Relative Convergence Speed

Almuth Meier

Computational Intelligence Group
Department of Computer Science
University of Oldenburg, Germany

Abstract. In this document, we describe our performance measure Relative Convergence Speed (RCS). We proposed it in [1] to evaluate the convergence speed of heuristic dynamic optimization algorithms more fairly than possible with existing measures. First, we explain the related measure Absolute Recovery Rate (ARR), and motivate our improvements that led to RCS. Afterwards, we visualize the differences of both measure to further emphasize the advantages of RCS.

1 Absolute Recovery Rate

Absolute recovery rate (ARR) is a behavior-based measure for the speed an optimization algorithm starts converging to the global optimum before the next change happens [2,3]. It is computed as follows:

$$\mathrm{ARR} = \frac{1}{P} \sum_{c=1}^{P} \frac{\sum_{t=1}^{g(c)} \big(f(\mathbf{x}^*_{c_t}, c) - f(\mathbf{x}^*_{c_1}, c)\big)}{g(c) \cdot \big(f(\mathbf{o}_c, c) - f(\mathbf{x}^*_{c_1}, c)\big)} \tag{1}$$

Variable $g(c)$ measures the number of generations during change period c, and P is the overall number of change periods. Variable $f(\mathbf{o}_c, c)$ signifies the global optimum fitness for change period c, while $f(\mathbf{x}^*_{c_t}, c)$ represents the best fitness the algorithm has found until generation t in change period c. Thus, $f(\mathbf{x}^*_{c_t}, c)$ is the best fitness the algorithm has found in the first generation of change period c. Following [2,3], the best value achievable for ARR is one, the worst is zero. However, in Section 3 we show that also negative values can occur. Therefore, in our implementation we use absolute differences to circumvent negative values.

2 Relative Convergence Speed

In [1], we introduced the measure relative convergence speed (RCS) since in dynamic optimization it is not only important to quickly *leave* poor solutions, considered by ARR, but also to *find* the global optimum with as few generations as possible. RCS measures how fast the ES approaches the global optimum relatively to the other algorithms included in the comparison:

$$\mathrm{RCS} = \frac{1}{P} \sum_{c=1}^{P} \frac{\sum_{t=1}^{g(c)} t \cdot |f(\mathbf{x}^*_{c_t}, c) - f(\mathbf{o}_c, c)|}{\sum_{t=1}^{g(c)} t \cdot |f(\mathbf{x}^*_{c_{\mathrm{worst}}}, c) - f(\mathbf{o}_c, c)|} \tag{2}$$

The semantic of the symbols is the same as defined for ARR. The worst fitness any algorithm has achieved during change period c is denoted by $f(\mathbf{x}^*_{c_{\text{worst}}}, c)$. RCS ranges from zero as best to one as worst value.

ARR is designed so that it measures how fast an algorithm can get *far away* from its *first solution*, i. e., the best found fitness in the first generation after a change. Due to this, algorithms that start with a very poor solution may achieve a larger ARR than those starting with a better solution. We illustrate this with an example. Assuming the best fitness values found by algorithm A during one change period are $[9, 6, 6, 0]$, those of algorithm B are $[4, 3, 3, 0]$, and zero is the global optimum fitness. Then, A has ARR $= 0.42$ and B has ARR $= 0.36$. Thus, A outperforms B although B in every but the last generation achieves a better fitness than A.

To overcome this unexpected behavior, we proposed RCS that takes into account how fast an algorithm finds solutions *near to* the *global optimum*. There exist five main differences to the definition of ARR. 1) The global optimum $\mathbf{o}_c$ has to be available. 2) As motivated above, we employ the difference $f(\mathbf{x}^*_{c_t}, c) - f(\mathbf{o}_c, c)$ instead of the difference $f(\mathbf{x}^*_{c_t}, c) - f(\mathbf{x}^*_{c_1}, c)$ in order to measure how near a solution is to the optimum. 3) In order to scale this difference into the interval $[0, 1]$ it is divided by the difference between the global optimum fitness and the worst fitness of any algorithm. By this, RCS measures the convergence speed of the algorithms relative to the worst fitness. If instead the difference $f(\mathbf{x}^*_{c_1}, c) - f(\mathbf{o}_c, c)$ would be incorporated like in ARR, the RCS would exhibit behavior similar to ARR in cases like the above presented example, but with inverted range. 4) The difference terms are weighted with the generation number in order to reward early good fitness values and penalize high fitness values at the and of change periods. 5) In contrast to ARR, the best value for RCS is zero and the worst is one. In Section 3, we scrutinize ARR and RCS deeper and show further disadvantages of ARR as division by zero and negative ARR values.

3 Examination of RCS and ARR

We examine the differences between ARR and RCS by means of a simplified optimization scenario where three discrete fitness values are possible (0,1, and 2) with zero as best fitness, and the optimization is conducted for one change period with two generations. Therefore, 3^2 possible optimization runs exist for one algorithm. For two algorithms, called A and B, there are $\binom{9}{2} = 36$ possible combinations disregarding the cases where both algorithms have the same sequence of fitness values. For each of these combination, Figure 1 shows the ARR, while Figure 2 visualizes the RCS. The plots depict for each generation the best fitness value the respective algorithm has. The legend contains the ARR or RCS value, respectively, while the color of the lines for the fitness progress is chosen according to the metric value. The darker the color, the better the metric value. Thus, a dark color represents a high ARR and a low RCS, respectively. Due to exponentially increasing number of plots, we do not visualize scenarios

with more fitness levels or generations. To signify a certain plot, for example the third from left in the second row, we write (2,3).

The most obvious disadvantage of ARR is that negative ARR values are possible though the authors proposing this measure claim that it ranges between zero and one [3]. In case the fitness level increases during a change period, the ARR becomes negative, e.g., in plot (5,2). That plot also shows the second limitation of ARR. If an algorithm starts a change period with the best possible fitness, division by zero occurs so that the ARR cannot be computed. In our implementation, we handle that case by setting ARR to one. However, this does not reflect the algorithm's behavior since the algorithm has ARR = 1 independent of whether its fitness stays low or increases, see e.g. plot (1,2). The third drawback are unintuitive ARR values in case algorithms start at different fitness levels, and their fitness decreases or stays the same. If the algorithm that started on a lower fitness always has equal or better fitness than the other algorithm, its ARR unexpectedly is not better. This happens in plots (8,1), (8,2), and (6,4). This behavior especially disadvantages prediction-based optimizers, since they might often obtain a lower fitness level at the beginning of change periods than algorithms without prediction because the prediction is supposed to lead them to promising regions in the solution space.

Comparing the results for ARR and RCS, it can be obtained that RCS solves these disadvantages. RCS cannot become negative, since in minimization problems both $f(\mathbf{x}^*_{c_t}, c)$ and $f(\mathbf{x}^*_{c_{\text{worst}}}, c)$ always are greater or equal to $f(\mathbf{o}_c, c)$ so that $f(\mathbf{x}^*_{c_t}, c) - f(\mathbf{o}_c, c)$ and $f(\mathbf{x}^*_{c_{\text{worst}}}, c) - f(\mathbf{o}_c, c)$ are positive. For maximization, these differences have a negative result, hence RCS is positive as well. Thus $|\cdot|$, used in the original publication [1], is not required. Division by zero takes place in the computation of RCS if all algorithms achieve the best possible fitness in all generations of the respective change period. Then, all algorithms get RCS = 0 for that change period. In contrast to ARR's behavior when division by zero occurs, this represents the situation that all algorithms perfectly converge to the global optimum. Furthermore, RCS does not disadvantage prediction-based optimizers as can be observed in plots (8,1), (8,2), and (6,4).

References

1. A. Meier and O. Kramer. Prediction with recurrent neural networks in evolutionary dynamic optimization. In *Applications of Evolutionary Computation (EvoApplications)*, pages 848–863, 2018.
2. T. T. Nguyen. *Continuous Dynamic Optimization using Evolutionary Algorithms*. PhD thesis, University of Birmingham, 2011.
3. T. T. Nguyen and X. Yao. Continuous dynamic constrained optimization – The challenges. *Transactions on Evolutionary Computation*, 16(6):769–786, 2012.

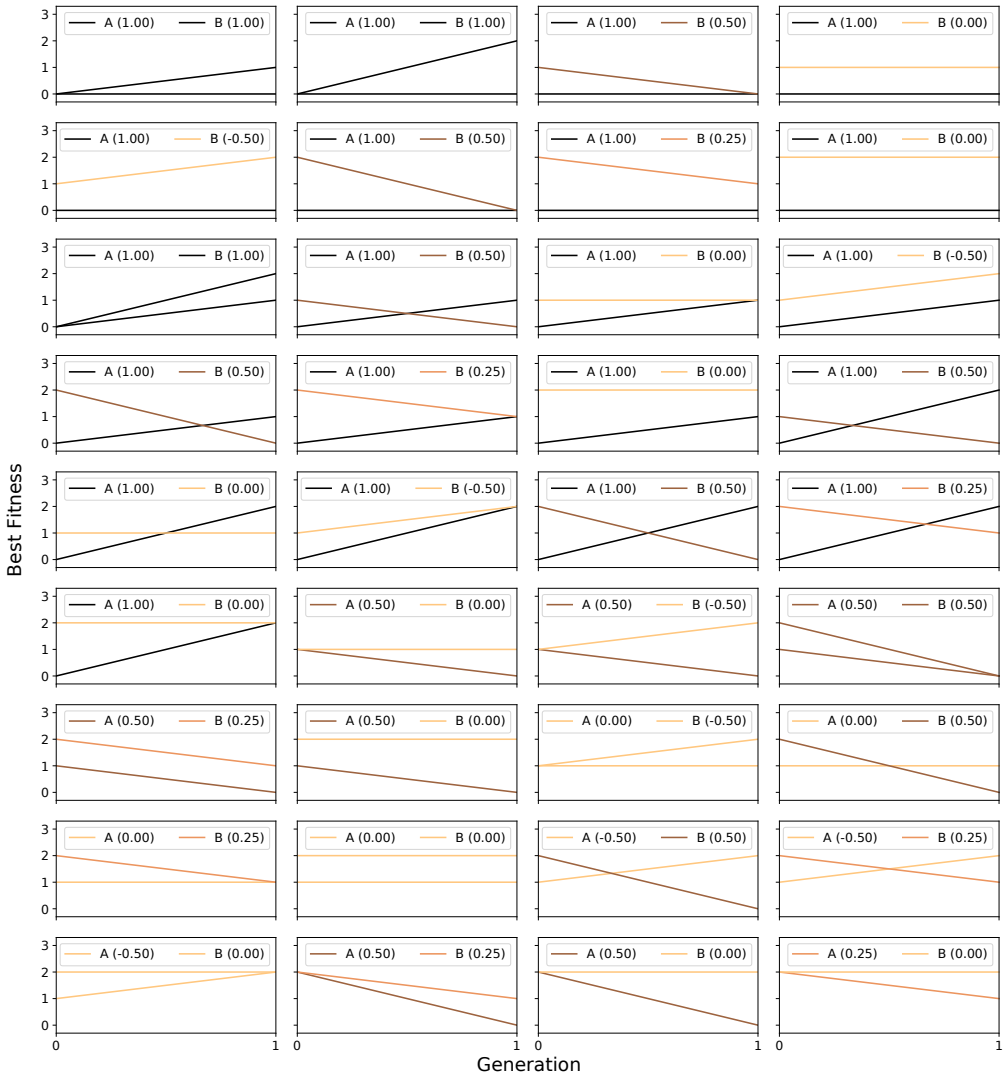

Fig. 1: ARR (in brackets) for all possible combinations of two algorithms (A and B) with three fitness levels (0,1,2) and two generations. Fitness plots with better ARR values, i.e., larger ones, have darker colors

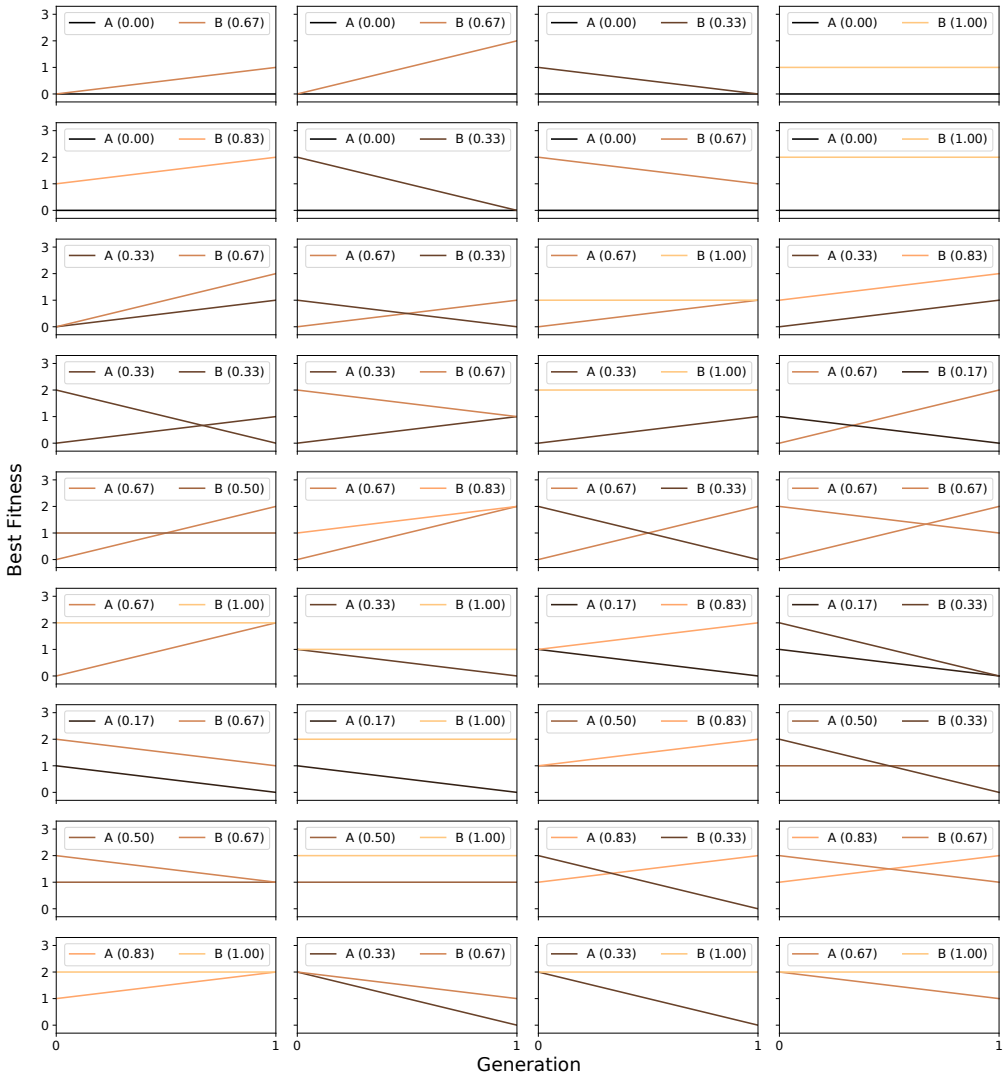

Fig. 2: RCS (in brackets) for all possible combinations of two algorithms (A and B) with three fitness levels (0,1,2) and two generations. Fitness plots with better RCS values, i.e., lower ones, have darker colors