Swarm and Evolutionary Computation 44 (2019) 228–246

Contents lists available at ScienceDirect

Swarm and Evolutionary Computation

journal homepage: www.elsevier.com/locate/swevo

Reinforcement learning versus evolutionary computation: A survey on hybrid algorithms

Madalina M. Drugan

Technical University of Eindhoven, The Netherlands

ARTICLE INFO

Keywords:
Reinforcement learning
Evolutionary computation
Natural paradigms
Hybrid algorithms
Survey

ABSTRACT

A variety of Reinforcement Learning (RL) techniques blends with one or more techniques from Evolutionary Computation (EC) resulting in hybrid methods classified according to their goal, new focus, and their component methodologies. We denote this class of hybrid algorithmic techniques as the evolutionary computation versus reinforcement learning (ECRL) paradigm. This overview considers the entire spectrum of algorithmic aspects and proposes a novel methodology that analyses the technical resemblances and differences in ECRL. Our design analyses the motivation for each ECRL paradigm, the underlying natural models, the sub-component algorithmic techniques, as well as the properties of their ensemble.

1. Introduction

Natural paradigms are a major inspiration source for all areas of science and especially in statistics and computer science. Machine Learning models map inputs (like sensors and other empirical data structures) to outputs (like predictions and classifications). A recent trend in machine learning transfers expertise from and to areas of optimisation. An interesting novel symbiosis considers: 1) *Reinforcement learning* (RL) where an agent interacts with a dynamic and changing environment by taking actions that affect the state of the environment to complete a task [169,190]. 2) *Evolutionary Computation* (EC) is a sub-field of global optimisation that uses evolutionary principles for automated and parallel problem solving [4,52,89]. Populations of interactive organisms inspired EC, and learning in animals and humans inspired RL. Although they seem very different, RL and EC are two optimisation techniques that address the same problem: the optimisation of a function, the maximisation of an agent's reward in RL and the fitness function in EC, respectively, in potentially unknown environments.

Hybrid evolutionary computation versus reinforcement learning (ECRL) techniques have raised constant interest. Therefore, current state-of-the-art in ECRL comprises a rather broad set of optimisation and learning algorithms. ECRL solves methodological and theoretical challenges in EC or RL, while representing realistic models with incomplete observations, large stochastic and changing environments. The main contribution of this study is a *unified* methodology for the ECRL algorithms with the goal of understanding the algorithmic essence of the two influential paradigms and what integrates them in the ECRL common framework. Our research is non-exhaustive, focusing on the major trends and practices in the field. Overall, there is a tremendous potential to develop new hybrid ECRL algorithms for already identified or new tasks.

Our approach has a *broader scope* than similar studies by summarising the major and latest trends in ECRL. A high-level conceptual comparison between RL and EC is given in Ref. [149]. Recent overview studies [22,40,122,151], however, focus on a particular subclass of ECRL algorithms when describing the targeted ECRL algorithmic subcomponent. Using natural paradigms as motivation for reinforcement learning [158] is novel for some hybrid reinforcement learning algorithms such as multi-objective reinforcement learning [44,48,111,145]. This study is *complementary* to the other studies collecting points of view from the perspective of both EC and RL. Some of the reviewed ECRL techniques are experimental studies, with no theoretical analysis. There are theoretical works without any empirical results, and algorithmic studies focus on the algorithmic analysis of some benchmark problems. The proposed method of classification of hybrid techniques is novel describing best the properties, or characteristics, of any ECRL algorithm.

The paper is structured as follows. Section 2 describes our methodology for analysing ECRL algorithms. Section 3 and 4 introduce preliminary notation and denomination for the EC and RL paradigms, respectively. Section 5 presents a high level classification criteria for the ECRL algorithms. Section 6 extrapolates their properties. Section 7 compares two common RL and EC paradigms. Section 8 surveys works on EC for direct policy search. Section 9 summarises EC methods that

E-mail address: madalina.drugan@gmail.com.

https://doi.org/10.1016/j.swevo.2018.03.011
Received 10 April 2017; Received in revised form 4 August 2017; Accepted 26 March 2018
Available online 29 March 2018

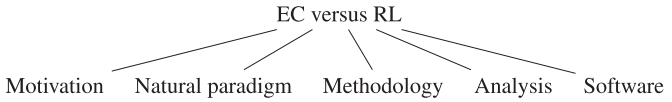

Fig. 1. A unified taxonomy for the ECRL paradigms.

adapt online with RL. Section 10 investigates the hierarchy of structures in ECRL. Section 11 presents multi-objective RL strategies. Section 12 describes ECRL in adversarial and dynamic environments. Section 13 concludes the paper.

2. A unified methodology for the analysis of ECRL

Understanding RL's underlying theoretical properties, and the empirical power of EC's mechanisms is crucial for designing hybrid algorithms. In this section, we introduce a methodology that analyses ECRL techniques in a common framework. Our framework includes both specific methods, like the natural intuition and the analysis of algorithmic sub-components of ECRL, and general methods, such as the motivation, the algorithmic analysis and applications. Fig. 1 illustrate our unified methodology for ECRL. Later in the paper, we specialise the taxonomies.

Motivation. Any new paradigm finds its motivation in either the challenge raised by real-world applications or existing models with different constraints. Increasing the computation power allows for accurate models of practical problems solvable by composing two or three methods. A general goal when combining EC and RL is that the compound performs better than the standard setting alone. Famous practical examples include robotics and web search engines that blend many techniques from machine learning and optimisation. For each paradigm in ECRL, we acknowledge at least one motivating application and the biological-like intuition on its base.

Natural intuition. Some ECRL paradigms explicitly highlight (simulation of) evolution components whereas, for few ECRL methods, the natural intuition was never stated before. One major effort of this work is to label the revised ECRL class of techniques using the natural paradigms. In Ref. [186], the evolutionary function approximation for RL uses Lamarckian evolution, which assumes that the offsprings acquire knowledge from their parents. The described parameter control techniques are accompanied by the explanations for the evolutionary techniques.

Methodological analysis of ECRL's sub-components. Another classification criteria considers the RL methodology used in the EC algorithms with learning components, and vice-versa, EC techniques in RL. To understand the properties of algorithms resulting from RL methods that mingle with EC techniques, we describe ECRL using a detailed description of Markov decision processes (MDPs) down to the underlying equations and the algorithmic recipe used by EC. We summarise methods that combine the algorithmic EC and RL compounds in an ECRL with a new focus and properties. We highlight the properties of sub-components and compare them with the features of their assembly.

Algorithmic analysis. Often, the algorithmic works validate the proposed methodology using either experimental or/and theoretical analysis. Novel algorithms require the re-investigation of the empirical methodology for the new optimisation or learning target of the algorithm, whereas theoretical properties explain the behaviour of the ECRL algorithms in the limit. Markov decision processes (MDP) have well-studied convergence properties; the value iteration approach with discounting reward values has geometrical convergence rates to the optimal policy [154]. An RL strategy with a shorter convergence time to the optimal policy than another approach is considered a better strategy.

In opposition, EC algorithms are often compared only empirically, with no formal analysis on their properties. Empirical studies constitute the foremost throughput for EC, where the EC's performance mainly depends on the landscape of the analysed problem. In fact, some of the fancy techniques imported from simulated evolution or the EC algorithms, with the best empirical performance have undetermined theoretical properties. For example, learning classifiers systems [22] is a thriving research area with many applications, but related technical studies were published only recently [128].

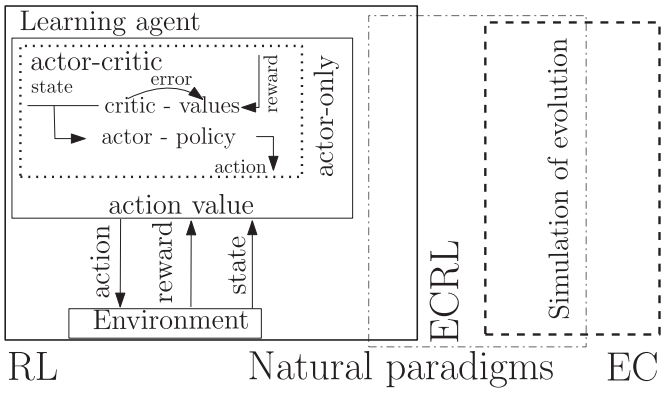

Fig. 2. RL uses a learning agent that continuously interacts with the environment by receiving rewards and performing actions in different states. RL has three different subclasses: actor critic, actor only, and critic only; the figure sketches the actor critic flow. The ECRL paradigm represents algorithms using techniques from EC, RL, and natural paradigms.

Software development tools. Nowadays, the increasing amount of algorithms makes the development of the corresponding software efficient when building on existing software and methods. Hence, we review some of the existing freeware software for RL, EC and ECRL, and we indicate some of their basic properties, such as the programming language and the targeted algorithms.

The next two sections fit RL and EC into our methodological study.

3. Short introduction in reinforcement learning

This section gives preliminaries on the RL techniques in ECRL instances using the methodology presented in the previous section.

3.1. The intuition: an agent interacts with an environment

RL problems involve a learning agent that interacts with the environment to optimise its long-term reward. An immediate reward is returned to the environment as a result of an action of the agent; the reward value that can be stochastic, i.e. drawn according to a probability distribution. An immediate negative reward expresses a punishment, and positive values show a positive immediate reward. The cumulative reward value (or long term reward) is maximised using the knowledge accumulated previously and the future predictions.

Fig. 2 sketches the intuition underlying the RL techniques. Consider the goal of learning the optimal policy. RL can be classified into three sub-classes: actor only, critic only and actor-critic. The actor-only paradigm searches directly in the policy space, including the majority of ECRL algorithmic instances [84].

3.2. Markov decision process

An MDP formulates the problem of decision-making under uncertainty with the objective to maximise a cumulative reward value. More formally, an MDP is a tuple $< S, A, T, R >$ characterised by:

1. A set of *states* $S = \{s_1, s_2, \dots, s_N\}$ where s_t is a state in S;
2. A set of *actions* $A = \{a_1, a_2, \dots, a_M\}$ available to the agent in each state s;
3. A *transition* distribution $T(s' \mid s, a)$ maps a pair composed of a state s and an action a to a probability distribution of state s';

4. A *reward* function $R : S \times A \times S \to \mathbb{R}$ gives the expected reward when the agent makes the transition from state s to state s' using action a.

We denote with r_t the immediate scalar reward obtained at time t, where

$$r_t = R(s_{t+1} = s' \mid s_t = s, a_t = a) = \mathbb{E}\{r_t \mid s_{t+1} = s', s_t = s, a_t = a\}$$

This process is Markovian since the distribution of the next states and rewards is independent of the past through the current state and action.

$$T(s_{t+1} \mid s_t, a_t) = T(s_{t+1} \mid s_t, a_t, \dots, s_1, a_1)$$

The action-selection mechanism in an MDP is described by a *policy* $\pi : S \times A \to [0, 1]$ that specifies a probability of selecting an action a in a specific state s. The expected discounted sum of future rewards gives the quality of a policy π in state s

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[R_t \mid s_t = s\right] = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t \cdot r_t \mid s_t = s\right]$$

where R_t denotes the return and γ is a real value denoted as the *discount factor* to weigh more the immediate reward than the reward received in the far future, $0 \leq \gamma < 1$. V^{π} is the expected return of an agent following policy π. Alternatively, the *action-value* function for policy π, $Q^{\pi}(s, a)$, is defined as the expected return when taking action a in state s under policy π. Thus

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}\left[R_t \mid s_t = s, a_t = a\right]$$

The goal of any MDP is to find the best policy π^* that maximises the expected return. The optimal state value function, for any state s, is $V^*(s) = \max_{\pi} V^{\pi}(s)$.

A classification criterion for RL approaches is the strategy used to determine optimal policies.

1) *Value iteration* updates each iteration a policy given its value function. The Bellman equation [8] is an update rule that extends the planning horizon with one step, meaning that the current value function $V_t^{\pi}(s)$ updates to

$$V_{t+1}^{\pi}(s) = \max_{a \in A} \sum_{s' \in S} T(s' \mid s, a)\left(R(s' \mid s, a) + \gamma \cdot V_t^{\pi}(s')\right) \quad (1)$$

The sequence of $V_t^{\pi}(s)$ converges as the number of iterations t goes to infinity, and there are several alternative ways to solve it.

2) *Policy iteration* improves the quality of the policy π over all possible actions, after evaluating the value function V^{π} of a fixed policy π. Q-learning is a value iteration like RL algorithm, whereas SARSA is a policy iteration RL algorithm. Actor-critic matches with some temporal difference learning (TD) variants where the prediction error is the difference between consecutive states and used to update both the actor and the critic.

3) *Direct policy search* considers that policies are parameterised by some real-valued vector, and EC strategies are often used to optimise these parameters. In this case, there is no need to learn the value function.

Table 1
Reviews on RL algorithms.

Reinforcement learning RL	Kaelbing et al. [93], Wiering and van Otterlo [190]
Deep learning in neural networks	Bengio [9], Schmidhuber [150,151]
Linear function approximators for RL	Geramifard et al. [67]
Bayesian RL	Ghavamzadeh [68]
Approximate dynamic programming	Powell [137]
Markov decision processes	Burnetas and Katehakis [25]
Multi-agent reinforcement learning	Bowling and Veloso [14]
Game theory and reinforcement learning	Cesa-Bianchi and Lugosi [30]

3.3. Reinforcement learning variants

RL defines a large class of algorithms; a comprehensive overview of RL variants is given in Sutton and Barto's book [169]. Table 1 shows review papers on RL variants. The standard MDP also assumes an entirely observable state (or CO-MDP), whereas the partially observable MDP (PO-MDP) strategies do not have direct access to the current state.

Dynamic programming (DP) assumes that the transition distribution is entirely known, which is a strong assumption unreasonable for most practical applications. DP methods assume that reward functions are known. For DP's alternatives, like *Monte Carlo* sampling, the transition and the reward functions do not need to be known apriori. *Temporal difference* (TD) learning uses observations of consecutive states to update value predictions. Thus, TD learns the values of states based on the estimates of other values, called bootstrapping. We have on-policy, i.e. SARSA, and off policy, i.e. Q-learning, TD algorithms.

Model-based RL speeds up the process by learning a model of the environment, such as storing the frequencies of each outcome of a state- action pair in a table that creates a model. Intuitively, a model should help the exploitation, because of the information that is back-propagated to update the state-action values. However, as pointed out by Ref. [170], this is not always the case, i.e. the model is sometimes not adequate or properly used. Furthermore, when the environment changes over time, model-based RL might not exit from the learning loops, even for small, but dynamical, environments.

3.4. Applications and freeware software

Real world applications of RL range from playing games to autonomous robots. Atari games have been learned with generative RL and Bayes rules [123], Monte-Carlo Tree Search [76] and EC [99]. A combination of Monte Carlo Tree Search and multi-armed bandits famously plays GO [66]. Othello [178,179], Pacman [12], and mazes have been successfully solved with Q-learning algorithms.

The current hype in RL is deep learning whose variants are adamant in playing games. In robotics, we find different components, such as vision or movement control, modelled with variants of RL: 1) model-based RL (state-action trajectories) is used in mobile robots [196], 2) Q-learning is used to control a robot arm [116], and 3) navigation of a robot using planning [173].

A number of software tools for the development of RL algorithms are freely available, and we give their summary in Table 2. Weka is a popular and general machine learning tool in Java that implements

Table 2
Reinforcement learning development (freeware) software tools.

Tool	Reference	Prog	Website
Weka – Waikato Environment for Knowledge Analysis	Frank et al. [62]	Java	cs.waikato.ac.nz/ml/
PyBrain – Python-Based Reinforcement Learning, Artificial Intelligence and Neural Network Library	Schaul et al. [148]	Python	http://pybrain.org/
DL4J – Deep Learning for Java		Java	https://deeplearning4j.org/lstm
OpenAIGym	Brockman et al. [17]	Python	gym.openai.com/

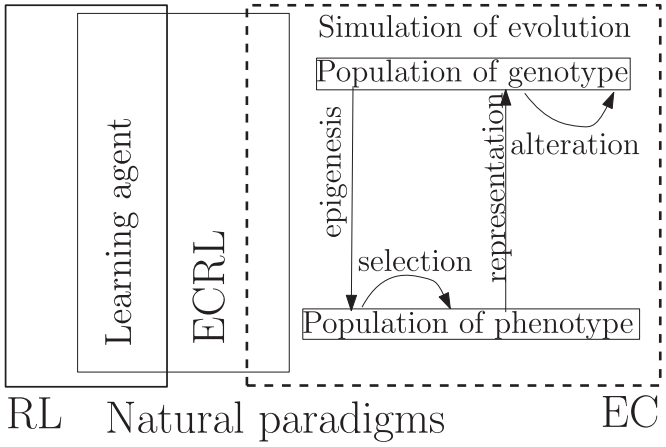

Fig. 3. Simulation of evolution was one of the motivations for EC. Individual genotypes are transformed through epigenesis into individual phenotypes; selection selects the best individuals from the population of phenotypes. The representation translates any phenotype back to its genotype, and alteration generates new genotypes and updates the population.

also several RL and EC methods. PyBrain is implemented in Python and contains a large number of RL and neural network techniques. DL4J focusses on deep learning implemented with neural networks. OpenAI is a library with lots of simulators and applications.

4. Brief introduction in evolutionary computation

In this section, we give an introduction to EC techniques in ECRL. Similarly with the introduction to RL, the presentation follows the pattern of our framework.

4.1. The intuition: simulation of evolution

EC has its roots and motivation in *simulation of evolution* [59] characterised by a set of individuals, or population, that interact. The population is maintained and updated according to naturally inspired mechanisms. Fig. 3 summarises the simulation of the evolution process.

Genetic algorithm (GA) is considered the most biologically accurate EC model. At first, a population of individual solutions is initialised uniformly at random. Every single solution contains a *genotype* that gives the underlying genetic encoding, and a *phenotype*, which encodes the behavioural response to the environment. The interaction between individuals occurs at different levels, i.e. genotype, phenotype or mixed, through (natural) selection and genetic operators, like recombination. *Epigenesis* is defined as a mapping between genotypes and phenotypes and includes rules of genetic structures, growing and development of cells under local context. Each gene corresponds to one phenotype; one phenotype contains many genotypes. A second mapping, called *selection*, represents the operations between phenotypes like selection, immigration, etc. The selection at phenotype level propagates in the genotype state space. Phenotypes are translated back to genotypes by *representation*. *Alternation* with genetics like operators, e.g. mutation and recombination, is the last mapping of an exact biological process, where the current genotypes are translated into novel genotypes.

Both evolutionary strategies ES and genetic programming GP operate directly on phenotypes; each phenotype is associated with a fitness value. In ES, the alteration is the substitute for self-adaptation such that the distribution of phenotypes generates new individuals. Like genetic algorithms, GP allows for recombination; two tree structures exchange sub-structures.

4.2. Variants of evolutionary computation

There are three major EC classes which emerged independently: 1) genetic algorithms [86], 2) evolutionary strategies [140,153], and 3) genetic programming [60,104]. Table 3 shows review papers on EC variants. A history on the beginnings of EC is presented in Ref. [35].

Genetic algorithms. GA is the most popular EC instance due to its intuitive usage and ease of implementation. A population of individual solutions is randomly initialised; each individual has a fitness value associated with a fitness function. The representation of an individual solution is the key for an efficient algorithm. GA was initially designed for binary string representations and later extended to real-valued strings and permutation problems. When apriori domain knowledge is available, GA is initialised with fit individuals for improved performance. Individuals that are mated or altered are selected each generation, with or without replacement, from the population. This first selection strategy is either uniform or proportional with the fitness distribution. The genetic operators have to be re-designed for each problem instance because of invalid resulting strings, or ineffective usage of current operators. The newly generated solutions are mixed with their parents, and a second selection round is applied by selecting the highly fitted individuals. The population is iterated until the stopping criterion holds, i.e. all the solutions in the population are the same, or the algorithm runs above a time threshold.

Evolutionary strategies. ES [10] considers continuous solution spaces, and uses mutation to generate new individual solutions and, afterwards, deterministically selects the best individuals. Manually tuning parameters is a time and resource consuming process. Therefore, ES considers that a well-performing optimiser tunes its parameters automatically. Covariance Matrix Adaptation Evolution Strategies (CMA-ES) [80] is a parameter free global optimiser for real-valued functions. CMA-ES generates new solutions from a multivariate normal distribution that is learned from the previous solutions. CMA-ES is popular due to its wide applicability on non-linear, non-convex, continuous optimisation functions and it is often used in combination with RL and supervised learning, i.e. for hyper-parameter tuning.

Genetic programming. GP [136] addresses the ambitious task of optimising computer programs. Each computer program is represented by a tree where the inner nodes are the operators, and the leaves are the variables. The genetic operators include nodes and sub-tree deletion and addition.

4.3. Analysis of evolutionary computation

Most EC algorithms converge to sub-optimal solutions for which the quality is determined by specific features. Landscape difficulty, the

Table 3
Reviews on EC algorithms.

EC paradigms	Review papers
Evolutionary computation EC	de Jong et al. [35], Back et al. [4]
Theoretical aspects of EC	Kallel et al. [94]
Evolutionary multi-objective optimisation EMO	Coello Coello [32], Deb [36]
Evolutionary dynamic optimisation	Nguyen et al. [129]
Genetic algorithms GA	Goldberg [74]
Estimation of distribution algorithm EDA	Hauschild and Pelikan [82]
Genetic programming GP	Poli [136]
Evolutionary Strategies ES	Beyer and Schwefel [10]

Table 4
Evolutionary computation development (freeware) software tools.

Tool	Reference	Prog	Website
ECJ23 – Java-based Evolutionary Computation Research System	White [184]	Java	cs.gmu.edu/eclab/projects/ecj/
PISA – A Platform and Programming Language Independent Interface for Search Algorithms	Bleuler et al. [11]	C++	tik.ee.ethz.ch/pisa/
jMetal – Metaheuristic Algorithms in Java	Durillo and Nebro [50]	Java	jmetal.github.io/jMetal/
ParadisEO – A software framework for Metaheuristics	Chaon et al. [28]	Java	paradiseo.gforge.inria.fr/
MOEA – A Free and Open Source Java Framework for multi-objective Optimisation		Java	moeaframework.org/
BorgMOEA – Borg multi-objective Evolutionary Algorithm	Hadka and Reed [78]	C++	borgmoea.org/
HyperNEAT – Hypercube-based NeuroEvolution of Augmenting Topologies	Gauci and Stanley [65]	C++	eplex.cs.ucf.edu/hyperNEATpage/
JavaXCSF – Learning Classifier System for function approximation	Stalph and Butz [163,164]	Java	

appropriateness of representation and the type of genetic operators strongly influence the performance measures. Approaches converging to global optima consider specific EC algorithmic instances, such as the standard GA that use the simplest sequence of mutation, recombination and selection. Standard GA keeps the best solution found in the current population, but does not converge to the optimal solution [146]. To prove the convergence to the global optimum, [53] proposes a simulated-like annealing mechanism for GA that gradually decreases fitness values using a temperature parameter. Some theoretical studies include in the analysis the function to be optimised; simple heuristics are competitive on simple functions, and challenging functions, i.e. deceptive functions, are challenging for any EC [188]. There are few EC instances, i.e. CMA-ES, that are transformed in learning algorithms with well studied theoretical behaviour, like natural evolution strategies [1].

The trade-off between computational time and the quality of the final solution is thoroughly studied for binary encoded GAs using the schema theory [86]. In the absence of mutation, one individual solution takes over the entire population in a finite number of generations. The take-over time gives an indication of selection pressure for a particular selection function [75]. The relation between the population size and the performance of GA is given in Ref. [81]. The number of individual fitness evaluations until finding the optimal solution defines the computation complexity. Landscape analysis is a methodology that correlates the performance of heuristics with the properties of the optimised problem [118].

EC's parameters depend on the algorithmic variant, whereas the running time of EC depends on the parameters used to tune the functionality of algorithmic components. For example, a high mutation rate means generating new individuals far away from their parents, whereas a low mutation rate explores the proximity of each individual. The selection pressure, the type of selection and the mechanisms to keep diversity, contribute all to the computational time of EC, complexifying the theoretical analysis of the practical algorithms.

4.4. Applications

The success of EC in practical applications implies a design phase to select the most appropriate EC variant and a test phase to select the best parameters. Between successful applications of EC we mention robotics [87,107,120], games [69,92] and control [132,141]. A large range of medical applications use variants of EC [108,195]. Some applications are common for both RL and EC, and thus prone to hybrid algorithms that exploit the advantages of both paradigms.

There are many (freeware) software tools for EC that are frequently utilised by both academia and companies; these software packages could be either specialised on particular EC paradigms or be a collection of generic methods. Table 4 reviews free software for EC. For example, the Java-based evolutionary computation research system (ECJ23) [184] is a flexible and generic software tool for EC. jMetal [50] is a Java toolbox that collects evolutionary algorithms for multi- and single objective problem instances. A Java framework for meta-heuristics ParadisEO [28] is a combination of EC and local search. Notable usages of ParadisEO for applications include the optimisation of cellular networks in mobile telecommunication systems [171] and the optimisation of layers of conducting polymers for protecting the material from the proliferation of electromagnetic inferences [152]. BorgMOEA [78] is an adaptive evolutionary multi-objective tool successfully used for decision making in the management of drinking water [141]. As an example of specialised EC software, we recall Hypercube-based NeuroEvolution of Augmenting Topologies (HyperNEAT) [65]. Other applications for neuroevolution [122] include optimisation in games and evolvable systems.

5. A taxonomy on the conceptual classification in ECRL

In this section, we develop on the classification criteria of the ECRL paradigm based on the unified methodology of analysis introduced in Section 2. The conceptual principles are: 1) the *motivation* for a particular algorithm, 2) the *intuition* given by the natural models underlying each concept, and 3) the *focus* of the hybrid algorithms. Fig. 4 specifies the taxonomy for this conceptual classification.

5.1. Practical motivation

Any new algorithmic technique is motivated by at least one of the following reasons: 1) a new problem, 2) the computational efficiency or 3) the theoretical soundness.

New applications. In our view, novel problems or environments have a very broad meaning. For example, a *novel setting* for a well-studied problem is the use of RL environments with reward vectors instead of reward values in the multi-objective RL paradigm [111]. Or, the *relaxation of constraints* of a Markov decision process (MDP) in RL to non-Markovian processes in the evolutionary function approximation for RL algorithm from Ref. [186].

Effectiveness. Improving the computational efficiency is the motivation for many algorithmic studies. EC and RL cooperate to solve challenging computational problems from robotics, like real-time control, online learning and constraint satisfaction. Online adaptation of the probability to select a genetic operator, or adaptive operator selection, is the most common method to generate EC algorithms with adaptive behaviour [51,73,159,172].

Theoretical soundness. There are few theoretical studies to guarantee the behaviour of ECRL methods. For example, there are no theoretical bounds for the adaptive operator selection, even though the RL component that learns the distribution of the operators converges to the corresponding stationary distribution.

5.2. Natural intuition

Natural paradigms are often invoked as motivation/intuition for EC and RL algorithms. Natural models could represent a novel selection

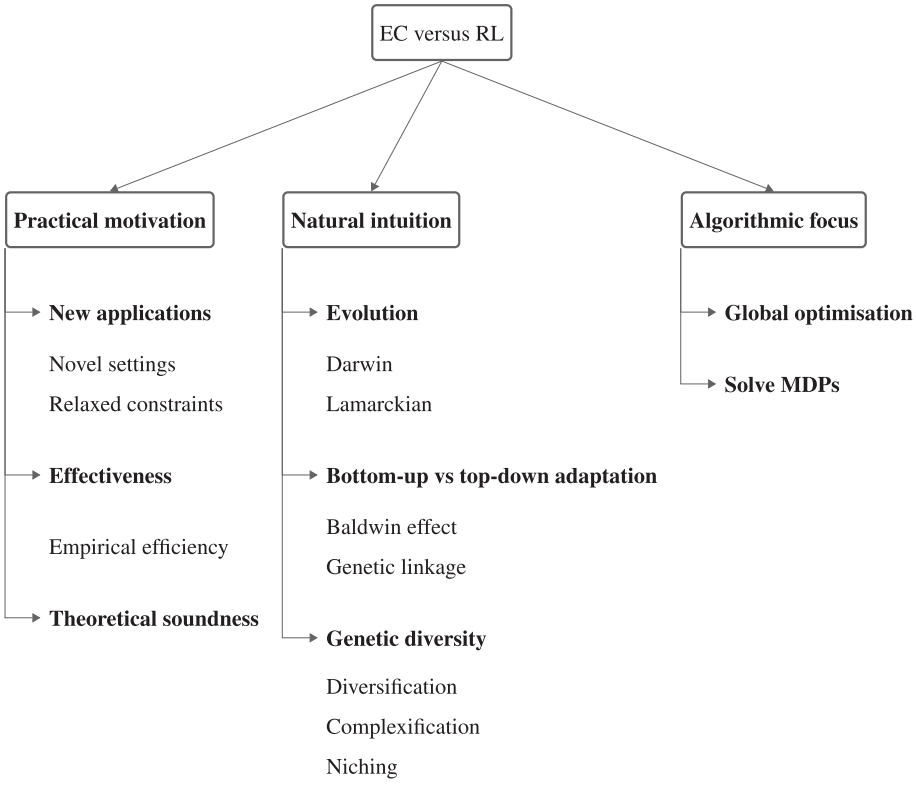

Fig. 4. A taxonomy for the conceptual classification of ECRL paradigms.

technique, interaction methods between agents or a new focus on evolution. We have paired the presented paradigms to highlight their antagonist effect. The following list of algorithmic instances is non-exhaustive; the reviewed ECRL models include at least one of these paradigms.

Darwinian versus Lamarckian evolution. The Darwinian evolutionary theory and its relation with the simulation of evolution and genetic algorithms are detailed in Ref. [2]. Lamarckian evolution is combined with a simple variant of GA in Ref. [7].

With the Darwinian evolution, the fittest individuals are selected for the next generation. An example of selection strategies in the standard genetic algorithms is the proportional selection, which chooses individuals proportionally with their fitness value. In antithesis to Darwinian evolution, Lamarck argued that organisms change with phenotype: the newly acquired skills are needed for survival, and the capabilities that are not used are lost.

Darwinism assumes that living creatures differentiate due to variations resulting from random mutations in their genes. Lamarckian evolution pretends that the capabilities acquired by individuals during their lifetime can be passed to their offsprings.

Darwinism assumes that the children inherit the parents' characteristics. Lamarck premises that organisms evolve from simple to complex, whereas Darwinism considers only the survival of the fittest.

Bottom-up versus top-down adaptation. In nature, the adaptation occurs bottom-up, meaning that the population assimilates the lessons learned by the individual solutions. In simulated evolution, i.e. evolutionary computation, successful algorithms often have a top-down approach. Genetic linkage specialises the impact of learning from the population to individuals. In contrast, the Baldwin effect has a bottom-up adaptation that generalises learning from individual solutions to their community. The Baldwin effect is the ability of lifetime learning of individuals for adaptive advantages of the population [176]. The Baldwin effect studies the behaviour in groups of individuals rather than on the individual itself. The Baldwin effect has two steps. In the first step, learning can accelerate evolution with favourable selection pressure. But since learning has a high cost, the second phase represents the selection pressure in states where learning is irrelevant. Often, the Baldwin effect is combined with Darwinian or Lamarckian evolution. In Ref. [189], the combination of heuristics and the Baldwin effect converges to the global optimum, whereas the Lamarckian evolution converges to the local optimum.

The genetic linkage is the process of grouping alleles that are similar according to some similarity measure. The competent genetic algorithm uses the genetic linkage to generate probabilistic models of the fit individuals [175]. The estimation of distribution algorithm (EDA) maintains a Bayesian model of data; each generation new solutions are created from the model [127].

The genetic diversity. Diversification introduces new individuals in the current population to avoid premature convergence to a low-quality individual solution. Diversification is complementary to mechanisms as genetic linkage that models the information in the newly generated solutions. The representation generated by the genetic linkage techniques corresponds to exploitation, and diversification is an exploration method. There are several ways to promote and maintain the diversity of the genetic material in the population.

Niching [156] keeps multiple sub-populations converging simultaneously to several individual solutions. Niching was motivated initially by landscapes with multiple optima where the standard GA would converge to a single solution. Niching splits the population into several sub-populations that interact only at certain moments in time by exchanging individuals. Two solutions from different sub-populations are distanced compared with the solutions from the same population. Niching had such a success that the mechanism was borrowed by other EC like evolutionary multi-objective optimisation and evolutionary strategies.

5.3. Algorithmic focus

EC and RL belong to two different types of algorithms. RL is part of the larger group of machine learning algorithms, whereas EC is a global optimisation technique. RL solves environments modelled as MDPs by rewarding good actions and punishing bad actions. Most EC techniques are empirical studies with no theoretical guarantees confirmed only by the goodness of the returned solution. Some hybrids, such as neuroevo-

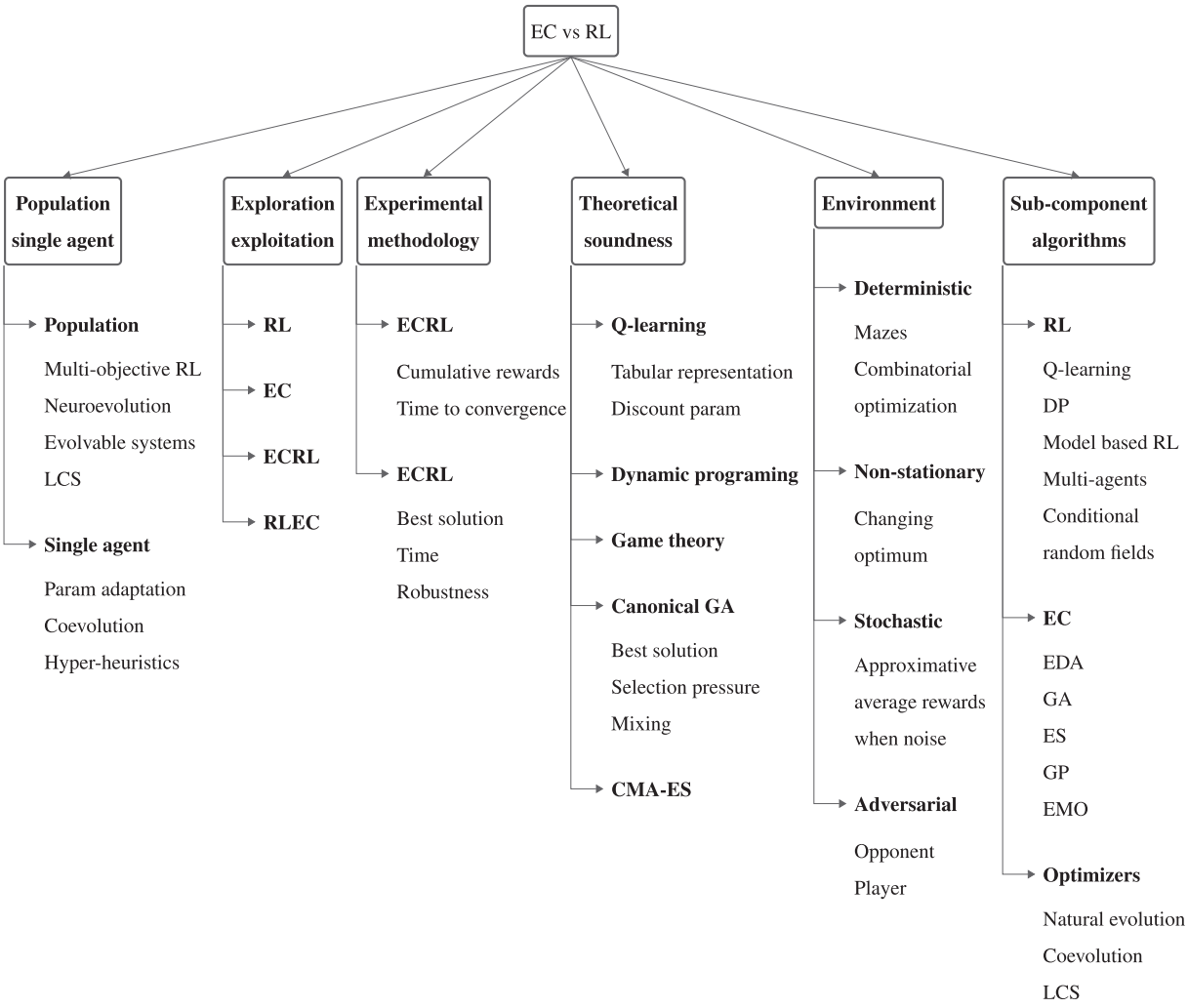

Fig. 5. An analytical taxonomy for evolutionary computation versus reinforcement learning.

lution and multi-objective RL, could be classified as RL methodologies with well preserved underlying properties; other approaches, such as parameter free GA and hyper-heuristics belong to EC. Few ECRL paradigms can not be classified as either RL or EC, like co-evolution and natural evolution strategies. The vast majority of ECRL paradigms, however, are methodological studies with properties inherited from their compounds. Some ECRL are EC with learning components.

6. A taxonomy on the algorithmic analysis of ECRL paradigms

In the previous sections, we have described some differences between EC and RL at a high, conceptual, level. Here, we extrapolate these properties to the general ECRL framework by classifying ECRL given their focus and motivation. Fig. 5 specifies the taxonomy for the proposed algorithmic analysis.

Population of individuals versus single agent. EC uses a population of solutions exchanging information with mechanisms resembling recombination and mutation from genetics and biology. Note that a population of solutions is not the current practice in optimisation and machine learning techniques, including RL, which updates a single solution at the time.

The exploration/exploration trade-off. An important aspect of RL is the exploration/exploitation dilemma [93]: The agent should explore actions for which the outcome is still uncertain, whereas exploitation refers to selecting actions which have shown to be good in the past. In a realistic environment with limited resources, an RL algorithm that explores a lot has little time left for exploitation and vice-versa. The parameters involved in tuning this dilemma are sometimes optimised with EC. A recent review of the exploration/exploitation methods for EC is provided in Ref. [34].

In EC, exploration implies the evaluation of new solutions that could have poor fitness and the exploitation means the usage of already known good solutions through selection. Thus, the exploration/exploitation trade-off is an attribute of successful adaptation in both RL and EC. Selecting and using these trade-off strategies are not trivial and actually, they can increase the time needed to find an acceptable solution.

Experimental methodology for hybrid ECRL. The experimental section of RL papers includes a comparative analysis of their cumulative rewards; the best algorithms have the largest long time reward. In infinite time, the algorithm reaches the optimal policy for which the total reward is maximised. The stopping rule for RL is often empirical and relies either on a maximum computational budget, time limit, or the stagnation of improvement acknowledged with graphical methods.

The performance measures focus on the best individual solution achieved in limited time and during a single run, or the percentage of occasions the returned solution reaches a threshold value. All the other solutions except the best solutions are discarded. ECRL either optimise, like EC algorithms, or learn, like RL methods, with new constraints or in new environments. For example, adaptive pursuit [172] often pursuits the best EC operator with an RL kind of algorithm.

Table 5
Overview on the review papers on the evolutionary computation versus reinforcement learning paradigms. Some ECRL paradigms are recent, such as hierarchical bandits, and do not have reviews.

EC versus RL	Review papers
EC for RL	Moriarty et al. [125], Schmidhuber [149]
Learning classifier systems for RL	Bull [22], Lanzi [109]
NeuroEvolution	Miikkulainen [122]
Evolvable systems	Whiteson [185], Doncieux et al. [40]
Parameter control in EC	Lobo et al. [114]
Generic parameter control	Karafotias et al. [98]
Hyper-heuristics	Burke et al. [23]
Evolutionary multi-objective RL	Liu et al. [111], Roijers et al. [142], Drugan [44]
Evolutionary dynamic optimisation	Nguyen et al. [129]
Co-evolution	Krawiec and Heywood [105]

The experimental behaviour of adaptive pursuit greatly depends on the intrinsic learning parameters. Hence, another indicator is the robustness in parameter variations.

Theoretical soundness. Q-learning for tabular representations is proven to converge with geometrical rates to the optimal policy [182]. Various analytical studies are conducted on a simplified version of the real world problems. Multi-armed bandits [71] is a simplified mathematical formalism with a single state and multiple actions. The multi-armed bandits studies upper and lower bound the convergence rates and convergence times in a multitude of contexts.

The canonical genetic algorithm [146] converges to the optimal solution, but the example used is simplified and impractical. Alternatively, algorithmic analysis includes selection pressure [29], sizing of the population [81], the mixing time of building blocks [31], and the geometry of the optimised function [188].

The goal of the ECRL algorithms dictates the soundness of the technique of interest. Hybrids with a new focus, like multi-objective reinforcement learning, require new tools for theoretical analysis.

The type of environments. The environment gives another classification criteria for ECRL. Deterministic state spaces could have a finite number of solutions or states could have continuous values. In EC, deterministic environments are challenging either because of their large state space, whereas in RL, the difficulty comes from the task to the solved, such as finding the exit in a maze. A multi-objective stochastic environment is generated from multivariate distributions with mean reward vectors.

A stochastic environment generates solutions using random, e.g. uniform or (sub)-Gaussian, distributions concentrated by unknown means. The task of an RL algorithm is to generate accurate solutions despite the noise in the environment. In some environments, the generating distribution of a solution changes with time. Evolutionary dynamic optimisation tracks the changing optimal solution, whereas RL mechanisms use the discounted factor that weights the current rewards more than future rewards.

Adversarial environments assume the existence of an opponent that maximises a different reward function than the agent. The adversarial environments are used to model games and are often considered the most general environment since it does not make assumptions on the reward functions or the Markov properties.

6.1. Sub-component algorithmic techniques of ECRL

Consider classes of algorithmic instances for each paradigm and all their possible combinations. For example, the learning classifier systems model incorporates GA into a variant of RL, and the adaptive operator selection fuses the adaptive pursuit with GA. Because of the assembling methodology, similar EC and RL techniques have a very different output when mixed. Not all combinations have actual algorithmic instances, because of the differences in representation or conflicting conditions. Furthermore, the unsuccessful experimental work often remains unpublished, and the experimental studies outnumber by far the theoretical research. Therefore, it's hard to enumerate all the possible ECRL instances; the backbone of this article is the success stories in the ECRL framework. Table 5 summarises the representative ECRL algorithms and their reviews. In fact, the next sections are all accompanied by tables with the reviewed algorithms and their decomposition in sub-component algorithms. As for combining paradigms, we distinguish between either integrating EC in RL or blending RL in EC.

Reinforcement learning in EC. RL is often used to improve the performance of EC algorithms. Hierarchical bandits use Monte Carlo tree search for learning in continuous search spaces [138] or with binary settings [42]. Multi-objectivisation solves a single-objective RL by adding objectives. Tuning the parameters of a complex optimisation algorithm such as EC is a complicated process done preferably in an automated way. Self-adapting GA strategies [56,172] use a greedy strategy to select an operator, and adapt the selection probability distribution based on feedback.

Evolutionary computation in RL. EC techniques are sometimes used to evolve RL policies that scale up RL to practical tasks such as vast state spaces, partially observable environments, rarely occurring events and non-stationary environments. Instead of using a single agent, a population of agents is used and evolved, having the advantage of better dealing with changing environments or local optima. EC for RL methods [185], like several versions of CMA-ES [83], are direct policy search methods that optimise a set of policies. GPs have been used for feature discovery, meaning to evolve sets of features representing states of RL [70].

General purpose optimisers. There are hybrid algorithms that are neither RL or EC, and they can be considered general purpose optimisers. Natural evolution strategies [192] are black box optimisation algorithms for continuous spaces. It is interesting to note that there is proven the equivalence between CMA-ES and natural evolution strategies [1]. The learning classifier systems [86] (LCS) precedes genetic algorithms; LCS evolves a set of rules to characterise an agent with the explicit goal of maximising reward intake. From a game theory perspective, adversarial RL closely resembles competitive coevolutionary EC [85], i.e. prey - predator models.

7. A practical example: Q-learning vs canonical GA

Next, we elaborate on the similarities and contrasts between RL and EC. At first, we briefly present Q-learning, cf Algorithm 1, and the canonical GA, cf Algorithm 2. Both algorithms have well understood theoretical and empirical properties, and are extensively used in practice. Then, we exemplify the combination between these two common EC and RL models with an instance of LCS, cf Algorithm 3, and neuroevolution, cf Algorithm 4. This section concludes with a discussion on the reviewed algorithmic methods.

Algorithm 1 Q-learning algorithm.

```
Initialise Q-values arbitrary ;
for each episode do
    Initialise s as the starting state ;
    repeat
        Select action a in state s ;
        Perform action a ;
        Observe a new state s' and
         receive reward r_t ;
        Update Q(s, a) values ;
        s' ← s ;
    until s' is a goal state;
```

Algorithm 2 Genetic algorithm.

```
t ← 0 ;
Evaluate a population of random
 solutions P^(t) ;
while termination condition not
 satisfied do
    Select solution(s) {x_1, ..., x_p} to
     be altered ;
    Generate children {y_1, ..., y_c}
     with operators ;
    Evaluate children individuals ;
    Select best individuals in P^(t) ;
    t ← t + 1 ;
```

Algorithm 3 Holland's LCS.

```
Initialise a population of random
 "state-action" rules ;
while not terminated do
    Generate classifiers to describe
     the current state s ;
    Match in M(s) rules containing s
     Evaluate the rules in M(s) ;
    Select a with the highest
     activation in Q-learning;
    Update the total value for rules in
     M(s) ;
```

Algorithm 4 NEAT-Q algorithm.

```
Initialise a population of random
 neural networks ;
for a given number of generations do
    Generate a network N ;
    while not terminated do
        Evaluate N on current state s ;
        Select action a with the
         highest activation ;
        Update the new state given a ;
        Update weights of N using
         Q-learning ;
        Update the total reward of
         network N ;
    Update current population ;
```

7.1. Q-learning

Q-learning is a popular model-free RL that estimates the value of taking action a in a state s incrementally based on immediate rewards and the current Q-value function [182]. Algorithm 1 learns a policy π while searching for the optimal policy π^* independently of the agent actions. The Q-learning algorithm's inner loop selects an action based on Q-values and an exploration strategy, i.e. the ϵ-greedy policy selects with high probability the best action, and with a small chance a suboptimal action. After performing the action a in the current state s, a new state s' is observed, and the agent receives reward r_t. The Q-values are learned through the update rule

$$Q(s,a) \leftarrow Q(s,a) + \alpha \cdot \left(r_t + \gamma \cdot \max_{a'} Q(s',a') - Q(s,a) \right) \quad (2)$$

The system converges to the optimal policy π^* under mild conditions: the environment is Markovian, the agent gradually decreases the learning rate, and the state-action pairs have a tabular representation.

7.2. The canonical genetic algorithm

Algorithm 2 shows the pseudo-code for the canonical GA. At first, a population of individual solutions $P^{(0)}$ is generated using some random distribution. The algorithm iterates until it fulfils the termination condition, i.e. all solutions in the population are of the same type, or no improvement is detected for several generations. One or several parent solutions $\{x_1, \ldots, x_p\}$ are selected from the current population $P^{(t)}$ to generate one or several children solutions $\{y_1, \ldots, y_c\}$ using genetic operators like mutation and recombination. *Mutation* is a local perturbation that generates new solutions (also called individuals) in the proximity of the current solution. *Recombination* starts from two (or more) parent solutions and exploits their current structure given by building blocks to generate one or more offspring solutions. The individuals on which mutation and recombination generate are called children and the target of the stochastic selection process. Each child is evaluated using a fitness function; the least fit individuals in the current population are replaced with competent children. The algorithm iterates with this updated population $P^{(t)}$.

7.3. Examples of ECRL instances

There are many possible ways to combine these two basic algorithmic instances, aka Q-learning and canonical GA.

7.3.1. Genetic algorithms for policy search

In early years, GA was often used for policy search in RL. An example of canonical GAs with individual agents that interact is described in Ref. [125]. In Algorithm 1, each agent is represented as a string of state-action pairs; the average total reward of the policy represents its fitness value. A mutation operator considers a non-zero transition probability from each state-action pair to another action in that state. The one point recombination operator exchanges action-state pairs between policies, meaning that the first part of a child policy is generated with one parent and the second part by another child. The goal of this GA is to find the optimal policy. In a similar setting [121], Q-learning is combined with local search to solve combinatorial optimisation problems. There is made a distinction between states and "coarse"-states over a local part of the search space where the principle of locality applies. Both approaches consider a tabular version of Q-learning with a small number of possible state-action pairs. The modular Q-learning [130] evolves policies using sub-populations focused on a specific sub-task.

7.3.2. Learning classifier systems and Q-learning

There are many variants of LCS with different encodings for agents, e.g. as classification rules, and with different operators and selection rules. Each classifier C is associated to a prediction value p, a prediction error ϵ, and a fitness value f. A set of classifiers describes the current state s. The matching set $M(s)$ is defined as the set of rules from the population that contains state s. The system uses twice Q-learning. For each action a in M, the prediction set $p(a,s)$ is updated using "bucket-brigade" that is a Q-learning variant. The accuracy is defined as a function or prediction error ϵ; both the accuracy and the rule values are updated with Q-learning. Classifiers with higher accuracy have more offsprings than the less accurate classifiers.

In the *Michigan* approach, each individual is a "state-action" rule with binary encoding that competes for space and priority in the population. An entire population represents a policy which is a cooperative set of rules. Algorithm 3 gives the pseudo-code of Holland's LCS. In the *Pittsburgh* approach, each policy is described by a single individual with a variable set of rules. Each gene in a chromosome is a condition-action rule that maps a set of states to an action, and thus a policy is a set of state-action rules. The fitness is the individual performance of the policy [91].

Discussion. There are pros and cons for each of these two representations. With only one Q-learning component, the Pittsburgh approach has a simpler behavioural control than the Michigan approach, which implies a strict command of the evolutionary process in generating the best policy. Further, the Pittsburgh approach is a straightforward combination of GA and Q-learning, whereas the Michigan approach considers several GA populations that could be evolved separately.

7.3.3. NeuroEvolution of Augmenting Topologies with Q-learning (NEAT+Q)

NEAT + Q back-propagates the value estimates using Q-learning [187]. Algorithm 4 shows the pseudo-code of NEAT + Q in growing network structures for neuroevolution. Each input node of a network is a state feature such that their values represent the agent's state. An action value is computed using outputs of the neural network. Each node has associated a weight in a neural network, and a network can have several layers of hidden nodes. NEAT starts with simple, random networks, to which both edges and nodes can be added to improve the accuracy of the function approximation. Similar solutions could have entirely different topologies, and thus innovation has to be protected with speciation such that the evolutionary process is not too destructive.

Discussion. Let's compare NEAT + Q and GA, cf. Algorithm 2. The main differences are 1) NEAT + Q is specialised in optimising neural networks with all consequences in the generation of new individuals, and 2) the inner loop of NEAT + Q contains a Q - learning algorithm for updating the Q-values of a network. Thus, NEAT + Q can be considered an instance of the generic EC framework, and at the same time, NEAT + Q can be regarded as a generalisation of Q-learning.

7.3.4. Controlling genetic algorithms' behaviour with Q-learning

The evolutionary process of GA is sometimes controlled with Q-learning. In Ref. [135], an entire GA population represents a state, whereas an action represents the probability to generate another population with mutation and recombination operators.

7.4. Algorithmic contrasts and similarities

Let's first compare Q-learning and GA algorithms, cf Algorithm 1 and 2, respectively. The selection mechanisms are similar although the representation and the adaptation are dissimilar because they are both optimisation paradigms, which select the best individual, or policy.

Representation. The two algorithms, GA and RL, have different representation needs and concepts. When the state-action pairs are limited, the Q-learning values are stored in a table updated each iteration. This Q-learning variant is denoted as tabular Q-learning. In GA, each individual is a string of bits or real numbers that are assigned a fitness value entirely determined by its representation. The GA's search space is often too large to enumerate. Since there is no population, there is no equivalent for the genetic operators in RL.

Adaptation. Q-learning has a single agent that learns in episodes. GA has a population of individuals that interact through recombination and selection mechanisms. An individual is removed when it is replaced with a better, stronger individual. Thus, in GA, an individual solution is not evolved, the entire population of individuals adapts over time.

Selection. The selection mechanisms are quite similar for the two strategies, since RL as well as EC pursuit fit solutions. In RL, an elitist selection mechanism often chooses the best action for a state, whereas GA replaces parents when they are better than their children. A probabilistic selection is selecting the best action in RL, or individual in GA.

Global convergence. When the transition distribution and the rewards are known, the convergence of an RL in an MDP to the optimal policy resumes to the properties of the controlled Markov chain.

Similarly, Markov chain Monte Carlo is used to prove the global convergence of some GA instances [146]. The entire population of individual solutions $P^{(t)}$ represents a state of a Markov chain. Let ℓ be the size of a particular bit string, where ℓ a positive integer. A novel solution y_i is generated from x_i by flipping each bit with the probability p_m, where $p_m \in (0,1)$. There is a non-zero probability of generating any solution from another solution. Consider a probabilistic selection rule; the transition probability from each population to another population is non-zero. The speed of convergence is supplied by the second eigenvalue of the transition matrix.

8. Evolutionary computation for reinforcement learning

EC techniques evolve RL policies that scale up RL to realistic tasks such as vast state spaces, partially observable environments, rarely occurring events and non-stationary environments. Using EC is, however, not always computationally advantageous: dynamic programming can optimally solve problems in polynomial time, whereas EC has an unbounded running time. Furthermore, value iteration performs poorly in the environments with continuous spaces where not all state-action pairs can be characterised. The number of papers on the evolutionary computation for reinforcement learning developments shows the large interest in these techniques [125,185]. Table 6 shows presents some references on ECRL.

Table 6
Evolutionary computation for reinforcement learning: the ECRL algorithmic instances, their RL and EC sub-components and the underlying natural models.

ECRL	RL comp	EC comp	Natural paradigm
Learning classifier systems (LCS) [22]	Q-learning	GA	Lamarckian evolution
Population of agents (GA + RL) [121]			Lamarckian and Darwinian evolution
Evolutionary Q-learning [6]			
NeuroEvolution of Augmenting Topologies (NEAT) [122] [166]	Direct policy search		Lamarckian evolution, complexification, speciation
NEAT with Q-learning (NEAT + Q) [185]	Q-learning		
Estimation of distribution algorithms for RL (EDA-RL) [79] [133]	Q-learning	EDA	Linkage-learning, Lamarckian evolution
CMA-ES for RL [186]	Direct policy search	CMA-ES	Darwinian evolution
CMA-ES for policy improvement [168]			
QGP [95]	Q-learning	GP	Lamarckian evolution
Reinforced genetic programming [41]			
Evolutionary hierarchical RL [55]	Hierarchical Q-learning		
Genetic network programming with RL [117]	Q-learning	GNP	
GP for function approximation [70] [88]	Direct policy search		

8.1. Practical motivation

The class of EC for RL are practical algorithms that enhance the performance of RL.

Automatic knowledge representation and discovery. Learning classifier systems (LCS) is a general algorithm for function approximation and their usage for reinforcement learning was advocated in Ref. [160]. Learning classifier systems (LCS) [13,22,86] evolves if-then rules called classifiers that map input states to actions. LCS algorithms are in the same time: 1) GAs with populations of rules that use genetic operators for rule discovery, and 2) a variant of Q-learning that learns how to interact with the system (or bucket brigade). LCS performs knowledge discovery, i.e. automatically discovering redundant state features to ignore [26], but performs quite poorly on discontinuous domains because of the representation of abrupt decision boundaries [103].

In Ref. [70], GP evaluates both policies and value functions represented as linear combination of features, and learns how to play Tetris.

Evolvable systems. Evolvable systems include an extensive range of algorithms from artificial life systems to social robots and evolvable hardware [40]. A self-adapting differential evolution algorithm is used to evolve controllers for a hexacopter [87].

In Refs. [83,126], CMA-ES automatically and simultaneously evolves weights and topologies of neural networks with covariance matrix adaptation ES (CMA-ES). CMA-ES is used in Ref. [88] to play the complex game of SZ-Tetris.

NeuroEvolution of Augmenting Topologies (NEAT) [165,167] evolves topologies and weights of a neural network that represent policies. EC evolves neural networks to approximate complex functions [186]. NEAT neuro-controllers proved their efficiency in robot control [49,166] to solve tasks like controlling a robot arm, which can avoid stationary or moving obstacles. NEAT has also been used for playing GO [167] and pole balancing [165].

Following the main trend in machine learning, deep learning which combines neural network architectures are further combined with EC variants. The non-dominated sorting genetic algorithm (NSGA-II) [37] is amalgamated with HyperNEAT [165] for playing the game of Tetris [69]. Network problems are optimised with deep neural networks and neuroevolution in Ref. [100]. The parameters of deep neural networks are optimised online in Ref. [115].

GP is often used with RL for solving a task with autonomous agents such as carrying a box across a room [95].

LCS, e.g. Pittsburg approach, is used in Ref. [120] for online real-time motion learning in robots. The first step of the hybrid algorithm is to adjust manually the representation of individual solutions of LCS; the output of LCS are evolved rules that feed a standard Q-learning algorithm.

Combinatorial optimisation. The population of RL that interacts through genetic operators is combined with local search to solve various combinatorial optimisation problems. NP-hard instances of the asymmetric travelling salesman and quadratic assignment problem are optimised with ECRL methods that encode local search as actions in RL and use GAs to evolve the corresponding agents [121].

8.2. Taxonomies

Representation. A policy's representation depends on the problem that RL needs to solve. In the most intuitive string encoding, a policy is a set of state-action pairs where its fitness function is approximated by a series of Monte Carlo trials. Each policy is described by a genotype/phenotype encoding such that new policies could be created with EC operators, e.g. real-valued operators are applied to strings of weights for neural networks, and learning classifier systems use triggered recombination. When the number of states is quite large, a mere enumeration of all possible state-action pairs is impractical. A distributed rule-based representation of a policy considers several EC algorithms evolved separately; each chromosome represents a set of rules and a strategy mingles several chromosomes from independent populations. Because an agent continuously interacts with the environment, the fitness values are averaged over time to obtain an equivalent of fitness function from EC. The best policies are scattered to the next generation.

The RL sub-components. We distinguish between the following ECRL types: 1) The direct policy search methods evolve entire policies by mapping state features to actions, such as CMA-ES for RL and NeuroEvolution of Augmenting Topologies (NEAT). 2) Value function between methods, mainly Q-learning construct estimates for state-action pairs using EC, like learning classifier systems and NeuroEvolution of Augmenting Topologies with Q-learning (NEAT + Q). 3) Model-based RL learns the associated model of the environment with EDAs or GPs.

The EC sub-components. RL policies evolved with EC have a large number of EC variants: 1) GAs are extensively used by early variants of EC for RL, such as NEAT with a real-valued GA. An interesting historical fact is that learning classifier systems (LCS) precedes EC algorithms and, in fact, EC was proposed as a special case of LCS. 2) EDA-RL uses a probabilistic model for computational efficiency. 3) CMA-ES is a very efficient algorithm for real-value optimisation and thus a logical choice for direct policy search, or function approximation. 4) GP is combined in several ECRL variants, such as QGP and GP for *RL*.

The simulation of evolution sub-component. Complexification is employed in learning NEAT controllers by generalising from simple to complex structures. Complexification [166] assumes that the purpose of evolution is finding the optimal structure that could be complicated rather than simply the optimisation of a fitness function. Speciation,

which is a variant of niching, is commonly used in neuroevolution to protect innovation [185]. Also in neuroevolution [186], the artificial neural networks inherit well-performing weights from their neural network parents using Lamarckian evolution.

Some direct policy search algorithms, i.e. CMA-ES for RL [83], select the best strategy from a population; thus they use Darwinian evolution. Reinforced genetic programming [41] enhances GP with both Baldwin and Lamarckian evolution using a Q-learning type of selection mechanism. Instances of learning classifier systems [193] use Lamarckian evolution to encode improvement, such as the distance between the current and the best individual solution, in the genotype.

8.3. Algorithmic instances of evolutionary computation for reinforcement learning

Estimation of distribution algorithms for reinforcement learning (EDA-RL) estimates policies with conditional random fields (CRF) [79]. *Conditional random fields* (CRF) uses Markov networks to generate conditional probability distributions to estimate policies. EDA-RL patterns the interaction models between the agent and environment with a Markov network that, similarly with EDA, learns the conditional probability distribution to estimate parameters describing the transition model. In linear parameterised CRF, an episode is represented as a sequence of state-action pairs and the corresponding transition probabilities from the current to the next action. The log-likelihood function computed in each phase is used to assess the performance of the graphical model.

RL is often used to improve the performance of GP. Reinforced genetic programming combines genetic programming and Q-learning, and evolutionary Monte Carlo tree mixes Monte Carlo tree search with ECs [41]. Consider a GP structure; each internal node is federated with an RL instance such that the adaptation occurs at a node level. Hierarchical reinforcement learning methods, such as MAXQ, have been combined with GP for improving performance [55].

9. Reinforcement learning for online control in evolutionary computation

Next, we summarise the ECRL methods that control EC algorithmic instances with RL to increase EC's performance. We distinguish several techniques for online adaptation: 1) online parameter control for EC [114], and 2) hyper-heuristics that generate and select heuristics to control performance. Table 7 presents literature on ECRL.

9.1. Practical motivation

Online parameter control. The environment's properties are explored using RL, as opposed to off-line operator selection where the parameters of each operator are set before usage. The motivation for online parameter adaption comes from the challenging and important task of parameter tuning in EC; we name as parameters: the mutation and recombination rates, population size and selection of parents for genetic operators [98]. Mutation and recombination operators have different competences in various regions of the landscape. Small size populations are computationally inexpensive but could severely diminish the performance of EC due to high selection pressure and premature convergence.

Practical applications of adaptive operator selection includes smart buildings design [58], and combinatorial optimisation problems [24]. RL is currently used by many robotic systems, thus, the combination between EC like techniques and RL arises naturally. Social robots [77] use EC (CMA-ES) and RL algorithms to control autonomous robots with social behaviour that need to solve a simple task like crossing rooms and to avoid obstacles where knowledge is shared locally between friendly robots.

Combinatorial optimisation. The performance of heuristics on combinatorial optimisation problems is regulated by optimising the sequence of parameters or the sequence of heuristics [23]. Hyper-heuristics are an ensemble of heuristics that automatically adapt to create the optimal ensemble of heuristics to search for the optimal solutions. Thus, unlike meta-heuristics that search directly for the optimal solution(s) using a mixture of EC and local search methods, hyper-heuristics learns, in most of the cases, a sequence of simple heuristics with standard techniques from RL, like Q-learning. Hyper-heuristics selects the best heuristic for a given task and have various applications like artificial immune systems [157], combinatorial optimisation problem instances [161].

9.2. Taxonomy

Online parameter control. The parameter control methods can be classified according to the number of adapted parameters, but also after the type of parameters. Some algorithms adapt only one parameter that is considered the most important for a good performance. Adaptive pursuit [172] and multi-armed bandits [56] are used for online mutation or recombination operator selection. In Ref. [46], a variant of multi-objective RL optimises two parameters at once, whereas [96] uses a variant of Q-learning to optimise all the parameters at once.

Hyper-heuristics. An overview of hyper-heuristic methods that use RL to automatically design meta-heuristics for optimising hard problems is given in Ref. [23]. There is a large variety of heuristics alternated with hyper-heuristics, but most studies use Q-learning to adjust the sequence of heuristics to the best performance. Hyper-heuristics depends on the particular problem applied on. There are two main classes of hyper-heuristics: selection and generation of heuristics. It is interesting to note that when the component heuristics differ in a single parameter, a selective hyper-heuristics is similar with online operator selection.

The RL sub-component. In our knowledge, actual hyper-heuristics uses exclusively Q-learning. The range of RL variants for online parameter control is considerable broader including one state RL, i.e. multi-armed bandits and adaptive pursuit, on policy and off-policy algorithms, like Q-learning and SARSA.

The EC sub-component. The online parameter control is exclusively utilised to genetic algorithms; CMA-ES has a different way to adapt to the search space whereas GP's challenges are related to tree structure rather than the parameter of genetic operators. Hyper-heuristics, however, include artificial immune systems, which uses a sort of co-

Table 7
Reinforcement learning for online control in Evolutionary Computation: algorithmic instances and sub-components.

Alg class	ECRL	RL sub-component	EC sub-component
Adapt a single parameter	Adaptive pursuit for operator selection [172]	Adaptive pursuit	Genetic algorithms
	Upper confidence bound for online operator selection [56]	Upper confidence bound	
Multiple operator selection	Adaptive multi-operator meta-heuristics [46]	Adaptive pursuit	Evolutionary multi-objective optimisation
	RL control of GA [135]	Q-learning	Genetic algorithms
Adapt all parameter at once	Generic parameter control with SARSA [96]	On-line policy SARSA	ES, GAs
Selective Hyper-heuristics	Great deluge hyper-heuristics [131]	Function approximation	Meta-heuristics
Generative hyper-heuristics	Adaptive representation control [155]		

evolution, and genetic programming.

Natural paradigms. There is not much work on the natural paradigms for control of EC, mostly because the adaptation mechanism is RL that is always represented by the interaction between an agent and the environment.

9.3. Algorithmic instances for controlling EC with RL

9.3.1. Online operator selection

Adaptive pursuit for online parameter selection. Adaptive pursuit (AP) [172] is an iterative algorithm that often selects the operator with the maximal average reward. AP associates to each operator a a probability value $P^{(t)}(a)$ at time t and an estimated reward value $Q^{(t)}(a)$. Each step, the immediate reward value for the selected operator a, $R^{(t)}(a)$, is returned, and the average reward value $Q^{(t)}(a)$ is updated. The method ranks the reward distribution $Q^{(t)}(a)$ and sets these ranked values to update the corresponding selection probabilities with a variant of the Bellman equation. The selection probability of the current best operator $a^* \leftarrow \text{argmax}_a Q^{(t+1)(a)}$ increases, whereas the probability of selecting all other operators deemed as sub-optimal, $\forall a \neq a^*$ decreases

$$P_{a^*}^{(t+1)} \leftarrow P_{a^*}^{(t)} + \beta \cdot [P_{\max} - P_{a^*}^{(t)}] \quad P_a^{(t+1)} \leftarrow P_a^{(t)} + \beta \cdot [P_{\min} - P_a^{(t)}]$$

where β is the learning rate. Note that the target probabilities $[P_{\max}, P_{\min}, \dots, P_{\min}]$ are fixed with a step like distribution, where $P_{\max}$ is much larger than $P_{\min}$, $P_{\max} \gg P_{\min}$ and $P_{\max} + (K\text{-}1)P_{\min} = 1$, where K is the number of operators. The larger is the learning parameter β, the slower converges the algorithm to the target distribution.

Upper confidence bound for online operator selection. UCB1 [3] is one of the most multi-armed bandits, due to its simplicity and its generality. Multi-armed bandits had been used in numerous studies on online operator selection [57]. Similarly to adaptive pursuit, each operator a is considered an arm with unknown probability of getting an immediate reward r_a. The cumulative reward function of operator a contains: 1) the estimated value of the operator r_a, and 2) an exploration coefficient $C \cdot \sqrt{\frac{2 \log \sum_j n_j}{n_a}}$, where n_a is the number of times the operator a was selected. Each time step, the arm that maximises $r_a + C \cdot \sqrt{\frac{2 \log \sum_j n_j}{n_a}}$ is selected.

The initial UCB1 algorithm suffers several changes required by practice. For example, UCB1 detects changes in the environment but reacts quite slow, because the rewards are averaged instead of used with a discount factor [174]. Alternatively, an UCB1 variant detects variations in the landscape with Page-Hinckley statistical tests [33] for an improved performance of the operator selection algorithm. Originally, UCB1 has positive sub-unitary values, whereas landscapes analysed with EC could take positive or negative values in any range. Hence, setting up the value of the constant C for the variant UCB1 is more relevant than for the standard UCB1; alternatively, one could normalise the reward function [174]. Other techniques to improve the performance of UCB1 are: 1) the operators are weighted based on their frequency, 2) the area under the curve is used as measure of improvement, and 3) the extreme value operator selection encourages exploration.

9.3.2. Multiple operator selection

Often, similar parameters have similar performance. In this case, an adaptive algorithm has difficulties in selecting one optimal operator causing the performance deploy. Generalised adaptive pursuit [47] assumes that exploiting a set of related operators is beneficial for the algorithm of selecting the optimal mutation rate for restarting the local search. Two (or more) operators take the maximum value $P_{\max}$; the rest of operators are selected with the minimum probability of $P_{\min}$. Recall that the sum of probabilities is always 1. Consequently, the larger is the amount of operators with the maximal selection value $P_{\max}$, the smaller is the maximal value $P_{\max}$, and, thus, the smaller the probability that an optimal operator is selected.

Another variant of the generalised adaptive pursuit simultaneously optimises the usage of two or more operators with techniques resembling multi-objective RL in the sense that multiple reward values assess the performance of an algorithm at a particular time, each reward value corresponds to the realisation of a single operator [46].

9.3.3. Generic parameter control

On-policy learning: SARSA. Q-learning has a number of variants based on different Q-learning update rules. State-action-reward-state-action (SARSA) [147] is an on-policy algorithm that optimised the policy that is executed simultaneously with the exploration path. The updating rule for the Q-value replaces the max operator from Q-learning with the action estimate a' according to the following policy

$$Q(s,a) \leftarrow Q(s,a) + \alpha \cdot \left(r^{(t)} + \gamma \cdot Q(s',a') - Q(s,a) \right)$$

This algorithm converges in infinite time to the optimal policy when all states and actions are tried infinitely often and the exploration coefficient decreases over time.

On fly parameter tuning with SARSA. A compressive overview of the parameter control strategies is given in Ref. [98]. In Ref. [96], EC uses a population of parameters correlated with fitness improvements monitoring the performance of the algorithm. The list of parameters contains: 1) genotypic diversity, 2) phenotypic diversity, 3) fitness standard deviation, 4) fitness improvement, and 5) stagnation counter. Note that this list includes variables that measure the current state of the population and the progress made over some generations. SARSA is used to select the optimal parameter, and the states are represented in a binary decision tree. Each state-action pair has associated a value that determines how much impact this pair has on the averaged reward values. The decision tree starts at the root of the state tree and traverses the tree based on the observations down to a leave node.

For the empirical study, there are selected several EAs, naming Evolutionary Strategies (ES), and cellular EAs. The learning algorithm improves the performance of ES by controlling the following parameters: 1) population size, 2) the generation gap (the ratio of offsprings), 3) mutation step size and 4) tournament size for survivor selection.

9.3.4. Hyper-heuristics algorithmic instances

Selective hyper-heuristics selects the best hyper-heuristics from a given set of heuristics. For reducing the designing effort, existing heuristics are combined in sequences that are at their turn optimised with RL [131].

In Ref. [155], hyper-heuristics evolves the representation of EC. In a recent study, the reward values of the Q-learning are modified as a feedback loop for the control of parameters [97]. We classify the previous approaches as generative hyper-heuristics.

10. Hierarchical structures in evolutionary computation

Hierarchical structures are of ordinarily used in optimisation and RL. In this section, we focus on ECRL for optimisation that allows tree structures.

10.1. Taxonomy

The usage of tree structures in EC is not new, GP are programs structured as trees. There are many variants combining GP and Q-learning, like reinforced genetic programming [41].

As a different research direction, the hierarchical bandits is an emerging EC technique that does not use genetic operators to generate new individuals by using a synergy with Monte Carlo tree search. Schemata bandits [45] has binary representation and its practical application is physical activity recognition [5].

10.2. Evolutionary Monte Carlo search

The optimisation in EC is steered by making an analogy with schema representation of solutions.

Schemata theory. The common part in the representation of several individuals is called a schema [86]. According to John Holland, the schema theorem explains the success of genetic algorithms stating that although the EC implicitly operates at an individual level, EC processes information about schemata (subsets of the search space) *in parallel*. Moreover, it samples the most interesting schemata called building blocks in a near-optimal way using the analogy between schemata and arms in the bandit problem. That is schemata with the fitness mean above the average are grouped as a bandit arm, and the schemata with the fitness below the mean are considered to be the second arm. The schema theorem shows that selection increasingly focuses on the schemata with the fitness average above the mean.

Monte Carlo Tree Search. MCTS [18] is a heuristic used to solve intractable problems, i.e. vast search spaces, like playing computer GO. MCTS builds a search tree using a search policy selecting the most probable nodes to expand. MCTS uses a top-down approach, i.e. root to leaves, with the following four steps: 1) MCTS *selects* the most promising children. 2) MCTS *expands* to create new nodes using a tree policy. 3) In the *simulation* phase, MCTS plays at random from the current node to the end of the game, and 4) MCTS *back-propagations* the information on the explored path. In MCTS, exploration means generating new branches in the tree, and exploitation implies to focus the search on the tree branches that returned good rewards.

10.2.1. Schemata bandits

The *baseline schemata bandits algorithm* builds a tree where each node is a schema. The starting point for each iteration of this algorithm is the root that is the most general schema. Considering the steps specific to the Monte Carlo tree search algorithm, cf MCTS, the schemata bandits algorithm consists of three steps:

Selection. Starting from the root, select child nodes that expand the schemata net towards the most promising parts of the search space down to a leaf node. A node is expandable if it is unvisited. A traditional policy to select the next node to expand is UCB1 that upper bounds the loss resulting from choosing non-optimal arms.

Expansion. If in the selection step, a child node that is not in the schemata graph is selected a node in the net is created. If the selected child node is already in the schemata graph, only the counters are incremented. The expansion finishes with the generation of a leaf node. When a leaf node is reached, a representative set of the corresponding bit-strings is generated and evaluated.

Propagation. We update the mean values of all the schemata in the schemata graph that contain that solution. The root schema is updated for all bit-strings. A schema higher in MCTS is updated more often than a schema lower in the hierarchy because more bit-strings match to a higher schema than to a lower schema in MCTS.

The schemata with the maximal estimated mean fitness are often selected using a multi-armed bandit algorithm. A schemata is a L dimensional hypercube, and there are 2^L such binary strings and L is the size of a bitstring. Schemata bandits is a parameter free optimisation algorithm. Since schemata are densely connected, only a part of the solutions in a schemata net is sampled.

10.2.2. Hierarchical bandits for continuous functions

Monte Carlo tree search variants are used in optimisation of real-coded multi-dimensional functions by partitioning the search space in sub-domains [42,138]. Each node in MCTS contains a multi-dimensional domain that focuses the search on the most promising partitions, i.e. which contain the best solutions. The other regions are explored with small probability.

Simultaneous optimistic optimisation (SOO) [138] is successfully applied on many dimensional test problems from the CEC 2014 competition on single objective real-parameter numerical optimisation. The hierarchical CMA-ES solver [42] uses CMA-ES solvers in each node of MCTS.

11. Evolutionary multi-objective optimisation and reinforcement learning

Evolutionary multi-objective optimisation (EMO) [32,36] optimises multi-objective environments. Some MORLs have as intuition a multi-criteria decision maker that makes decisions given not one but several criteria. The goal of some multi-criteria decision makers is to generate a single solution of the Pareto front that weights the criteria [143]. A smaller group of MORL considers the generation of the entire Pareto front, and the decision on returning the best solution is postponed. Methods from EMO are incorporated into MDP to assemble effective algorithms to learn Pareto optimal policies.

11.1. Practical motivation

Multi-objective RL [44,48,142] (MORL) optimises reward vectors instead of reward values. Multi-objective dynamic programming [63] and multi-objective or multi-criteria MDP (MOMDP) [183] find their roots in the 80s where the immediate reward values are replaced with reward vectors. *Multi-criteria decision making* (MCDM) is concerned with optimisation of more than one objective. A decision maker decides which solutions are relevant and when to show these solutions to the decision maker.

Real-world applications motivate the usage of *multi-objective reinforcement learning* (MORL): 1) the control theory [20], 2) traffic light control [21,101], 3) planning for the health system [108,112,113], and 4) gaming [69]. In a multi-agent setting, MORL techniques optimise the schedule of a mining company [38,144]. Practical applications of multi-objective reinforcement learning are reviewed in Ref. [177], and a review on multiobjective sequential decision making in given in Ref. [142]. An algorithmic interpretation of multiobjective RL is provided in Ref. [111]. In Ref. [145], multiobjective reinforcement learning is motivated by likely real-world life scenarios for scheduling in mine industry. A generic and tunable problem instance generator [64] proposes large and challenging multiobjective environments. Another benchmark with test problems for MORL is proposed in Ref. [177].

11.2. Multi-objective reinforcement learning algorithmic setup

For the infinite horizon MOMDP, the value vector of a state s under the policy π is the expected cumulative discounted sum of reward vectors

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \boldsymbol{r}_{t+1} \mid s_t = s\right]$$

resulting in a vector with m dimensions, here denoted as objectives. The associated immediate reward vector with size m, $\boldsymbol{r} = \left[r_1, \ldots, r_m\right]^T$. The cumulative reward function is $\boldsymbol{R}(s' \mid s,a) = \left[R_1(s' \mid s,a), \ldots, R_m(s' \mid s,a)\right]^T$, where $\boldsymbol{r}_t = \boldsymbol{R}(s' \mid s,a)$.

In MORL, several actions can be considered to be the best according to their reward vectors. Thus, in general, multiple optimal policies are Pareto optimal. A policy π^i, $\boldsymbol{V}^i(s)$, is *non-dominated* by another policy π^j, $\boldsymbol{V}^j(s)$, if there is *no* objective o for which $V^i_o(s) < V^j_o(s)$. $\boldsymbol{V}^i(s)$ is *dominated* by $\boldsymbol{V}^j(s)$, $\boldsymbol{V}^i(s) \prec \boldsymbol{V}^j(s)$, if there is at least one objective o for which $V^i_o(s) < V^j_o(s)$, and for all other objectives $V^i_o(s) \leq V^j_o(s)$. The *set of non-dominated value functions* is denoted with $\mathbb{V}^*(s)$, where all component policies $\boldsymbol{V}^i(s)$ are non dominated in the state s. Thus, there is no policy $\boldsymbol{V}^j(s) \in \mathbb{V}(s)$ such that $\boldsymbol{V}^j(s) \prec \boldsymbol{V}^i(s)$.

Table 8
Evolutionary multi-objective reinforcement learning: algorithmic instances and sub-components.

EMORL	RL comp	EC comp
EMO for direct policy search [72]	Function approximation	Objective reduction
Scalarised multi-armed bandits [194]	multi-armed bandits	Scalarisation functions
Hypervolume based multi-objective Q-learning [124]	Q-learning	Hypervolume exploration
Hypervolume-based Monte Carlo tree search [181]	Monte Carlo tree search	
Multi-objectivisation for GAs [27]	Q-learning	GA
Multi-objectivisation of RL [19]		multi-objectivisation

The set of non-dominated Q-values is denoted with $\mathbb{Q}^*(s,a)$, where all component policies $\boldsymbol{Q}(s,a) = \left[Q(s,a,1),\ldots,Q(s,a,m)\right]^T$ are non-dominated in state s and action a. The Pareto optimal operator is a non-dominated operator defined for Q-values

$$PO(\mathbb{Q}(s,a)) = \{\boldsymbol{Q}^i(s,a) \mid \boldsymbol{Q}^i(s,a) \in \mathbb{Q}(s,a) \wedge ND(\boldsymbol{Q}^i(s,a), \mathbb{Q}(s,a))\}$$

where the non-dominated operator $ND(\boldsymbol{Q}^i(s,a), \mathbb{Q}(s,a))$ tells whether $\boldsymbol{Q}^i(s,a)$ is non-dominated by any value function in the set $\mathbb{Q}(s,a)$. The dynamic programming operator for deterministic environments is

$$DP(\mathbb{Q}(s,a)) = \{\boldsymbol{R}(s' \mid s,a) \oplus \gamma \mathbb{V}(s') \mid \boldsymbol{T}(s' \mid s,a) = 1.0\}$$

where $\oplus$ is the maximum operator for sets and the transition probability vector $\boldsymbol{T}(s' \mid s,a)$ is defined for each objective. The bootstrapping rule for MOMDPs is similar with the bootstrapping rule from Ref. [169] to assure the convergence of Q-vectors even for a stochastic environment. The goal of MOMDPs is to find the Pareto-optimal set of policies $\mathbb{P}^*$ that receive the most rewards. This dynamic programming algorithm converges to the maximal Pareto optimal set of policies because it eliminates all dominated policies [191].

11.3. Taxonomy

Few multiobjective reinforcement learning algorithms have an evolutionary component. We denote this class of algorithms as *evolutionary multiobjective reinforcement learning* (EMORL). In the following, we classify EMORLs according to the type of RL. Table 8 summarises algorithmic classes of EMORL.

Evolutionary multi-objective optimisation for direct policy search. A method to optimise in multi-objective spaces is to simplify the search space. In Ref. [16], the number of objectives is diminished with principle component analysis. A multi-objective version of the evolutionary computation for direct policy search optimises water reservoirs [72].

Scalarisation functions transform the reward vector into reward values using a weight vector of real values that is either randomly generated from a distribution or selected by a user [54]. In Refs. [43,194], the weights of scalarisation functions are adapted from a set.

Evolutionary multi-objective optimisation for value function approximations. The *hypervolume unary indicator* [61] commonly assesses the performance of evolutionary multi-objective optimisation. A successful MORL algorithm uses the hypervolume indicator in the exploration mechanism [124]. In the dynamics of hypervolume-based MORL algorithms, there were noticed the same downside as for multi-objective optimisation algorithms that use the hypervolume indicator, meaning this algorithm do not scale up with the number of objectives.

Hypervolume-based Monte Carlo tree search. Hypervolume-based search is computed with a Monte Carlo tree search method in Ref. [181]. The advantage of hypervolume-based search over scalarisation functions is that there is no need to search for a set of functions to generate the Pareto front. The disadvantage is that the decision maker has no control over the output Pareto optimal solutions.

Multi-objectivisation. We consider multi-objectivisation [102] to be a type of complexification, where a single objective optimisation problem is transformed into a multi-objective optimisation instance by adding ancillary objectives for computational efficiency. In Ref. [27], an "helper" or extra RL speeds up the optimisation process of GA. In Ref. [19], multiple copies of a single objective RL policy solve a given single objective MDP environment.

To conclude, reinforcement learning with reward vectors is a novel and promising research area with an initial slow development because of severe computational problems that were somewhat solved with the incorporated advanced multi-objective optimisation techniques.

12. Dynamic and adversarial environments

In this section, we discuss ECRL classes build on dynamic and adversarial environments.

12.1. Evolutionary dynamic optimisation

EDO concerns evolutionary computation methods in dynamic and changing environments. A survey on EDO [129], discusses its important provocations from the prism of EC. Generically, EDO is a dynamic optimisation problem solved with EC algorithms that are enlarged with tools that predict the change and track the global optima. However, the up-mentioned review does not discuss ECRL with the same focus.

RL is often used in learning in non-stationary environments [39]. From the associated applications, we recall stochastic mazes [174], and scheduling with uncertainty [110], and games.

Evolutionary reinforcement learning [186] consists of experimental algorithms that solve reinforcement learning tasks for online learning. In a dynamic environment, the experimental section is usually different from the empirical evaluation of RL algorithms since the focus is now on the exact optimal solution rather than the algorithms' long time behaviour. Online algorithms monitor the quality of solutions observed so far, or for easier environments where the optimal solution is known, how many times the target evolves.

Evolutionary multi-objective RL has variants for optimising in non-stationary environments. EA + RL uses multi-objectivisation with auxiliary objectives [134]. To adapt to changing auxiliary objectives, RL uses the discounted factor in a Q-learning algorithm. Multi-objective RL [119] is applied on the dispatcher problem that involves scheduling for electricity generators to meet the customers demand while minimising the fuel cost and emissions.

12.2. Coevolutionary computation

CEC [90] are EC instances where the fitness function, expressing the quality of an individual, is based on the interaction between different solutions in the evolutionary system. Coevolutionary algorithms do not have a pre-determined fitness function; the assembly of speciated individuals determine a relative value for each species.

Competitive CEC is usually associated with games resembling the adversarial setting from reinforcement and multi-armed bandits where the success of some players means the failure of others. Competitive CEC is combined with RL to play GO [106].

Competitive CEC is combined with neuroevolution in Ref. [15]. A population of mazes is coevolved with a population of neural networks for maze navigation, where each neural network solves at least one maze. The model is motivated by natural evolution paradigms such as

chromaria [162] that copies of genotypes are created to continue one's lineage.

Cooperative CEC decomposes the system into sub-problems that co-evolve testing how well the sub-ensembles work together. Cooperative CEC is considered a type of resource allocation problem, which is sometimes solved with RL, where species need to coexist for their survival, a multi-armed algorithm evolves the most promising species [139].

Cooperative CEC is combined with neuroevolution for optimising many parameters for playing games [180]. The algorithm is using as encoding for the neural network weights wavelet coefficients that are successful in signal processing and image analyses.

13. Conclusions

This overview proposes a unified framework for the algorithmic techniques at the confluence between evolutionary computation and reinforcement learning. Some ECRL methods are evolutionary computation algorithms for reinforcement learning, some paradigms are reinforcement learning methods for evolutionary computation, whereas some are hybrids that cannot be easily classified as any of the two approaches.

The proposed methodology considers all the aspects of algorithmic design. Practical motivation, such as innovative approaches for challenging real-world applications, or improved computational efficiency, is essential for the novel introduced techniques. Both reinforcement learning and evolutionary computation are motivated by natural processes like the adaptation of an agent to the environmental changes or the evolution of a population. A methodological analysis of the evolutionary computation versus reinforcement learning's sub-components helps in the comprehension of the ensemble. All the ECRL studies allow a thorough comparison between algorithms. Most comparisons are pragmatical with innovative empirical measurements and methodologies; theoretical works investigate the convergence properties in the limit. ECRL often considers a static environment, i.e. with optima that are invariant with time. Dynamical and adversarial environments are demanding in computational resources, but essential in many applications concerning gaming and robotics.

Using our framework, we group ECRL into classes of algorithms. Evolutionary computation for reinforcement learning is RL with an improved empirical performance for EC. Reinforcement learning is sometimes used for the on-line control of genetic algorithms. Hierarchical structures were used before in EC, see genetic programming methods; here, we advocate their usage with RL. Evolutionary multi-objective reinforcement learning is a sub-area of reinforcement learning with reward vectors that uses evolutionary computation for efficiency. Co-evolution paradigms are often combined with RL to solve problems modelled as two populations that compete for resources. For ease of discussion, we present and discuss in detail few ECRL algorithmic instances.

To conclude, this overview shows the main developments in the field of evolutionary computation versus reinforcement learning and gives hits on the feature advancements in the field.

Appendix A. Supplementary data

Supplementary data related to this article can be found at https://doi.org/10.1016/j.swevo.2018.03.011.

References

[1] Y. Akimoto, Y. Nagata, Isao Ono, S. Kobayashi, Bidirectional relation between CMA evolution strategies and natural evolution strategies, in: Parallel Problem Solving from Nature (PPSN), LNCS, Springer, 2010, pp. 154–163.

[2] W. Atmar, Notes on the simulation of evolution, IEEE Trans. Neural Netw. (TNN) 5 (1) (1994) 130–147.

[3] P. Auer, N. Cesa-Bianchi, P. Fischer, Finite-time analysis of the multiarmed bandit problem, Mach. Learn. 47 (2–3) (2002) 235–256.

[4] T. Back, D.B. Fogel, Z. Michalewicz (Eds.), Handbook of Evolutionary Computation, IOP Publishing Ltd., 1997.

[5] A. Baldominos, P. Isasi, Y. Sáez, B. Manderick, Monte Carlo schemata searching for physical activity recognition, in: Int Conf Intelligent Networking and Collaborative Systems (INCoS), 2015, pp. 176–183.

[6] Akram Beigi, Nasser Mozayani, A simple interaction model for learner agents: an evolutionary approach, J. Intell. Fuzzy Syst. 30 (5) (2016) 2713–2726.

[7] R.K. Belew, When both individuals and populations search: adding simple learning to the genetic algorithm, in: Proc. International Conference on Genetic Algorithms, Morgan Kaufmann, 1989, pp. 34–41.

[8] R.E. Bellman, Dynamic Programming, Princeton University Press, 1957.

[9] Y. Bengio, Learning deep architectures for AI, Found. Trends Mach. Learn. 2 (1) (2009) 1–127.

[10] H.G. Beyer, H.P. Schwefel, Evolution strategies: a comprehensive introduction, J. Nat. Comput. 1 (1) (2002) 3–52.

[11] S. Bleuler, M. Laumanns, L. Thiele, E. Zitzler, PISA — a platform and programming language independent interface for search algorithms, in: Evolutionary Multi-criterion Optimization (EMO), LNCS, Springer, 2003, pp. 494–508.

[12] L. Bom, R. Henken, M. Wiering, Reinforcement learning to train ms. Pac-Man using higher-order action-relative inputs, in: Adaptive Dynamic Programming and Reinforcement Learning (ADPRL), SSCI, IEEE, 2013, pp. 156–163.

[13] L.B. Booker, D.E. Goldberg, J.H. Holland, Classifier systems and genetic algorithms, Artif. Intell. 40 (1–3) (1989) 235–282.

[14] M. Bowling, M. Veloso, Multiagent learning using a variable learning rate, Artif. Intell. 136 (2) (2002) 215–250.

[15] J.C. Brant, K.O. Stanley, Minimal criterion coevolution: a new approach to open-ended search, in: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO, 2017.

[16] D. Brockhoff, E. Zitzler, Objective reduction in evolutionary multiobjective optimization: theory and applications, Evol. Comput. 17 (2) (2009) 135–166.

[17] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, Jie Tang, W. Zaremba, Openai gym, CoRR, abs/1606.01540, 2016.

[18] C. Browne, E. Powley, D. Whitehouse, S.M. Lucas, P.I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, S. Colton, A survey of Monte Carlo tree search methods, Trans. Comp. Intel AI Games 4 (1) (2012) 1–46.

[19] T. Brys, A. Harutyunyan, P. Vrancx, M.E. Taylor, D. Kudenko, A. Nowé, Multi-objectivization of reinforcement learning problems by reward shaping, in: Inter Joint Conf Neural Networks (IJCNN), IEEE, 2014, pp. 2315–2322.

[20] T. Brys, K. Van Moffaert, K. Van Vaerenbergh, A. Nowé, On the behaviour of scalarization methods for the engagement of a wet clutch, in: Inter Conf on Machine Learning and Applications (ICMLA), IEEE, 2013, pp. 258–263.

[21] T. Brys, T.T. Pham, M.E. Taylor, Distributed learning and multi-objectivity in traffic light control, Connect. Sci. 26 (1) (2014) 65–83.

[22] Larry Bull, A brief history of learning classifier systems: from CS-1 to XCS and its variants, Evol. Intell. 8 (2015) 55–70.

[23] E.K. Burke, M. Gendreau, M.R. Hyde, G. Kendall, G. Ochoa, E. Özcan, Rong Qu, Hyper-heuristics: a survey of the state of the art, J. Oper. Res. (JORS) 64 (12) (2013) 1695–1724.

[24] E.K. Burke, M. Gendreau, G. Ochoa, J.D. Walker, Adaptive iterated local search for cross-domain optimisation, in: Genetic and Evol Comp Conf (GECCO), ACM, 2011, pp. 1987–1994.

[25] A.N. Burnetas, M.N. Katehakis, Optimal adaptive policies for Markov decision processes, Math. Oper. Res. 22 (1) (1997) 222–255.

[26] M.V. Butz, D.E. Goldberg, P. Luca Lanzi, Gradient descent methods in learning classifier systems: improving XCS performance in multistep problems, Trans. Evol. Comput. (TEC) 9 (5) (2005) 452–473.

[27] A. Buzdalova, A. Matveeva, G. Korneev, Selection of auxiliary objectives with multi-objective reinforcement learning, in: Genetic and Evolutionary Computation Conference, (GECCO), ACM, 2015, pp. 1177–1180.

[28] S. Cahon, N. Melab, E.-G. Talbi, ParadisEO: a framework for the reusable design of parallel and distributed metaheuristics, J. Heuristics 10 (3) (2004) 357–380.

[29] E. Cantú-Paz, Migration policies, selection pressure, and parallel evolutionary algorithms, J. Heuristics 7 (4) (2001) 311–334.

[30] N. Cesa-Bianchi, G. Lugosi, Prediction, Learning, and Games, Cambridge University Press 2006, 2006, pp. I–XII, 1–394.

[31] T. Chabin, A. Tonda, E. Lutton, How to Mislead an Evolutionary Algorithm Using Global Sensitivity Analysis, Springer, 2016, pp. 44–57.

[32] C.A. Coello Coello, Evolutionary multi-objective optimization: a critical review, in: Evolutionary Optimization, vol. 48, Springer, 2002, pp. 117–146.

[33] L. Da Costa, Á. Fialho, M. Schoenauer, M. Sebag, Adaptive operator selection with dynamic multi-armed bandits, in: Genetic and Evolutionary Computation Conference (GECCO), MIT, 2008, pp. 913–920.

[34] M. Crepinsek, Shih-Hsi Liu, M. Mernik, Exploration and exploitation in evolutionary algorithms: a survey, Comput. Surv. 45 (3) (2013), 35:1–35:33.

[35] K. de Jong, D.B. Fogel, H.P. Schwefel, A history of evolutionary computation, in: Handbook of Evolutionary Computation, Oxford University Press, 1997.

[36] K. Deb, Multi-objective optimisation using evolutionary algorithms: an introduction, in: Multi-objective Evolutionary Optimisation for Product Design and Manufacturing, Springer, 2011, pp. 3–34.

[37] Kalyanmoy Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Trans. Evol. Comput. (TEC) 6 (2002) 182–197.

[38] M.R. Diederik, S. Whiteson, F.A. Oliehoek, Computing convex coverage sets for faster multi-objective coordination, J. Artif. Intell. Res. (JAIR) 52 (2015) 399–443.

[39] G. Ditzler, M. Roveri, C. Alippi, R. Polikar, Learning in nonstationary environments: a survey, IEEE Comp. Int. Mag. 10 (4) (2015) 12–25.
[40] S. Doncieux, N. Bredèche, Jean-Baptiste Mouret, A.E. Eiben, Evolutionary robotics: what, why, and where to, Front. Robot. AI 2 (2015) 4.
[41] K.L. Downing, Reinforced genetic programming, Genet. Program. Evolvable Mach. 2 (3) (2001) 259–288.
[42] M.M. Drugan, Efficient real-parameter single objective optimizer using hierarchical CMA-ES solvers, in: EVOLVE - a Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation, Springer, 2015.
[43] M.M. Drugan, Linear scalarization for Pareto front identification in stochastic environments, in: Evolutionary Multi-objective Optimization, (EMO), Springer, 2015.
[44] M.M. Drugan, Multi-objective optimization perspectives on reinforcement learning algorithms using reward vectors, in: European Symposium on Artificial Neural Networks (ESANN), 2015.
[45] M.M. Drugan, P. Isasi, B. Manderick, Schemata bandits for binary encoded combinatorial optimisation problems, in: Simul Evolution and Learning (SEAL), Springer, 2014, pp. 299–310.
[46] M.M. Drugan, E.-G. Talbi, Adaptive multi-operator metaheuristics for quadratic assignment problems, in: EVOLVE - a Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation, Springer, 2014, pp. 149–163.
[47] M.M. Drugan, D. Thierens, Generalized adaptive pursuit algorithm for genetic Pareto local search algorithms, in: Genetic and Evolutionary Computer Conference (GECCO), ACM, 2011, pp. 1963–1970.
[48] M.M. Drugan, M. Wiering, P. Vamplew, M. Chetty, Special issue on multi-objective reinforcement learning, Neurocomputing 263 (2017) 1–2.
[49] T. D'Silva, R. Miikkulainen, Learning dynamic obstacle avoidance for a robot arm using neuroevolution, Neural Process. Lett. 30 (1) (2009) 59–69.
[50] J.J. Durillo, A.J. Nebro, jMetal: a Java framework for multi-objective optimization, Adv. Eng. Software 42 (10) (2011) 760–771.
[51] A.E. Eiben, R. Hinterding, Z. Michalewicz, Parameter control in evolutionary algorithms, Trans. Evol. Comput. (TEC) 3 (2) (1999) 124–141.
[52] A.E. Eiben, J.E. Smith, Introduction to Evolutionary Computing. Natural Computing Series, Springer, 2003.
[53] A.E. Eiben, E.H.L. Aarts, K.M. van Hee, Global convergence of genetic algorithms: a Markov chain analysis, in: Parallel Problem Solving from Nature (PPSN), LNCS, Springer-Verlag, 1991, pp. 4–12.
[54] G. Eichfelder, Adaptive Scalarization Methods in Multiobjective Optimization. Springer, 2008.
[55] S. Elfwing, Eiji Uchibe, Kenji Doya, H.I. Christensen, Evolutionary development of hierarchical learning structures, IEEE Trans. Evol. Comput. 11 (2) (2007) 249–264.
[56] A. Fialho, L. Da Costa, M. Schoenauer, M. Sebag, Extreme value based adaptive operator selection, in: Parallel Problem Solving from Nature PPSN, Springer, 2008, pp. 175–184.
[57] Á. Fialho, L. Da Costa, M. Schoenauer, M. Sebag, Analyzing bandit-based adaptive operator selection mechanisms, Ann. Math. Artif. Intell. 60 (2010) 25–64.
[58] Á. Fialho, Y. Hamadi, M. Schoenauer, A multi-objective approach to balance buildings construction cost and energy efficiency, in: Eur Conf on Artif Intel (ECAI), 2012, pp. 961–966.
[59] D.B. Fogel, Phenotypes, genotypes, and operators in evolutionary computation, in: Congress on Evolutionary Computation (CEC), vol. 1, IEEE, 1995.
[60] L.J. Fogel, A.J. Owens, M.J. Walsh, Intelligent decision making through a simulation of evolution, Behav. Sci. 11 (4) (1966) 253–272.
[61] C.M. Fonseca, J.D. Knowles, L. Thiele, E. Zitzler, A tutorial on the performance assessment of stochastic multi-objective optimizers, in: Evolutionary Multi-criterion Optimization (EMO), Springer, 2005.
[62] E. Frank, M.A. Hall, G. Holmes, R. Kirkby, B. Pfahringer, I.H. Witten, L. Trigg, WEKA - a machine learning workbench for data mining, in: The Data Mining and Knowledge Discovery Handbook, Springer, 2005, pp. 1305–1314.
[63] N. Furukama, Characterization of optimal policies in vector-valued Markovian decision processes, Math. Oper. Res. 5 (2) (1980) 271–279.
[64] D. Garrett, J. Bieger, K.R. Thórisson, Tunable and generic problem instance generation for multi-objective reinforcement learning, in: Adaptive Dynamic Programming and Reinforcement Learning (ADPRL), SSCI, IEEE, 2014, pp. 1–8.
[65] J. Gauci, K.O. Stanley, Generating large-scale neural networks through discovering geometric regularities, in: Genetic and Evolutionary Computation Conference (GECCO), 2007, pp. 997–1004.
[66] S. Gelly, L. Kocsis, M. Schoenauer, M. Sebag, D. Silver, C. Szepesvári, O. Teytaud, The grand challenge of computer GO: Monte Carlo tree search and extensions, Commun. ACM 55 (3) (2012) 106–113.
[67] A. Geramifard, T.J. Walsh, S. Tellex, G. Chowdhary, N. Roy, J.P. How, A tutorial on linear function approximators for dynamic programming and reinforcement learning, Found. Trends Mach. Learn. 6 (4) (2013) 375–451.
[68] M. Ghavamzadeh, S. Mannor, J. Pineau, A. Tamar, Bayesian reinforcement learning: a survey, Found. Trends Mach. Learn. 8 (5–6) (2015) 359–483.
[69] L.E. Gillespie, G.R. Gonzalez, J. Schrum, Comparing direct and indirect encodings using both raw and hand-designed features in Tetris, in: Proceedings of the Genetic and Evolutionary Computation Conference, (GECCO), 2017, pp. 179–186.
[70] S. Girgin, P. Preux, Feature discovery in reinforcement learning using genetic programming, in: European Conference on Genetic Programming, EuroGP, Springer, 2008, pp. 218–229.
[71] J.C. Gittins, Bandit processes and dynamic allocation indices, J. R. Stat. Soc. 41 (2) (1979) 148–177.
[72] M. Giuliani, S. Galelli, R. Soncini-Sessa, A dimensionality reduction approach for many-objective markov decision processes: application to a water reservoir operation problem, Environ. Model. Software 57 (2014) 101–114.
[73] D.E. Goldberg, Probability matching, the magnitude of reinforcement, and classifier system bidding, Mach. Learn. 5 (1990) 407–425.
[74] D.E. Goldberg (Ed.), The Design of Innovation: Lessons from and for Competent Genetic Algorithms, Springer, 2013.
[75] D.E. Goldberg, K. Deb, A comparative analysis of selection schemes used in genetic algorithms, in: Foundations of Genetic Algorithms (FOGA), 1990, pp. 69–93.
[76] X. Guo, S. Singh, H. Lee, R.L. Lewis, X. Wang, Deep learning for real-time Atari game play using offline Monte-Carlo tree search planning, in: Proc of NIPS Deep Learning Workshop, 2014.
[77] E. Haasdijk, N. Bredeche, S. Nolfi, A.E. Eiben, Evolutionary robotics, Evol. Intell. 7 (2) (2014) 69–70.
[78] D. Hadka, P. Reed, Borg: an auto-adaptive many-objective evolutionary computing framework, Evol. Comput. 21 (2) (2013) 231–259.
[79] Hisashi Handa, EDA-RL: estimation of distribution algorithms for reinforcement learning problems, in: Genetic and Evolutionary Computation Conference (GECCO), 2009, pp. 405–412.
[80] N. Hansen, A. Ostermeier, Completely derandomized self-adaptation in evolution strategies, Evol. Comput. 9 (2) (2001) 159–195.
[81] G.R. Harik, E. Cantú-Paz, D.E. Goldberg, B.L. Miller, The gambler's ruin problem, genetic algorithms, and the sizing of populations, Evol. Comput. 7 (3) (1999) 231–253.
[82] M. Hauschild, M. Pelikan, An introduction and survey of estimation of distribution algorithms, Swarm Evol. Comput. 1 (3) (2011) 111–128.
[83] V. Heidrich-Meisner, C. Igel, Uncertainty handling CMA-ES for reinforcement learning, in: Genetic and Evolutionary Computation Conference (GECCO), ACM, 2009, pp. 1211–1218.
[84] V. Heidrich-Meisner, M. Lauer, C. Igel, M.A. Riedmiller, Reinforcement learning in a nutshell, in: European Symposium on Artificial Neural Networks (ESANN), 2007, pp. 277–288.
[85] W.D. Hillis, Co-evolving parasites improve simulated evolution as an optimization procedure, Phys. D 42 (1–3) (1990) 228–234.
[86] J.H. Holland, Adaptation in Natural and Artificial Systems, U. Michigan Press, 1975.
[87] G.D. Howard, On self-adaptive rate restarts for evolutionary robotics with real rotorcraft, in: Proceedings of the Genetic and Evolutionary Computation Conference, (GECCO), 2017, pp. 123–130.
[88] W. Jaskowski, M.G. Szubert, P. Liskowski, K. Krawiec, High-dimensional function approximation for knowledge-free reinforcement learning: a case study in SZ-Tetris, in: Genetic and Evolutionary Computation Conference, GECCO, ACM, 2015, pp. 567–573.
[89] K.A. De Jong, Evolutionary Computation: a Unified Approach, MIT, 2006.
[90] K.A. De Jong, M.A. Potter, Evolving complex structures via cooperative coevolution, in: Evolutionary Programming, MIT, 1995, pp. 307–317.
[91] K.A. De Jong, W.M. Spears, D.F. Gordon, Using genetic algorithms for concept learning, Mach. Learn. 13 (1993) 161–188.
[92] N. Justesen, Sebastian Risi, Continual online evolutionary planning for in-game build order adaptation in starcraft, in: Proceedings of the Genetic and Evolutionary Computation Conference, (GECCO), 2017, pp. 187–194.
[93] L.P. Kaelbling, M.L. Littman, A.W. Moore, Reinforcement learning: a survey, J. Artif. Intell. Res. 4 (1996) 237–285.
[94] L. Kallel, B. Naudts, A. Rogers (Eds.), Theoretical Aspects of Evolutionary Computing, Springer, 2013.
[95] Shotaro Kamio, Hideyuki Mitsuhasi, Hitoshi Iba, Integration of genetic programming and reinforcement learning for real robots, in: Genetic and Evolutionary Computation (GECCO), 2003, pp. 470–482.
[96] G. Karafotias, A.E. Eiben, M. Hoogendoorn, Generic parameter control with reinforcement learning, in: Conf on Genetic and Evol Comp (GECCO), ACM, 2014, pp. 1319–1326.
[97] G. Karafotias, M. Hoogendoorn, A.E. Eiben, Evaluating reward definitions for parameter control, in: Applications of Evolutionary Computation (EvoApplications), 2015, pp. 667–680.
[98] G. Karafotias, M. Hoogendoorn, Á.E. Eiben, Parameter control in evolutionary algorithms: trends and challenges, Trans. Evol. Comput. (TEC) 19 (2) (2015) 167–187.
[99] S. Kelly, M.I. Heywood, Multi-task learning in atari video games with emergent tangled program graphs, in: Proceedings of the Genetic and Evolutionary Computation Conference, (GECCO), 2017, pp. 195–202.
[100] S. Khadka, Jen Jen Chung, K. Tumer, Evolving memory-augmented neural architecture for deep memory problems, in: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO, 2017, pp. 441–448.
[101] M.A. Khamis, W. Gomaa, Adaptive multi-objective reinforcement learning with hybrid exploration for traffic signal control based on cooperative multi-agent framework, Eng. Appl. Artif. Intell. 29 (2014) 134–151.
[102] J.D. Knowles, R.A. Watson, D. Corne, Reducing local optima in single-objective problems by multi-objectivization, in: Evolutionary Multi-criterion Optimization (EMO), Springer, 2001, pp. 269–283.
[103] N. Kohl, R. Miikkulainen, Evolving neural networks for fractured domains, in: Genetic and Evolutionary Computation Conference, (GECCO), ACM, 2008, pp. 1405–1412.
[104] John R. Koza, Genetic Programming: on the Programming of Computers by Means of Natural Selection, MIT Press, 1992.

[105] K. Krawiec, M.I. Heywood, Solving complex problems with coevolutionary algorithms, in: Genetic and Evolutionary Computation Conference (GECCO), Tutorials, 2017, pp. 782–806.
[106] K. Krawiec, W. Jaskowski, M.G. Szubert, Evolving small-board GO players using coevolutionary temporal difference learning with archives, Appl. Math. Comput. Sci. (AMCS) 21 (2011) 717–731.
[107] S. Kriegman, N. Cheney, F. Corucci, J.C. Bongard, A minimal developmental model can increase evolvability in soft robots, in: Proceedings of the Genetic and Evolutionary Computation Conference, (GECCO), 2017, pp. 131–138.
[108] E.B. Label, D.J. Lizotte, B. Ferguson, Set-valued dynamic treatment regimes for competing outcomes, Biometrics 70 (1) (2014) 53–61.
[109] P.L. Lanzi, Learning classifier systems from a reinforcement learning perspective, Soft Comput. 6 (3–4) (2002) 162–170.
[110] M. Laskey, J. Mahler, Z. McCarthy, F.T. Pokorny, S. Patil, J.P. van den Berg, D. Kragic, P. Abbeel, K. Goldberg, Multi-armed bandit models for 2d grasp planning with uncertainty, in: Proc Conf on Automation Science and Engineering (CASE), IEEE, 2015, pp. 572–579.
[111] C. Liu, X. Xu, D. Hu, Multiobjective reinforcement learning: a comprehensive overview, Trans. Syst. Man Cybern.: Systems 45 (2015) 385–398.
[112] D.J. Lizotte, M. Bowling, S.A. Murphy, Linear fitted-Q iteration with multiple reward functions, J. Mach. Learn. Res. (JMRL) 13 (2012) 3253–3295.
[113] D.J. Lizotte, M. Bowling, S.A. Murphy, Efficient reinforcement learning with multiple reward functions for randomized clinical trial analysis, in: Int Conf on Machine Learning (ICML), 2010.
[114] F.G. Lobo, C.F. Lima, Z. Michalewicz (Eds.), Parameter Setting in Evolutionary Algorithms, Volume 54 of Studies in Computational Intelligence, Springer, 2007.
[115] P.R. Lorenzo, J. Nalepa, M. Kawulok, L. Sánchez Ramos, J. Ranilla Pastor, Particle swarm optimization for hyper-parameter selection in deep neural networks, in: Proceedings of the Genetic and Evolutionary Computation Conference, (GECCO), 2017, pp. 481–488.
[116] S. Maas, M. Wiering, B. Verhaar, Reinforcement learning of a pneumatic robot arm controller, in: Eur Workshop on Reinforcement Learning (EWRL), 2005, pp. 23–24.
[117] Shingo Mabu, Kotaro Hirasawa, Jinglu Hu, A graph-based evolutionary algorithm: genetic network programming (GNP) and its extension using reinforcement learning, Evol. Comput. 15 (3) (2007) 369–398.
[118] K. Malan, A.P. Engelbrecht, A survey of techniques for characterising fitness landscapes and some possible ways forward, Inf. Sci. 241 (2013) 148–163.
[119] P. Mannion, S. Devlin, K. Mason, J. Duggan, E. Howley, Policy invariance under reward transformations for multi-objective reinforcement learning, Neurocomputing 263 (2017) 60–73.
[120] D. Maravall, J. de Lope, J.A. Martin H, Hybridizing evolutionary computation and reinforcement learning for the design of almost universal controllers for autonomous robots, Neurocomputing 72 (4–6) (2009) 887–894.
[121] V.V. Miagkikh, W.F. Punch, An approach to solving combinatorial optimization problems using a population of reinforcement learning agents, in: Conference on Genetic and Evolutionary Computation (GECCO), Morgan Kaufmann Publishers Inc., 1999, pp. 1358–1365.
[122] R. Miikkulainen, Neuroevolution, in: Encyclopedia of Machine Learning and Data Mining, 2017, pp. 899–904.
[123] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller, Playing Atari with deep reinforcement learning, in: Deep Learning Workshop, NIPS, 2013.
[124] K. Van Moffaert, M.M. Drugan, A. Nowé, Hypervolume-based multi-objective reinforcement learning, in: Evolutionary Multi-criterion Optimization (EMO), Springer, 2013, pp. 352–366.
[125] D.E. Moriarty, A.C. Schultz, J.J. Grefenstette, Evolutionary algorithms for reinforcement learning, J. Artif. Intell. Res. (JAIR) 11 (1999) 241–276.
[126] H. Moriguchi, S. Honiden, CMA-TWEANN: efficient optimization of neural networks via self-adaptation and seamless augmentation, in: Genetic and Evol Comp Conf (GECCO), ACM, 2012, pp. 903–910.
[127] H. Mühlenbeinz, G. Paass, From recombination of genes to the estimation of distributions i. binary parameters, in: Parallel Problem Solving from Nature PPSN IV, 1996, pp. 178–187.
[128] Masaya Nakata, Will N. Browne, Tomoki Hamagami, Keiki Takadama, Theoretical XCS parameter settings of learning accurate classifiers, in: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO, 2017, pp. 473–480.
[129] Trung Thanh Nguyen, Shengxiang Yang, J. Branke, Evolutionary dynamic optimization: a survey of the state of the art, Swarm Evol. Comput. 6 (2012) 1–24.
[130] Isao Ono, Tetsuo Nijo, Norihiko Ono, A genetic algorithm for automatically designing modular reinforcement learning agents, in: Genetic and Evolutionary Computation Conference (GECCO), 2000, pp. 203–210.
[131] E. Özcan, M. Misir, G. Ochoa, E.K. Burke, A reinforcement learning - great-deluge hyper-heuristic for examination timetabling, Int. J. Appl. Metaheuristic Comput. (IJAMC) 1 (2010) 39–59.
[132] A. Parker, G.S. Nitschke, Autonomous intersection driving with neuro-evolution, in: Genetic and Evolutionary Computation Conference, GECCO, Evolutionary Machine Learning Workshop, 2017, pp. 133–134.
[133] Topon Kumar Paul, Hitoshi Iba, Reinforcement learning estimation of distribution algorithm, in: Genetic and Evolutionary Computation GECCO, ACM, 2003, pp. 1259–1270.
[134] I. Petrova, A. Buzdalova, M. Buzdalov, Improved selection of auxiliary objectives using reinforcement learning in non-stationary environment, in: International Conference on Machine Learning and Applications, ICMLA, 2014, pp. 580–583.
[135] J.E. Pettinger, R.M. Everson, Controlling genetic algorithms with reinforcement learning, in: Genetic and Evolutionary Computation Conference (GECCO), 2002, p. 692.
[136] R. Poli, W.B. Langdon, N.F. Mcphee, A Field Guide to Genetic Programming, Lulu.com, 2008, http://www.gp-field-guide.org.uk/.
[137] W.B. Powell, Perspectives of approximate dynamic programming, Ann. Oper. Res. 241 (1) (2016) 319–356.
[138] P. Preux, R. Munos, M. Valko, Bandits attack function optimization, in: Congress on Evolutionary Computation (CEC), IEEE, 2014, pp. 2245–2252.
[139] F.-M. De Rainville, M. Sebag, C. Gagné, M. Schoenauer, D. Laurendeau, Sustainable cooperative coevolution with a multi-armed bandit, in: Genetic and Evol Computation Conference (GECCO), ACM, 2013, pp. 1517–1524.
[140] I. Rechenberg, Evolutionsstrategie, Frommann-Holzboog, 1994. Reprint PhD thesis 1971.
[141] P.M. Reed, D. Hadka, J.D. Herman, J.R. Kasprzyk, J.B. Kollat, Evolutionary multiobjective optimization in water resources: the past, present, and future, Adv. Water Resour. 51 (2013) 438–456.
[142] D.M. Roijers, P. Vamplew, S. Whiteson, R. Dazeley, A survey of multi-objective sequential decision-making, J. Artif. Intell. Res. (JAIR) 48 (2013) 67–113.
[143] D.M. Roijers, S. Whiteson, F.A. Oliehoek, Computing convex coverage sets for multi-objective coordination graphs, in: Algorithmic Decision Theory (ADT), Springer, 2013, pp. 309–323.
[144] D.M. Roijers, S. Whiteson, F.A. Oliehoek, Linear support for multi-objective coordination graphs, in: Inter Conf on Autonomous Agents and Multi-agent Systems (AAMAS), 2014, pp. 1297–1304.
[145] D.M. Roijers, S. Whiteson, P. Vamplew, R. Dazeley, Why multi-objective reinforcement learning? in: European Workshop on Reinforcement Learning (EWRL), 2015.
[146] G. Rudolph, Convergence analysis of canonical genetic algorithms, Trans. Neural Netw. (TNN) 5 (1) (1994) 96–101.
[147] G.A. Rummery, M. Niranjan, Online Q-learning Using Connectionist Systems, Technical report, Cambridge University Engineering Department, 1994.
[148] T. Schaul, J. Bayer, D. Wierstra, Yi Sun, M. Felder, F. Sehnke, T. Rückstieß, J. Schmidhuber, PyBrain, J. Mach. Learn. Res. (JMLR) 11 (2010) 743–746.
[149] J. Schmidhuber, Evolutionary computation versus reinforcement learning, in: Industrial Electronics Society (IECON), IEEE, 2000.
[150] J. Schmidhuber, Deep learning in neural networks: an overview, Neural Netw. 61 (2015) 85–117.
[151] J. Schmidhuber, Deep Learning, 2017, pp. 338–348.
[152] O. Schuetze, L. Jourdan, T. Legrand, E.-G. Talbi, J.L. Wojkiewicz, A multi-objective approach to the design of conducting polymer composites for electromagnetic shielding, in: Evolutionary Multi-criterion Optimization (EMO), LNCS, Springer, 2007.
[153] H.P. Schwefel, Evolution and Optimum Seeking, Wiley, 1995.
[154] P.J. Schweitzer, A. Federgruen, The asymptotic behavior of undiscounted value iteration in Markov decision problems, Math. Oper. Res. 2 (4) (1977) 360–381.
[155] E.O. Scott, J.K. Bassett, Learning genetic representations for classes of real-valued optimization problems, in: Conference on Genetic and Evolutionary Computation GECCO, ACM, 2015, pp. 1075–1082.
[156] O.M. Shir, Niching in Evolutionary Algorithms, Springer, 2012, pp. 1035–1069.
[157] K. Sim, E. Hart, An improved immune inspired hyper-heuristic for combinatorial optimisation problems, in: Genetic and Evol Computation Conference (GECCO), ACM, 2014, pp. 121–128.
[158] S.P. Singh, R.L. Lewis, A.G. Barto, J. Sorg, Intrinsically motivated reinforcement learning: an evolutionary perspective, IEEE Trans. Autonom. Mental Dev. 2 (2) (2010) 70–82.
[159] J. Smith, T.C. Fogarty, Operator and parameter adaptation in genetic algorithms, Soft Comput. 1 (2) (1997) 81–87.
[160] R.E. Smith, D.E. Goldberg, Reinforcement learning with classifier systems: adaptive default hierarchy formation, Appl. Artif. Intell. 6 (1) (1992) 79–102.
[161] J.A. Soria-Alcaraz, G. Ochoa, J. Swan, J.M. Carpio, H. Puga, E.K. Burke, Effective learning hyper-heuristics for the course timetabling problem, Eur. J. Oper. Res. (EJOR) 238 (2014) 77–86.
[162] L.B. Soros, K.O. Stanley, Identifying necessary conditions for open-ended evolution through the artificial life world of Chromaria, in: Proceedings of the International Conference on the Synthesis and Simulation of Living Systems ALife, 2014, pp. 793–800.
[163] P.O. Stalph, M.V. Butz, Current XCSF capabilities and challenges, in: Learning Classifier Systems (IWLCS), 2009, pp. 57–69.
[164] P.O. Stalph, M.V. Butz, Documentation of JavaXCSF, 2009.
[165] K.O. Stanley, R. Miikkulainen, Evolving neural network through augmenting topologies, Evol. Comput. 10 (2) (2002) 99–127.
[166] K.O. Stanley, R. Miikkulainen, Competitive coevolution through evolutionary complexification, J. Artif. Intell. Res. (JAIR) 21 (2004) 63–100.
[167] K.O. Stanley, R. Miikkulainen, Evolving a roving eye for GO, in: Genetic and Evolutionary Computation (GECCO), 2004, pp. 1226–1238.
[168] F. Stulp, O. Sigaud, Path integral policy improvement with covariance matrix adaptation, in: International Conference on Machine Learning (ICML), 2012.
[169] R.S. Sutton, A.G. Barto, Reinforcement Learning: an Introduction, MIT Press, 1998.
[170] I. Szita, C. Szepesvári, Model-based reinforcement learning with nearly tight exploration complexity bounds, in: Int Conf on Machine Learning (ICML), 2010, pp. 1031–1038.
[171] E.-G. Talbi, S. Cahon, N. Melab, Designing cellular networks using a parallel hybrid metaheuristic on the computational grid, Comput. Commun. 30 (4) (2007) 698–713.

[172] D. Thierens, An adaptive pursuit strategy for allocating operator probabilities, in: Genetic and Evolutionary Computation Conference (GECCO), ACM, 2005, pp. 1539–1546.
[173] S. Thrun, Exploring Artificial Intelligence in the New Millenium, Chapter Robotic Mapping: a Survey, Morgan Kaufmann, 2002.
[174] A.D. Tijsma, M.M. Drugan, M.A. Wiering, Comparing exploration strategies for q-learning in random stochastic mazes, in: IEEE Symposium Series on Computational Intelligence, SSCI, 2016, pp. 1–8.
[175] M. Tsuji, M. Munetomo, Linkage Analysis in Genetic Algorithms, Springer Berlin Heidelberg, 2008, pp. 251–279.
[176] P.D. Turney, Myths and Legends of the Baldwin Effect, CoRR, cs.LG/0212036, 2002.
[177] P. Vamplew, R. Dazeley, A. Berry, R. Issabekov, E. Dekker, Empirical evaluation methods for multiobjective reinforcement learning algorithms, Mach. Learn. 84 (1–2) (2011) 51–80.
[178] S. van den Dries, M.A. Wiering, Neural-fitted TD-leaf learning for playing Othello with structured neural networks, Trans. Neural Netw. Learn. Syst. (TNN) 23 (11) (2012) 1701–1713.
[179] M. van der Ree, M. Wiering, Reinforcement learning in the game of Othello: learning against a fixed opponent and learning from self-play, in: Adaptive Dynamic Programming and Reinforcement Learning (ADPRL), SSCI, IEEE, 2013, pp. 108–115.
[180] S. van Steenkiste, J. Koutník, K. Driessens, J. Schmidhuber, A wavelet-based encoding for neuroevolution, in: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO, 2016, pp. 517–524.
[181] W. Wang, M. Sebag, Multi-objective Monte Carlo tree search, in: Asian Conference on Machine Learning, Springer, 2012, pp. 1–16.
[182] C.J.C.H. Watkins, P. Dayan, Technical note Q-learning, Mach. Learn. 8 (1992) 279–292.
[183] C.C. White, K.W. Kim, Solution procedures for vector criterion Markov decision processes, J. Large Scale Syst. 1 (1980) 129–140.
[184] David R. White, Software review: the ECJ toolkit, Genet. Program. Evolvable Mach. 13 (1) (2012) 65–67.
[185] S. Whiteson, Evolutionary computation for reinforcement learning, in: Reinforcement Learning: State-of-the-art, Springer, 2012, pp. 325–355.
[186] S. Whiteson, P. Stone, Evolutionary function approximation for reinforcement learning, J. Mach. Learn. Res. (JMLR) 7 (2006) 877–917.
[187] S. Whiteson, P. Stone, On-line evolutionary computation for reinforcement learning in stochastic domains, in: Genetic and Evol Comp Conf (GECCO), ACM, 2006, pp. 1577–1584.
[188] D. Whitley, MK landscapes, NK landscapes, MAX-kSAT: a proof that the only challenging problems are deceptive, in: Conference on Genetic and Evolutionary Computation (GECCO), ACM, 2015, pp. 927–934.
[189] L.D. Whitley, V.S. Gordon, K.E. Mathias, Lamarckian evolution, the Baldwin effect and function optimization, in: Parallel Problem Solving from Nature - PPSN, Springer, 1994, pp. 6–15.
[190] M. Wiering, M. van Otterlo (Eds.), Reinforcement Learning: State-of-the-art, Springer, 2012.
[191] M.A. Wiering, E.D. de Jong, Computing optimal stationary policies for multi-objective markov decision processes, in: Approximate Dynamic Programming and Reinforcement Learning (ADPRL), SSCI, IEEE, 2007, pp. 158–165.
[192] D. Wierstra, T. Schaul, T. Glasmachers, Yi Sun, J. Peters, J. Schmidhuber, Natural evolution strategies, J. Mach. Learn. Res. (JMLR) 15 (1) (2014) 949–980.
[193] D. Wyatt, L. Bull, A Memetic Learning Classifier System for Describing Continuous-valued Problem Spaces, Springer Berlin Heidelberg, 2005, pp. 355–395.
[194] S.Q. Yahyaa, M.M. Drugan, B. Manderick, Thompson sampling in the adaptive linear scalarized multi-objective multi-armed bandit, in: International Conference on Agents and Artificial Intelligence, (ICAART), 2015.
[195] George Zhu, D.J. Lizotte, Jesse Hoey, Scalable approximate policies for markov decision process models of hospital elective admissions, Artif. Intell. Med. 61 (1) (2014) 21–34.
[196] V. Zhumatiy, F. Gomez, M. Hutter, J. Schmidhuber, Metric state space reinforcement learning for a vision-capable mobile robot, in: Intelligence Autonomous Systems 9, IOS Press, 2003.