequential encoding of possible actions. On the other hand, Monte Carlo tree search iteratively builds a search tree, where every path from the root to a leave is a possible sequence of actions. These are evaluated and new leaves are added to the tree until a predefined budget is reached. The most promising path, i.e. the one with the highest reward is selected finally [37].

Table 9 provides a short overview on approaches used for fighting games since 2014. The FooAI, second place of the official 2017 competition, is based on Monte Carlo tree search (MCTS) [37], for example, whereas the MizunoAI [38] predicts the opponent's next action using the k-nearest neighbour algorithm [39] and tries to find a useful counteraction. Table 10 gives a brief overview over some AI fighters of the last competitions.

In this paper, a novel AI player for the FTGAIC based on neuroevolution is introduced. Furthermore, the question, whether using multiple conflicting fitness functions and employing multi-objective neuroevolution results in agents that can cope with different opponents, is investigated.

In the case of success, our results may show that it is possible to derive general well-performing fighting strategies. This may deliver an answer on the main research questions the ICE Laboratory defined:

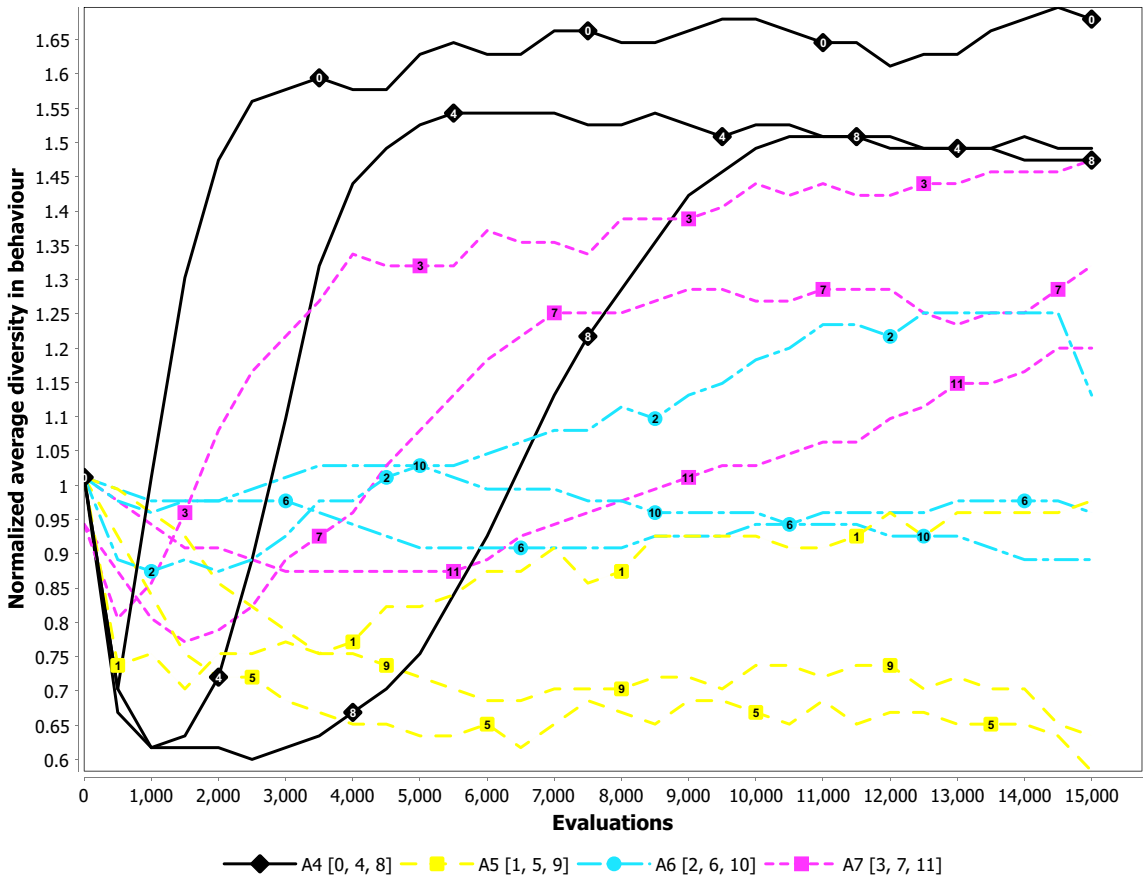

Fig. 6 Average diversity (behaviour) of all samples at the corresponding number of evaluations each. All values are in range $|0.034, 0.099| = 0.065$. μ: {0, 1, 2, 3} = 50; {4, 5, 6, 7} = 200; {8, 9, 10, 11} = 500

- Is it possible to realize general fighting game AI? In other words an AI that is strong against any opponent, human and artificial, in any play mode.
- If the first question can be answered with yes, how to realize such an AI?

As stated in the introduction, several researchers have considered ANNs as controllers in fighting games, for example [5, 32, 36, 40, 41]. The approaches show that neural networks may operate as well-performing controllers. The approaches range from traditional neural networks over neuroevolution to deep learning. Yoon and Kim [42], for example, used deep Q-Networks to evolve an AI controller for the FTGAIC, based on visual input only. A drawback is that all are using only a scalar fitness function which results in only one objective that can be optimized. While multiple fitness functions can be mapped into a single one using scalarization, where each fitness function is assigned a weight determining its importance; this has the disadvantage that the user has to assign the weights a priori without further knowledge about the range of solutions existing [20, p. 196]. To avoid this and other disadvantages, multiple fitness functions can be optimized concurrently. Multi-objective optimization for digital games has been considered before. For example, Schmitt and Köstler [43] apply a multi-objective genetic algorithm for finding optimal parameter configurations for opposing players' units in the real-time strategy game StarCraft II. However, we are unaware of approaches for neuroevolution and fighting games.

This paper considers an ANN-controlled agent for the FTGAIC. The ANN is evolved using the novel multi-objective neuroevolutionary algorithms previously described taking several conflicting fitness functions into account. Neuroevolution necessitates the simulation of a large number of fights, i.e. games. Every game in the case of the FTGAIC consists of three rounds each with a duration of 60 seconds. For evolving ANNs efficiently, a considerable reduction in the simulation time is required. For this reason, several changes to the source code[6] of the FTGAIC have become necessary.

[6] The original source code is available publicly on GitHub: https://github.com/TeamFightingICE/FightingICE, last accessed: 2019-09-12.

Table 9 Selected approaches for fighting games

Authors	Year	Methods	Platform	References
Park and Kim	2014	Multi-armed bandit	FightingICE	[44]
Yamamoto et al.	2014	k-nearest neighbour	FightingICE	[38]
Asayama et al.	2015	Linear extrapolation, k-nearest neighbour	FightingICE	[45]
Majchrzak et al.	2015	Dynamic scripting reinforcement learning	FightingICE	[2]
Mendoça et al.	2015	Deep reinforcement learning neural networks	Boxer	[5]
Sato et al.	2015	Switching, rule based	FightingICE	[4]
Zuin and Macedo	2015	Hidden Markov models	Own development	[46]
Kristo and Maulidevi	2016	Neuroevolution	FightingICE	[36]
Zuin et al.	2016	Evolutionary algorithms	Own development	[47]
Demediuk et al.	2017	Monte Carlo tree search	FightingICE	[48]
Kim and Kim	2017	Monte Carlo tree search	FightingICE	[49]
Neufeld et al.	2017	Hierarchical task network	FightingICE	[34]
Nguyen et al.	2017	Deep convolutional neural networks	FightingICE	[50]
Yoon and Kim	2017	Deep Q learning	FightingICE	[42]
Martínez et al.	2017	Genetic programming	M.U.G.E.N.	[35]
Pinto and Coutinho	2018	Hierarchical reinforcement learning	Monte Carlo tree search FightingICE	[3]

Table 10 Overview over the opponents selected

Name	References	Remarks
FooAI	–	Optimized version of MctsAi
JayBot_GM	[51]	Combination of genetic algorithm and MCTS for character ZEN
JerryMizunoAI	[53]	Improved version of MizunoAI using fuzzy control to avoid MizunoAI's "cold start problem", when no data for prediction are available
KotlinTestagent	–	Optimized version of MCTS-based GigaThunder with respect to performance and behaviour
MctsAi	–	Sample AI of FTGAIC, based on MCTS
Thunder	–	New version of GigaThunder, improvements in agent behaviour

The column *References* lists literature available about the corresponding fighter

The game is limited to 60 frames per second (fps). In every frame both opponents are required to return an input to the simulation. This means that each controller has a timeslot of 1 s/60 $\approx$ 16.67 ms to perform necessary computations. Because each controller runs in a separate thread, it is necessary that both threads and the main thread of the FTGAIC are synchronized each frame. The synchronization is handled in the AI controller's class. As a consequence of multi-threading, the FTGAIC may not run with 60 fps on older, respectively, slower hardware.

For evolving ANNs as controllers for AI players in the FTGAIC, it is necessary to remove unused features from the implementation. Among these is the 60 fps limitation of the game, all classes belonging to visual or acoustical effects and all logging functions. By removing graphics and sound from the game, the storage size of the game resources could be reduced which significantly increases the loading speed. The result of operating without these features is a comparatively fast simulating version of the game. The drawback is that due to the synchronization of multiple controller threads, the controllers' commands are only executed in very few frames because the simulation proceeds without waiting for the threads.

To solve this problem, the multi-threading component has been replace by a single thread in which all computations are performed. The advantage of a single-threaded simulation is that it is independent of the executing computer's hardware. Additionally, synchronization between multiple threads is not necessary any more. The drawback of this approach is that the simulation only uses a single CPU core for computation. This drawback can be

compensated in turn by running multiple simulations concurrently on a single computer.

Another drawback of our lightweight implementation of FTGAIC is that it cannot be used to evolve visual-based AI, because all methods and dependencies handling visual data have been removed.

5 ANNBot: an ANN-controlled FTGAIC agent

This section provides the details of applying our neuroevolutionary approaches to derive a controller for the FTGAIC. To this end, several design decisions are required which range from the information made available to the network over the outputs and actuators of the resulting ANNBot to the fitness functions used during the training. Finally, a controller evolved by our multi-objective neuroevolutionary algorithms is evaluated against several other AI fighters. First of all, an interface between the neuroevolution and the FightingICE platform needed to be established.

5.1 The ANNBot

This section describes the new AI for the FTGAIC, called ANNBot. The ANNBot is controlled by an artificial neural network which can either be set by reference to a Java object or loaded from an XML-file. The first variant is used for the training and evaluation of the ANNs, while the latter can be applied for participating in competitions using a full version of FTGAIC.

It is worth to mention that our ANNBot is much less time-consuming than MCTS-based AI opponents, e.g. FooAI and MctsAi. Due to the fact that MCTS needs a large number of iterations to find a successful strategy at each frame, while the neural network controller of ANNBot needs only a single iteration, our ANNBot is (1) computationally efficient and (2) its quality is independent from the hardware running on. Of course, it must be trained beforehand. Furthermore, it can be used as a reference to develop new AI players for the Fighting Game AI Competition.

5.1.1 The neural network: input layer

The ANNBot extracts different information about the game state every frame. All necessary information is normalized to values in the interval [0, 1], respectively, set to -1 if not existing.

After conducting preliminary experiments to assess the effects of input combinations, the following seven input values have been selected:

Let $w = 960$ be the width of the arena and $h = 640$ the height of the arena. First, the ANNBot is provided its current horizontal location x_p which can take a value in range $[0, w-1]$. Note that a character's location in x- and y-direction is always the centre of its hitbox[7] throughout this paper. Concerning the vertical location, the ANNBot is only provided an information whether the character is currently in air (1) or standing or crouching on the ground (0). The current location determines which attacks are available. Additionally, the ratio of available EP of the character and maximum EP necessary for starting the most EP consuming attack is given to the ANNBot. The ANNBot further provided the current distance in x- and y-direction to the opponent. As the FTGAIC always converts all values and inputs of each character p as if it would stand left of the opponent o and looking towards it, the x-distance can be computed as

$$d_x(p,o) = \frac{|x_p - x_o|}{w}.$$

In contrast, y-values are not converted by the simulation. Therefore, the ANNBot has to distinguish between three scenarios: equal y-location of both fighters ($d_y = 0.5$), the player is above the opponent ($d_y < 0.5$) and the player is below the opponent ($d_y > 0.5$)

$$d_y(p,o) = \frac{1}{2}\left(\frac{y_p - y_o}{h} + 1\right).$$

The last two input values are the distance in x- and y-direction to the nearest dangerous projectile. As projectiles fired by the own character cause no damage to itself, only the opponent's projectiles are dangerous. These values are determined in the same way as the distance to the opponent. If no dangerous projectile exists, both values are set to -1. This sums up to seven input values each taking a value between [0, 1] or -1.

The vector of input values does not take any velocities into account. Furthermore the controller is not aware of what action is currently performed by its character or the one of the opponent. When the opponent attacks, it might be useful to provide the ANN the location or distance to the hitbox of the attack. Additionally, predicting some information like the expected position of the next attack's hitbox or the location of its own or the opponent's character in the next frame(s). All these and further information might be taken into account in a future version of the ANNBot. For sure, these can also be prepared and combined in a

[7] Each character and each attack have a certain rectangle that describes their location, called hitbox. In FTGAIC, this is an abstracted version of collision detection: If the hitboxes of a character and an attack collide or overlap, the attack was successful and the character gets hit by that attack.

smart way in order to let the ANNBot learn the meaning of the input values quickly.

5.1.2 Output layer

In the FTGAIC, there are seven keys a player can hit each frame: attacks *A*, *B*, *C* and movement directions *up*, *left*, *down*, and *right*. Most combinations of keys cause simple actions as *movement* or an attack. Additionally, there are combinations of keys that are performed over a certain number of frames, for example *1. Down, 2. Down + Forward, 3. Forward + Attack-Button C* causes a fireball to be thrown. There are only a few valid combinations of keys over one or more frames. Namely there are 56 different actions a player can perform. Only 40 of the 56 actions [51] can be triggered by the player, whereas the remaining actions are performed as a consequence of a previous action, for example, recovering from a suffered hit or a performed attack. The 40 actively triggered actions consist of movement, guarding, close, and distant combat. As seven keys allow 128 different permutations while only 40 of these are linked to a valid action, the ANN's output cannot be simply seven values determining which keys are hit in the current frame. This would lead to a large part of the output values causing invalid key combinations. Zuin et al. [47] have investigated evolutionary algorithms to find combos and unexpected key sequences. Although their results are promising and could easily be adopted to our needs, for example by defining an additional fitness function counting the proportion of valid output values, we decided to employ another procedure which always produces valid combinations.

The proposed approach aims at keeping the controller network as small and simple as possible by reducing the number of output values to a minimum. Namely there are three output values o_1 to o_3 which always will contain a value between zero and one each. Two types of actions a character can perform need to be distinguished: Movement and attacks. While movement comprises all movement actions, including jumping and crouching, and guarding positions, attacks can be in close or far distance. The output value o_3 determines whether the agent will move or perform an attack in the next frame(s): If $o_3 \leq 0.5$, the agent will move; otherwise, it will try to start an attack.

The output values o_1 and o_2 determine the *x*- and *y*-position the next action of the agent aims to. Here, movement (Fig. 7) and performing an attack (Fig. 8) need to be distinguished.

The range of length one (for o_1 and o_2) is split into several intervals of different length, as shown in Fig. 7. One can image this selection approach like the analogue stick of a game controller. The output values (0.5, 0.5) indicate the initial position of the analogue stick, which is denoted by *A* here. Intuitively, this should result the agent in doing nothing; however, in order to avoid receiving unnecessary damage, the agent should always "do something". Therefore, the agent would take a standing guarding position in that case. The example output values (0.8, 0.2) which are marked by *B* here, i.e. a movement of the analogue stick into the upper right direction, will make the ANNBot perform a jump forwards. Note that all actions are selected based on the $\leq$ operator: For example (0.5, 0.25) will select the *Jump* action and (0.75, 0.75) the *Guard (Standing)* action. If the character is currently not standing on the ground, it does not have any influence on its movement until it lands on touches the ground again. In that case, the character will always perform the in-air guarding position. Furthermore, keep in mind that the pixels of the arena in FTGAIC are numbered according to the pixels of a monitor starting by (0, 0) in the top-left-corner and ending at $(w-1=959, h-1=639)$ in the bottom-right-corner, which explains our choice of the layout.

In the case $o_3 > 0.5$, an attack will be started in the next frame. The way how the next attack is selected is shown in Fig. 8. Note that the attacks available for execution depend on the character's current *y*-location: There is a different set of attacks when the player is standing or crouching on the ground as if it were jumping. Furthermore, close and far combat attacks are distinguished. If $o_3 \leq 0.75$, a close attack will be chosen; otherwise, if $o_3 > 0.75$ a far attack. This comprises four sets of different attacks, and Fig. 8 shows a single exemplary set containing five attacks.

For the selection, some initial preparations are carried out. Each attack contains information about the associated hitbox. The hitbox defines a rectangle relative from the fighter's position starting at $(x_{\min}, y_{\min})$ and ending at $(x_{\max}, y_{\max})$. First, all attacks are sorted into four groups, depending on the location where these can be started (ground, air) and the location of effect (close, far). Then, the smallest and largest *x*- and *y*-values in each group are determined, which allows to normalize all values to the interval [0, 1] as shown in Fig. 8. This in turn allows to create a map of the attacks' hitboxes as shown exemplary in Fig. 8. As empty and overlapping areas may occur, unlike Fig. 7, this map contains the challenge to handle different situations:

1. Exactly one hitbox matching the position is selected by output values o_1 and o_2. The attack the corresponding hitbox belongs to is chosen as next action. Example: Location *A* in Fig. 8.
2. At least two hitboxes covering the position selected by the two output values. In that case, the attack with the higher expected damage is carried out. Example: Location *B* in Fig. 8.

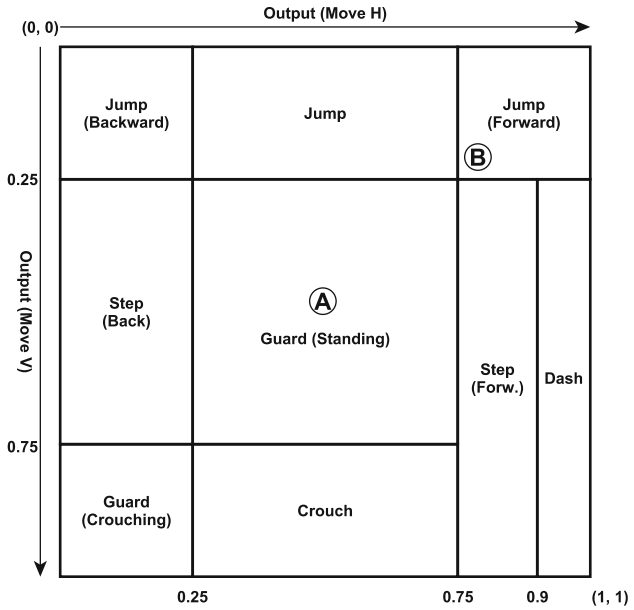

Fig. 7 Map for selecting the next movement or guarding action. Note that *Move H* equals o_1 and *Move V* equals o_2

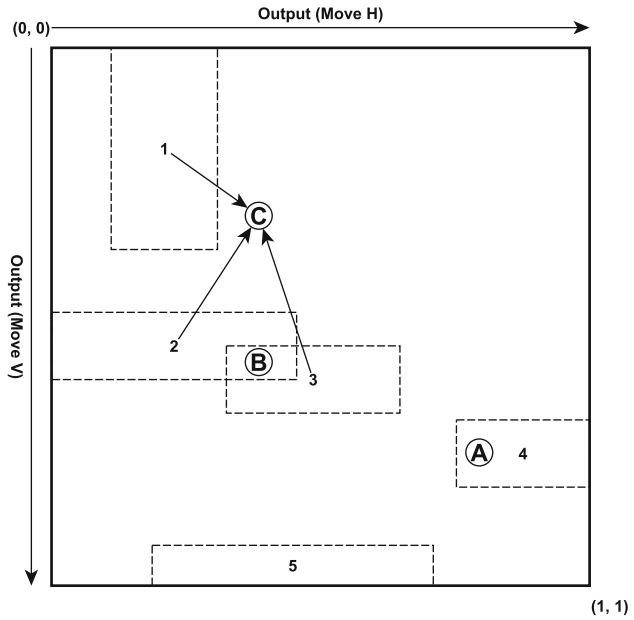

Fig. 8 An exemplary map for selecting the next attack. The rectangles labelled from 1 to 5 are the corresponding hitboxes. Note that there exist four different maps depending on the character's location and whether a close or far combat attack is to be performed. Each map is filtered by the EP the character possesses and the EP the attacks require to be started. This ensures that the ANNBot does only select and try to start an attack that it really can perform. Note that *Move H* equals o_1 and *Move V* equals o_2

3. There is no hitbox at the position that has been selected. As this situation may arise quite often, depending on the number of attacks available and their corresponding hitboxes, the agent has to select an adequate alternative action. Here, the shortest (squared) euclidean distances from the centre of each hitbox in the map to the selected location are determined. The hitbox with the smallest distance to the given location will get its corresponding attack performed. Example: Location *C* in Fig. 8. The figure only depicts the distances from attacks 1, 2, and 3 hitboxes to location *C*, as the other hitboxes obviously do not come into question. Finally, attack *1* will be selected.

Before the corresponding attack for o_1 and o_2 is selected, the map of attacks possible is filtered for attacks that the character can start due to energy constraints: $\mathrm{EP}_{\mathrm{attack}} \leq \mathrm{EP}_{\mathrm{character}}$. Three of the groups contain attacks that do not require any EP to be started, only the group *(Ground, Far)* consists of attacks that all require a certain amount e of EP. In the case that $\mathrm{EP}_{\mathrm{attack}} < e$ it may happen that no attack is found in that group. Then, the ANNBot will fall back to perform a movement action.

The selected action, movement or attack, is executed using the `CommandCenter`-class over at least one frame in the simulation. If a multi-frame-input has been selected in a previous frame, an input is already defined in the current frame. In this case, the player is not allowed to perform another action in the current frame, and the ANN is not updated to save computation time.

5.2 Training

The training of the ANNBot can be done in two different ways: Either against an already existing opponent like *FooAI* or *MctsAi*, or against other ANN-controlled opponents. Because FooAI and MctsAi are both based on MCTS, see e.g. [1, p. 45f], they repeat as many MCTS iterations as possible in the 16.67 ms timeslot per frame. Thus, the evolution would progress very slow (in nearly real time) fighting against those AI opponents. Additionally fighting against a single opponent could lead the ANNs to being overfitted to exactly the opponent they have been trained with.

To avoid overfitting and to perform as many evaluations as possible in very little time, a competitive coevolutionary approach may be useful, see e.g. [20, p. 223f] for more information. This first case study follows the first approach using a small number of evaluations. The coevolutionary approach will be investigated in future research.

Every solution is evaluated for five fights against FooAI and JayBot_GM each and not against any other ANNBot. Every solution is only evaluated once for ten fights, i.e. if it has just been created. The advantage of this procedure compared to the coevolutionary approach is that it generally requires less fights for evaluating a solution. To

evaluate the first generation 10 μ fights are necessary, in every following generation only the λ new solutions need to be evaluated, requiring 10λ fights. Furthermore, this approach focuses on beating the specified opponents FooAI and JayBot_GM and also opponents that behave similarly. As more than one reference opponent is applied for evaluation, this approach can also derive a generally good performing fighting game AI against different types of opponents. Every fight goes over three rounds. The fitness of each solution is averaged over its results in all fights. The fitness of newly created solutions is also determined as the average fitness over all fights.

5.3 Fitness functions

To evolve controllers of desired behaviour, several fitness functions are defined. These are promoting properties like win–loss rate, offensive and defensive behaviour. All fitness values are in range 0 to 1 and have to be minimized. The performance of an ANNBot a fighting against an opponent b can be assessed by:

1. The ratio of the rounds won by a in relation to the total number of rounds. This fitness function shall promote ANNs that win more rounds than their respective opponent. Note that by default $rnd_{\text{total}} = 3$, other settings are also possible. Note that if a lost the fight against b, this fitness function always returns 1. Thereby, it promotes solutions that are not only able to win many rounds but also enough rounds to win the corresponding fight
$$f_1(a) = \begin{cases} 1, & \text{if } rnd_{\text{won}}(a) < rnd_{\text{won}}(opponent(a)) \\ 1 - \dfrac{rnd_{\text{won}}(a)}{rnd_{\text{total}}}, & \text{otherwise} \end{cases}.$$
2. The total damage the opponent of a has received. As this value is to be maximized its value has to be subtracted from 1 in order to convert it to a minimization problem. The constant $dmg_{\max} = 1000$ denotes the maximum regarded damage per round. If the damage exceeds this limit, the damage rate is capped to 1. This fitness function fosters controller's that are offensively strong and able to cause much damage, not only by direct attacks but also indirect possibilities to cause damage to the opponent, for example by performing attack combos
$$f_2(a) = 1 - \min\left\{1, \frac{dmg(opponent(a))}{rnd_{\text{total}} \times dmg_{\max}}\right\}.$$
3. The total damage the ANNBot a has expected. Contrary to f_2 this value has to be minimized naturally. It promotes controllers that either are capable of avoiding getting in contact with the opponent, respectively, the effects of the opponent's attacks or is able to defend the opponent's attacks
$$f_3(a) = \min\left\{1, \frac{dmg(a)}{rnd_{\text{total}} \times dmg_{\max}}\right\}.$$
4. The total number of close combat attacks started by the ANNBot a. Throughout preliminary experiments with only the three previous fitness functions, the ANNBot almost always discovered the strategy to fire projectiles on the opponent permanently. To foster close combat attacks f_4 has been introduced, the plus one in the divisor ensures to avoid division by zero in case that no close combat attacks were started
$$f_4(a) = \frac{1}{attacks_{\text{close}}(a) + 1}.$$

The fitness functions described above promote different behaviours simultaneously: First, the generally desirable property having won more fights than lost (f_1) and causing more damage (f_2) than receiving (f_3) are strengthened. These properties lead to a controller that is either good in offensive or defensive behaviour or both. Promoting the ANNBot to use as many close combat attacks as possible in addition is an auxiliary fitness function in order to avoid the inflationary use of projectiles and to foster human-like behaviour. The fitness functions are conflicting, in so far they force the training of defensive and offensive behaviour simultaneously. Note that—for the final competition—f_1 is the most important fitness function, because our goal is to evolve ANN controllers that are capable of defeating existing opponents. The remaining fitness functions mainly support evolving promising behaviour.

5.4 Experimental results and discussion

As described in Sect. 5.2, ANNs have been trained against the well-performing opponents FooAI and JayBot_GM. A single run of the experiment using the algorithm nNEAT with the QD measure for sorting has been conducted. The population size was set to $\mu = 50$ individuals. The control parameters and the activation function remain unchanged.

The experiment was terminated after 5000 solutions have been evaluated. Note that the MCTS algorithm used in FooAI and JayBot_GM achieves better results with an increasing number of repetitions. The FooAI performs as many repetitions as possible during the 16.67 ms available per frame, our test system used 100–200 repetitions per frame on average. The performance of FooAI heavily depends on the performance of the computer executing the simulation. Many fighters are based on MCTS, while FooAI has some random components other fighters always result in the same behaviour when the number of MCTS

iterations remains the same. For avoiding some fighters behaving deterministically and making our results comparable, a random number of MCTS iterations within between 200 and 225 is done every frame. As the number of iterations is drawn randomly, on average, there will be 212 iterations.

As noise is present, although desired, measures have to be taken to reduce the influence of noise when evaluating and especially comparing the solutions among each other. Playing ten games over three rounds per solution equals resampling with ten samples. The fitness is the average value for each fitness function over the ten games played [52]. Finally, the population is sorted using the QD measure.

The best performing ANN with respect to f_1, i.e. the ratio of won and lost rounds, of the final external fitness evaluation was selected for further investigation. The ANN has been found after 2,100 evaluations and consists of seven input, three output and no hidden neurons. There are 23 links of which no one is recurrent. It is interesting to see that no hidden layer is required for a well-performing bot. This may hint at several phenomena. One possible reason may be that the opponents may be exploited by a learning approach which is able to detect non-obvious weak points. Further research is required, however. The selected individual was evaluated for 200 fights consisting of three rounds each against the opponents listed in Table 10.

The results are shown in Table 11 and the corresponding fitness values in Table 12.

Note that our experiments are subject to the following restrictions: The FTGAIC offers three different characters to play, each with strengths and weaknesses. All experiments were performed only with the character Zen (vs. Zen). Additionally, the ANNBot always played as player 1 starting in the left half of the fighting arena. This has only small to no influence on the character's fitness, because in most matches the players are moving and switching positions within the first few frames of the match.

Table 11 gives an overview over the fights that have been won (or lost) by the ANNBot against the six opponents. Remarkable is that the ANNBot using the evolved controller was capable of winning at least 51.6% of all fights (without ties being taken into account) that have been fought against four of the six opponents. Against FooAI, JerryMizunoAI and MctsAi, the ANNBot was able to win nearly all fights. The opponents KotlinTestAgent and Thunder could not be beaten by the ANNBot in more than 11, respectively, 1% of all fights. On the other hand, in preliminary experiments, the ANNBot was trained against Thunder and KotlinTestAgent, the evolved controllers were able to win a large number of fights against these opponents. They were able to discover the weaknesses of the corresponding opponents but were only successful against the opponents they have been trained with. As this is not a generally superior AI, we do not list these experiments' results here. There may be several causes for the observations made here. First of all, the opponents may not show a sufficient diverse behavioural range in order to provide the necessary information for a generalization. While we strongly suspect that this may be a significant contributing factor, a closer look at other potential contributing factors, e.g. the definition of objective functions is required.

Compared to the results shown in [40], where an ANN of fixed topology has been trained to fight against MctsAi and other AI opponents, our ANNBot performs much better: The ANNBot was able to win 99% of all fights against MctsAi, the ANN evolved in [40] is only successful in four of 30 rounds (with character Zen), which comprises a success rate of 13.3%. Note that MctsAi is the only mutual opponent that has been investigated in this paper and [40]. As our ANN is compared against the state-of-the-

Table 11 Number of fights that have been won, tied, or lost by the ANNBot using the corresponding controller

Opponent	Won	Tie	Lost	Success rate
FooAI	198	0	2	99.0
JayBotGM	95	16	89	47.5 (51.6)
JerryMizunoAI	199	0	1	99.5
KotlinTestAgent	22	0	178	11.0
MctsAi	192	0	8	96.0
Thunder	2	0	198	1.0

Additionally, the proportion of successful fights is provided, the number in brackets is the success rate without ties being taken into account

Table 12 Fitness values of the ANNBot against the different opponents

Opponent	f_1	f_2	f_3	f_4
FooAI	0.053	0.599	0.163	0.594
	±0.147	±0.039	±0.051	±0.411
JayBot_GM	0.612	0.833	0.182	0.481
	±0.384	±0.048	±0.079	±0.427
JerryMizunoAI	0.043	0.758	0.147	0.975
	±0.126	±0.028	±0.027	±0.124
KotlinTestAgent	0.925	0.917	0.431	0.633
	±0.215	±0.066	±0.155	±0.396
MctsAi	0.115	0.712	0.136	0.673
	±0.228	±0.061	±0.055	±0.399
Thunder	0.993	0.953	0.302	0.293
	±0.066	±0.041	±0.069	±0.387

art opponents of FTGAIC, it can be concluded that our ANNBot performs better than the bot evolved in [40].

The fitness values of our ANNBot against the different opponents are given in Table 12. By definition the value of f_1 is proportional to the number of won fights. The large standard deviation against FooAI and JerryMizunoAI is caused by a small number of outliers. The values for f_2 range from 0.6 to 0.95 which comprises that the opponents received between 400 and 50 damage points per round on average. The worst values in f_2 were achieved against Thunder and KotlinTestAgent. On the other hand, the values for f_3 reach from 0.14 to 0.43 which means that the ANNBot received between 140 and 430 damage points per round on average. The most damage points the ANNBot received against KotlinTestAgent and Thunder. The fitness function f_4 has been introduced in order to foster close combat attacks too. The values in Table 12 indicate that the selected controller does not apply close combat attacks in general—only one to three attacks per fight. All good performing controllers with respect to f_1 were not using close but far combat. Thus, the fitness function f_4 could not keep the ANNBot from using far combat attacks constantly. A solution could be not to divide close and far combat attacks in the selection procedure described in Sect. 5.1.2. We assume that the ANNBot then will also choose close combat attacks, and this in turn would likely require a larger number of evaluations to train the ANNBot.

To achieve this and enable to perform a large number of evaluations within very few time, future work will conduct two necessary steps: First ANN controllers will be trained to copy their opponents' behaviour, for example from previously recorded fights. Then, a set of sufficiently similarly behaving copies of existing AI opponents will be used as reference to train ANNs. This will probably enable us to evolve more complex structures and behaviours of our ANNBot.

We additionally analysed the ANNBot using the official FTGAIC simulation, and thereby found that the controller confirmed our expectation regarding f_4: The ANNBot tries to reach one of the corners of the fighting arena and then continuously fires projectiles in the opponent's direction while jumping in certain intervals. The projectiles are hindering the opponent from getting close to the ANNBot and thereby avoid the ANNBot from getting hit by close combat attacks. Furthermore by jumping, the ANNBot can avoid some of the opponent's projectiles.

The results show that the success of our example controller is based on (1) the controller strategy, which allows to fire a projectile nearly every frame or (2) the exploitation of a weakness of the FTGAIC simulation which allows to fire projectiles without further limitations. This leads us to the conclusion that the currently evolved ANNBot will be successful against many opponents on the one hand, but on the other hand do not behave "typically human". Mendonça et al. [5] investigated in how far ANNs are capable of mimicking human behaviour and being successful in the fighting game Boxer.[8] To this end, they utilized data of recorded battles between human players. This approach may also be used in future work to train ANNs, for example with an additional fitness function describing the "humaneness" of a controller.

6 Summary

This paper focused on multi-objective neuroevolutionary approaches which are seldom considered in research. It continued our work first begun in [13] by focusing on the selection procedure which plays an important role in evolutionary multi-objective optimization. To this end, novel sorting mechanisms for our multi-objective algorithms were presented: The QD measure that promotes finding solutions of high quality as well as maintaining diversity and the iterative $R2$ contribution introduced by Díaz-Manríquez et al. [14]. Furthermore, the paper introduced a novel procedure for computing the exact $R2$ contribution of all solutions in a set that is faster than the existing procedures by factor μ (population size).

A first series of experiments concerning a multi-objective double pole balancing problem has been conducted. Therefore, ANN controllers for the cart were evolved using our neuroevolutionary variants mNEAT and nNEAT and furthermore the well-performing NEAT-PS [10] and NEAT-MODS [12]. The results of the experiments have shown that our algorithms were able to outperform the existing multi-objective variants of NEAT not only with respect to quality but also population diversity. Furthermore, the nNEAT converges faster towards the Pareto front than mNEAT, NEAT-PS and NEAT-MODS. The performance of nNEAT will be investigated more closely in the future using further application cases in order to analyse whether it remains the best performing technique.

Since the results were promising, we then focused on the application of nNEAT in the area of fighting games. This game genre represents a challenging test case for AI-controlled characters. In addition, it constitutes a major part of commercial games. Also multi-objective formulations arise naturally and appear beneficial for deriving a well-performing fighting bot.

Therefore, the paper examined whether multi-objective neuroevolution is able to evolve artificial neural networks as controllers for AI players for fighting games focusing on

[8] https://gamejolt.com/games/boxer/29331/, last accessed: 2019-09-12.

the Fighting Game AI Competition. Although multi-objective optimization appears promising, it has not been explored before for fighting games.

To investigate the capabilities of nNEAT for fighting games, we first created a novel AI player called *ANNBot*. Here, we also had to define the inputs and outputs for the neural network, as well as the fitness functions that are meant to train a controller that performs well in both, attacking and defending, and thus, is capable of defeating other AI opponents. The training phase of the ANNBot required the development of a lightweight version of the Fighting Game AI Competition. This variant is usable as a kind of framework for AI player evolution for similar approaches.

Finally, we applied our nNEAT in order to evolve a controller for the ANNBot in the FTGAIC. The best performing controller with respect to the opponents FooAI and JayBot_GM was selected for further investigation. Here, we let fight the ANNBot with the corresponding controller for 200 fights against six different AI opponents (see Table 12). The results show: nNEAT is capable to evolve controllers that are able to beat most opponents considered in more than the half of all fights. In some cases, the success rate even exceeded 96%. Our ANNBot even outperforms previous AI players based on neural networks by at least the factor seven [40].

The Fighting Game AI Competition was started to shed more light on the research question whether general fighting game AI could be developed. The results of this paper show that neuroevolution in combination with multi-objective optimization may present one possible approach. While our ANNBot has only been tested against AI players and in one play mode, we are confident that it will perform well also against human players and in different play modes using different characters. This will be investigated in future work. Additionally, other controller strategies, e.g. arbitrary output of key combinations, whose validity determines the value of an additional fitness function [47] and input values for the neural networks should be investigated.

7 Conclusions and future work

To summarize the main results, this paper contributes to the research on multi-objective neuroevolution by introducing the novel QD measure for sorting solutions within our mNEAT and nNEAT, multi-objective variants of the well-known NEAT. The measure enables to drive the search towards both, high-quality and diverse solutions which is one of the main objectives in multi-objective optimization. Future work will investigate additional strategies for applying quality and diversity measures in order to gain maximum profit for the search.

Furthermore, a novel efficient way of determining the $R2$ contribution of solutions has been proposed which in turn improves the performance of the neuroevolutionary algorithm. A first experimental comparison between mNEAT, nNEAT and other multi-objective variants of NEAT has been conducted. While the first results are promising, further experiments will be carried out in future work to allow more general statements about the performance of mNEAT and nNEAT. Since deriving good settings of the control parameters is a difficult problem that depends on the task at hand, we are also currently working on parameter adjustment in order improve the usability of the algorithms.

Aside from the double pole balancing problem, this paper described a lightweight and efficient simulation environment for the Fighting Game AI Competition and an exemplary implementation of an ANN-based bot. The experimental results are promising. However, as in the case of the competing research, it can be seen only as a first step towards a general fighting game AI. Therefore, more complex experiments will be considered. This also includes to train ANN controllers to copy their opponents behaviour. This will allow achieving challenging competitors much faster. In addition, the optimization of the ANNBot's input and output strategy, supporting it becoming more powerful against other opponents, and aspects like the "humanness" of an AI fighter will be investigated.

Acknowledgements Open Access funding provided by Projekt DEAL.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

Ethical standard The focus of the paper lies on fighting games which constitute a major and well-accepted variant of commercial digital games. Since they are difficult for AI players they have attracted research interest in the artificial and computational game community. The particular game which the paper considers uses a highly stylized, comic-like graphic style which does not depict any injuries or blood. This article does not contain any studies involving human participants performed by any of the authors.

Appendix 1: An efficient computation of the exact *R*2 contribution

The $R2$ indicator [54] is based on a utility function $u : \mathbb{R}^K \rightarrow \mathbb{R}$ which assigns a utility value to each solution a in set A. The utility function is based on the standard weighted Tchebycheff utility function

$$u(z) = -\max_{i\in\{1,\ldots,K\}} \left\{\lambda_i \left|z_i^* - z_i\right|\right\}$$

for a point or solution z, a reference point z^* and a weight vector $\lambda = (\lambda_1, \ldots, \lambda_K) \in \Lambda$. Following [55], the point z^* should be an ideal point that cannot be dominated by any feasible solution. The set Λ (usually) comprises uniformly distributed weight vectors over the weight space [55]. Note that the expression *R2 value* is used to describe the $R2$ value of a whole set and *R2 contribution* for the $R2$ value of a single individual of a set, here. The $R2$ value is defined as

$$\begin{aligned} R2(A, \Lambda, z^*) &= -\frac{1}{|\Lambda|}\sum_{\lambda\in\Lambda}\max_{a\in A}\{u(a)\} \\ &= -\frac{1}{|\Lambda|}\sum_{\lambda\in\Lambda}\max_{a\in A}\left\{-\max_{k\in\{1,\ldots,K\}}\left\{\lambda_k\left|z_k^* - a_k\right|\right\}\right\}. \\ &= \frac{1}{|\Lambda|}\sum_{\lambda\in\Lambda}\min_{a\in A}\left\{\max_{k\in\{1,\ldots,K\}}\left\{\lambda_k\left|z_k^* - a_k\right|\right\}\right\} \end{aligned}$$

The $R2$ indicator returns the averaged sum of the minimal distances for all $a \in A$ in any dimension for each weight vector $\lambda \in \Lambda$. Thus, if a set A has a lower $R2$ value than another set B, its individuals are located closer to the reference point on average than B's individuals with respect to the selected weight vectors. The $R2$ contribution of a single individual $a \in A$ is determined as

$$R2(a, A, \Lambda, z^*) = R2(A\backslash\{a\}, \Lambda, z^*) - R2(A, \Lambda, z^*).$$

The $R2$ value is computed in time $O(K\mu|\Lambda|)$ [55], while the computation of the $R2$ contribution of all individuals in a set A with $|A| = \mu$ requires time $O(K\mu^2|\Lambda|)$. This makes the $R2$ indicator very efficient to compute, especially for large number of dimensions ($K > 3$) [56], compared to other quality indicators as the Hypervolume, which requires time $O\left(\mu \log \mu + \mu^{\frac{K}{2}+1}\right)$ [16] and thus grows exponentially with the number of dimensions.

The original computation of the $R2$ contribution works as follows: For each individual $i \in \mathcal{P}$ the $R2$ contribution of i is the difference between $\mathcal{P}$'s $R2$ value and the $R2$ value of $\mathcal{P}\backslash\{i\}$. Therefore, the $R2$ value of $\mathcal{P}$ is computed once in time $O(K\mu|\Lambda|)$ first. Next, the $R2$ value of each subset $\mathcal{P}\backslash\{i\}$ is computed $\forall i \in \mathcal{P}$ in time $O(\mu(K(\mu-1)|\Lambda|))$. Finally, the difference between both contributions is computed for every individual $i \in \mathcal{P}$. This sums up to time $O(K\mu|\Lambda| + \mu(K(\mu-1)|\Lambda|)) = O(K\mu^2|\Lambda|)$.

Shang et al. [57] propose a new $R2$ indicator R_2^{new} based on a new Tchebycheff function introduced by Zhang et al. [58]. The new $R2$ indicator is defined as follows:

$$R_2^{\text{new}}(A, \Lambda, z^*) = \frac{1}{|\Lambda|}\max_{a\in A}\left\{\min_{k\in\{1,\ldots,K\}}\left\{\frac{\left|z_k^* - a_j\right|}{\lambda_k}\right\}\right\}^m$$

The only difference to previously proposed $R2$ indicators is the exponential m which leads, properly set, to a linear relation with the true Hypervolume of a set A. Thereby, the R_2^{new} can be used to efficiently approximate the Hypervolume of a set A [57]. The runtime of the algorithm comprises $O(K\mu|\Lambda|)$, which leads to equal runtime as the conventional approach when the $R2$ (or approximated Hypervolume) contribution of all individuals in the set A has to be computed. Nevertheless, the R_2^{new} is an interesting alternative to the utilization of the Hypervolume indicator for sorting populations upon multiple fitness functions. Therefore, it should be investigated in future work.

As described in Sect. 2, Díaz-Manríquez et al. [14] introduced a novel procedure, which approximates the $R2$ contribution of all individuals $i \in \mathcal{P}$ in time $O(K\mu|\Lambda|)$. For each weight vector $\lambda \in \Lambda$, their approach first determines the individual $i_\lambda \in \mathcal{P}$ with the shortest weighted distance to the reference point z^*. The $R2$ contribution of each of the previously identified individuals $i_\lambda \in \mathcal{P}$ is then approximated with the weighted distance from i_λ to z^* with respect to λ. The weighted distance is defined later in this paper.

In the case that an individual $i \in \mathcal{P}$ exists which has the shortest weighted distance for more than one weight vector expressed by $\exists\lambda_1, \lambda_2 \in \Lambda : \lambda_1 \neq \lambda_2 \wedge i_{\lambda_1} = i_{\lambda_2}$, the $R2$ contribution of i is the sum of all weighted distances for all weight vectors.

Whereas the approach above is computationally efficient, a drawback is that it only delivers an approximation of each individual's $R2$ contribution, because the weighted distance of the second nearest individual for each weight vector is not taken in account. The resulting approximation influences the order of the individuals in $\mathcal{P}$ based on their $R2$ contribution.

In the following, the weighted distance and minimum weighted distance to reference point z^* are defined and a theorem about the efficient computation of the $R2$ contribution of a population's individuals is introduced. Finally, a proof of the theorem and the pseudocode of the corresponding algorithm are given.

Definition 1 (*Weighted distance to z^**) Let there be an individual i, a weight vector λ and a reference point z^*. Then, $d(i, \lambda, z^*)$ is defined as the weighted distance of the individual i to z^* weighted by λ

$$d(i, \lambda, z^*) = \max_{k \in \{1,\dots,K\}} \{\lambda_k |z_k^* - i_k|\} = -u(i). \tag{1}$$

Definition 2 (*Minimum weighted distance to z^**) Given a population $\mathcal{P}$, a weight vector λ, and a reference point z^*. Then, the minimum weighted distance to z^* for the weight vector λ for all individuals $a \in \mathcal{P}$ $d_{\min}(\mathcal{P}, \lambda, z^*)$ is defined as

$$d_{\min}(\mathcal{P}, \lambda, z^*) = \min_{a \in \mathcal{P}} \{d(a, \lambda, z^*)\}. \tag{2}$$

An individual $i \in \mathcal{P}$ for which $d(i, \lambda, z^*) = d_{\min}(\mathcal{P}, \lambda, z^*)$ is called dominating λ ($dom_{\lambda,\mathcal{P}}$) throughout this paper.

Theorem 1 (Computation of the $R2$ contribution) *Let $\mathcal{P}$ be a population and Λ a set of weight vectors. The set $X \subseteq \Lambda$ defines the set of weight vectors for which $\forall \lambda \in X : dom_{\lambda,\mathcal{P}} = i$. For all $\lambda \in X$, there exists an individual $j_\lambda \in \mathcal{P} : j_\lambda \neq i$ each with $dom_{\lambda,\mathcal{P}\setminus\{i\}} = j_\lambda$,* i. e., *the second shortest weighted distance for λ* in $\mathcal{P}$ to z^*. *Then, the $R2$ contribution of i is given by*

$$R2(i, \mathcal{P}, \Lambda, z^*) = \frac{1}{|\Lambda|} \sum_{\lambda \in X} d(j_\lambda, \lambda, z^*) - d(i, \lambda, z^*). \tag{3}$$

Proof (Computation of the $R2$ contribution) The $R2$ value of a population $\mathcal{P}$ is given by

$$R2(\mathcal{P}, \Lambda, z^*) = \frac{1}{|\Lambda|} \sum_{\lambda \in \Lambda} \min_{a \in \mathcal{P}} \{d(a, \lambda, z^*)\}. \tag{4}$$

Note that only the individual that dominates a weight vector $\lambda \in \Lambda$ contributes to the $R2$ value of $\mathcal{P}$. Let $i \in \mathcal{P}$ be an individual and $X \subseteq \Lambda$ a subset of weight vectors with $\forall \lambda \in X : dom_{\lambda,\mathcal{P}} = i$. Thus, i has the lowest weighted distance to z^* for all weight vectors in X.

If there exists another individual $j \in \mathcal{P}$ with $j \neq i$ for which $d(j, \lambda, z^*) = d(i, \lambda, z^*)$ for any weight vector $\lambda \in X$, then i has been selected as individual with lowest weighted distance and j as individual with the second lowest weighted distance although their weighted distance is the same. Switching the order of two or more individuals with the same weighted distance has neither an influence on the $R2$ contribution of a single individual nor on the whole population's $R2$ value as it comprises zero for that weight vector.

The $R2$ value of $\mathcal{P}$ using only the weight vectors $\lambda \in X$ is determined by

$$\begin{aligned} R2(\mathcal{P}, X, z^*) &= \sum_{\lambda \in X} \min_{a \in \mathcal{P}} \{d(a, \lambda, z^*)\} \\ &= \sum_{\lambda \in X} d(i, \lambda, z^*). \end{aligned} \tag{5}$$

Let $Y = \Lambda \setminus X$ be the set of weight vectors for which $\forall \lambda \in Y : dom_{\lambda,\mathcal{P}} \neq i$. This implies

$$\begin{aligned} R2(\mathcal{P}, Y, z^*) &= \sum_{\lambda \in Y} \min_{a \in \mathcal{P}} \{d(a, \lambda, z^*)\} \\ &= \sum_{\lambda \in Y} \min_{a \in \mathcal{P}\setminus\{i\}} \{d(a, \lambda, z^*)\} \end{aligned} \tag{6}$$

because the individual i has no influence on the $R2$ value in this case. Since $X \cup Y = \Lambda$ and $X \cap Y = \emptyset$, the $R2$ value of $\mathcal{P}$ can be computed as

$$\begin{aligned} &R2(\mathcal{P}, \Lambda, z^*) \\ &= \frac{1}{|\Lambda|} \left(\sum_{\lambda \in Y} \min_{a \in \mathcal{P}} \{d(a, \lambda, z^*)\} + \sum_{\lambda \in X} \min_{a \in \mathcal{P}} \{d(a, \lambda, z^*)\} \right) \\ &\overset{\text{Eq. } 5 \text{ and } 6}{=} \frac{1}{|\Lambda|} \left(\sum_{\lambda \in Y} \min_{a \in \mathcal{P}\setminus\{i\}} \{d(a, \lambda, z^*)\} + \sum_{\lambda \in X} d(i, \lambda, z^*) \right) \end{aligned} \tag{7}$$

and the $R2$ value of $\mathcal{P}\setminus\{i\}$ as

$$\begin{aligned} &R2(\mathcal{P}\setminus\{i\}, \Lambda, z^*) \\ &= \frac{1}{|\Lambda|} \left(\sum_{\lambda \in Y} \min_{a \in \mathcal{P}\setminus\setminus\{i\}} \{d(a, \lambda, z^*)\} + \sum_{\lambda \in X} \min_{a \in \mathcal{P}\setminus\{i\}} \{d(a, \lambda, z^*)\} \right). \end{aligned} \tag{8}$$

The $R2$ contribution of the individual $i \in \mathcal{P}$ is determined as follows:

$$\begin{aligned} &R2(i, \mathcal{P}, \Lambda, z^*) = R2(\mathcal{P}\setminus\{i\}, \Lambda, z^*) - R2(\mathcal{P}, \Lambda, z^*) \\ &\qquad \overset{\text{Eq. } 7 \text{ and } 8}{=} \\ &\frac{1}{|\Lambda|} \left(\sum_{\lambda \in Y} \min_{a \in \mathcal{P}\setminus\{i\}} \{d(a, \lambda, z^*)\} + \sum_{\lambda \in X} \min_{a \in \mathcal{P}\setminus\{i\}} \{d(a, \lambda, z^*)\} \right) \\ &\quad - \frac{1}{|\Lambda|} \left(\sum_{\lambda \in Y} \min_{a \in \mathcal{P}\setminus\{i\}} \{d(a, \lambda, z^*)\} + \sum_{\lambda \in X} d(i, \lambda, z^*) \right) \\ &= \frac{1}{|\Lambda|} \sum_{\lambda \in X} \left(\min_{a \in \mathcal{P}\setminus\{i\}} \{d(a, \lambda, z^*)\} - d(i, \lambda, z^*) \right). \end{aligned} \tag{9}$$

Per definition the term $\min_{a \in \mathcal{P}\setminus\{i\}} \{d(a, \lambda, z^*)\}$ is the weighted distance from the second dominating individual $j_\lambda \in \mathcal{P}$ for each weight vector $\lambda \in X$ to z^*. Therefore, it is replaced by $d(j_\lambda, \lambda, z^*)$ resulting in

$$R2(i, \mathcal{P}, \Lambda, z^*) = \frac{1}{|\Lambda|} \sum_{\lambda \in X} d(j_\lambda, \lambda, z^*) - d(i, \lambda, z^*) \tag{10}$$

□

Our proof shows that the $R2$ contribution of i only depends on the weight vectors $\lambda \in X$ for which i is dominating and the individual $j_\lambda = dom_{\lambda,\mathcal{P}\setminus\{i\}}$ (second shortest weighted distance) for each of these weight vectors.

The $R2$ contribution of i then equals the sum of its contribution for all weight vectors $\lambda \in X$, divided by $|\Lambda|$. This means if the two individuals $dom_{\lambda,\mathcal{P}}$ and $dom_{\lambda,\mathcal{P}\setminus\{dom_{\lambda,\mathcal{P}}\}}$ are known for all weight vectors $\lambda \in \Lambda$, the $R2$ contribution of every individual can be computed by summing up the difference of $dom_{\lambda,\mathcal{P}\setminus\{dom_{\lambda,\mathcal{P}}\}}$'s and $dom_{\lambda,\mathcal{P}}$'s weighted distance to z^* for all weight vectors $\lambda \in \Lambda$ and dividing it by $|\Lambda|$.

$dom_{\lambda,\mathcal{P}\setminus\{i\}} = j$. Now, the difference of both distances is added to the $R2$ contribution of individual i

$$r2c[i] = r2c[i] + (d(j,\lambda,z^*) - d(i,\lambda,z^*)).$$

After iterating over all weight vectors, the array $r2c$ contains the $R2$ contribution of each individual $p \in \mathcal{P}$ multiplied by factor $|\Lambda|$. To get the exact $R2$ contribution of each individual, all values in $r2c$ are divided by $|\Lambda|$. This has no influence on the sorting of the population and makes this step redundant. The algorithm of Díaz-Manríquez et al. does not divide the $R2$ contribution values by $|\Lambda|$ [14]. The algorithm provided here terminates in time $O(K\mu|\Lambda|)$, which is equal to the algorithm's runtime presented in [14].

Algorithm 1: Efficient computation of the $R2$ contribution. Note that further optimizations, e.g. line 9 are considered to avoid unnecessary iterations and keep the runtime as low as possible.

Input: Population $\mathcal{P}$, weight vectors Λ, reference point z^*
Output: $r2c$ = $R2$ contribution of each individual $i \in \mathcal{P}$

```
1
2  r2c ← real array of length |P|
3  foreach λ ∈ Λ do
4      λ^1, λ^2 ← ∞
5      index ← −1
6
7      foreach i ∈ P do
8          v ← λ_1 |z*_1 − i_1|
9          if v < λ^2 then
10             for 2 ≤ k ≤ K do
11                 v ← max(v, λ_k |z*_k − i_k|)
12             end for
13             if v < λ^1 then
14                 λ^2 ← λ^1
15                 λ^1 ← v
16                 index ← indexOf(i)
17             end if
18             else if v < λ^2 then
19                 λ^2 ← v
20             end if
21         end if
22     end foreach
23
24     r2c[index] ← r2c[index] + λ^2 − λ^1
25 end foreach
26
27 for 1 ≤ i ≤ |r2c| do
28     r2c[i] ← (1/|Λ|) r2c[i]
29 end for
```

This results in the following algorithm (see Algorithm 1 for pseudocode): To compute the $R2$ contribution of all $p \in \mathcal{P}$, an array $r2c$ of length $|\mathcal{P}|$ that contains the $R2$ contribution of each individual, is created and initialized with zero each. The algorithm then iterates over all weight vectors $\lambda \in \Lambda$ and determines the distances of the individuals $i,j \in \mathcal{P}$ to z^* for which $dom_{\lambda,\mathcal{P}} = i$ and

Because Díaz-Manríquez et al. do not consider the distance of the second dominating individual for each weight vector [14], the algorithm results in an approximation of the $R2$ contribution of each individual. The approximation has influence on the sorting of the population's individuals based on $R2$ contribution. Contrary to that, our algorithm delivers exactly the same results as the original

computation of the $R2$ contribution does, but has a runtime that is reduced by factor μ.

Appendix 2: Description of the control parameters

In the following, we provide an overview over the control parameters that either mNEAT, nNEAT or both algorithms depend on.

Weight mutation range (WMR)
The maximum amount a link's weight can be perturbed by in a single mutation operation.

Modify weight probability (MWP)
The probability that a link of an ANN will get its weight perturbed. The probability applies on each link of an ANN separately.

Add neuron probability (ANP)
The probability that a new neuron is added to an ANN during mutation.

Add link probability (ALP)
The probability that a new link at one or between two existing neurons is added to an ANN during mutation.

Looped connection probability (LCP)
The probability that a new link will be self recurrent.

Crossover probability (CP)
The probability that crossover between two parental solutions is performed during variation. Note that this parameter becomes obsolete, if the natural conditions for crossover are not fulfilled (e.g. less than two parents available).

Mutation probability (MP)
The probability that an offspring solution is mutated. This can be the result from crossover or a copy of an existing solution. Note that if crossover and mutation are not to be performed due to probability, one of both will be forced to be performed with equal probability randomly, unless stated otherwise.

Gene enabled on crossover probability (GEOCP)
Probability that a gene, i.e. link is enabled on crossover if it occurs in both parents and is disabled in either.

Mate by choosing probability (MBCP)
The probability that a common gene, i.e. link of two parents is taken over from one parent during crossover. Otherwise, the offspring's gene is averaged from the parents.

Interspecies mating rate (IMR)
The probability that an offspring solution is created from two parent solutions from different species instead of same species. mNEAT only.

Survival threshold (ST)
The proportion of a species' members (sorted by fitness) that are taken into account for parent selection during offspring creation. mNEAT only.

Replacement rate (RR)
The proportion of solutions of the population that will be replaced by offspring individuals during each epoch. nNEAT only.

Speciation coefficient (SC)
Multipurpose parameter that is applied for different settings around speciation, e.g. the speciation threshold.

Factor C1 excess (FCE)
Coefficient for excess genes for the determination of the distance between two ANNs. A larger value results in a larger influence of excess genes on the total distance.

Factor C2 disjoint (FCD)
Coefficient for disjoint genes for the determination of the distance between two ANNs. A larger value results in a larger influence of disjoint genes on the total distance.

Factor C3 weight difference (FCWD)
Coefficient for common genes' weight difference for the determination of the distance between two ANNs. A larger value results in a larger influence of weight differences (among common genes) on the total distance.

Age bonus malus (ABM)
The bonus or malus (in per cent) a young respectively old solution's fitness is decreased or increased. A lower value (bonus) results in better overall fitness.

Maximum stagnation (MS)
The maximum number of generations a species is allowed to survive without finding a new best solution among its members. mNEAT only.

Selection pressure (SP)
The selection pressure for parent selection. A larger value results in a greater chance for good performing solutions to be selected as parent as well as a lower chance for worse performing.

Age threshold young (ATY)
The number of generations a solution is stated to be a young solution (starting at its creation).

Age threshold old (ATO)
The number of generations from which a solution is stated to be an old solution.

References

1. Yannakakis GN, Togelius J (2018) Artificial intelligence and games. Springer, Berlin
2. Majchrzak K, Quadflieg J, Rudolph G (2015) Advanced dynamic scripting for fighting game AI. In: Chorianopoulos K et al (eds) Entertainment computing—ICEC 2015. Springer, Cham, pp 86–99. ISBN: 978-3-319-24589-8

3. Pinto IP, Coutinho LR (2018) Hierarchical reinforcement learning with Monte Carlo tree search in computer fighting game. In: IEEE transactions on games, pp 1–1. https://doi.org/10.1109/TG.2018.2846028
4. Sato N et al (2015) Adaptive fighting game computer player by switching multiple rule based controllers. In: 2015 3rd international conference on applied computing and information technology/2nd international conference on computational science and intelligence, pp 52–59. https://doi.org/10.1109/ACIT-CSI.2015.18
5. Mendonça MRF, Bernardino HS, Neto RF (2015) Simulating human behavior in fighting games using reinforcement learning and artificial neural networks. In: 2015 14th Brazilian symposium on computer games and digital entertainment (SBGames), pp 152–159. https://doi.org/10.1109/SBGames.2015.25
6. Heidrich-Meisner V, Igel C (2008) Evolution strategies for direct policy search. In: Rudolph G et al (eds) Parallel problem solving from nature—PPSN X. Springer, Berlin, pp 428–437. ISBN: 978-3-540-87700-4
7. Stanley KO (2004) Efficient evolution of neural networks through complexification. Ph.D. thesis. University of Texas, Austin
8. Schrum J, Miikkulainen R (2016) Discovering multimodal behavior in Ms. Pac-Man through evolution of modular neural networks. IEEE Trans Comput Intell AI Games 8(1):67–81
9. Deb K et al (2002) A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans Evol Comput 6(2):182–197
10. van Willigen W, Haasdijk E, Kester L (2013) Fast, comfortable or economical: evolving platooning strategies with many objectives. In: 16th international IEEE conference on intelligent transportation systems (ITSC 2013). IEEE, pp 1448–1455
11. Zitzler E, Laumanns M, Thiele L (2001) SPEA2: improving the strength Pareto evolutionary algorithm. Tech. rep. 103. Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH), Zurich, Switzerland
12. Abramovich O, Moshaiov A (2016) Multi-objective topology and weight evolution of neuro-controllers. In: IEEE congress on evolutionary computation (CEC), IEEE, pp 670–677
13. Künzel S, Meyer-Nieberg S (2018) Evolving artificial neural networks for multi-objective tasks. In: International conference on the applications of evolutionary computation. Springer, pp 671–686
14. Díaz-Manríquez A et al (2013) A ranking method based on the R2 indicator for many-objective optimization. In: 2013 IEEE congress on evolutionary computation (CEC). IEEE, pp 1523–1530
15. Stanley KO, Miikkulainen R (2002) Evolving neural network through augmenting topologies. Evol Comput 10(2):99–127
16. Beume N, Naujoks B, Emmerich M (2007) SMS-EMOA: multiobjective selection based on dominated hypervolume. Eur J Oper Res 181(3):1653–1669
17. Zitzler E, Künzli S (2004) Indicator-based selection in multiobjective search. In: Proceedings of the 8th international conference on parallel problem solving from nature, PPSN VIII. Springer, pp 832–842
18. Knowles JD, Watson RA, Corne DW (2001) Reducing local optima in single-objective problems by multi-objectivization. In: International conference on evolutionary multi-criterion optimization. Springer, pp 269–283
19. Jensen MT (2003) Guiding single-objective optimization using multi-objective methods. In: Workshops on applications of evolutionary computation. Springer, pp 268–279
20. Eiben AE, Smith JE (2015) Introduction to evolutionary computing. Springer, Berlin
21. Falcón-Cardona JG, Coello CAC, Emmerich M (2019) CRI-EMOA: a Pareto-front shape invariant evolutionary multi-objective algorithm. In: International conference on evolutionary multi-criterion optimization. Springer, pp 307–318
22. Ishibuchi H et al (2015) Modified distance calculation in generational distance and inverted generational distance. In: International conference on evolutionary multi-criterion optimization. Springer, pp 110–125
23. Hardin DP, Saff EB (2004) Discretizing manifolds via minimum energy points. Not AMS 51(10):1186–1194
24. Trautmann H, Wagner T, Brockhoff D (2013) R2-EMOA: focused multiobjective search using R2-indicator-based selection. In: International conference on learning and intelligent optimization. Springer, pp 70–74
25. Moriarty DE, Mikkulainen R (1996) Efficient reinforcement learning through symbiotic evolution. Mach Learn 22(1–3):11–32
26. Demšar J (2006) Statistical comparisons of classifiers over multiple data sets. J Mach Learn Res 7:1–30
27. Shaffer JP (1986) Modified sequentially rejective multiple test procedures. J Am Stat Assoc 81(395):826–831
28. Skillings JH, Mack GA (1981) On the use of a Friedman-type statistic in balanced and unbalanced block designs. In: Technometrics, pp 171–177
29. Auger A, Hansen N (2005) A restart CMA evolution strategy with increasing population size. In: 2005 IEEE congress on evolutionary computation, vol 2. IEEE, pp 1769–1776
30. Lu F et al (2013) Fighting game artificial intelligence competition platform. In: 2013 IEEE 2nd global conference on consumer electronics (GCCE). IEEE, pp 320–323
31. Wymann B et al, TORCS, the Open Racing Car Simulator. In: 4 (2000). Software available at https://www.torcs.sourceforge.net. Accessed 17 Sept 2019
32. Cho BH, Park CJ, Yang KH (2007) Comparison of AI techniques for fighting action games—genetic algorithms/neural networks/evolutionary neural networks. In: Ma L, Rauterberg M, Nakatsu R (eds) Entertainment computing—ICEC 2007. Springer, Berlin, pp 55–65. ISBN: 978-3-540-74873-1
33. Fighting game AI competition—introduction. http://www.ice.ci.ritsumei.ac.jp/ftgaic/index-1.html. Accessed 17 Sept 2019
34. Neufeld X, Mostaghim S, Perez-Liebana D (2017) HTN fighter: planning in a highly-dynamic game. In: 2017 9th computer science and electronic engineering (CEEC), pp 189–194. https://doi.org/10.1109/CEEC.2017.8101623
35. Martínez-Arellano G, Cant R, Woods D (2017) Creating AI characters for fighting games using genetic programming. In: IEEE transactions on computational intelligence and Ai in games 9.4, pp 423–434. https://doi.org/10.1109/TCIAIG.2016.2642158
36. Kristo T, Maulidevi NU (2016) Deduction of fighting game countermeasures using neuroevolution of augmenting topologies. In: 2016 international conference on data and software engineering (ICoDSE), pp 1–6. https://doi.org/10.1109/ICODSE.2016.7936127
37. Browne CB et al (2012) A survey of Monte Carlo tree search methods. IEEE Trans Comput Intell AI Games 4(1):1–43
38. Yamamoto K et al (2014) Deduction of fighting-game countermeasures using the K-nearest neighbor algorithm and a game simulator. In: 2014 IEEE conference on computational intelligence and games, pp 1–5. https://doi.org/10.1109/CIG.2014.6932915
39. Cover T, Hart P (1967) Nearest neighbor pattern classification. IEEE Trans Inf Theory 13(1):21–27
40. Robison AD (2017) Neural network AI for FightingICE. California Polytechnic State University, San Luis Obispo (Thesis)
41. Osés Laza A (2017) Reinforcement Learning in Videogames. B.S. thesis. Universitat Politècnica de Catalunya
42. Yoon S, Kim K (2017) Deep Q networks for visual fighting game AI. In: 2017 IEEE conference on computational intelligence and

games (CIG), pp 306–308. https://doi.org/10.1109/CIG.2017.8080451
43. Schmitt J, Köstler H (2016) A multi-objective genetic algorithm for simulating optimal fights in starcraft II. In: 2016 IEEE conference on computational intelligence and games (CIG). IEEE, pp 1–8
44. Park H, Kim K (Aug. 2014) Learning to play fighting game using massive play data. In: 2014 IEEE conference on computational intelligence and games, pp 1–2. https://doi.org/10.1109/CIG.2014.6932921
45. Asayama K et al (2015) Prediction as faster perception in a real-time fighting video game. In: 2015 IEEE conference on computational intelligence and games (CIG), pp 517–522. https://doi.org/10.1109/CIG.2015.7317672
46. Zuin GL, Macedo YPA (2015) Attempting to discover infinite combos in fighting games using hidden markov models. In: 2015 14th Brazilian symposium on computer games and digital entertainment (SBGames), pp 80–88. https://doi.org/10.1109/SBGames.2015.15
47. Zuin GL et al (2016) Discovering combos in fighting games with evolutionary algorithms. In: Proceedings of the genetic and evolutionary computation conference 2016. GECCO'16. ACM, Denver, Colorado, USA, pp 277–284. ISBN: 978-1- 4503-4206-3. https://doi.org/10.1145/2908812.2908908
48. Demediuk S et al (2017) Monte Carlo tree search based algorithms for dynamic difficulty adjustment. In: 2017 IEEE conference on computational intelligence and games (CIG), pp 53–59. https://doi.org/10.1109/CIG.2017.8080415
49. Kim M, Kim K (2017) Opponent modeling based on action table MCTS-based fighting game AI. In: 2017 IEEE conference on computational intelligence and games (CIG), pp 178–180. https://doi.org/10.1109/CIG.2017.8080432
50. Nguyen DTT, Quang Y, Ikeda K (2017) Optimized non-visual information for deep neural network in fighting game. In: Proceedings of the 9th international conference on agents and artificial intelligence (ICAART 2017), pp 676–680
51. Kim M-J, Ahn CW (2018) Hybrid fighting game AI using a genetic algorithm and Monte Carlo tree search. In: Proceedings of the genetic and evolutionary computation conference companion. ACM, pp 129–130
52. Rakshit P, Konar A, Das S (2017) Noisy evolutionary optimization algorithms—a comprehensive survey. Swarm Evol Comput 33:18–45
53. Ishihara M et al (2015) Investigating kinect-based fighting game AIs that encourage their players to use various skills. In: 2015 IEEE 4th global conference on consumer electronics (GCCE). IEEE, pp 334–335
54. Hansen MP, Jaszkiewicz A (1994) Evaluating the quality of approximations to the non-dominated set. IMM, Department of Mathematical Modelling, Technical University of Denmark
55. Brockhoff D, Wagner T, Trautmann H (2012) On the properties of the R2 indicator. In: Proceedings of the 14th annual conference on Genetic and evolutionary computation. ACM, pp 465–472
56. Gómez RH, Coello CAC (2013) MOMBI: a new metaheuristic for many-objective optimization based on the R2 indicator. In: 2013 IEEE congress on evolutionary computation (CEC). IEEE, pp 2488–2495
57. Shang K et al (2018) A new R2 indicator for better hypervolume approximation. In: Proceedings of the genetic and evolutionary computation conference. GECCO'18. Kyoto, Japan. ACM, pp 745–752. ISBN: 978-1-4503-5618-3. https://doi.org/10.1145/3205455.3205543
58. Ma X et al (2018) On Tchebycheff decomposition approaches for multiobjective evolutionary optimization. IEEE Trans Evol Comput 22(2):226–244