Bicriteria optimisation of average and worst-case performance using coevolutionary algorithms

Benford, Alistair; Olhofer, Markus; Rodemann, Tobias; Lehre, Per Kristian

Bicriteria optimisation of average and worst-case performance using coevolutionary algorithms

Alistair Benford*, Markus Olhofer†, Tobias Rodemann† and Per Kristian Lehre*
*School of Computer Science, University of Birmingham, Birmingham, B15 2TT, UK
†Honda Research Institute Europe, Carl-Legien-Straße 30, 63073 Offenbach/Main, Germany

***Abstract*—A common aim in real-world optimisation problems is to seek a solution offering highest performance on expected scenarios, but at the same time guaranteeing an at least acceptable performance on worst-case scenarios. Competitive coevolution evolves a population of solutions alongside a population of difficult scenarios in order to find so-called robust solutions. However, solutions with maximal worst-case performance often exhibit poor performance on more typical scenarios. Existing coevolutionary approaches generally favour such solutions over ones which sacrifice only a small amount of average performance for an almost as large gain in worst-case performance, despite the latter being favourable in most practical applications.**

We present a new coevolutionary algorithm which treats average performance and worst-case performance as two objectives of a bicriteria optimisation problem and seeks the corresponding Pareto front. Such an algorithm enables the discovery of solutions with strong performance in both of these metrics, which would otherwise be rejected if optimising for only one. Our algorithm constitutes the first coevolutionary approach to this solution concept. We also provide experimental results on the performance of this algorithm on the design of smart controllers for the management of energy flow between buildings, renewable energy sources, and electric vehicles.

I. Introduction

For many real-world optimisation problems, the observed payoff of a solution may depend on a number of scenario variables which cannot be controlled. Handling this uncertainty is a central consideration of robust optimisation (see, for example, [10]), where a commonly adopted approach is to seek the solution whose worst-case performance across all scenarios is the highest.

Finding such a solution is especially difficult if the set of possible scenarios is so large that directly calculating worst-case performance is computationally impractical. In such cases, competitive coevolutionary algorithms can instead be used to seek robust solutions. Coevolutionary algorithms evolve a population of solutions (referred to as the *predator population*) alongside a population of scenarios (referred to as the *prey population*). The predators constantly adapt to seek better payoffs with respect to their contemporary preys, while the preys evolve to minimise the payoffs of the predators. This approach requires very little information about the underlying payoff function, and so while various robust optimisation methods are available (see, for example, [2]), coevolution is an especially appealing technique for settings such as derivative-free or simulation-based optimisation [9, 15, 17].

Robust optimisation delivers performance guarantees, making it practical for a wide range of applications – a collection of renewable energy sources should meet a certain demand in all weather conditions; an investment portfolio may be subject to a maximum acceptable loss. However, solutions delivered by robust optimisation are often excessively conservative, as optimising to insure against poor performance for the worst-case scenario may result in less good performance for the more typical scenarios. This trade-off between average performance and worst-case performance is a critical consideration for practitioners, for whom the most desirable solutions are those which sacrifice only a small amount of average performance for a large gain in worst-case performance (or sacrifice only a small amount of worst-case performance for a large gain in average performance). To illustrate, consider the objective space indicated by the shaded region in Figure 1. Robust optimisation will always favour the solution with the best worst-case performance (in this case, C) regardless of average performance. On the other hand, optimising for average performance alone will return solution A. However both approaches fail to identify solution B, which has good performance in both metrics. Thus, either of these approaches taken alone are inadequate to address the trade-off faced by practitioners.

Fig. 1: Average versus worst-case performance

What is then needed are algorithms which concurrently optimise average and worst-case performance for scenario-based optimisation problems. In the case of linear programming, Chassein and Goerigk [4] describe the solution concept of an average-case-worst-case curve (AC-WC curve) and present a corresponding algorithm. Our main contribution is the first coevolutionary algorithm for finding the AC-WC curve.

In Section II we state the problem setting, consider the trade off between average and worst-case performance, and define precisely the AC-WC curve. In Section III we introduce AMCA, a coevolutionary algorithm adapted to this setting.

Finally, in Section IV, we analyse the performance of AMCA on a real-world energy management problem.

Notation: Given a set S, we use $\mathcal{P}(S)$ to denote the set of probability distributions on S. For a function $g : S \to \mathbb{R}$ and a finite set $A \subseteq S$, we write $y = \operatorname{argmin}_{x \in A} g(x)$ to indicate that y is chosen uniformly at random from the set $\{x \in A : g(x) = \min_{x' \in A} g(x')\}$. Given $n \in \mathbb{N}$, we have $[n] = \{1, \dots, n\}$.

II. Problem setting and solution concepts

We consider problems defined by a payoff function $f : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$, where $f(x, y)$ is the performance of solution x (a vector of design variables) when evaluated in the scenario defined by y (a vector of uncertainty variables). The general aim is to choose x to give a high value of $f(x, y)$, given that controlling y is not possible.

A. Maximising worst-case performance

A solution concept often used for this setting in robust optimisation is to identify the solution with the highest worst-case performance. Precisely, we aim to maximise the function

$$f_{\text{wc}}(x) = \min_{y \in \mathcal{Y}} f(x, y). \tag{1}$$

This provides performance guarantees for all scenarios representable by $\mathcal{Y}$. As discussed in Section I, situations where $\mathcal{Y}$ is too large for $\min_{y \in \mathcal{Y}} f(x, y)$ to be computed directly motivate the use of coevolution for this solution concept.

B. Maximising average performance

Even if all scenarios appearing in $\mathcal{Y}$ are realistic, those that incur worst-case performance may be atypical or occur with low probability. While worst-case optimisation has clear benefit, in applications we often ask that solutions also have a good average performance. If the scenario space is equipped with an underlying probability measure $p \in \mathcal{P}(\mathcal{Y})$, this corresponds to maximising the function

$$f_{\text{ac}}(x) = \mathbb{E}[f(x, Y)] \tag{2}$$

where Y is sampled according to p. In practice, p may not be available to the optimiser, or (2) may again be too expensive to compute directly. Such cases can be addressed by replacing f_{ac} with an inexpensive approximation. A natural choice is to fix a small number of benchmark scenarios $y_1, \dots, y_k$ and set $f_{\text{ac}}(x) = \frac{1}{k} \sum_{i \in [k]} f(x, y_i)$, thus making the cost of approximating f_{ac} equivalent to k evaluations of f.

C. Trading off between average and worst-case performance

In practice, adopting just one of worst-case performance or average performance is rarely enough to deliver practical solutions for a scenario optimisation setting; practitioners often aim to find solutions that perform well in both of these metrics. As one should not generally expect the optima of f_{wc} and f_{ac} to coincide, a solution concept corresponding to this aim should lend careful consideration to the trade-off between f_{ac} and f_{wc}.

Several approaches have been proposed to this end. To evaluate the performance of game-playing agents, Lactot et al. [12] use a linear aggregation of average and worst-case performance (while claiming equal weighting to be most natural for their setting, as the units are identical). In a linear programming setting with uncertain constraint vectors, Bertsimas and Sim [3] control the level of conservatism using a parameter Γ. Further methods exist for balancing typical performance with risk more generally. Assuming access to the distribution of scenarios, Garatti and Campi [10] propose finding the best performance that can be guaranteed with probability $1-\varepsilon$, so that ε is a tuning parameter for robustness (with $\varepsilon = 0$ corresponding to maximising worst-case performance alone). In finance, mean-variance analysis is used to maximise expected return while minimising variance, which used instead of worst-case performance to represent risk [16].

Common to these approaches is the need to assert a preference for the relative value of f_{ac} and f_{wc} before the optimisation process (through parameters, weightings, or otherwise). However, this preference is itself naturally informed by the relative combinations of f_{ac} and f_{wc} available for the problem. We therefore adopt a multi-objective approach which views all best-possible combinations of f_{ac} and f_{wc} as desirable.

For a multi-objective problem with payoff function $F(x) = (f_1(x), \dots, f_k(x))$, we say x_1 *Pareto dominates* x_2 if $f_i(x_1) \geqslant f_i(x_2)$ for every $i \in [k]$ and $f_j(x_1) > f_j(x_2)$ for at least one $j \in [k]$. The *Pareto set* is then defined to be the set of $x \in \mathcal{X}$ which are not Pareto dominated by any other element of $\mathcal{X}$ (so called *Pareto-optimal* solutions). The *Pareto front* is then the image of the Pareto set under F. Chassein and Goerigk [4] define the *average-case-worst-case curve (AC-WC curve)* as the Pareto front of the bicriteria problem defined by $F(x) = (f_{\text{wc}}(x), f_{\text{ac}}(x))$ (so in Figure 1, the AC-WC curve for Figure 1 is indicated by the blue boundary). The AC-WC curve encompasses the full breadth of best-possible combinations of average and worst-case performance, making it a uniquely informative and powerful tool for the trade-off between f_{ac} and f_{wc}. However, despite extensive research into the usefulness of evolutionary computing for worst-case optimisation (in the case of coevolutionary algorithms) and multi-objective optimisation (in the case of multi-objective evolutionary algorithms [6]), our work constitutes the first evolutionary approach to computing the AC-WC curve.

D. Using alternatives to average performance

For the remainder of this paper, we use f_1 in place of f to denote the underlying payoff function of the optimisation problem, and we use f_2 in place of f_{ac}. Thus, we seek the Pareto set of the bi-objective problem defined by

$$F(x) = (\min_{y \in \mathcal{Y}} f_1(x, y), f_2(x)). \tag{3}$$

As discussed in Section II-B, f_2 will be assumed to be an inexpensive substitute for f_{ac}, thus making our algorithm practical even if the underlying probability distribution for $\mathcal{Y}$ is unknown or difficult to compute.

Our methods make no assumption about the relation between f_1 and f_2, and so may also be practical for any bi-objective optimisation problem of the form (3). As an example,

Algorithm 1 AMCA

Require: Population sizes $\mu, \lambda \in \mathbb{N}$
Require: Predator mutation operator $\mathcal{M}_{\mathcal{X}} : \mathcal{X} \to \mathcal{P}(\mathcal{X})$
Require: Prey mutation operator $\mathcal{M}_{\mathcal{Y}} : \mathcal{Y} \to \mathcal{P}(\mathcal{Y})$
Require: Payoff functions $f_1 : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ and $f_2 : \mathcal{X} \to \mathbb{R}$
1: Initialise populations $P_0 \in \mathcal{X}^\mu$ and $Q_0 \in \mathcal{Y}^\mu$
2: **for** $t \in \mathbb{N}$ until termination criterion met **do**
3: **for** $i \in [\mu]$ **do**
4: **for** $j \in [\lambda]$ **do**
5: Sample $x_{i,j} \sim \mathcal{M}_{\mathcal{X}}(P_t(i))$
6: Sample $y_{i,j} \sim \mathcal{M}_{\mathcal{Y}}(Q_t(i))$
7: **end for**
8: **end for**
9: Set $P'_t = \{x_{i,j} : i \in [\mu], j \in [\lambda]\}$
10: Set $Q'_t = \{y_{i,j} : i \in [\mu], j \in [\lambda]\}$
11: Sort P'_t using the ranking method of NSGA-II [7] with respect to the bi-objective function

$$F_t(x) = (\min_{y \in Q'_t} f_1(x, y)\,,\ f_2(x)\,); \tag{4}$$

after sorting, we may regard $P'_t \in \mathcal{X}^{\mu\lambda}$.
12: **for** $i \in \mu$ **do**
13: Set $P_{t+1}(i) = P'_t(i)$
14: Set $Q_{t+1}(i) = \text{argmin}_{y \in Q'_t} f_1(P'_t(i), y)$
15: **end for**
16: **end for**

if $f_1(x, y)$ measures the speed of an aircraft x subject to weather conditions y and $f_2(x)$ measures the monetary cost of manufacturing x, then the Pareto front of (3) trades off between worst-case speed and manufacturing cost.

III. Algorithm description

The Adversarial Multicriteria Coevolutionary Algorithm (AMCA) uses standard coevolutionary principles of mutation and selection to evolve a predator population $P_t \in \mathcal{X}^\mu$ alongside a prey population $Q_t \in \mathcal{Y}^\mu$. Central to its design is the choice of predator mutation operator $\mathcal{M}_{\mathcal{X}} : \mathcal{X} \to \mathcal{P}(\mathcal{X})$ and prey mutation operator $\mathcal{M}_{\mathcal{Y}} : \mathcal{Y} \to \mathcal{P}(\mathcal{Y})$; given $x \in \mathcal{X}$, $\mathcal{M}_{\mathcal{X}}(x)$ is the probability distribution on $\mathcal{X}$ corresponding to a random mutation of x (and similar for $\mathcal{M}_{\mathcal{Y}}$). The choice of mutation operators is often tailored to the problem, and may also incorporate more advanced coevolutionary methods such as self adaptation [1]. Thus, we leave these operators unspecified in our description.

A full description of AMCA is provided by Algorithm 1. At the beginning, populations $P_0 \in \mathcal{X}^\mu$ and $Q_0 \in \mathcal{Y}^\mu$ are initialised, typically by randomly sampling μ elements of $\mathcal{X}$ and $\mathcal{Y}$ according to some initial distribution. Then the t^{th} generation coevolves P_t and Q_t into P_{t+1} and Q_{t+1} as follows. First, for each predator $P_t(i)$, λ mutants of $P_t(i)$ are added to a set P'_t of predator offspring. Likewise, λ mutants of each prey $Q_t(i)$ are added to a set Q'_t of prey offspring. After this, P'_t and Q'_t both contain $\mu\lambda$ offspring in total, from which the strongest must be selected. To do this, each predator offspring x is given a bi-objective fitness value $F_t(x) = (\min_{y \in Q'_t} f_1(x, y)\,,\ f_2(x)\,)$, which serves as a proxy for (3). The predator offspring are then ranked according to their fitness values, and the best μ ranked elements become the next predator population P_{t+1}. Finally, the next prey population Q_{t+1} is taken to be the μ preys in Q'_t which deliver the worst-case performance on f_1 for the μ predators in P_{t+1}.

Because F_t is a bi-objective function, ranking P'_t requires a choice of method for ranking multi-objective fitness values. Here we propose that of the celebrated NSGA-II algorithm [7]. This method iteratively removes nondomination levels from P'_t, consisting of elements P'_t which are not Pareto dominated by any other yet-to-be-removed elements. Elements of P'_t are then ranked first by their nondomination level, and then within those levels by a measure of crowding distance which prioritises diversity in objective values. Under this regime, elements of P'_t receive positive selective pressure for strong performance on $\min_{y \in Q'_t} f_1(x, y)$ and $f_2(x)$, and elements of Q'_t receive selective pressure for representing the worst-case scenarios for the strongest elements of P'_t. While one could adopt a different ranking method, we emphasise that by assigning highest ranks to non-dominated elements, this approach will always select the best possible approximation to the AC-WC curve, making it suitable choice for our goal.

A. Algorithm cost

As is standard for evolutionary algorithms, we measure runtime by counting the number of times the objective function is called, the assumption being that operations such as mutation and sorting have negligible cost in comparison (see [8]). As our setting relies on both f_1 and f_2, we use c_1 and c_2 to denote the computational costs of evaluating f_1 and f_2 respectively, and report the cost of AMCA in terms of these values.

The cost of AMCA can be inferred from line 11 of Algorithm 1. For every $x \in P'_t$, evaluating $\min_{y \in Q'_t} f_1(x, y)$ has a cost of $c_1|Q'_t| = c_1\lambda\mu$ and evaluating $f_2(x)$ has a cost of c_2. Because P'_t contains $\lambda\mu$ elements in total, line 11 therefore has a total cost of

$$\lambda\mu \cdot (c_1\lambda\mu + c_2) = c_1 \cdot \mu^2\lambda^2 + c_2 \cdot \mu\lambda.$$

The only other line of the algorithm which queries f_1 or f_2 is line 14; however, these required values of f_1 can be inferred from the queries made in line 11, and so line 14 can be performed with no additional cost. Thus, the total cost to perform one generation of AMCA is $c_1 \cdot \mu^2\lambda^2 + c_2\mu\lambda$.

B. A cheaper version of AMCA

While AMCA appears suitable for finding AC-WC curves, its computational cost may be prohibitively expensive for practical applications. Accordingly, we now present a cheaper version of the algorithm (C-AMCA) designed to approximate the dynamics of AMCA with only $O(\mu(\mu + \lambda))$ function evaluations per generation, instead of $O(\mu^2\lambda^2)$.

C-AMCA mirrors the process of Algorithm 1 where possible, but saves function evaluations by making two key changes:

(a) Instead of applying mutation and selection to the predator and prey populations simultaneously, divide each generation into a separate *prey improvement phase* and *predator improvement phase*.

(b) In line 11, if x is an offspring of $P_t(i)$, then C-AMCA uses $f_1(x, Q_t(i))$ as an initial estimate for $\min_{y\in\mathcal{Y}} f_1(x, y)$ instead of an expensive minimisation calculation. (Then, after discarding the worst predator offspring using this initial estimate, the fitness of those remaining can be computed more carefully.)

Note that the estimate in (b) is only effective as $Q_t(i)$ was chosen to deliver worst possible performance on f_1 for $P_t(i)$. For this reason, it is notationally convenient to regard the population of C-AMCA as a set of predator-prey pairs, rather than two separate populations. Formally, at the beginning of generation t the population of C-AMCA will be a multiset $P_t \subseteq \mathcal{Z} := \mathcal{X}\times\mathcal{Y}$ of size μ. In the next section we state several subprocedures which operate on subsets of $\mathcal{Z}$ before stating C-AMCA in full. The specific design choices of the algorithm can be motivated, but only with detailed discussion about their relationship to the AC-WC problem setting. Accordingly, we opt to present a description which is concise and reproducible, rather than motivated in detail.

C. Building blocks of C-AMCA

The main loop of C-AMCA is built from four subprocedures: two randomised mutation procedures called PREDMUTATE and PREYMUTATE, and two deterministic selection procedures called SETENEMIES and BESTPREDS. Each of these procedures are stated formally by Algorithms 2-5, however they are perhaps better understood using the following one-sentence descriptions.

PREDMUTATE(P, λ)
Input: $P \subseteq \mathcal{Z}$, $\lambda \in \mathbb{N}$.
Output: $P' \subseteq \mathcal{Z}$ with $|P'| = \lambda \cdot |P|$.
Description: Mutate the predator component of each element of P (create λ mutants for every parent).

PREYMUTATE(P, λ)
Input: $P \subseteq \mathcal{Z}$, $\lambda \in \mathbb{N}$.
Output: $P' \subseteq \mathcal{Z}$ with $|P'| = \lambda \cdot |P|$.
Description: Mutate the prey component of each element of P (create λ mutants for every parent).

SETENEMIES(P, Q)
Input: $P, Q \subseteq \mathcal{Z}$.
Output: A stable set $P' \subseteq \mathcal{Z}$ with $|P'| = |P|$.
Description: Replace the prey component of each element (x, y) of P by searching Q for the prey which delivers worst-case performance on f_1 for x.

BESTPREDS(P, μ)
Input: $P \subseteq \mathcal{Z}$, $\mu \in \mathbb{N}$
Output: $P' \subseteq P$ with $|P'| = \mu$
Description: Sort P with respect to the bi-objective function

$$F((x, y)) = (f_1(x, y), f_2(x));$$

after sorting, return the best μ ranked elements.

Algorithm 2 PREDMUTATE(P, λ)

Require: $P \subseteq \mathcal{X} \times \mathcal{Y}$, $\lambda \in \mathbb{N}$
Require: Mutation operator $\mathcal{M}_\mathcal{X} : \mathcal{X} \to \mathcal{P}(\mathcal{X})$
1: Label $P = \{(x_1, y_1), \ldots, (x_\mu, y_\mu)\}$
2: **for** $i \in [\mu]$ **do**
3: Sample $x_{i,j} \sim \mathcal{M}_\mathcal{X}(x_i)$ for each $j \in [\lambda]$
4: **end for**
5: **return** $\{(x_{i,j}, y_i) : i \in [\mu], j \in [\lambda]\}$

Algorithm 3 PREYMUTATE(P, λ) (with elitism)

Require: $P \subseteq \mathcal{X} \times \mathcal{Y}$, $\lambda \in \mathbb{N}$
Require: Mutation operator $\mathcal{M}_\mathcal{Y} : \mathcal{Y} \to \mathcal{P}(\mathcal{Y})$
1: Label $P = \{(x_1, y_1), \ldots, (x_\mu, y_\mu)\}$
2: **for** $i \in [\mu]$ **do**
3: Set $y_{i,1} = y_i$
4: Sample $y_{i,j} \sim \mathcal{M}_\mathcal{Y}(y_i)$ for each $j \in \{2, \ldots, \lambda\}$
5: **end for**
6: **return** $\{(x_i, y_{i,j}) : i \in [\mu], j \in [\lambda]\}$

Algorithm 4 SETENEMIES(P, Q)

Require: $P, Q \subseteq \mathcal{X} \times \mathcal{Y}$
Require: Payoff function $f_1 : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$
1: Let $X \in \mathcal{X}^{\mu_1}$ be the $\mathcal{X}$-components of elements of P
2: Let $Y \in \mathcal{Y}^{\mu_2}$ be the $\mathcal{Y}$-components of elements of Q
3: **for** $i \in [\mu_1]$ **do**
4: Set $y_i = \text{argmin}_{y\in Y} f_1(X(i), y)$
5: **end for**
6: **return** $\{(X(i), y_i) : i \in [\mu_1]\}$

Algorithm 5 BESTPREDS(P, μ)

Require: $P \subseteq \mathcal{X} \times \mathcal{Y}$, $\mu \in \mathbb{N}$
Require: Payoff functions $f_1 : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ and $f_2 : \mathcal{X} \to \mathbb{R}$
1: Sort P using the ranking method of NSGA-II [7] with respect to the bi-objective function

$$F((x, y)) = (f_1(x, y), f_2(x)) \tag{5}$$

2: Let Q consist of the first μ elements of P after sorting
3: **return** Q

The mutation procedures PREDMUTATE and PREYMUTATE correspond closely to the mutation steps of AMCA. However, one specific difference is that PREYMUTATE is explicitly *elitist*, in the sense that we ensure that at least one of the mutants of each prey is identical to its parent (see line 3 of Algorithm 3). The effect is that preys must compete with their own parents for their place in the next generation. In fact, this elitism is also present for the predators, but manifested in the wider design of AMCA rather than in PREDMUTATE. The benefits and drawbacks of elitism is a much discussed topic

in evolutionary computing (see, for example, [5]), and it can easily be removed from C-AMCA (or indeed added to AMCA) with some simple modification.

Roughly speaking, BESTPREDS is used to replicate the operation of line 11 of Algorithm 1. However, in bypassing the minimisation computation in (4), it relies on the implicit assumption in (5) that, for each $(x, y) \in P$, $f_1(x, y)$ is a suitable substitute for $\min_{(\,\cdot\,,y')\in P} f_1(x, y')$. For this reason, BESTPREDS is most effective when each predator appearing in P has been paired with a prey representing its worst-case scenario. In this context, SETENEMIES is then a useful tool for pairing predators with their worst-case preys so that BESTPREDS may be applied later. In fact, its utility is twofold, as this additionally removes the weakest preys from the algorithm's population while retaining the strongest.

D. Statement of C-AMCA

Algorithm 6 provides a complete description of C-AMCA in terms of the subprocedures of Section III-C. First, a set of predator-prey pairs P_{init} is initialised and $P_0 :=$ SETENEMIES$(P_{\text{init}}, P_{\text{init}})$ is then taken as the initial population. The main loop then consists of two phases: a prey-improvement phase (lines 3-10) which takes as input P_t and returns P'_t, and a predator-improvement phase (lines 11-14) which takes as input P'_t and returns P_{t+1}.

In the prey-improvement phase, the predator-prey pairs of P_t are first labelled as $z_1, \ldots, z_\mu$. Separately for each $z_i := (x_i, y_i)$, PREYMUTATE is used to generate a set Y_i, effectively consisting of λ mutants of y_i. After this, we set $\{u_i\} =$ SETENEMIES$(\{z_i\}, Y_i)$, so that u_i is a copy of $z_i = (x_i, y_i)$, but with y_i replaced by the mutant of y_i causing the worst-case performance on f_1 for x_i. The predator-prey pairs are then collected to form the set $U = \{u_1, \ldots, u_\mu\}$. Finally, we set $P'_t =$ SETENEMIES(U, U), so that newly-discovered strong preys are shared among the wider population.

In the predator-improvement phase, first the predator components of P'_t are mutated using PREDMUTATE to form a very large pool Q of $\mu\lambda$ predator-prey pairs. A set R is then formed, consisting of the μ elements of Q with the highest-ranked predators (using BESTPREDS). As each predator in R has only had worst-case performance estimated using the prey paired with its parent (as per (b)), SETENEMIES is then used to pair the predators of R with the worst-case prey appearing in P'_t for a more accurate estimate. The result is then merged with the parent population P'_t and BESTPREDS is used for a second time to identify the μ elements with the strongest predators to form the next population P_{t+1}.

Let us now verify that the cost of a generation of C-AMCA is indeed only $O(\mu(\mu + \lambda))$, recalling the framework set out in Section III-A where computational costs of c_1 and c_2 are associated with f_1 and f_2 respectively. PREDMUTATE and PREYMUTATE rely on mutation alone and do not query f_1 or f_2, and so these operations have a cost of 0. Calculating SETENEMIES(P, Q) requires knowledge of the value of $f_1(x, y')$ for every $(x, y) \in P$ and $(x', y') \in Q$, and so this has a cost of $c_1|P||Q|$. BESTPREDS(P, μ) requires $f_1(x, y)$ and

Algorithm 6 C-AMCA

Require: $\mu, \lambda \in \mathbb{N}$
Require: Payoff functions $f_1 : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ and $f_2 : \mathcal{X} \to \mathbb{R}$.
1: Initialise a population $P_{\text{init}} \subseteq \mathcal{X} \times \mathcal{Y}$ with $|P_{\text{init}}| = \mu$
2: Set $P_0 =$ SETENEMIES$(P_{\text{init}}, P_{\text{init}})$
3: **for** $t \in \mathbb{N}$ until termination criterion met **do**
4: Label $P_t = (z_1, \ldots, z_\mu)$
5: **for** $i \in [\mu]$ **do**
6: Sample $Y_i \sim$ PREYMUTATE$(\{z_i\}, \lambda)$
7: Set $\{u_i\} =$ SETENEMIES$(\{z_i\}, Y_i)$
8: **end for**
9: Set $U = \{u_1, \ldots, u_\mu\}$
10: Set $P'_t =$ SETENEMIES(U, U)
11: Sample $Q \sim$ PREDMUTATE(P'_t, λ)
12: Let $R =$ BESTPREDS(Q, μ)
13: Let $S =$ SETENEMIES(R, P'_t)
14: Let $P_{t+1} =$ BESTPREDS$(P'_t \cup S, \mu)$
15: **end for**

$f_2(x)$ to be evaluated for every $(x, y) \in P$, and so has a cost of $(c_1 + c_2)|P|$ (except in line 14, where BESTPREDS$(P'_t \cup S, \mu)$ can be computed by recalling previous function evaluations and so has a cost of 0). Thus, if α_i denotes the cost of performing line i of Algorithm 6, we can deduce that the cost per generation is indeed

$$\begin{aligned}\mu \cdot \alpha_7 + \alpha_{10} + \alpha_{12} + \alpha_{13} &= \mu \cdot c_1\lambda + c_1\mu^2 + (c_1 + c_2)\mu\lambda + c_1\mu^2 \\ &= (2c_1 + c_2)\mu\lambda + 2c_1\mu^2 = O(\mu(\mu + \lambda)).\end{aligned}$$

IV. EXPERIMENTAL ANALYSIS OF C-AMCA ON THE DEVELOPMENT OF ROBUST CONTROLLERS

In this final section, we investigate the performance of our algorithm on a real-world energy management problem. As electric vehicles (EVs), battery storage and renewable energy sources have become more common in homes, there is increasing interest in the question of how to efficiently distribute power between these components while still meeting energy demands. This setting is inherently scenario-based – the payoff of a particular controller for such an energy system is sensitive to factors such as the household's energy demands, changing energy prices, and ambient weather conditions (if solar modules are in use). Accordingly, Mushtaq and Rodemann [17] used an archive-based coevolutionary algorithm to seek robust controllers which maximise worst-case performance. Such controllers lie on just one extreme end of the AC-WC curve; we will investigate the effectiveness of C-AMCA at identifying controllers with a range of average-case and worst-case performances.

A. Problem definition

We use a problem definition very similar to that of [17], and so here we opt to describe it concisely rather than explain in full detail and generality. The problem concerns an energy

identifier	description	$\overline{Q}_c$	$\overline{I}_c^{\text{in}}$	$\overline{I}_c^{\text{out}}$
PV	photovoltaic supply	∞	0.0	3.0
HH	household demand	∞	∞	0.0
EV	electric vehicle	35.5	6.6	0.0
B	stationary battery	7.5	3.0	3.0

TABLE I: Listing of energy system components. $\overline{Q}_c$ is given in kWh; $\overline{I}_c^{\text{in}}$ and $\overline{I}_c^{\text{out}}$ are given in kW.

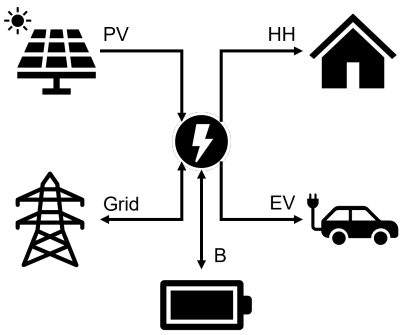

Fig. 2: Energy flow between components (adapted from [17]).

system consisting of a set of components $\mathcal{C} = \{\texttt{PV}, \texttt{HH}, \texttt{EV}, \texttt{B}\}$ (see Table I and Figure 2). The system is simulated over the course of one year, which is divided into n discrete time steps of length T (which we fixed at 30 minutes). The state of a component $c \in \mathcal{C}$ is described using several quantities:

- $I_{c,j}$ denotes the flow of charge (or current) into c at time step j (a negative value of $I_{c,j}$ indicates flow out of c).
- $Q_{c,j}$ denotes the charge of c at time step j.
- $\overline{I}_c^{\text{in}}$ and $\overline{I}_c^{\text{out}}$ denotes the maximum possible flow in and out of c.
- $\overline{Q}_c$ denotes the maximum capacity c.

For components that do not hold charge (PV and HH), we adopt the convention $Q_{c,j} = \infty$ and $\overline{Q}_c = \infty$. Given a controller and a scenario description, one year is simulated by performing the following actions for each time step j in turn.

1) Using the scenario description, EV is labelled as *absent* or *present* and the values of $I_{\texttt{PV},j}$ and $I_{\texttt{HH},j}$ are fixed.
2) Using information about the current state of the system, the controller determines target flows $I_{\texttt{EV},j}^{\text{target}}$ and $I_{\texttt{B},j}^{\text{target}}$ for the electric vehicle and the stationary battery.
3) The actual flows $I_{\texttt{EV},j}, I_{\texttt{B},j}$ are computed by modifying $I_{\texttt{EV},j}^{\text{target}}, I_{\texttt{B},j}^{\text{target}}$ according to the constraints implied by $\overline{I}_{\texttt{EV}}^{\text{in}}, \overline{I}_{\texttt{EV}}^{\text{out}}, \overline{Q}_{\texttt{EV}}, \overline{I}_{\texttt{B}}^{\text{in}}, \overline{I}_{\texttt{B}}^{\text{out}}, \overline{Q}_{\texttt{B}}$.
4) Finally, the charge of each component c at the next time step is computed according to

$$Q_{c,j+1} = Q_{c,j} + I_{c,j} \cdot T.$$

Any surplus or deficit in total energy flow $\sum_{c \in \mathcal{C}} I_{c,j} \cdot T$ is automatically offset by a separate *grid component*, from which energy may purchased at a fixed cost per unit energy C_{grid} (we assume no monetary reward for feeding energy back to the grid).

Once the simulation has concluded, the performance of the controller on the scenario is calculated according to two metrics: *normalised cost* and *customer satisfaction*.

Normalised cost is defined as the ratio between the total cost of energy purchased from the grid and the total amount of energy used by HH and EV (i.e., the components representing energy demands). Formally, this is

$$f_{\text{cost}} = \frac{\sum_{j \in [n]} C_{\text{grid}} \cdot \max\{0, \sum_{c \in \mathcal{C}} I_{c,j} \cdot T\}}{\sum_{j \in [n]} I_{\texttt{HH},j} \cdot T + \sum_{j \in \mathcal{P}_{\texttt{EV}}} I_{\texttt{EV},j} \cdot T},$$

where $\mathcal{P}_{\texttt{EV}}$ denotes the set of j such that EV is labelled as present at time step j. By normalising using the amount of energy used by the customer, scenarios with higher total energy demand are not automatically more difficult – instead, scenarios are difficult according to when and where energy is demanded in the system. Additionally, because here the only non-grid energy source is carbon neutral, f_{cost} may also be used as a proxy for normalised carbon emissions.

Customer satisfaction quantifies the ability of the controller to keep the electric vehicle charged at its departure times. Specifically, if we define $\mathcal{D}_{\texttt{EV}} \subseteq [n]$ to be the set of j such that EV is labelled as present at time step $j-1$ and absent at time step j, then we take

$$f_{\text{sat}} = \frac{1}{|\mathcal{D}_{\texttt{EV}}|} \sum_{j \in \mathcal{D}_{\texttt{EV}}} \alpha(Q_{\texttt{EV},j} / \overline{Q}_{\texttt{EV}}),$$

where $\alpha : [0,1] \to [0,1]$ is a function chosen so that $\alpha(q)$ represents the satisfaction resulting from a departure with a q proportion of charge.

Finally, to aggregate these sub-objectives, desirability functions *desirability functions* d_{cost} and d_{sat} are used to scale f_{cost} and f_{sat} to the interval $[0,1]$ (see [11, 17]). The aggregated performance is then the average of these two scaled values,

$$f = \frac{d_{\text{cost}}(f_{\text{cost}}) + d_{\text{sat}}(f_{\text{sat}})}{2}. \tag{6}$$

The scenarios were constructed using the process described in Figure 3, and the controllers used the process described in Figure 4 to determine $I_{\texttt{EV},j}^{\text{target}}$ and $I_{\texttt{B},j}^{\text{target}}$. From these descriptions, we see that the controllers and scenarios have parameter-based representations given by

$$x = (q_{\texttt{EV}}, q_{\texttt{B}}, a_1, \ldots, a_{12}, b_1, \ldots, b_{12}),$$
$$y = (i, s_1, \ldots, s_7, \ell_1, \ldots, \ell_7, p_1, \ldots, p_7).$$

Thus, we obtain a scenario optimisation problem with solution domain $\mathcal{X}$ consisting of all such x (with constraints outlined in Figure 4) and scenario domain $\mathcal{Y}$ consisting of all such y (with constraints outlined in Figure 3). The payoff $f_1(x, y)$ of solution x in the case of scenario y is given by (6). As a proxy for average case performance, we randomly sampled 10 benchmark scenarios $\hat{y}_1, \ldots, \hat{y}_{10} \in \mathcal{Y}$ and set $f_2(x) = \frac{1}{10} \sum_{i \in [10]} f_1(x, y)$ (the same set of benchmark scenarios were used for all runs). Accordingly, we associate evaluation costs of $c_1 = 1$ and $c_2 = 10$ with f_1 and f_2 respectively.

Scenario Description

Parameters: Dataset index $i \in [300]$; trip start times $s_1, \ldots, s_7 \in [7.0, 21.0]$; trip lengths $\ell_1, \ldots, \ell_7 \in [1.0, 5.0]$; weekday trip probabilities $p_1, \ldots, p_5 \in [0.5, 1.0]$; weekday trip probabilities $p_6, p_7 \in [0.0, 1.0]$.

Process:

- Set the flow of PV according to the i^{th} entry of the real-world solar energy production dataset.
- Set the flow of HH according to the i^{th} entry of the real-world household energy usage dataset.
- For every day in the year, if it is the j^{th} day of the week, then, with probability p_j, create an EV trip of length ℓ_j starting at time s_j which uses a fixed amount of energy (30 kWh).

Fig. 3: Method for constructing scenarios from a set of parameters. Note that while the construction appears random, we use a fixed random seed to ensure that the same set of parameters always produces the same scenario.

Controller Description

Parameters: EV charge threshold $q_{\text{EV}} \in [0.0, 1.0]$; EV charge rates $a_1, \ldots, a_{12} \in [0.0, 1.0]$; B charge threshold $q_{\text{B}} \in [0.0, 1.0]$; B charge rates $b_1, \ldots, b_{12} \in [-1.0, 1.0]$.

Process:

- At the given time step j, perform the following three tests about the state of the system.
  - **Test 1:** Is $-I_{\text{PV},j} > I_{\text{HH},j}$?
  - **Test 2:** Is $Q_{\text{B},j} < q_{\text{B}} \cdot \overline{Q}_{\text{B}}$?
  - **Test 3:** Is EV absent from the system? If not, is $Q_{\text{EV},j} < q_{\text{EV}} \cdot \overline{Q}_{\text{EV}}$?

  Map the results of these tests onto a unique number k between 1 and 12 (noting that Test 3 has three possible outcomes, making $2 \times 2 \times 3$ possible outcomes in total).
- Set $I^{\text{target}}_{\text{EV},j} = a_k \cdot \overline{I}^{\text{in}}_{\text{EV}}$ and $I^{\text{target}}_{\text{B},j} = b_k \cdot \overline{I}^{\text{in}}_{\text{B}}$.

Fig. 4: The process used by a controller to set $I^{\text{target}}_{\text{EV},j}$ and $I^{\text{target}}_{\text{B},j}$.

B. Results

The following method was used to compare the performance of k algorithms $\mathcal{A}_1, \ldots, \mathcal{A}_k$.

1) Each algorithm $\mathcal{A}_i$ is run for a budget of up to T_{max} function evaluations. The final predator population and prey population before the budget was reached is recorded as P_i and Q_i respectively.
2) We define $\overline{Q} = \cup_{i \in [k]} Q_i$, so that $\overline{Q}$ is the accumulated set of preys from across all k algorithms.
3) For each $i \in [k]$, we calculate the Pareto front of the function $F_i : P_i \to \mathbb{R}^2$ given by

$$F_i(x) = (\min_{y \in \overline{Q}} f_1(x, y), f_2(x)). \tag{7}$$

Minimising over accumulated set of preys $\overline{Q}$ is essential in (7), as computing the worst-case performance for a predator generated by $\mathcal{A}_i$ by minimising over only Q_i would effectively penalise algorithms which identified more difficult cases in $\mathcal{Y}$.

name	parameters	solution concept
C-AMCA	$\mu = 50$, $\lambda = 50$	AC-WC
$(\mu + \lambda)$ EA	$\lambda = 3000$, $\mu = 30$	AC
PDCoEA	$\lambda = 5000$	WC

TABLE II: Algorithms used for comparison.

300 independent runs of this process with $T_{\text{max}} = 10^7$ were performed for the three algorithms listed in Table II. $(\mu+\lambda)$ EA [8] was chosen as a benchmark for the best average performing solution and PDCoEA [13, 14] was chosen as a benchmark for the best worst-case performing solution (which form the extremes of the AC-WC curve). Figure 5 shows averages of the 300 resulting Pareto fronts. As expected, the solutions of the $(\mu + \lambda)$ EA show strong average performance but poor worst-case performance, whereas the reverse is true for those of PDCoEA. On the other hand, the Pareto front for C-AMCA has a range of average and worst-case performances spanning between the two extremes, demonstrating that C-AMCA can find solutions for this problem which offer strong improvement in either average or worst-case performance at the cost of only a small amount of the other.

The average hypervolume bounded by the AC-WC curve of C-AMCA was 0.45234, whereas the combined hypervolume bounded by $(\mu + \lambda)$ EA and PDCoEA together was 0.45121. A Wilcoxon signed-rank test established this difference to be statistically significant with a p-value of 0.0092.

V. Conclusion

We presented AMCA, an algorithm for finding solutions to scenario optimisation problems which are Pareto optimal with respect to metrics of average and worst-case performance. AMCA is the first algorithm for this setting which uses evolutionary computing. The design incorporates both competitive coevolutionary and multi-objective methods to make it suitable for a problem setting that is both robust an multi-objective.

We also presented a cheaper version of AMCA, which was tested using an established energy management problem. The results gave initial confirmation that our algorithm can discover solutions that exhibit average and worst-case performance in a range of combinations varying between the extremes produced by the benchmarking algorithms.

Future work will involve experimental analysis of the performance of AMCA on a wider range of scenario optimisation problems. Another topic for future research is the extension of AMCA to problem settings with alternative objectives in place of (or in addition to) average performance.

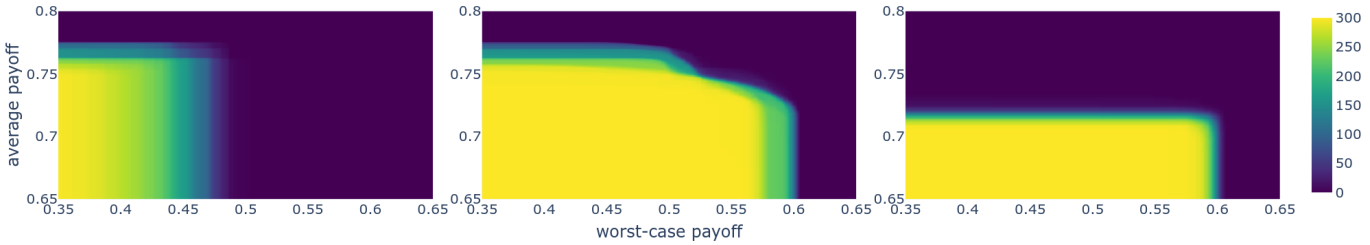

Fig. 5: Heatmaps representing the average Pareto front across the 300 runs of $(\mu + \lambda)$ EA (left), C-AMCA (centre), and PDCoEA (right). The value assigned to each point in the objective space is the number of runs in which that point was contained in the hypervolume of the resulting Pareto front.

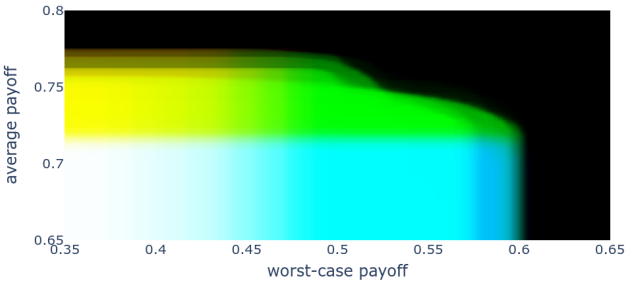

Fig. 6: Overlay of the heatmaps in Figure 5 using RGB channels corresponding to the three algorithms.

Acknowledgements

Benford and Lehre were supported by a Turing AI Fellowship (EPSRC grant ref EP/V025562/1). The computations were performed using the University of Birmingham's BlueBEAR HPC service. See http://www.birmingham.ac.uk/bear for more details.

References

[1] T. Bäck. Self-adaptation in genetic algorithms. *Proceedings of the first European conderence on artificial life*, pages 263–271, 1992.

[2] D. Bertsimas, D. B. Brown, and C. Caramanis. Theory and applications of robust optimization. *SIAM Review*, 53(3):464–501, 2011.

[3] D. Bertsimas and M. Sim. The price of robustness. *Operations Research*, 52(1):35–53, 2004.

[4] A. Chassein and M. Goerigk. A bicriteria approach to robust optimization. *Computers & Operations Research*, 66:181–189, 2016.

[5] D.-C. Dang, A. Eremeev, and P. K. Lehre. Escaping local optima with non-elitist evolutionary algorithms. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(14):12275–12283, 2021.

[6] K. Deb. *Multi-objective optimization using evolutionary algorithms*. Wiley, 2001.

[7] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[8] B. Doerr and F. Neumann. *Theory of Evolutionary Computation*. Springer, 2020.

[9] G. D'Angelo and F. Palmieri. A co-evolutionary genetic algorithm for robust and balanced controller placement in software-defined networks. *Journal of Network and Computer Applications*, 212:103583, 2023.

[10] S. Garatti and M. C. Campi. Modulating robustness in control design: Principles and algorithms. *IEEE Control Systems Magazine*, 33(2):36–51, 2013.

[11] J. Harrington. The desirability function. *Ind. Quality Contr.*, 21(10):494–498, 1965.

[12] M. Lanctot, J. Schultz, N. Burch, M. O. Smith, D. Hennes, T. Anthony, and J. Perolat. Population-based evaluation in repeated rock-paper-scissors as a benchmark for multiagent reinforcement learning. *Transactions on Machine Learning Research*, 2023.

[13] P. K. Lehre. Runtime analysis of competitive co-evolutionary algorithms for maximin optimisation of a bilinear function. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '22, page 1408–1416, 2022.

[14] P. K. Lehre, M. Hevia Fajardo, J. Toutouh, E. Hemberg, and U.-M. O'Reilly. Analysis of a pairwise dominance coevolutionary algorithm and defendit. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '23, page 1027–1035, 2023.

[15] M. Li, F. Guimarães, and D. A. Lowther. A competitive co-evolutionary algorithm for constrained robust design. In *9th IET International Conference on Computation in Electromagnetics (CEM 2014)*, pages 1–2, 2014.

[16] H. Markowitz. *Mean-Variance Analysis in Portfolio Choice and Capital Markets.* Basil Blackwell, 1987.

[17] M. B. Mushtaq and T. Rodemann. Adversarial optimization approach for development of robust controllers. In *Applications of Evolutionary Computation*, pages 384–399, 2020.