A Reproducible Unified Framework for Sequential Decision Problems

Aman Soni, Peter R. Lewis, Anikó Ekárt, Christopher Buckingham

Abstract—Sequential decision problems require new decisions in response to an environment that changes over time. Current decisions may change the value, or available choices, of future decisions. Algorithms from reinforcement learning, evolutionary dynamic optimisation, or hybrids, tackle forms of sequential decision problems. Our key contribution is a unified framework for sequential decision problems that allows algorithms from reinforcement learning and evolutionary dynamic optimisation to be applied to problems from both approaches. To overcome We demonstrate these techniques on both canonical benchmarks and the state of the art Open AI gym framework [1]. We contribute a technique that ensures experimental results are easily reproduced. We extend previous work on a Q-learning Based Evolutionary Algorithm to evaluate relative algorithm performance on continuous control problems where failure states have a high cost, or it is not possible to restart the problem instance. Unlike episodic problems, algorithms cannot use multiple training episodes to improve performance. We introduce the Double Mountain Car Problem as a continuous control version of the Mountain Car Problem. This work demonstrates a mechanism for Evolutionary Methods to utilise and extend the gym benchmarking framework to compare their research against methods from reinforcement learning using techniques that ensure experimental results are easily reproduced.

1 Introduction

Sequential Decision Problems (SDPs) require ongoing decisions for a problem environment that changes over time. The aim is to maximise the solution quality for the **complete sequence of decisions**. Solution quality encompasses *accumulated reward* from Reinforcement Learning (RL) and *fitness* from Evolutionary Dynamic Optimisation (EDO). This may not be the same as maximising the solution quality at each time step. The current decision may affect the available choices for future decisions. This effect is referred to as *time-linkage* between environmental states. An Evolutionary Computation (EC) approach to SDPs is to frame the problem as a Dynamic Optimisation Problem (DOP). DOPs, often tackled using Evolutionary Dynamic Optimisation (EDO) algorithms, are usually seen as tracking moving optima problems [2]. On the other hand, Reinforcement Learning (RL) commonly defines the Reinforcement Learning Problem (RLP) as Markov Decision Processes (MDPs) [3]. Fu et al. [4] proposed a definition of DOPs inspired by the idea of *multiple decision-making* in the RL community. The definition, outlined here in Section 3, draws a connection between the different types of SDPs studied in EDO and RL. This paper formalises this connection in a *Unified Framework* for SDPs.

The benefits of the SDP Framework are threefold. Firstly, it creates an opportunity to cross-pollinate ideas across well-established research. Secondly, it allows the different types of SDPs studied in EDO and RL to be explicitly categorised. Thirdly, it allows algorithms from RL and EDO to be applied to RLPs or DOPs. The problem definitions for RLPs and DOPs are analysed for equivalences between the different problem class instances. A software contribution is to create the Conceptual Moving-Peaks Benchmark (CMPB) [5] gym environment. Systematically study of algorithms used on the different types of SDPs shows that EDO and RL algorithms specialise in certain types of SDPs and can benefit each other in solving SDPs. Experiments are available online using Jupyter Notebooks and for download in our code repository. We propose and validate a method using virtualization to guarantee reproducible results.

There have been a number of recent significant advances in RL, particularly with game playing [6]. This has resulted in an explosion of work to create standardised benchmarking frameworks. Current state of the art is the OpenAI Gym [7], a widely adopted real time software platform for benchmarking RL algorithms. We support the SDP framework by extending Open AI Gym with the CMPB problem environment thus demonstrating that RL algorithms can be used for EC problems. An immediate benefit of the SDP Framework has been to highlight that the recent work in the RL community focuses primarily on *episodic-learning* tasks. This can be related to a body of work in EC on evolving controllers (e.g. neuro-evolution). In this paper we study the problem type classified as *continuous control* problems in RL, or online EDO in EC where **restarts may mean the loss of deployed hardware**, a costly option preferably avoided. For a further contribution, we compare the performance of algorithms from both approaches, and hybrids, on new EDO and other selected OpenAI Gym environments. We adopt a method, outlined in Section 3.5 to ensure reproducibility.

This paper is structured as follows. Section 2 provides background on RL and EDO. Section 3 outlines the SDP framework. Section 3.5 details benchmarks, experimental design, results and analysis. Section 6 concludes the paper and outlines future work.

• *A. Soni, P.R. Lewis, C.D. Buckingham and A. Ekárt are with the Aston Lab for Intelligent Collectives Engineering (ALICE), Department of Computer Science, Aston University, Birmingham, UK, B4 7ET.
E-mail: see https://alice.aston.ac.uk/*

• *Source code available at https://github.com/aliceresearch/gym-sdp*

Manuscript received April 13, 2019.

2 Background

2.1 Evolutionary Computing

Optimization has been long been studied using Evolutionary Algorithms (EAs) [8]. Generally speaking, optimization problems can be divided into two categories. One is Static Optimization Problems (SOPs), and the other is Dynamic Optimization Problems (DOPs). We can think of a SOP as a one-decision-making problem, where only one solution, i.e., one decision, is determined for a SOP. In contrast, there has not been a consensus about the definition of DOPs studied in the EDO community.

Definition 1. Given a fitness function f, which is a mapping from some set A, i.e., a solution space, to the real numbers $R : A \approx R$, a SOP is to find a solution, i.e., making a decision, $x \in A$ such that for all $x \in A$, $f(x_*) \geq f(x)$.

While there is an unambiguous definition for a SOP, a rigorous definition for DOPs is missing.

2.1.1 Dynamic Optimisation Problems

Historically, many definitions about DOPs have been proposed in EDO. Applying genetic algorithms to DOPs was initially studied by Goldberg and Smith [9]. While no explicit definition of DOPs was provided, and yet the authors studied the performance (best-of-generation) of a genetic algorithm with dominance and diploidy on a dynamic 0-1 knapsack problem [10]. Similarly, Branke studied the performance (off-line performance) of memory-based EAs on the moving peaks benchmark [11], without explicitly defining DOPs. In some other works [12] [13] [14], DOPs were simply defined as a sequence of SOPs over time, in which the goal for each SOP is to find a solution maximising the fitness function of that SOP. Hence, the performance of algorithms for such DOPs was measured by the average performance on each SOP during a considered time interval. Another type of definition of DOPs can be found in [15], in which DOPs were considered as problems of maximising an integration of a Dynamic Fitness Function (DFF) over a time period by determining a solution at each time point. Nguyen [16] also defined DOPs as maximising such an integral, but with a more flexible definition framework than Bosman [15]. Performance is in terms of quantifying how the DFF changes, or how previously determined solutions influence the dynamic, etc.

There also exist some descriptions about what DOPs should look like. Examples are optimization problems are considered dynamic only if the EA has to cope with the dynamic by Branke [17], If any of those uncertainties related to optimization problems are to be taken into account, we call the problem dynamic by Jin et al [18], DOPs are optimization problems which must be solved as time goes by by Bosman [15], and DOPs are a special class of dynamic problems which are solved online by an optimisation algorithm as time goes by by Nguyen et al [2]. The aforementioned four types of definitions of DOPs are limited in different ways. The first type does not define DOPs explicitly but considers DOPs as problems of adapting a solution to a changing fitness landscape using EAs. This largely restricts the possible domain of DOPs, as it unnecessarily assumes that DOPs should be solved by EAs. The second type considers DOPs as a sequence of SOPs over time, and it is assumed that the DOP is solved optimally if and only if each of the SOPs is solved optimally. This formulation is certainly true in some real-world scenarios but not in others. For example, solutions determined for previous SOPs can have an impact on what future SOPs look like, as argued by Bosman [15]. Hence, in such cases, the overall performance can be sub optimal even though an optimal solution is found for each SOP. The third type defines DOPs as maximizing an integration of a DFF over a time period. Yet, it is not straightforward as how the decision maker would provide solutions to such DOPs. With regard to the fourth type, descriptive text, the obvious drawback is that natural language descriptions are not rigorous as they are open to having the description interpreted differently.

2.2 Reinforcement Learning

RL is a local search method inspired by the psychological theory of learning through trial-and-error [3]. The learning agent tries different actions which may change the state of the environment. The agent learns the value of the action through a reward signal. In the most realistic cases, the learners' actions affect not only the immediate value of the reward but may also determine subsequent environment states, and the actions available for these subsequent states. These two characteristics, trial-and-error search and delayed reward, are the two most important distinguishing features of reinforcement learning.

Reinforcement Learning Problems (RLPs) are often defined as Markov Decision Processes (MDPs) [3]. The problem environment is represented as a set of variables referred to as the state. State variables may be continuous or discrete. If only a subset of the set of state variables is available to the agent, the problem is often modelled as a Partially Observable MDP (POMDP) [3].

The actions available to the agent at each decision point is called the decision or action space. The action set usually remains constant for a learning task. However, outcomes may be dependant on the state of the environment e.g. a agent in a maze may not be able to move forward if there is a wall in that position.

2.2.1 The Reinforcement Learning Problem

Sutton and Barto [3] define the general form of the RLP as a problem where an agent, for a sequence of discrete time steps, t, at each t evaluates its representation of the environment's state, $s_t \in \mathbb{S}$, where $\mathbb{S}$ is the set of all possible states, and selects an action to perform, $a_t \in \mathbb{A}(s_t)$, where $\mathbb{A}(s_t)$ is the set of available actions for the state s_t. As a consequence of the action, a_t, the environment moves to a new state, s_{t+1} and provides a numerical reward to the agent for the next time step, $r_{t+1} \in \mathbb{R}$. The agent builds a *policy* π for each t, where $\pi_t(s, a)$ is the probability that the selected action $a_t = a$ if $s_t = s$. The solution is a policy π that maximises[1] the total sum of expected rewards. If R denotes the reward function, an RLP can be defined as:

$$max \sum_{t=0}^{t_e} R(\arg\max[\pi_t(s_t, a_t)], s_t). \tag{1}$$

1. Maximization problems are considered without a loss of generality.

3 A Unified Framework for Sequential Decision Problems

Based on Fu et al. [4], we define a DOP as: *Given an optimization problem F, an optimization algorithm G to solve F, and an optimization period $[t_0, t_e]$, F is a DOP if during $[t_0, t_e]$ the underlying fitness landscape changes and G has to react by providing new solutions.* The function E calculates the estimated reward based on f_t at each time step. When the state of the environment at time t is a set of variables s_t, and the action taken at time t is a_t, a DOP can be defined[2] as:

$$max \sum_{t=0}^{t_e} E(f_t(s_t, a_t)). \tag{2}$$

Definition 2 captures the distinctive feature of DOPs, in comparison to Static Optimization Problems (SOPs), which is multiple-decision-making over time. Further support for the perspective that DOPs and RLPs are subsets of a broader class of SDPs [4] is found in the striking similarity for the problem definitions Definition 1 and Definition 2.

DOPs generally assume a computational model of the fitness function. We draw a parallel between fitness in DOPs and reward in RLPs. However, execution of the fitness function will not change the environment state. Conversely, for RLPs the value of the reward is determined by the environment and returned as a state variable when the environment. For both problems, the aim is to maximise the accumulated sum of the returned value over an interval $[t_0, t_e]$. Another similarity is that, at at every at time step t, EDO algorithms compare individuals in a population set, which is a collection of different state-actions pairs, while RL algorithms select from a policy-value set, which is a collection of different state-actions pairs.

For both RLPs (Definition 1) and DOPs (Definition 2) the value of making a decision is determined by the effect of the decision on the current state of the environment. RLPs assume that the reward signal is determined by the environment and returned as a state variable when the environment. Conversely, DOPs generally assume that there is a computational model of the fitness function. A further assumption is that executing the computational model will not change the environment state.

For both problem definitions, the aim is to maximise the accumulated sum of the returned value over an interval $[t_0, t_e]$. Another similarity is that, at at every at time step t, EDO algorithms compare individuals in a population set that is a collection of different state-actions, while RL algorithms select from a policy-value set that represent different state-actions.

For some problems the dynamics of state is linked to time [2], [3], [4]. A difference between the definitions is that DOP (Definition 2) accumulates the maximum reward at every t while RLP (Definition 1) maximises the accumulated sum of discounted future rewards. The second difference is that the state representation in DOP (Definition 2) is a part of the problem specification while RLP (Definition 1) tracks the received rewards using state-action pairs (s_t, a_t) in the learning policy π. This implies that DOP (Definition 2) does not cater for time-linkage between states, while the RLP (Definition 1) tracks state transitions. The difference is further supported in that time-linkage is a term used by reinforcement learning, while there is no common term used for this effect in EDO.

2. Notation has been adjusted to aid comparison to RLP (Definition 1).

3.1 DOPs as SDPs

DOPs are the online extension of EC methods [8]. EC methods assume some observability of environment state that is usually a part of the problem statement. EC methods cope well with large, continuous, constrained or sparse action-state sets [19]. Once a solution has been found is it applied to the problem instance. DOPs, an extension of EC methods, assume a simulation environment that includes a computational model of the fitness function. Solutions for DOPs are generated offline and applied to the problem instance. The assumption is that fitness evaluations are run without changing the environment state [8]. A common assumption is that there is the ability to perform fitness evaluations without implementing an action.

Typically DOPs disregard the time taken to generate a solution. DOP methods consider only the time spent interacting with the environment to apply the solution. This assumption of infinite computing resources for offline processing leads to inaccurate comparisons for algorithms that act only online. We introduced a method to compare relative algorithm performance for methods from EDO and RL [20].

3.2 RLPs as SDPs

Reinforcement learning assumes some observability of state where at the least a reward signal, generated by the environment on change of state, is available. There is an assumption of computational resources for the agent's internal data structure, and action selection methods. In general, all agents methods are are assumed to be online so that when the agent performs no operation the environment continues to change over time [3]. There are also many benchmarks developed in RL [10], which can be seen as DOP benchmarks according to our definition of DOPs. Benchmarks in RL have some common features that may favour only RL methods, such as a discrete action space, no evaluation model of the reward function, and full observability of states.

3.3 Assumptions about SDPs

SDPs assume a union of information and resources from RLPs and DOPs. The availability of these resources and information favour different approaches. SDPs seeks to calculate the optimal set of decisions rather than the set of optimal decisions.

While the problem of defining the difficulty of a problem is difficult, knowing whether an optimal can be calculated is a useful problem characteristic that seperate problem classes. We call benchmarks where the optimal can be calculated a canonical benchmark.

3.4 Naming the Problem Type

Russell and Norvig [21] define a decision-theoretic agent as one that makes a rational decision on an appropriate action to meet its goals. They call these one-shot or episodic decision problems to avoid confusion with decision problems from computational theory. We have seen several references to a singular decision as a decision-making problem. However, more complex problems, that require a sequence of decisions over time, are named Sequential Decision Problems (SDPs). We have previously referred to these as Sequential Decision-Making Problems (SDMPs) [20].

The aim of benchmarks is to provide a means of predicting the real-world performance of algorithms and as such are used for two different ways [22]. The first is to understand the behaviour of algorithms to determine the relative strengths on problem instances with particular characteristics. The second is to test relative algorithm performance on a benchmark problem instance with particular features. It is widely accepted that different algorithms will outperform others only on certain problem instances [23]. For a meaningful comparison of algorithm performance we need a wide range of problem instances, or the ability to tune the benchmark to change the problem instance characteristics, or to be able to restrict the resources available for solving the problem instances.

Sutton and Barto [3] define the environment as everything that is outside of the learning agent. For benchmarks, the environment is essentially a set of variables, the values for which represent the environment state at any given time. Agents can perform a set of actions. We call this the action space, and the range of reward signal returned by the environment the reward space. In addition, some problems are simple enough that we can calculate accurate solutions. In this section we outline these aspects of benchmark problems in more detail.

3.4.1 Continuous Control vs Episodic Learning

While Open AI labels the benchmarks problems as being of the continuous control type, their approach is to train algorithms on a number of episodes. We see these as episodic problems, which for game play, is a suitable approach as restarting a problem instance has a low cost. In many real-world situations failure states would have a high-cost in cases where there are expensive robots being used. As the majority of work recently published has been on episodic training, we choose to focus our efforts on continuous-control problems. This is important as algorithm performance can be compared on unseen problem instances.

In the Open AI Gym framework there is a repeated mention of continuous control. However, agents are trained across episodes of problem instances. Sutton and Barto [3] define learning tasks when the agent carries its learning across multiple runs, or episodes. Further, as there is done flag that signals the achievement of the goal, this is not a continuous-control problem. The greenhouse control problem where the process is ongoing is a good example of continuous control [24]. In the greenhouse control problem there is no completion state prior to the end of the problem instance. Training is a single episode that abandons learning from previous instances. Benchmarks for continuous-control may have failure states from which there is no recovery.

3.4.2 Optimal Solutions and Problem Complexity

When benchmarks are used to gain an understanding of algorithm behaviour it is useful to use problem instances that are simple enough so that changes in algorithm performance can be clearly linked to the changes in the problem definition, or tuning parameters of the algorithm. This allows algorithm designers to test whether changes in the algorithm cope better with a specific problem instance.

Some problem instances are simple enough that an optimal solution can be calculated. It is useful to be able to calculate the optimal solution as that allows us to measure the relative performance of different algorithms in relation to the optimal. In the real-world, it is far more common that an optimal solution cannot be calculated as there are unknown variables that effect the solution, or that the complexity of calculating an optimal is computationally expensive or intractable, making the calculation of an optimal unfeasible.

While the problem of defining the difficulty of a problem is difficult, knowing whether an optimal can be calculated is a useful characteristic of problem instance. Further, the optimal for a specific problem instance is a characteristic of a problem instance. We call benchmarks where the optimal can be calculated a canonical benchmark.

3.5 Challenges of Reproducible Experiments

The use of empirical comparison of the performance of algorithms on optimisation problems is extensive, with effective methods for comparison being discussed since inception [25]. For an excellent introduction to empirical research and experimental design see Cohen's book "Empirical Methods for Artificial Intelligence" [26]. Like Hooker [27], we use *scientific testing* to investigate **algorithmic behaviours** following a common sense list of dos and do not's [28] for experimentally analysing algorithms.

It is interesting that many, perhaps most, research papers focus on solution quality alone [29], ignoring efficiency, efficacy, robustness, complexity, impact, ability to generalise and innovation. In their classical work on reporting empirical results of heuristics Barr et al. [29] specify a loose experimental set up methodology and a guideline for reporting results to overcome the main challenges, list below, in creating reproducible research:

- clear stated method to create reproducible results [29] [30] [31]
- specify all influential factors (code, computing environment, etc) [29] [30]
- be precise regarding measures including parameter values [29]
- test against state-of-the-art [31]
- valid statistical inference [31]
- choose test instances that are relevant to objectives [30]
- results are provided without regard to research objectives [30]
- scope of generalised performance is generally too broad [30]

We have created an experimental architecture, outlined later in this section to address these concerns. The main benefit of this approach is to allow fellow researchers to easily extend and reuse the experimental platform.

3.6 The Action Space

The action, or decision, space is the range of actions available to the agent. Action space variables can be a combination of discrete and continuous values. A driving agent may use a discrete action control to indicate turning left while the level of force applied to the steering wheel may be a percentage of the total turn available, which is a continuous value. The range of actions can change during a task. In role-playing games a player may acquire new skills. This would increase the action set for the agent. Alternatively actions may become unavailable e.g. in a first-person shooter you not longer have ammunition for the machine-gun so cannot use this weapon until the agent acquires more ammunition of the correct type.

3.7 The State Space

The state space represents the environment state. This usually a set of variables that may be discrete or continuous. If all state variables are known then the state is fully-observable. Problems of this type are usually defined as a Markov Decision Process (MDP) [3]. The problem state may also be a Partially-Observable Markov Decision Process (POMDP) when not all the variables are available to the agent, or if the state contains noise [3]. There may be different level of noise with-in the environment. It is only for benchmark problems that it may be useful to specify state variables that are not noisy. In the real-world there will always be some level of sensing noise.

Reinforcement learning often stores values for the state-action space in the learning policies. This map is useful as it allows agents to track the value of being in a given state and taking the particular action.

Actions taken at the current state may affect available choices, or the value of the reward for future states. This is called time-linkage in RL [3]. Most real-world problems have some level of time-linkage between states.

3.7.1 The Dynamics of State

All state variables can be divided into two groups. The dynamic of the first group does not depend on previous actions, while the dynamic of the second group depends on previous actions. For benchmarks dynamics we there are 6 different dynamics suggested in CEC09 dynamic optimization competition benchmark generator [22] to update the state variables in the first group. The 6 dynamics are described as follows:

1) small step
2) large step
3) random
4) chaotic
5) recurrent
6) recurrent with noise

3.8 The Reward Space

Rewards can be continuous or discrete, and positive or negative. In RL, the reward is returned by the environment, and the agent usually does not have a mechanism to calculate the value of the reward for any given state, except by performing the action while in a given state. Current state-of-the-art RL work has been most successful with game playing [6]. With most computer games a restart of the problem instance is a low cost operation. However, in the real-world there may be a high cost to failure environments. You would not want to train your autonomous car that driving off a cliff has a high negative reward by driving it off a cliff. Negative rewards are often called regret with the aim of the problem to minimise the regret.

For the SDP framework we equate reward and fitness. DOPs use a fitness function to select the best solutions from the population of solutions that have been generated. Executing the fitness function does not change the environment state. This is in contrast to rewards from RL. Benchmarks for SDPs should provide a computational model of the reward function. The reard function model should execute without changing the environment state. The function would require the state-action pair as an input and return the reward. We believe that it is the role of the agent to track state transition information. Some benchmarks may force the agent to model the reward function for offline execution.

3.9 Creating a Repeatable Experiment Environment

The CERN Analysis Preservation and reuse framework relies on three pillars:

- Describe: adequately describe and structure the knowledge behind a physics analysis in view of its future reuse. Describe all the assets of an analysis and track data provenance. Ensure sufficient documentation and capture associated links.
- Capture: store information about the analysis input data, the analysis code and its dependencies, the runtime computational environment and the analysis workflow steps, and any other necessary dependencies in a trusted digital repository.
- Reuse: instantiate preserved analysis assets and computational workflows on the compute clouds to allow their validation or execution with new sets of parameters to test new hypotheses.

4 Environment Categories

Background: Why Gym? (2016)

Reinforcement learning (RL) is the subfield of machine learning concerned with decision making and motor control. It studies how an agent can learn how to achieve goals in a complex, uncertain environment. Its exciting for two reasons. Firstly, RL is very general. Second, RL algorithms have started to achieve good results in many difficult environments.

However, RL research is also slowed down by two factors. One, the need for better benchmarks. And two, lack of standardisation of environments, making it difficult to reproduce published research and compare results from different papers.

Gym is an attempt to fix both problems.

4.1 Evolutionary Environment Category

Existing environments are in categories including algorithmic, toy text, classic control, MuJoco physics simulator and robot control. Our work contributes an additional evolutionary environment category for benchmark problems from EDO.

Open AI Gym has a number of categories for the problem environments it contains. Our main contribution is to extend the categories to include select problems from evolutionary computing. This allows practitioners from evolutionary computing to test their algorithms against the existing state-of-the-art on the Gym benchmarking toolkit.

4.2 A Conceptual Moving-Peaks Benchmark

We generalise the Moving Peaks Benchmark [11] to define our benchmark. The concepts of height, width, and centre for the fitness function in the MPB are transferred. We add time-linkage between states using a bias that changes the future reward for the next decision when the current action chosen is less than 0. We call this new benchmark the Conceptual Moving Peaks Benchmark (CMPB).

There are two groups of things need to be specified in order to get a benchmark instance of CMPB. In the first group, the parameters regarding to the state space and the decision space need to be specified. The type of dynamic for the state needs to be specified as well.

In the second group, assumptions discussed earlier in this section need to be stated explicitly. It should be made clear whether the decision-maker is given the full analytic form of the reward function, the intermediate information of the reward function in which the decision-maker can pose questions about the immediate reward of any action without implementing such an action, or no information of the reward function where an action has to be implemented in order to get its immediate reward at a particular state. The assumption about the dynamic of state is determined once options in the first group are specified, e.g, whether the dynamic of state is Markovian.

Finally, it is necessary to specify to what extent the decision maker has information about the state s: s can be fully observable, partially observable in which, e.g., a white noise is added to s before it is passed to the decision maker, or non-observable.

4.2.1 The Action Space

An action is specified by a set of variables. For a state space that has the dimension of peak function being n, there are n variables to specify an action: $a = (a_1, a_2, \cdots, a_n)$. The action/solution space A_t for the state s_t is basically a subset of the n dimensional real numbers $\mathbb{R}^n$.

4.2.2 The State Space

A state is simply a set of variables in our benchmark. Each state consists of four parts, namely height, width, centre, and bias. Each height, width, and centre are associated with a peak function, whose meaning will be explained later in the reward function. The bias is just a single scalar. There are two parameters to control the dimension of state. One is called the number of peaks m, and the other is the dimension of peak n. Simply put, a state s is represented using the vector $(h_1, h_2, \cdots, h_m, w_1, w_2, \cdots, w_m, c_{i1}, ci2, \cdots, c_{mn}, b)$ of length $(n+2)m+1$, where h_i and w_i denote the height and the width of the ith peak function. c_i $c_i = (c_{i1}, c_{i2}, \cdots, c_{in})$ is the centre of the ith peak function, and b is the bias. Time-linkage is a part of the state space.

We divide all state variables in s into two groups. The dynamic of the first group does not depend on previous actions, while the dynamic of the second group depends on previous actions. The first group consists of all variables except for the bias b in s. In the CMPB [5], the dynamic of the environment variable c_t is $(c_0 = 5)$:

$$c_t = c_{t-1} \times -1 \tag{3}$$

Therefore c_t oscillates between c_0 and $-c_0$ and the environment returns only two states. We leave relating the other dynamics in benchmark problem environments for future work.

4.2.3 The Reward Space

The reward function takes a state and an action as inputs and outputs a real number. The reward function in our benchmark consists of the bias b and several peak functions $(h_i w_i || x c_i || 2$ is the ith peak function). Where s represents a state $s = (h_1, h_2, \cdots, h_m, w_1, w_2, \cdots, w_m, c_{i1}, c_{i2}, \cdots, c_{mn}, b)$, and a is an action. For all benchmark instances of the CMPB, the reward function at time step t is:

$$f_t(s_t, a_t) = 30 - 2|a_t - c_t| + b_t, \tag{4}$$

where s_t represents the state at time step t: $s_t = (c_t, b_t)$. a_t belongs to the interval $[-10, 10]$ and represents the decision at time step t. c_i denotes a vector $(c_{i1}, c_{i2}, \cdots, c_{in})$, and —— —— is the Euclidean norm. Without loss of generality, we require $w_i > 0(1im)$.

The dynamic of the bias b_t is as follows $(b_0 = \theta_b)$:

$$b_t = \begin{cases} \theta_b & \text{if } a_{t-1} \geq 0, \\ -\theta_b & \text{otherwise,} \end{cases} \tag{5}$$

where θ_b is a parameter, which controls the influence of a_{t-1} on b_t, with larger values being more influential on the effect of time-linkage. In the first benchmark instance, θ_b is set to 100, and in the second benchmark instance, θ_b is set to 15.

4.3 A Generalised m-Armed n-Power Bandit Benchmark

We generalize a bandit problem [3] to define an SDP benchmark. The concept of m-Arms is transferred and we add time-linkage between states in the problem a configurable parameter *m*. We add time-linkage between states using a bias that changes the future reward for the next decision when the current action chosen is less than 0. We call this new benchmark the Generalised m-Armed n-Power Bandit (GAP) problem.

There are 2 input parameters that need to be specified in order to get a benchmark instance of GAP. The first is *n* for the number of arms and the second is a value for *m* the time-linkage between states. The variable *m* is used to a bonus for the state. If the current bonus is 0 and the position of the selected arm, the action *a*, is a power of *m*, then the bonus for the next time step is equal to the reward returned for action a.

Finally, it is necessary to specify to what extent the decision maker has information about the state s: s can be fully observable, partially observable in which, e.g., a white noise is added to s before it is passed to the decision maker, or non-observable.

4.3.1 *The State Space*

A state is a set of variables $\mathbb{S}$ with each state s_t represented using the vector $(b_1, b_2, \cdots, b_m, p)$ of length $m+1$, where b_i denotes the position of the *i*th arm, and the value of p_t is 0 if $p_{t-1} \geq 0$, otherwise $p_t = i$ if $i \in \mathbb{I}$ where $\mathbb{I} = (i^1, i^2, \cdots, i^n)$.

4.3.2 *The Action Space*

For a state space that has m arms the action space $\mathbb{A} = a_1, a_2, \cdots, a_m$, there are m variables to specify an action a where $ain\mathbb{A}$ and $\mathbb{A} \subset \mathbb{Z}$

4.3.3 *The Reward Space*

The reward function at time step $t+1$ takes a state s_t and an action a_t as inputs and outputs a real number. The reward function consists of the position of the arm and the bonus set at $t-1$ from action a_{t-1}. For all benchmark instances of the GAP, the reward function at time step t is:

$$f_t(s_t, a_t) = i + p_t, \tag{6}$$

where s_t represents the state at time step t: $s_t = (p_t)$. a_t belongs to the interval $[0, 5]$ and represents the decision at time step t. i denotes a scalar value for the position of arm b where $\mathbb{B} = (b_0, b_1, \cdots, b_n)$. Without loss of generality, we maximise $i + p_t$ where $p_t = i_{t-1}$.

4.4 Double Mountain Car

Open AI Gym holds a number of classic control environments which include the Mountain Car Problem (MCP) [32]. The aim of the problem is to get a car to climb a mountain. However, the car cannot directly reach the goal as it is underpowered and cannot overcome gravity to climb the hill without building sufficient momentum. The task is considered complete when the car reaches the top of the mountain. The agent is randomly placed between -0.6 and -0.4 at the start of each problem instance. The goal is to solve test that the agents learns to explore the full environment space. The agent receives a reward of -1 at all position except the goal position of 0.5. At the goal position, the reward is 0.

Except for terminal failure states, continuous-control problems should run for the duration of the task. We introduce a new continuous-control version of the MCP, the Double Mountain Car Problem (DMCP), by mirroring the state space around the goal position as shown in Figure 5. Characteristic of RL benchmark problems both state variables, current position and velocity, provide time-linkage between states.

DMCP does not have a goal position. The agent receives a negative reward based on the car's distance to the top of the hill, except a the top of the hill, where the reward is 0. The agent has two different types of learning this problem. The first is as with the MCP, to learn to use momentum to reach the top of the goal hill. The second is to learn to balance itself at the top of the hill. The agent can only begin to learn to balance once it first reaches the top of the hill and finds that the reward returned is greater that previous positions encountered.

Agents that do not carry learning across episodes are unable to solve this problem in 200 time steps, the standard number of steps assigned to each classic control environment in the Gym tool-kit. Based on the Open AI Baselines algorithms, we used $100,000$ steps for each episode.

4.4.1 *The Action Space*

At each time step an agent has the choice to push or not push the car. If the agent decides to push, then the agent must choose whether to push to the left or the right. The action space is the set of variables $A = ['pushleft', 'nopush', 'pushright']$ which have numeric values of $(A) = [0, 1, 2]$ in the Open AI Gym environment.

4.4.2 *The State Space*

There are two state variables, the current position of the car, and it's velocity. The position is in the range $p = [-1.2, 2.2]$ where the goal state is 0.5. The velocity of the car is in the range $v = [-0.07, 0.07]$. When the velocity is positive the car is moving to the right, and when it is negative the car is moving to the left. Both state variables, current position and velocity, provide time-linkage between states. This is characteristic of RL benchmark problems

Agents that do not carry learning across episodes are unable to solve this problem in 200 time steps. Approaches from the literature include using a different reward function that is a based on the distance from the goal position. We use the distance from the goal to return a negative reward. At the goal position the reward is 0. The optimal agent would learn to hover at 0.5 for the duration of the problem.

4.4.3 *The Reward Space*

We borrow from the the literature a reward function that returns the negative of the difference between the goal position and the current position of the car, and subtracts the current velocity. When the car is at the top of the hill, $(p = 0.5)$ the reward is 0.

$$r_t = -10|goal - p_t| \tag{7}$$

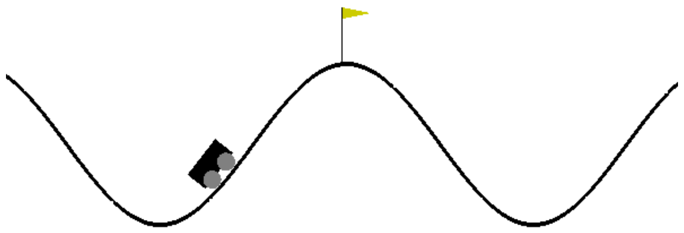
Fig. 1. Double Mountain Car - A continuous-control problem

5 Experiments and Analysis

TODO://Architecture Diagram/ Setup Guide

This subsection outlines selected algorithms from each community and discuss hybrid algorithms. We demonstrate the application of RL algorithms to EDO problems, then EDO algorithms to RL problems. We detail the experimental settings used to compare algorithm performance and present an analysis of the results. We are interested in supporting the SDP framework by demonstrating that algorithms can be interchanged.

The performance of EDO, Q-learning, and QBEA, together with the optimal accumulated rewards and a random method, on the first and second benchmark instances of CMPB are presented in Figure 2 and Figure 3 respectively. The performance of EDO, Q-learning, and QBEA, together with the optimal accumulated rewards and a random method, on the benchmark instances of the MCP and DMCP are presented in Figure 4 and Figure 5 respectively.

5.1 Experimental Set Up

For all benchmark instances, experiments were averaged over 100 runs for 2000 time steps. We generated a set of seeds. The same seeds were then used for repeated runs. The standard error is shown by the error bars. Where there is no variance of the standard error, the error bar is shown only once.

5.2 Representative Algorithms

5.2.1 Reinforcement Learning

Q-learning [33] (Algorithm 1) is a representative RL algorithm [3]. The internal learning policy is stored in a state-action matrix called the Q-values. An off-policy implementation [3] means that the policy update on line 7 uses the reward for the best known action at new state s_{t+1} rather that the selected action performed on line 5 to improve the tracking of time-linkage between states. While several RL methods [6], [34] require offline time, Q-learning (Algorithm 1) interacts directly with the environment in an online manner with no offline time requirements.

Algorithm 1 Representative off-policy RL algorithm

```
Require: discount factor, learning parameter
 1: INITIALIZE Q-values
 2: for $t = 0 \rightarrow t_e$ do
 3:     OBSERVE state $s_t$
 4:     CHOOSE action $a_t$, $a_t \in \mathbb{A}(s_t)$ from Q-values
 5:     PERFORM action $a_t$;
 6:     OBSERVE state $s_{t+1}$ and RECEIVE reward $r_t$
 7:     UPDATE Q-values
 8: end for
```

5.2.2 Evolutoinary Dynamic Optimisation

In general, EDO assumes that solutions are generated offline before being applied to the problem instance [19]. An EDO approach for dynamic environments is hyper-mutation, which increases the diversity of the population of candidate solutions after a change has been detected [2].

A general form of an EA, with state awareness at each time step, is described in the pseudo-code of Algorithm 2, denoted as EDO hereafter. This algorithm would run at each time step of the SDMP. Intuitively this means several offline computational steps are required to make a decision at every time step in comparison to Algorithm 1. We adopt an approach to restart the algorithm at every time step with the fitness function changing to reflect the new state of the environment.

Algorithm 2 EDO algorithm

```
Require: evaluation function $f(s, a)$
 1: for $t = 0 \rightarrow t_e$ do
 2:     OBSERVE state $s_t$
 3:     INITIALISE population randomly
 4:     for all generations do
 5:         EVALUATE members of population
 6:         REPRODUCE members of population
 7:     end for
 8:     CHOOSE action $a_t$, $a_t \in \mathbb{A}(s_t)$ from population
 9:     PERFORM action $a_t$
10: end for
```

5.2.3 An RL and EDO Hybrid

QBEA [5], described in Algorithm 3, is a RL-EDO hybrid. QBEA uses Q-values similarly to Algorithm 1. In addition, QBEA stores state transition information so that it can estimate the next probable state. The EA on line 4 uses the estimated state to search the reward function at each decision point, updating the Q-values for each action evaluated.

Algorithm 3 Pseudo code of QBEA

```
Require: discount factor $\lambda$, learning parameter $\alpha$, evaluation function $f(s, a)$
 1: INITIALISE Q-values, state-action
 2: for $t = 0 \rightarrow t_e$ do
 3:     OBSERVE state $s_t$
 4:     SEARCH reward function $f(s_t, a_t)$
 5:     for each evaluated $a_i$ on $f(s_t, a_i)$ do
 6:         UPDATE Q-values for predicted state
 7:     end for
 8:     CHOOSE action $a_t$, $a_t \in \mathbb{A}(s_t)$ from Q-values
 9:     PERFORM action $a_t$
10:     OBSERVE state $s_{t+1}$ and RECEIVE reward $r_t$
11:     UPDATE Q-values
12:     UPDATE state-action
13: end for
```

The update operation in line 6 when running an EA is different from that in line 11. As in an EA, we have not observed the next state yet. The agent makes use of an internal state transition matrix to determine the next state based on the past probabilities.

5.3 Experimental Results and Analysis

The performance of EDO, Q-learning, and QBEA, together with the optimal accumulated rewards and a random

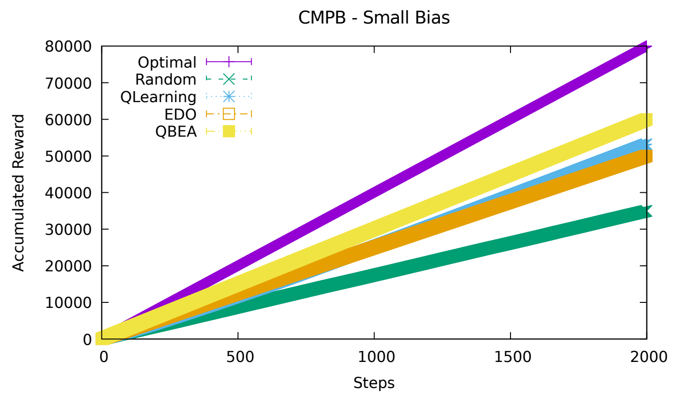

Fig. 2. The averaged accumulated rewards in Equation 4 over 100 runs with $\theta_b = 15$ for 2000 steps on the CMPB.

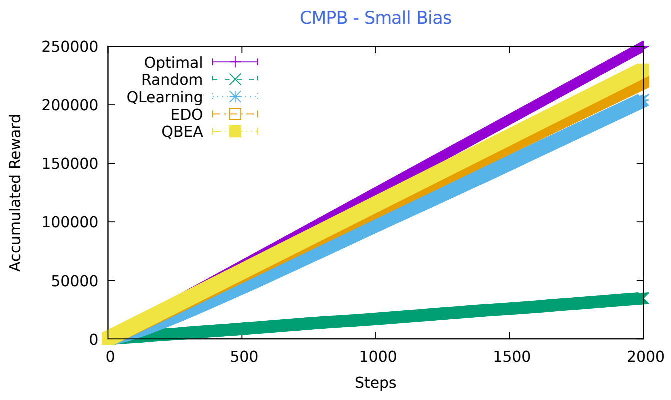

Fig. 3. The averaged accumulated rewards in Equation 4 over 100 runs with $\theta_b = 100$ for 2000 steps on the CMPB.

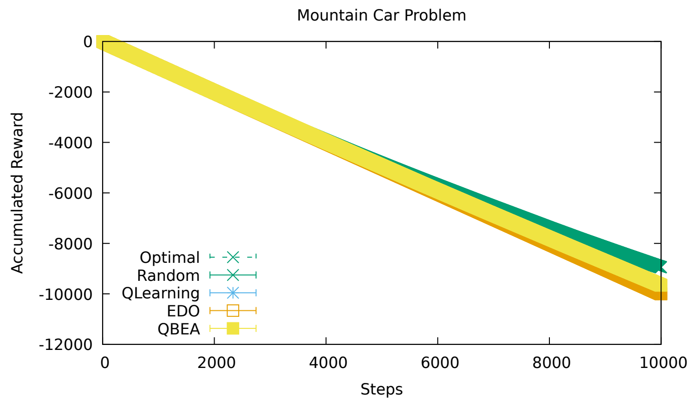

Fig. 4. The averaged accumulated rewards in Equation 4 for 10000 times steps over 100 runs on the Mountain Car Problem.

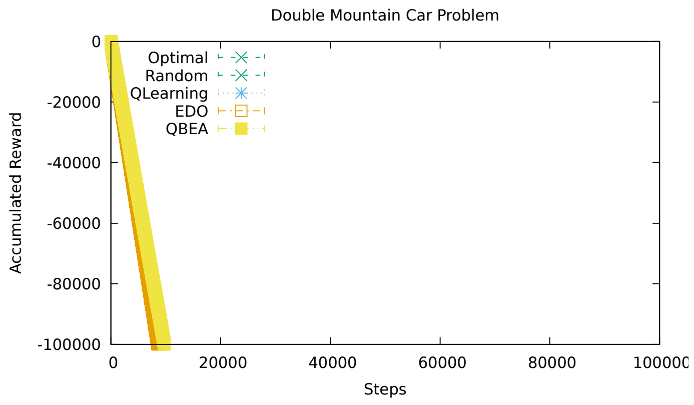

Fig. 5. The averaged accumulated rewards in Equation 4 for 10000 times steps over 100 runs on the Mountain Car Problem.

method, on the first and second benchmark instances of CMPB are presented in Figures. 2, 3 respectively.

For all benchmark instances, experiments were averaged over 100 runs for 2000 time steps. We generated a set of seeds. The same seeds were then used for repeated runs. The standard error is shown by the error bars. Where there is no variance of the standard error, the error bar is shown only once.

The performance of EDO, Q-learning, and QBEA, together with the optimal accumulated rewards and a random method, on the MCP and DMCP benchmark instances presented in Figures. 4, 5 respectively.

For all benchmark instances, experiments were averaged over 100 runs for 10000 time steps. We generated a set of seeds. The same seeds were then used for repeated runs. The standard error is shown by the error bars. Where there is no variance of the standard error, the error bar is shown only once.

5.3.1 *General Observations*

On the CMPB the results are in line with Fu et al. [5] and Soni et al. [20]. Q-learning achieves better performance than EDO on the first benchmark instances of the CMPB where time-linkage has a higher effect. EDO outperforms Q-learning on the second benchmark instances of the CMPB. QBEA consistently outperforms on all instances of the CMPB.

On the MCP and DMCP the results shown that there is only a small variance between the accumulated rewards across the methods. However, they do support our contribution for the SDP framework which is that EDO methods can be used for RLPs and RL methods can be used for DOPs.

6 CONCLUSIONS

In this paper we introduced the Unified SDP Framework. We demonstrated an implementation of the framework on the state-of-the-art Open AI Gym tool-kit. We extend the Gym environments with benchmark from EDO, and showed that RL methods can be used to solve DOPs. We further demonstrated that EDO methods can be used to solve RLPs.

We introduced a new continuous-control version of the Mountain Care Problem and called it the Double Mountain Car problem. In this problem agents are required to learn to use momentum to climb a hill, and then further to hold their position at the top of the hill. In real-world sequential decision problems, many tasks require learning agents to learn different skills in order to solve the task. Our results show that RL and EDO methods can be used to solve these benchmark problems.

Our results show that the relationship of time-linkage between states, the size of the state-action space and runtime resource constraints affect the online performance of algorithms. When these characteristics are specified for problem instances they can be used to help with algorithm selection. Developing a further understanding of how these constraints affect solution quality could help to design or select more efficient algorithms, particularly when specific resources need to be conserved.

One important consideration is that the paper only considers environmental dynamics where the environment returns to previous states. Q-learning was designed to learn the difference between states. In the presence of random or chaotic dynamics, or in the case of a large or continuous state-space, EDO may be more suitable. Useful extensions to the benchmark would be to include a wider range of varied sized state-action spaces, as well as state dynamics, to test relative algorithm performance under different SDP dimensions.

There are two key areas of interest for future work. First, problem instances that are exceptions to the assumption of equality between fitness and reward. And second, algorithm performance under uncertainty. Exceptions can be investigated by changing the reward estimation function (E in eq. 2). The problem definitions (eq. 2 and 1) can simulate noise in sensing by adding a Gaussian probability density function to the values of s_t.

References

[1] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[2] T. Nguyen, S. Yang, and J. Branke, "Evolutionary dynamic optimization: A survey of the state of the art," *Swarm and Evolutionary Computation*, vol. 6, pp. 1–24, 2012. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S2210650212000363\nhttp://www.sciencedirect.com/science/article/pii/S2210650212000363

[3] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1, no. 1.

[4] H. Fu, P. R. Lewis, B. Sendhoff, K. Tang, and X. Yao, "What are dynamic optimization problems?" in *Evolutionary Computation 2014. CEC14. IEEE Congress on*. IEEE, 2014, pp. 1550–1557.

[5] H. Fu, P. R. Lewis, and X. Yao, "A Q-learning Based Evolutionary Algorithm for Sequential Decision Making Problems," in *Workshop "In search of Synergies between Reinforcement Learning and Evolutionary Computation" at Parallel Problem Solving from Nature (PPSN)*. VUB AI Lab, 2014.

[6] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016. [Online]. Available: http://www.nature.com/doifinder/10.1038/nature16961

[7] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[8] T. T. Nguyen, S. Yang, and J. Branke, "Evolutionary dynamic optimization: A survey of the state of the art," *Swarm and Evolutionary Computation*, vol. 6, pp. 1–24, 2012.

[9] D. E. Goldberg and R. E. Smith, "Nonstationary function optimization using genetic algorithm with dominance and diploidy," in *Proceedings of the Second International Conference on Genetic Algorithms on Genetic Algorithms and Their Application*. Hillsdale, NJ, USA: L. Erlbaum Associates Inc., 1987, pp. 59–68. [Online]. Available: http://dl.acm.org/citation.cfm?id=42512.42521

[10] A. Dutech, T. Edmunds, J. Kok, M. Lagoudakis, M. Littman, M. Riedmiller, B. Russell, B. Scherrer, R. Sutton, S. Timmer *et al.*, "Reinforcement learning benchmarks and bake-offs ii," *Advances in Neural Information Processing Systems (NIPS)*, vol. 17, 2005.

[11] J. Branke, "Memory enhanced evolutionary algorithms for changing optimization problems," in *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, vol. 3. IEEE, 1999, pp. 1875–1882.

[12] K. Weicker, "An analysis of dynamic severity and population size," in *International Conference on Parallel Problem Solving from Nature*. Springer, 2000, pp. 159–168.

[13] V. S. Aragón and S. C. Esquivel, "An evolutionary algorithm to track changes of optimum value locations in dynamic environments," *Journal of Computer Science & Technology*, vol. 4, 2004.

[14] P. Rohlfshagen and X. Yao, "Dynamic combinatorial optimisation problems: an analysis of the subset sum problem," *Soft Computing*, vol. 15, no. 9, pp. 1723–1734, 2011.

[15] P. A. Bosman and H. La Poutre, "Learning and anticipation in online dynamic optimization with evolutionary algorithms: the stochastic case," in *Proceedings of the 9th annual conference on Genetic and evolutionary computation*. ACM, 2007, pp. 1165–1172.

[16] T. T. Nguyen and X. Yao, "Dynamic time-linkage problems revisited," in *Workshops on Applications of Evolutionary Computation*. Springer, 2009, pp. 735–744.

[17] J. Branke and H. Schmeck, "Designing evolutionary algorithms for dynamic optimization problems," in *Advances in evolutionary computing*. Springer, 2003, pp. 239–262.

[18] Y. Jin and J. Branke, "Evolutionary optimization in uncertain environments-a survey," *IEEE Transactions on evolutionary computation*, vol. 9, no. 3, pp. 303–317, 2005.

[19] A. E. Eiben and M. Schoenauer, "Evolutionary computing," *Information Processing Letters*, vol. 82, no. 1, pp. 1–6, 2002.

[20] A. Soni, P. R. Lewis, and A. Ekárt, "Offline and online time in sequential decision-making problems," in *IEEE CIDUE*. IEEE Press, 2016.

[21] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach (3rd edition)*. Prentice Hall, 2009.

[22] X. Li, K. Tang, M. N. Omidvar, Z. Yang, K. Qin, and H. China, "Benchmark functions for the cec 2013 special session and competition on large-scale global optimization," *gene*, vol. 7, no. 33, p. 8, 2013.

[23] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997.

[24] P. Gutman, P. Lindberg, I. Ioslovich, and I. Seginer, "A non-linear optimal greenhouse control problem solved by linear programming," *Journal of agricultural engineering research*, vol. 55, no. 4, pp. 335–351, 1993.

[25] J. Brownlee *et al.*, "A note on research methodology and benchmarking optimization algorithms," *Complex Intelligent Systems Laboratory (CIS), Centre for Information Technology Research (CITR), Faculty of Information and Communication Technologies (ICT), Swinburne University of Technology, Victoria, Australia, Technical Report ID*, vol. 70125, 2007.

[26] P. R. Cohen, *Empirical methods for artificial intelligence*. MIT press Cambridge, MA, 1995, vol. 139.

[27] J. N. Hooker, "Testing heuristics: We have it all wrong," *Journal of heuristics*, vol. 1, no. 1, pp. 33–42, 1995.

[28] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, pp. 237–285, 1996.

[29] R. S. Barr, B. L. Golden, J. P. Kelly, M. G. Resende, and W. R. Stewart, "Designing and reporting on computational experiments with heuristic methods," *Journal of heuristics*, vol. 1, no. 1, pp. 9–32, 1995.

[30] A. E. Eiben and M. Jelasity, "A critical note on experimental research methodology in ec," in *Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on*, vol. 1. IEEE, 2002, pp. 582–587.

[31] E. Peer, A. Engelbrecht, and F. Van Den Bergh, "Cirg@ up optibench: A statistically sound framework for benchmarking optimisation algorithms," in *Evolutionary Computation, 2003. CEC'03. The 2003 Congress on*, vol. 4. IEEE, 2003, pp. 2386–2392.

[32] S. P. Singh and R. S. Sutton, "Reinforcement learning with replacing eligibility traces," *Recent Advances in Reinforcement Learning*, pp. 123–158, 1996.

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

[34] M. Castronovo, D. Ernst, A. Couëtoux, and R. Fonteneau, "Benchmarking for bayesian reinforcement learning," *PLoS ONE*, vol. 11, pp. 1–37, 2016.

Aman Soni Biography text here.

Peter R. Lewis Biography text here.

Anikó Ekárt Biography text here.

Christopher Buckingham Biography text here.