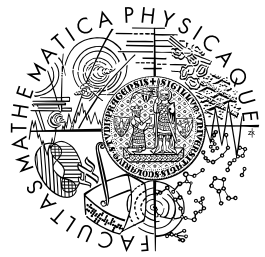

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

**BACHELOR THESIS**

Michal Pospěch

# Thesis title is N/A

Department of Theoretical Computer Science and Mathematical Logic

Supervisor of the bachelor thesis: Mgr. Roman Neruda, CSc.
Study programme: Computer science
Study branch: General computer science

Prague 2021

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date .....  .....

Author's signature

Dedication. It is nice to say thanks to supervisors, friends, family, book authors and food providers.

Title: Thesis title is N/A

Author: Michal Pospěch

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: Mgr. Roman Neruda, CSc., Department of Theoretical Computer Science and Mathematical Logic

Abstract: Abstracts are an abstract form of art. Use the most precise, shortest sentences that state what problem the thesis addresses, how it is approached, pinpoint the exact result achieved, and describe the applications and significance of the results. Highlight anything novel that was discovered or improved by the thesis. Maximum length is 200 words, but try to fit into 120. Abstracts are often used for deciding if a reviewer will be suitable for the thesis; a well-written abstract thus increases the probability of getting a reviewer who will like the thesis.

Keywords: key words

# Contents

# Introduction

# Chapter 1

# Important first chapter

# Chapter 2

# Theory

## 2.1 Reinforcement learning

The key problem that reinforcement learning is trying to solve is controlling an *agent* in an *environment*. The agent interacts with the environment by selecting some action and the environment responds by presenting the agent with a new situation. The environment also provides a numerical reward to the agent whose task is to maximise it over time.

More formally, the agent interacts with the environment in discrete (if the problem has continuous time, it is discretised) time steps $t = 1, 2, 3, \ldots$. At each time step the agent recieves $S_t \in \mathcal{S}$, a representation of internal state of the environment. Based on that it selects an action $A_t \in \mathcal{A}$. This results in the agent receiving a reward $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$ in addition to a new state $S_{t+1}$ in the next step generating a sequence or *trajectory* beginning like this:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots$$

The random variables $S_t$ and $R_t$ depend only on the preceding state and action based on function $p : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$ characterising *dynamics* of the problem wiht following definition:

$$p(s', r|s, a) = \Pr\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\}.$$

This can, however, be viewed as an restriction on information contained in state rather than restriction on the environment. If the state does contain all information that influence the future then it has the *Markovian property* and the problem at hand is a *Markov decision process*.

To formally describe what is the goal of reinforcement learning *expected return* $G_t$ needs to be defined first. It is a function of the reward sequence, in the

simplest case an ordinary sum of rewards

$$G_t = \sum_{i=t+1}^{T} R_i, \tag{2.1}$$

where $T$ is the final time step.

This approach works only if the there is a final step in the interaction which would divide the interaction into subsequences called *episodes*. Each episodes ends when the environment reaches a *terminal state* and is reset into its initial state afterwards. Such tasks are called *episodic tasks*.

Contrary to that there are tasks which do not break into episodes, they continue indefinitely. They are called *continuing tasks* For these the definition 2.1 doesn't work as the sum could possibly be infinite. To calculate expected reward for such tasks a modified approach is needed. A *discount rate* $\gamma \in [0, 1]$ is introduced to define *discounted return*

$$G_t = \sum_{i=0}^{\infty} \gamma^i R_{t+i+1}. \tag{2.2}$$

The discount rate makes future reward worth less at the moment of decision. If $\gamma < 1$ and the reward are bounded, then the infinite sum 2.2 is finite. For $\gamma = 0$ the agent is said to be "myopic" taking into account only the immediate reward and ignoring the future. Otherwise as $\gamma$ approaches $1$ the future rewards have bigger weight and influence the agent's decision more. Succesive returns are related in a way

A common occurence in reinforcement learning is an estimating *value function* which estimate how beneficial it is for the agent to be in that state. This is defined by the expected return which is dependent on the actions that the agent will perform. Therefore a value function must be defined with respect to such way of acting called *policy*.

Policy is a function $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ returning probability of selecting each action in given state meaning that agent following policy $\pi$ at time $t$ would select action $A_t = a$ if in state $S_t = s$ with probability $\pi(a|s)$. The policy function is changed by reinforcement learning based on the agent's experience.

For state $s$ and policy $\pi$ the *state-value function* $v_\pi(s)$ gives the expected return when starting in state $s$ and following policy $\pi$, formally (for MDPs)

$$v_\pi(s) = \mathrm{E}_\pi[G_t|S_t = s], \tag{2.3}$$

where $\mathrm{E}_\pi[.]$ is expected value when following policy $\pi$ in every timestep.

*Action-value function* $q_\pi(s, a)$ for policy $\pi$ is defined similarly. It gives the expeted return of taking action $a$ in state $s$ while following policy $\pi$, formally

$$q_\pi(s, a) = \mathrm{E}_\pi[G_t|S_t = s, A_t = a]. \tag{2.4}$$

- various methods
  - criterion of optimality
  - direct policy search (and various methods)

## 2.2 Evolutionary algorithms

Evolutionary algorithms (EA) are a type of optimisation metaheuristics inspired by the process of bilogical evolution. At first a number of possible solutions to the problem at hand is generated (*population*) and each solution (*individual*) is encoded (via a domain-specific encoding) and evaluated giving us the value of its *fitness*. Fitness is a function describing how good that particular individual is and it is everything that is needed for creation of Then a new population is created using a *crossover* (combination) of 1 or more individuals which are selected using the operator of *parental selection*. Each of the newly created individuals has a chance to be mutated via the *mutation* operator. Finally a new population is selected from *offsprings* and possibly the parents based of fitness and enters the next iteration of the EA and the following generation is chosen using *environmental selection* operator. The algorithm repeats until the stop condition is met, usually a set number of iterations or small improvement of fitness between 2 generations.

There are many variands of EAs such as genetic algorithms (most common), genetic programming, evolutionary programming, neuroevolution or evolutionary strategies that are further described in following chapter. [1] [2]

**Algorithm 1** Evolutionary algorithm

initialize population $P^0$ with $n$ individuals
set $t = 0$
**repeat**
    $Q^t = \{\}$
    **for** $i \in \{1 \dots m\}$ **do**
        $p_1, \dots, p_\rho = ParentalSelection(P^t)$
        $q = Crossover(p_1, \dots, p_\rho)$
        $q = Mutation(q)$ with chance $p$
        $Q^t = q \cup Q^t$
    **end for**
    $P^{t+1} = EnvironmentalSelection(Q^t \cup P^t)$
    increment $t$
**until** stop criterion fullfilled

## 2.3 Evolutionary strategies

Evolutionary strategies (ES) are a type of optimisation metaheuristic which further specialises EA and restricts their level of freedom. The selection for crossover is unbiased, mutation is parametrised and thus controllable, individuals which should be put to next generation are chosen ordinally based on fitness and individuals contain not only the problem solution but also control parameters.

More formally ES $(\mu/\rho, \kappa, \lambda)$ has $\mu$ individuals in each generation, which produces $\lambda$ offsprings, each created by crossover of $\rho$ individuals and each individual is able to survive for up to $\kappa$ generations as described in algorithm 2. This notation further generalizes the old $(\mu, \lambda)$ and $(\mu + \lambda)$ notations, where the "," notation means $\kappa = 1$ and "+" notation $\kappa = \infty$.

**Algorithm 2** $(\mu/\rho, \kappa, \lambda)$-ES

1: initialize population $P^0$ with $\mu$ individuals
2: set age for each $p \in P^0$ to 1
3: set $t = 0$
4: **repeat**
5: $Q^t = \{\}$
6: **for** $i \in \{1 \dots \lambda\}$ **do**
7: select $\rho$ parents $p_1, \dots, p_\rho \in P^t$ uniformly at random
8: $q = variation(p_1, \dots, p_\rho)$ with age 0
9: $Q^t = q \cup Q^t$
10: **end for**
11: $P^{t+1}$ = select $\mu$ best (wrt. fitness) individuals from $Q^t \cup \{p \in P^t : age(p) < \kappa\}$
12: increment age by 1 for each $p \in P^{t+1}$
13: increment $t$
14: **until** stop criterion fullfilled

To design an ES one must first select an appropriate representation for an individual and the most natural one is prefered in most cases, if all parameters are of one type (e.g. a real number) a simple vector will suffice, if the types are mixed, a tuple of vectors is required. This however causes an increased complexity of the variation operator.

As for design of the variation operator there are some guidelines that should be followed when designing it.

**Reachability** every solution should be reachable from any other solution in a finite number of applications of the variation operator with probability

$p > 0$

**Unbiasedness** the operator should not favour any particular subset of solution unless provided with information about problem at hand

**Control** the operator should be parametrised in such way that the size of the distribution can be controlled (practice had shown that decreasing it as the optimal solution is being approached is necessary)

kovariance

[3] [1]

### 2.3.1 CMA-ES

TODO [4]

## 2.4 Evolutionary strategies as replacement for reinfocement learning

Black-box optimisation is an alternative approach to solving RL tasks also known as Direct policy search or neurevolution when applied to neural networks. It has several attractive properties such as indifferenco to distribution of rewards, no need for backpropagation and tolerance of arbitrarily long episodes.

Compared to reinfocement learning using evolutionary strategies has the advantage of not needing a gradient of the policy performance. Also as the state transition function is not known the gradient can't be computed using backpropagation-like algorithm. Thus some noise needs to be added to make the problem smooth and the gradient to be estimable. Here is where reinfocement learning and evolutionary strategies differ, reinfocement learning adds noise in the action space (actions are chosen from a distribution) while evolutionary strategies add noise in the parameter space (parameters perturbed while actions are deterministic).

Not requiring backpropagation has several advantages over other RL methods. First the amount of computation necessary for one episode of ES is much lower (about one third, potentially even less for memory usage). Not calculating gradient using analytic methods also protects these methods from suffering from *exploding gradient* which is a common issue with recurrent neural networks. And last, the network can contain elements that are not differentiable such as hard attention.

Also ES are easily paralellisable. First, contrary to RL, where value function is inherently linear procedure and has to be performed more times to improve a given policy. Furthermore it operates on whole whole episodes, therefore it does

not require frequent commuincation between workers. And finally, as the only information received by each worker is the reward, it is possible to communicate only the reward inbetween workers. That, however, requires synchronising seeds known to other workers beforehand and recreating perturbations based on that information. Thus the required is extremely low comapred to communication of whole gradients which would be required for paralellisation of a policy gradient based algorithm.

### 2.4.1 OpenAI Evolutionary Strategy

These ideas are explored in OpenAI-ES algorithm. The agent acting in the envrionment is represented be a policy $\pi_\theta$ with parameter vector $\theta$ and $f(\cdot)$ is the reward returned by the environment. As we need to introduce some noise, the population is distribution $p_@y$ is instantiated as an an isotropic multivariate Gaussian with mean $\psi$ and covariance $\sigma^2 I$. Then $\mathbb{E}_{\theta\sim p_\psi} f(\theta) = \mathbb{E}_{\epsilon\sim N(0,I)} f(\theta + \sigma\epsilon)$. Thus the gradient approximation is calculated as follows:

$$\nabla_\theta \mathbb{E}_{\theta\sim p_\psi} f(\theta) = \nabla_\theta \mathbb{E}_{\epsilon\sim N(0,I)} f(\theta + \sigma\epsilon) \approx \frac{1}{n\sigma} \sum_{i=1}^{n} f(\theta_i)\epsilon_i, \tag{2.5}$$

where $\theta_i = \theta + \sigma\epsilon_i, \epsilon_i \sim N(0, I)$ and $n$ is the population size.

The resulting algorithm uses SGD (or Stochastic Gradient Ascent - SGA in this case) or other gradient based optimisation technique for parameter vector $\theta$ update.

As the ES could be seen as method for computing a derivative estimate using finite differences in randomly chosen direction it would suggest that it would scale poorly with dimensions of parameters $\theta$ same as the finite differences method. In theory the number of necessary optimisation steps should scale linearly with the dimension. That however doesn't mean that larger networks optimised using ES will perform worse than smaller ones, that depends on the difficulty (intrinsic dimension) of the problem. The network will perform the same however it will take more optimisation steps to do so.

In practice ES performs slightly better on larger networks and it is hypothesised that it is for the same reason as why it is easier to optimise large networks using standard gradient based methods: larger networks have fewer local minima.

Due to perturbing the parameters and not the actions ES are invariant to the frequency at which the agent acts in the envirionment. Tradtional MDP-based reinforcement learning methods rely on *frameskip* as one their parameters that is crucial to get right for the optimization to be successful. While this is solvable for problems that do not require long term planning and actions, long term

**Algorithm 3** OpenAI-ES

**Input**: Learning rate $\alpha$, noise standard deviation $\sigma$, initial policy parameters $\theta_0$, $n$ number of workers
initialize $n$ workers with known random seeds, and initial parameters $\theta_0$
set $t = 0$
**repeat**
    **for** $i \in \{1 \dots n\}$ **do**
        sample $\epsilon_i \sim \mathcal{N}(0, I)$
        compute returns $f_i = f(\theta_t + \sigma\epsilon_i)$
    **end for**
    Send all scalar returns $f_i$ from each worker to every other worker
    **for** $i \in \{1 \dots n\}$ **do**
        reconstruct all perturbations $\epsilon_j$ for $j \in \{1, \dots, n\}$ using known random seeds
        set $\theta_{t+1} = \theta_t + \alpha \frac{1}{n\sigma} \sum_{j=1}^{n} f_i \epsilon_i$
    **end for**
    increment $t$
**until** stop criterion fullfilled

strategic behaviour poses a challenge and reinfocement learning needs hiearchy to be succesful unlike evolutionary strategy.

### 2.4.2 Novelty search

While the main drive in of improvement in ES is the value of fitness (how "good" the result is), novelty search takes a different approach. Novelty search is focused on finding different solutions, as it is inspired by nature's drive towards diversity. Each policy has its novelty calculated with respect to previous policies and search is directed to parts of search space with high novelty. This approach makes it less succeptible to local optima created by deceptive rewards than reward-based method.

Each policy $\pi$ gets assigned its domain-dependent behavorial characteristics $b(\pi)$ (e.g. final position of the agent) and it is added to an archive set $A$ of characteristics of previous policies. Then the novelty $N(b(\pi_\theta), A)$ is calculated as average distance from $k$ nearest neighbours from the archive set $A$.

$$N(\theta, A) = N(b(\pi_\theta), A) = \frac{1}{|S|} \sum_{j \in S} \|(\pi_\theta) - b(\pi_j)\|_2$$
$$S = kNN(b(\pi_\theta), A)$$

### Combination with evolution strategies

To find and follow the gradient of expected novelty with respected to $\theta^t$ we use the framework outlined in 2.4.1.

With archive $A$ and sampled parameters $\theta_t^i = \theta_t + \sigma\epsilon_i$, the gradient estimate can be calculated via following formula:

$$\nabla_{\theta_t}\mathbb{E}_{\epsilon\sim\mathcal{N}(0,I)}[N(\theta_t + \sigma\epsilon)|A] \approx \frac{1}{n\sigma}\sum_{i=1}^{n} N(\theta_t^i, A)\epsilon_i \tag{2.6}$$

It is possible because archive $A$ is fixed during one iteration and is updated only at the end. Only characteristics corresponding to each $\theta^t$ are added to $A$, as adding each sampled would cause the archive $A$ to inflate too much increasing the complexity of calculation of nearest-neighbours.

To encourage additional diversity an initial meta-population of $M$, selection of $M$ is domain dependent, agents is created. While it is possible to optimise the behaviour of a single agent and reward it for behaving differently than its ancestors, this way we get the benefits of population-based exploration. Each agent has parameters $\theta^m$ and is being rewarded for behaviour different from all prior agents, thus we get $M$ differently behaving policies.

Describe?

$M$ random parameter vectors are initialised and in each iteration one is selected to be updated. The selection probability is proportional to its novelty.

$$P(\theta^m) = \frac{N(\theta^m, A)}{\sum_{i=1}^{M} N(\theta^i, A)} \tag{2.7}$$

To perform the update step, we need to calculate the gradient estimate of expected novelty with respect to $\theta_t^m$ using equation 2.6 where $n$ is the number of perturbations. When we get the gradient estimate we use SGD (or in this case Stochastic Gradient Ascent) with learning rate $\alpha$ to update the parameters $\theta^m$

$$\theta_{t+1}^m \coloneqq \theta^m + \alpha\frac{1}{n\sigma}\sum_{i=1}^{n} N(\theta_t^{i,m}, A)\epsilon_i. \tag{2.8}$$

After updating the individual, a new behavorial characteristics $b(\pi_{\theta_{t+1}^m})$ is calculated and added to the archive $A$.

This process is repeated for a predetermined number of times as novelty search is not supposed to converge to a "best" solution and returns the best performing policy which is being preserved during the run of the algorithm.

### Combination with reward based exploration

While *NS-ES* helps agents avoid local optima and deceptive reward signals, it also completly discards reward which might cause the performance to suffer.

**Algorithm 4** NS-ES

**Input**: Learning rate $\alpha$, noise standard deviation $\sigma$, initial policy parameters $\theta_0$, $n$ number of workers, $T$ number of iterations
initialize $n$ workers with known random seeds, empty archive $A$, and initial parameters $\{\theta_0^1, \ldots, \theta_0^M\}$
set $t = 0$
**for** $i \in \{1 \ldots M\}$ **do**
    calculate $b(\pi_{\theta_0^i})$
    add $b(\pi_{\theta_0^i})$ to $A$
**end for**
**for** $t \in \{0 \ldots T-1\}$ **do**
    sample $\theta_t^m$ from $\{\theta_t^1, \ldots, \theta_t^M\}$
    **for** $i \in \{1 \ldots n\}$ **do**
        sample $\epsilon_i \sim \mathcal{N}(0, I)$
        compute characteristics $b(\theta_t^m + \sigma\epsilon_i)$
        compute $n_i = N(b(\theta_t^m + \sigma\epsilon_i), A)$
    **end for**
    Send all novelties $n_i$ from each worker to every other worker
    **for** $i \in \{1 \ldots n\}$ **do**
        reconstruct all perturbations $\epsilon_j$ for $j \in \{1, \ldots, n\}$ using known random seeds
        set $\theta_{t+1}^m = \theta_t^m + \alpha \frac{1}{n\sigma} \sum_{j=1}^{n} n_i \epsilon_i$
        add $b(\theta_{t+1}^m)$ to $A$
    **end for**
**end for**

Therefore NSR-ES, an improved version of NS-ES, uses both reward (fitness) and novelty for computation of the update step. NSR-ES is in many ways similar to NS-ES, it calculates both novelty and reward at once and it operates on entire episodes. The only difference is, that the calculation of the gradient estimate is based on the average of reward and novelty.

Specifically, for parameter vector $\theta_t^{m,i} = \theta_t^m + \sigma\epsilon_i$ we calculate the reward $f(\theta_t^{m,i})$ and novelty $N(\theta_t^{m,i}, A)$, rank-normalise both values independently (as both values usually have completely different scales), calculate the average and set it as weight for corresponding $\epsilon_i$ for gradient estimation. Then the estimated gradient is used to update the parameter vector via SGD (or other gradient based optimisation method) simirally as in equation 2.8:

$$\theta_{t+1}^m \coloneqq \theta^m + \alpha \frac{1}{n\sigma} \sum_{i=1}^{n} \frac{N(\theta_t^{i,m}, A) + f(\theta_t^{i,m})}{2} \epsilon_i. \tag{2.9}$$

Intuitively, following the approximated gradient based on both novelty and reward directs the search areas of the parameter-space with both high novelty and reward. This can, however, be improved further.

While NSR-ES uses a linear combination of reward and novelty to approximate that is static for the whole duration of the training. Contrary to that NSRAdapt-ES (NSRA-ES) dynamically changes the ratio of reward gradient $f(\theta_t^{i,m}$ and novelty gradient $N(\theta_t^{i,m}, A)$ based on how the training is currently progressing. This way, it will give more weight to the reward (thus following the performance gradient) when making progress and more weight to the novelty (following novelty gradient) when stuck in a local optima to give more incentive to find different approaches.

Formally, a parameter $w$ is used to control the ratio of reward and novelty used for calculation of gradient estimate. For a specific $w$ at a given generation parameter vector $\theta_t^m$ is updated (SGD used) via following expression:

$$\theta_{t+1}^m \coloneqq \theta_t^m + \alpha \frac{1}{n\sigma} \sum_{i=1}^{n} (1-w) N(\theta_t^{i,m}, A)\epsilon_i + w f(\theta_t^{i,m})\epsilon_i. \tag{2.10}$$

[5]

**Algorithm 5** NSR-ES

**Input**: Learning rate $\alpha$, noise standard deviation $\sigma$, initial policy parameters $\theta_0$, $n$ number of workers, $T$ number of iterations
initialize $n$ workers with known random seeds, empty archive $A$, and initial parameters $\{\theta_0^1, \ldots, \theta_0^M\}$
set $t = 0$
**for** $i \in \{1 \ldots M\}$ **do**
    calculate $b(\pi_{\theta_0^i})$
    add $b(\pi_{\theta_0^i})$ to $A$
**end for**
**for** $t \in \{0 \ldots T-1\}$ **do**
    sample $\theta_t^m$ from $\{\theta_t^1, \ldots, \theta_t^M\}$
    **for** $i \in \{1 \ldots n\}$ **do**
        sample $\epsilon_i \sim \mathcal{N}(0, I)$
        compute characteristics $b(\theta_t^m + \sigma\epsilon_i)$
        compute $n_i = N(b(\theta_t^m + \sigma\epsilon_i), A)$
        compute $f_i = f(\theta_t^m + \sigma\epsilon_i)$
    **end for**
    Send all novelties and rewards $n_i, f_i$ from each worker to every other worker
    **for** $i \in \{1 \ldots n\}$ **do**
        reconstruct all perturbations $\epsilon_j$ for $j \in \{1, \ldots, n\}$ using known random seeds
        set $\theta_{t+1}^m = \theta_t^m + \alpha \frac{1}{n\sigma} \sum_{j=1}^{n} \frac{n_i + f_i}{2} \epsilon_i$
        add $b(\theta_{t+1}^m)$ to $A$
    **end for**
**end for**

**Algorithm 6** NSRA-ES

**Input**: Learning rate $\alpha$, noise standard deviation $\sigma$, initial policy parameters $\theta_0$, $n$ number of workers, $T$ number of iterations
initialize $n$ workers with known random seeds, empty archive $A$, and initial parameters $\{\theta_0^1, \dots, \theta_0^M\}$
set $t = 0, t_{best} = 0, f_{best} = -\infty$
**for** $i \in \{1 \dots M\}$ **do**
    calculate $b(\pi_{\theta_0^i})$
    add $b(\pi_{\theta_0^i})$ to $A$
**end for**
**for** $t \in \{0 \dots T-1\}$ **do**
    sample $\theta_t^m$ from $\{\theta_t^1, \dots, \theta_t^M\}$
    **for** $i \in \{1 \dots n\}$ **do**
        sample $\epsilon_i \sim \mathcal{N}(0, I)$
        compute characteristics $b(\theta_t^m + \sigma\epsilon_i)$
        compute $n_i = N(b(\theta_t^m + \sigma\epsilon_i), A)$
        compute $f_i = f(\theta_t^m + \sigma\epsilon_i)$
    **end for**
    Send all novelties and rewards $n_i, f_i$ from each worker to every other worker
    **for** $i \in \{1 \dots n\}$ **do**
        reconstruct all perturbations $\epsilon_j$ for $j \in \{1, \dots, n\}$ using known random seeds
        set $\theta_{t+1}^m = \theta_t^m + \alpha \frac{1}{n\sigma} \sum_{j=1}^{n} \frac{n_i + f_i}{2} \epsilon_i$
        add $b(\theta_{t+1}^m)$ to $A$
    **end for**
    **if** $f(\theta_{t+1}^m) > f_{best}$ **then**
        $w = max(1, w + \delta_w)$
        $f_{best} = f(\theta_{t+1}^m)$
        $t_{best} = 0$
    **else**
        $t_{best} += 1$
    **end if**
    **if** $t_{best} \geq t_w$ **then**
        $w = min(0, w - \delta_w)$
        $t_{best} = 0$
    **else**
    **end if**
**end for**

# Chapter 3

# Results and discussion

# Conclusion

# Bibliography


[1] Günter Rudolph. "Evolutionary Strategies". In: *Handbook of Natural Computing*. Ed. by Grzegorz Rozenberg, Thomas Bäck, and Joost N. Kok. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 673–698. ISBN: 978-3-540-92910-9. DOI: 10.1007/978-3-540-92910-9_22. URL: https://doi.org/10.1007/978-3-540-92910-9_22.

[2] P. A. Vikhar. "Evolutionary algorithms: A critical review and its future prospects". In: *2016 International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC)*. 2016, pp. 261–265. DOI: 10.1109/ICGTSPICC.2016.7955308.

[3] Hans-Paul Schwefel and Günter Rudolph. "Contemporary evolution strategies". In: *Advances in Artificial Life*. Ed. by Federico Morán et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 891–907. ISBN: 978-3-540-49286-3.

[4] Nikolaus Hansen. *The CMA Evolution Strategy: A Comparing Review*. 2006.

[5] Edoardo Conti et al. *Improving Exploration in Evolution Strategies for Deep Reinforcement Learning via a Population of Novelty-Seeking Agents*. 2018. arXiv: 1712.06560 [cs.AI].