Synergies between Evolutionary Algorithms and Reinforcement Learning

Madalina M. Drugan
Technical University of Eindhoven
madalina.drugan@gmail.com
Agenda

• Reinforcement Learning: short introduction
• Evolutionary Computation: short introduction

I. Evolutionary Computation in Reinforcement Learning
   1. Learning classifier systems / Artificial neural networks
   2. Multi-criteria decision making and reinforcement learning

II. Reinforcement Learning in Evolutionary Computation
   1. On-line adaptive operator selection
   2. Monte Carlo Tree Search (MCTS)
   3. Hyper-heuristics

III. Hybrids: Natural Evolution, Co-evolutionary algorithms
• Discussion
Reinforcement Learning (RL): an introduction

[Sutton and Barto, 1998] [Wiering and van Otterlo, 2012]

• On-line / off-line learning technique with the goal of maximizing the long term reward

• RL solves environments that are modelled as Markov decision processes (MDP) by rewarding good actions and punishing bad actions

• The exploration / exploitation dilemma
  • exploration -> try out actions of which the outcome is still uncertain
  • exploitation -> select actions which have shown to be good in the past

• The best actions are identified by trying them out and evaluating their long term consequences

• Applications: game theory, robot control, control theory, planning, operations research, etc.
Markov decision processes (MDPs)

• MDP studies decision making under uncertainty

• An MDP is characterized with a tuple $< S, A, T, R >$
  • the state – search space $S = \{s_1, s_2, \ldots, s_N\}$, where $s_t \in S$
  • a set of actions $A = \{a_1, a_2, \ldots, a_N\}$ available to the agent in each state.
  • a transition probability $T(s, a, s')$ mapping state action pairs to a probability distribution over successor states
  • a reward function $R: S \times A \times S \rightarrow \mathbb{R}$ to denote the expected reward when the agent makes the transition from state $s$ to state $s'$ using action $a$.
  • $r_t$ is the immediate scalar reward obtained at time $t$.

• This process is Markovian $\rightarrow$ the distribution over the next states is independent on the past given the current state and action.
Markov decision processes (MDP)

• An example

• A policy $\pi$ is a sequence of state-action pairs
• Policies can perform deterministic or stochastic action selection

• **Goal of MDPs**: to find the best policy which is the policy that receives the most rewards
MDPs paradigms: Dynamic programming (DP)

- Bellman equation (1957)
  - *expected value* of a state $V^\pi(s)$ is defined in terms of the immediate reward and the expected future rewards
  - the value of a state $s$ under an optimal policy $\pi^*$ is
    \[
    V^*(s) = \max_{a \in A} \sum_{s' \in S} T(s, a, s') (R(s, a, s') + \gamma V^*(s'))
    \]

- DP
  - breaks the problem up in smaller problems
  - assumes that the transition model $T$ is known $\rightarrow$ strong assumption
  - Assumes that the reward function $R$ is known $\rightarrow$ mild assumption
Algorithms for solving MDPs

• **Value iteration**
  • Each iteration,
    • the value function is updated
    • a new optimal policy is computed for the new value function

  • An optimal policy maximizes the expected cumulative reward for any initial state

• **Policy iteration**
  • First, fully evaluate a policy
  • Then improve the policy
Fundamental DP algorithms: policy iteration

• Two phases:
  • **policy evaluation:**
    • evaluate the value function $V^\pi$ of a fixed policy $\pi$
    • Bellman equation is an update rule that extends the planning horizon with one step
    • The current value function $V_k^\pi(s)$ is updated to $V_{k+1}^\pi(s)$
      \[
      V_{k+1}^\pi(s) = \sum_{s' \in S} T(s, \pi(s), s') (R(s, \pi(s), s') + \gamma V_k^\pi(s))
      \]
    • The sequence of $V_k^\pi(s)$ converges as $k$ goes to infinity
  • **policy improvement:**
    • improves the quality of the policy $\pi$
      \[
      \pi'(s) = \text{argmax}_{a \in A} \sum_{s' \in S} T(s, a, s') (R(s, a, s') + \gamma V^\pi(s'))
      \]
    • policy iteration
      \[
      \pi_0 \rightarrow V^{\pi_0} \rightarrow \pi_1 \rightarrow V^{\pi_1} \rightarrow \ldots \rightarrow \pi_k \rightarrow V^{\pi_k} \rightarrow \ldots \rightarrow \pi^*
      \]
Fundamental DP algorithms: value iteration

• A simplified version of policy iteration that merges the policy evaluation and policy improvement in a single step

• The value function is updated on the fly

\[ V_{k+1}(s) = \max_{a \in A} \sum_{s' \in S} T(s, a, s') (R(s, a, s') + \gamma V_k(s)) \]

• \( V_k \) converges to the optimal value \( V^* \)

\[ V_0 \rightarrow V_1 \rightarrow \ldots \rightarrow V_k \rightarrow V_{k+1} \ldots \rightarrow V^* \]

• Very popular due its simplicity

• Other MDPs

Partially observable Markov decision processes (POMDPs)

• Standard MDPs \( \rightarrow \) completely observable MDPs (CO-MDPs)

• POMDPs \( \rightarrow \) do not have direct access to the current state
Model free RL paradigms: Q-learning [Watkins, 1989]

- The transition model and the reward functions are not known a-priori
  - Value function denotes cumulative rewards
  - The goal of RL is to optimize the long term reward intake

- Off-policy learning algorithm meaning that it learns a policy \( \pi \) while searching for the optimal policy \( \pi^* \)
- Q-values are estimated based on immediate rewards and actions
- \( \epsilon \)-greedy policy selects often the best action and with small probability another action

- Decrease the learning rate such that Q-values converge to the true values
- Explore all actions an infinite number of times for convergence
Model free RL paradigms: Q-learning

[Watkins, 1989]

- Q-learning algorithm’s inner loop
  - select an action $a$ based on $Q$-values and an ($\varepsilon$- greedy) exploration strategy
  - perform action $a$
  - observe the new state $s'$ and receive reward $r_t$
  - update $Q$-values
    \[
    Q(s, a) \leftarrow Q(s, a) + \alpha \cdot \left( r_t + \gamma \cdot \max_{a' \in A(s')} Q(s', a') - Q(s, a) \right)
    \]
  - $s \leftarrow s'$

- Converges to the optimal policy $\pi^*$
Model free RL

• **SARSA**
  • On-policy algorithm that optimizes the policy that is executed
    \[ Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha (r_t + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)) \]
  • max operator in Q-learning is replaced with the action estimate according to the policy
  • converges in infinite time to the optimal policy if
    • all states and actions were tried infinitely often
    • exploration decreases over time

• **QV learning**
  • On policy algorithm that uses also states values to speed up learning
    \[ Q_{t+1}(s_t, a_t) = r_t + \gamma V_t(s_{t+1}) \]
    \[ V_{t+1}(s_t) = r_t + \gamma V_t(s_{t+1}) \]
Real-world applications for reinforcement learning

- Playing online games:
  - Atari games (Pacman)
    - generative RL using Bayesian rules
    - Q-learning
    - Monte Carlo Tree Search

- Go games --> Monte Carlo Tree Search using Multi-armed bandits

- Othello, Pacman

- Mazes --> Q-learning
Real-world applications for RL

• Robotics
  • Vision with model-based RL
  • Control of a robot arm with Q-learning
  • POMDPs for navigating a mobile robot

• Traffic light control
  • Model based RL and game theory
  • Multi-agent systems in
    • traffic distribution in a network

• RL is considered slow and resource consuming compared with other heuristics but can learn online under changing environmental conditions
Evolutionary Computation (EC): an introduction

- EC is a subfield of global optimization
  - popular for its large number of variants for different types of real-world applications
  - Heuristic that finds a fit solution that could be used in practice

- Genetic algorithms (GA) → most popular EC for solutions with binary representation
- Evolutionary Strategies (ES) → parameter free EC for continuous optimization
- Evolutionary Multi-objective optimization (EMO) → a branch of MCDM concerned with multi-objective optimization
- Co-evolutionary algorithm → assume two populations of individuals that compete or cooperate for survival
- Learning classifier systems (LSC) → evolves if-then rules called classifiers and it is related to evolving neural networks
Evolutionary Computation (EC): highlights

• EC uses a **population of individual** solutions that exchange useful information with mechanisms that resembles recombination and mutation
  • **Selection** is a stochastic process of selecting the best individuals from the population
    • The quality of each solution is evaluated using a **fitness function**
  • **Mutation** is a local perturbator that generates children in the proximity of the parents
  • **Recombination** is a non-local perturbator that generates children that preserve (structural) communalities in the parents
  • **Convergence**: all solutions are the same, or a good enough solution has been reached

• Exploration / exploitation trade-off
  • Exploration → generation of new solutions
  • Exploitation → selection of good solutions
I.1. Evolutionary Algorithms for RL

[Moriarty et al, 1999]

- EARL evolves policies with EAs
  - Generate new policies using genetic operators, i.e. mutation
  - Associate each policy a fitness value
  - Select the best policies to produce the next generation

- Policy representation
  - Rule-based policy representation
    - each gene is a condition-action rule that maps a set of states to an action
    - distributed rule-based representation of a policy over several EAs evolved separately
  - Parameter representation for evolving neural networks
    - each gene is a weight in a neural network
    - distributed network based policies constructs different parts of a neural network that optimise different tasks using EAs
Evolutionary Algorithms for Reinforcement Learning

• Fitness and credit assignment
  • an agent interacts with environment
  • fitness values are averaged over time
    • sub-policy credit assignment for distributed policies
  • what is the effect of current action vs past actions on the current reward for sparse pay-offs like reaching the goal with a robot

• Selection
  • roulette selection proportional with fitness values

• Genetic operators → specific to each policy representation
  • triggered operators for learning classifier systems (LCS)
  • real coded operators for strings of weights for neural networks
Evolutionary Algorithms for RL: variants

• Neuro-evolution
  • evolves neural networks to approximate complex functions
• Topology and Weight evolving artificial neural networks (TWEANN)
  • a state is represented by features, each feature is an input in an network
  • an action value is the output of a separately evolved network
• NeuroEvolution of Augumented Topologies (NEAT)

• Learning classifier systems (LCS) [Holland, 1975]
  • Used by RL to optimise and learn in complex non-stationary environments
  • Precedes and generalizes evolutionary computation
  • Population of rules that represent a policy
  • Uses EC to evolve policies
  • Michigan approach ➔ an entire population is a policy
  • Pittsburgh approach ➔ an individual is a single policy
Concluding remarks: Evolutionary Algorithms for RL

• scale-up to large state spaces
  • *policy generalization* groups together states for which the same action is required
    • varies considerably with the rules they encode
    • level of abstraction is higher than for a normal policy
  • *policy selectivity* means the knowledge about bad decisions is not represented
    • Reduce computational effort by focusing on promising actions

• non-stationary environments
  • tracking in non-stationary environments
  • a statistical model of agent uncertainty

• incomplete state information
  • reward policies that avoid the ambiguous states
  • model hidden states
I.2. Evolutionary multi-objective optimization (EMO) vs Multi-criteria decision making (MCDM)

- MCDM assumes the involvement of a decision maker (DM)
- EMO automatically optimizes in many objectives
- Classification of MCDM methods
  - *Pareto-based ranking* $\Rightarrow$ DM is indifferent (all solutions are equally important)
    - Pareto dominance relation $\Rightarrow$ partial order relation where two solutions can be better in one objective and worse in another objective
  - Scalarized relation $\Rightarrow$ DM prefers a region of the search space
    - A scalarization function transforms the value vector of a solution in a scalar value
  - Preference ranking $\Rightarrow$ DM has preference for an objective
  - Interactive methods $\Rightarrow$ DM interacts with the environment
Pareto dominance relation

- A reward vector can be better than another reward vector in one objective and worse in another objective.
- The natural order relationship for multi-objective search spaces.
- Examples of relationships between reward vectors.

<table>
<thead>
<tr>
<th>relationship</th>
<th>notation</th>
<th>relationships</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \mu_1 ) dominates ( \mu_2 )</td>
<td>( \mu_2 &lt; \mu_1 )</td>
<td>( \exists j, \mu_2^j &lt; \mu_1^j ) and ( \forall o, j \neq o, \mu_2^o \leq \mu_1^o )</td>
</tr>
<tr>
<td>( \mu_1 ) weakly dominates ( \mu_2 )</td>
<td>( \mu_2 \leq \mu_1 )</td>
<td>( \forall j, \mu_2^j \leq \mu_1^j )</td>
</tr>
<tr>
<td>( \mu_1 ) is incomp with ( \mu_2 )</td>
<td>( \mu_2 \parallel \mu_1 )</td>
<td>( \mu_2 \not\approx \mu_1 ) and ( \mu_1 \not\approx \mu_2 )</td>
</tr>
<tr>
<td>( \mu_1 ) is non-domin by ( \mu_2 )</td>
<td>( \mu_2 \not\in \mu_1 )</td>
<td>( \mu_2 &lt; \mu_1 ) or ( \mu_2 \parallel \mu_1 )</td>
</tr>
</tbody>
</table>

- The Pareto front is the set of expected reward vectors that are non-dominated by the other expected reward vectors.
- All the solutions in the Pareto front are considered equally important.
\( L_p \) scalarization function

- **Goal:** \( L_p \) transforms the multi-objective search space into a single objective space using a scalarization function.

- Weighted power \( p \) sums of reward values, where a set of predefined weights is considered:
  \[
  f_p(\mu_i) = \left( \sum_{j=1}^{D} \omega_j \cdot (\mu_i^j - z^j)^p \right)^{\frac{1}{p}}
  \]

- \( L_p \) function can find all solutions of any shape, i.e. non-convex.
- The reference point \( z = (z_1, z_2, \cdots, z_D) \) is an extra parameter.
- \( L_1 \) function is a *linear scalarization* function.
- \( L_\infty \) function is a *Chebyshev scalarization* function.
Multi-objective reinforcement learning (MORL)

- Early research (80s): multi-objective dynamic programming and multi-objective MDPs
- Main differences with (single objective) MDPs
  - Immediate reward values are replaced with reward vectors
  - A set of optimal policies of incomparable quality in all objectives
  - Computationally intractable even for simple small environments
- Techniques from MOO and MCDC are incorporated in RL with reward vectors
- **Goal**: to identify one or a set of Pareto optimal policies
- When compared with RL, MORL has:
  - Extra computational challenges
  - Different exploration / exploitation trade-offs
  - More complicated experimental setups
Evolutionary multi-objective optimisation (EMO) in RL

- **Multi-objective Markov Decision Processes (MOMDPs)**
  - compute all Pareto optimal policies
  - tuples of rewards instead of a single reward
  - stationary and non-stationary deterministic environments

- **Multi-objective Reinforcement Learning (MORL)**
  - important differences with single objective reinforcement learning
    - several actions can be considered to be the best according to their reward tuples.
  - techniques from EMO should be used in the multi-objective RL framework to improve the exploration/exploitation trade-off
  - complex and large multi-objective environments.

- **Multi-objective multi-armed bandits (MOMABs)**
  - single state reinforcement learning algorithm
  - various variants of multi-armed bandits extended to reward vectors
Multi-objective Markov decision processes (MOMDPs)


• Multi-objective dynamic programming (MODP)
  • A reward vector $r = (r_1, r_2, \ldots, r_m)$ with m dimensions
  • A reward function $R(s, a, s') = (R_1(s, a, s'), \ldots, R_m(s, a, s'))$
  • A set of Pareto optimal policies
  • An agent will select one or more Pareto optimal policies
  • Value or policy iteration multi-objective dynamic programming

• Pareto dominance relation or scalarization functions are used to compute and track the Pareto optimal policies
• For both non-stationary and deterministic environments
• The corresponding algorithms converge to the unique Pareto optimal set of policies
Value iteration multi-objective dynamic programming

- [Wiering and de Jong, 2007] [Wiering et al 2014]
- The value function is a vector $V^i(s) = (V_1^i(s), V_2^i(s), \ldots, V_m^i(s))$
- The set of non-dominated value functions is denoted with $V^O(s) = \{V^i(s) \mid V^i(s) \text{ isn’t dominated by a policy in } s\}$
- The set of non-dominated Q-values $Q^O(s, a) = \{Q^i(s, a) \mid Q^i(s, a) \text{ isn’t dominated in } s, a\}$
- The non-dominated operator tells whether a policy $i$ is non-dominated in the state $s$ by any value function in $V^D(s)$ $\text{ND}(V^i(s), V^D(s)) \iff \#V^j(s) \in V^D(s) ; V^j(s) \succ V^i(s)$
- The Pareto optimal operator $PO(Q^D(s, a)) = \{Q^i(s, a) \mid Q^i(s, a) \in Q^D(s, a) \land \text{ND}(Q^i(s, a), Q^D(s, a))\}$
- The dynamic programming operator for deterministic environments $DP(Q^D(s, a)) = \{R(s, a, s') \oplus \gamma V^O(s') \mid P(s, a, s') = 1.0\}$
The optimal set of Q-values vectors \( Q^0^* \)
\[
Q^0^* = (PO(DP(Q_0)))^*
\]
where * means the operator is repeated an infinite number of times

This dynamic programming algorithm converges to the maximal Pareto optimal set of policies because it eliminates all dominated policies

The set of Pareto optimal policies is independent of the usage of the Pareto optimal operator inside the dynamic programming operator
\[
PO(DP'(Q_0))^* = (PO(DP(Q_0)))^*
\]
where \( DP' \) does not make use of the \( PO \) operator at all
\[
DP'(Q^D(s, a)) = \{ R(s, a, s') \oplus \gamma V^D(s') | P(s, a, s') = 1.0 \} \]
Multi-objective RL (MORL) algorithms

• **Pareto Q-learning** algorithm applies a Pareto operator for sets to keep track of Pareto optimal policies [van Moffaert et al, 2014]

• **Scalarization based RL** [Vamplew et al, 2010][Brys et al, 2014] [van Moffaert et al, 2013a]
  • Weighted linear scalarization functions (most algorithms)
  • Goal: identifying all Pareto optimal policies on a convex front

• **Hyper-volume unary** indicator is used to measure the quality of solutions [Wang & Sebag, 2013] [van Moffaert et al, 2013b]
  • Another function that transforms the multi-objective search space

• **Preference based** dominance relations are used with multi-criteria RL [Gabor et al, 1998]
  • Only a region on the Pareto front is required
  • Interaction with the user to select the preferred solutions
Multi-objective Q-learning

- $Q$-values are extended to $Q$-vectors $\rightarrow$ one value for each objective
  $Q(s, a) = (Q^1(s, a), \ldots, Q^M(s, a))$
- A set of linear scalarization functions to identify convex fronts
- $L_\infty$ norm identifies solutions on any Pareto front
- $Q$-values are updated in each objective $\rightarrow$ convergence properties are inherited from $Q$-learning
- The selection of next actions depends on the used scalarization function
- The main differences with single objective RL
  - **Pareto front** includes several reward values that are equally good (incomparable) and a set of $Q$-vectors with incomparable best values.
  - **Updating rules** for the $Q$-vectors and the selection of the best next action when compared with single objective MDPs.
Scalarized multi-objective $Q$-learning

- Multi-objective MDPs $\rightarrow$ immediate reward vectors
  \[ R(s, a) = (R^1(s, a), R^2(s, a), \ldots, R^m(s, a)) \]

- $Q$ – values
  \[ Q(s, a) = (Q^1(s, a), Q^2(s, a), \ldots, Q^m(s, a)) \]

- Scalarized $Q$-values
  \[ SQ(s, a) = \sum_{o=1}^{m} w^o Q^o(s, a) \]

- $\epsilon$-greedy selects the action $a$ with the largest $SQ$ value with a high probability

- Try action $a$ and collect reward $r(s, a)$

- For each objective $o$
  \[ Q(s, a, o) \leftarrow Q(s, a, o) + \alpha \cdot [r(s, a, o) + \gamma Q(s', \text{max}_{a'} o) - Q(s, a, o)] \]

- $s \leftarrow s'$
A scalarized $Q$-learning algorithm
Model based multi-objective reinforcement learning

[Wiering et al, 2014]

• Most MORL are model free algorithms that use scalarization functions to transform the reward vectors into reward values

• Model based MORL
  • estimate the model of the environment using frequencies stored in a table
  • solve this model based on value iteration multi-objective DPs

• Exploration strategies
  • least visited exploration
    • counts the times each action was taken
  • random exploration
    • actions are selected randomly
Hypervolume based MORL

[Wang & Sebag, 2013]

- Multi-objective Monte Carlo tree Search (MOMCTS)
- Hypervolume unary indicator is used to select a node in MOMCTS
- MOMCTS inherits computational problems for hypervolume unary indicator
- MOMCTS is combined with Pareto dominance to improve the diversity of solutions
- MOMCTS performs better than scalarized MORL on the bi-objective Deep Sea environment

[van Moffaert et al, 2013b]

- Multi-objective $Q$-learning uses hypervolume indicator to evaluate the quality of the Pareto front
- Action selection mechanism that maximises the hypervolume indicator
- Compared with scalarization based MORL, hypervolume based MORL has
  - better performance
  - narrower Pareto front
Real-world applications for MORL

• Many real-world problems are inherently multi-objective

• **Traffic light control** --> linear scalarized multi-objective Q-learning

• **Control problems** (like wet clutch) with two or more objectives --> adaptive linear scalarized MORL

• Hand or automatically generated multi-objective MDPs --> these problems were approached with both scalarized and Pareto MORL

• RL that were transformed into multi-objective environments
  • Mounting car problems
  • Maze like problems, i.e. Deep Sea World

• Preference based MORL that requires user interactions → scalarized MORL
RL paradigms: Multi-armed bandits (MAB)

• Mathematical formalism that studies the convergence properties of RL with a single state

• An example: a gambler faces a row of slot machines and decides
  • which machines to play,
  • how many times to play each machine
  • in which order to play them

• When played, each machine provides a reward generated from an unknown distribution specific to that machine.

• The goal of the gambler is to maximize the sum of rewards earned through a sequence of lever pulls.
Multi-armed bandits (MAB) algorithms

• **Intuition for stochastic MAB**
  - The algorithm starts by fairly exploring $N$-arms (= actions)
  - An agent selects between $N$-arms such that the expected reward over time is maximized
    - The distribution of the stochastic payoff of the different arms is assumed to be unknown to the agent.
  - Gradually focus on the arm with the best performance

• **Exploration / exploitation trade-off**
  - Explore sub-optimal arms that might have been unlucky
  - Exploit optimal arm as much as possible

• **Performance measures**
  - Cumulative regret is a measure of how much reward a strategy loses by playing the suboptimal arms
Multi-objective multi-armed bandits

[Drigan & Nowe, 2013]

• Multi-armed bandits use reward vectors
• Evolutionary Multi-objective optimization (EMO) techniques are used to design computationally efficient MOMABs
• The exploration / exploitation trade-off
  • EMO $\rightarrow$ exploration means evaluation of new solutions in a very large search space where states cannot be enumerated
    $\rightarrow$ exploitation means to focus the search in promising regions where the global optimum could be located
  • MAB $\rightarrow$ exploration means to pull arms that have suboptimal mean reward values
    $\rightarrow$ exploitation means to pull the currently identified best arm(s)
• MOMABs $\rightarrow$ a finite set of arms and reward vectors generated from stochastic distributions
Multi-objective multi-armed bandits (MOMABs)

- **Goal**: maximize the returned reward; or to minimize the cumulative or immediate regret of pulling suboptimal arms
- We assume that all Pareto optimal arms are equally important and need to be identified
- **Performance measures**
  - Pareto regret $\rightarrow$ sum of the distances between each suboptimal arm and the Pareto front
  - Variance regret $\rightarrow$ variance in using the Pareto optimal arms
  - KL divergence measure
- **Theoretical analysis**
  - Upper and lower bounds on expected cumulative regret
- **Challenges**
  - Large and complex *stochastic* multi-objective search spaces
  - Non-convex Pareto fronts
The bi-objective transmission problem of wet clutch

- **Wet clutch**: an application from control theory
- **Optimise the functionality of the clutch**:  
  - the optimal *current profile* of the electro-hydraulic valve that controls the pressure of the oil to the clutch  
  - the *engagement time*.
- **Stochastic output data** —> some external factors, such as the surrounding temperature, cannot be exactly controlled.
- **Optimise parameters** —> minimise the clutch's profile and the engagement time in varying environmental conditions.
Stochastic discrete MOMAB problems

• $K$-armed bandit, $K \geq 2$, with independent arms
• The reward vectors have $D$ –objectives, where $D$ fixed

• An arm $i$ is played at time steps $t_{1,i}, t_{2,i}, \ldots$
• The corresponding reward vectors $X_{i,t_1}, X_{i,t_2}, \ldots$ are independently and identically distributed according to an unknown law with unknown expectation vectors

• The goal of MOMAB:
  • Identify the set of best arms by maximising rewards in all objectives
  • The arms in the Pareto front are considered equally important and should be equally pulled
  • Minimise the regret (or the loss) of not selecting Pareto optimal arms
Performance metric: Pareto regret

- $\Delta_i = \|\nu_i^* - \mu_i\|_2 \rightarrow$ empirical distance between an arm $\nu$ and the Pareto front

- Let $\nu_i^*$ be the virtual reward vector of the arm $\nu$ such that $\mu_i$ has the minimum distance to $\nu_i^*$.
  - $\nu_i^* = \|\mu_i - \epsilon_i\|_2$ is incomparable with all reward vectors in the Pareto front

- The expected Pareto regret for a learning algorithm after $\nu$ arm pulls is
  $$\mathbb{E}[R_n] = \sum_{i=1}^{K} \Delta_i \cdot \mathbb{E}[T_i(n)]$$
Pareto MAB algorithms

- **Definition:** a multi-objective MAB algorithm that uses the *Pareto partial order relationship*

- The Pareto regret metric is used to upper bound the performance of the designed Pareto MAB algorithms

- **Challenges** in designing Pareto MAB algorithms:
  - *Pareto front identification selects a representative Pareto set of arms*
  - *The exploitation/exploration trade-off:*
    - Exploration: pull suboptimal arms that might be unlucky
    - Exploitation: pull as much as possible the optimal arms
  - Optimising the performance of Pareto MABs in terms of upper and lower bounds on expected and/or immediate regret
  - Ameliorate the performance of Pareto MABs for large sets of arms
Pareto Upper Confidence Bound (PUCB1)

- Straightforward generalisation of UCB1
  - operator selection [Fialho et al, 2009]
  - learning the utility of swap operations in combinatorial optimisation [Puglierin et al, 2013]

- Maximises the reward index \( \hat{\mu}_i + \sqrt{\frac{2 \ln(n \sqrt[D]{|A^*|})}{n_i}} \)

**PUCB1 algorithm**

- Each iteration, a Pareto front is calculated using
  \[
  \hat{\mu}_h + \sqrt{\frac{2 \ln(n_h \sqrt[D]{|A^*|})}{n_h}} > \hat{\mu}_i + \sqrt{\frac{2 \ln(n_i \sqrt[D]{|A^*|})}{n_i}}
  \]
- One of the arms from the Pareto front is selected and pulled
- Means are updated
  \[
  \sum_{i \notin A^*} \frac{8 \cdot \log(n \sqrt[D]{|A^*|})}{\Delta_i} + (1 + \frac{\pi^2}{3}) \cdot \sum \Delta_i
  \]

- The upper bound is
- The worst-case performance of this algorithm is when the number of arms \( K \) equals the number of optimal arms
Pareto front identification

This policy is an extension of the best arm identification algorithm [Audibert et al, 2010] for a set of arms of equal quality.

The $m$-best arm identification algorithm assumes that the $m$-best arms can be totally ordered.

The algorithm

Let $A_1 = \{1, \ldots, K\}$, $\log(K) = \frac{1}{2} + \sum_{i=2}^{K} \frac{1}{i}$, $n_0 = 0$ and for $k \in \{1, \ldots, K - 1\}$

$$n_k = \left\lfloor \frac{\log(D|A^*|)}{\log(K) + \log(D|A^*|)} \cdot \frac{n - K}{K + 1 - k} \right\rfloor$$

For all rounds $k = 1, 2, \ldots, K - 1$

1. For each arm $i \in A_k$, select it for $n_k - n_{k-1}$ rounds
2. Let $A_{k+1} = A_k \setminus \text{argmin}_{i \in A_k} \hat{x}_{i,n_k}$ the arm to dismiss in this round

Let the remaining set of arms be the Pareto optimal set of arms $A^*$
Annealing Pareto Knowledge gradient

[Yahyaa et al, 2014]

- **Knowledge gradient policy** is a reinforcement learning algorithm where the reward vectors are updated using Bayesian rules.
- Decrease uncertainty around the arms.
- The algorithm:
  - All arms are pulled once.
  - Iteratively, extreme arms are identified as either Pareto optimal or deleted as suboptimal arms.
  - Stop when there are no more arms to classify.
Challenges in designing scalarized MOMABs

[Drugan & Nowe, 2014] [Drugan, 2015a][Drugan, 2015b]

• Identify the entire Pareto front
  • Large and non-convex Pareto fronts
  • Non-uniform distributions of arms on the Pareto front

• Optimise the performance of scalarized MOMABs in terms of upper and lower regret bounds
  • Scalarized / Pareto regret metric
  • The Kullback-Leibler divergence regret metric

• Exploitation/exploration trade-off:
  • Exploration: sample scalarization functions, and pull arms that might be unluckyly identified as suboptimal
  • Exploitation: pull as much as possible the Pareto optimal arms of relevant scalarization functions
Concluding remarks on multi-objective reinforcement learning

• Multi-objective reinforcement learning (MORL)
  • Follows closely the latest developments in RL and MOO, but also MCDM

• Multi-objective multi-armed bandits (MOMAB)
  • New theoretical tools needed to study the performance of MORL algorithms

• Open research questions
  • Computationally efficient exploitation / exploration trade-off
  • Adequate performance measures for MORL and MOMABs
  • Advanced MOO and MCDM techniques to improve the performance of MORL and MOMAB algorithms
  • Challenging real world problems to motivate MORL and MOMABs paradigms
II. Reinforcement learning in Evolutionary Computation

II.1. Adaptive operator selection for Evolutionary Computation (EC)

- Online parameter selection as opposite to off-line parameter selection
- Uses reinforcement learning to select operators
  - Adaptive pursuit → pursuit often the operator that improves the most the results
  - Multi-armed bandits like UCB1 to adaptively select the best operator
  - SARSA
- Applied in tuning the parameters of EC algorithms

II.2. Monte Carlo decision trees for optimizing in continuous and discrete spaces

II.3. Hyper-heuristics
II.1. Adaptive operator selection

• Motivation:
  • the performance of EAs depends on the used parameters
  • the performance of a genetic operator depends on the landscape
  • an operator can have different performance in different regions of the landscape

• Tuning genetic operators
  • Selection of parameters
  • Mutation rates / Recombination exchange rates
  • Population size
  • Variable neighbourhood size (local search)

• Online learning strategy
  • The algorithms should learn relatively fast the best operator
  • There are several operators that perform similarly
Adaptive pursuit strategy (AP)  
[Thierens, 2005]

• Each operator $\mathbf{U}$ has associated a probability value $P_i^{(t)}$ of selection and an estimated reward value $Q_i^{(t)}$

• Online operator selection algorithm with fixed target probabilities is a step like distribution $D = \{p_M, p_m, \ldots, p_m\}$
  • $p_M$ has a large probability value to select often the best operator
  • $p_m$ has a small non zero probability to select any suboptimal operator

• The iterative algorithm
  • Pursuit with probability $P_v^{(t)}$ the operator $v$ with the maximal estimated reward $Q_v^{(t)}$
  • Get reward vector $R_v^{(t)}$ for the operator $v$
  • Update reward value $Q_v^{(t)}$ using the immediate reward $R_v^{(t)}$
  • High rank the estimated reward distribution $Q_v^{(t)}$ and set the values in vector $r$
  • For each operator $U$, update the selection probabilities:
    $$ P_i^{(t+1)} \leftarrow P_i^{(t)} + \beta \cdot (D_r[i] - P_i^{(t)}) $$
UCB1 for online operator selection

[Fialho at al, 2010]

• Each operator is considered an arm with unknown probability of getting a reward

• The reward function for operator $\mathcal{U}$ contains

  • the estimated value for the operator $\hat{\mu}_i$
  • the exploitation coefficient $c \sqrt{\frac{2 \log \sum_j n_j}{n_i}}$

  $n_i$ is the number of times the operator $\mathcal{U}$ was selected

• Remarks

  • Originally, UCB1 has positive sub-unitary values
  • Setting up $C$ is important for any fitness landscape
  • UCB1 detects changes in the environment but reacts quite slow
  • UCB1 is combined with other optimisation techniques to improve the performance of the online operator selection algorithm
UCB1 for online operator selection

- Performance of operator selection depends on
  - the improvement measure considered (difference in fitness value and / or diversity)

- Techniques to improve the performance of UCB1
  - Detect a change in the distribution with Page-Hinkley statistical tests
  - Weigh the operators using their frequency in applying it
  - Area under curve is also used as a measure of improvement in UCB1
  - Extreme values operator selection focuses on extremes to encourage exploration

- Hyper-parameter tuning, or tuning the tuner

- Off-line parameter tuning with F-race

- UCB1 is used to select solutions that adapt the CMA-ES matrix in continuous MO-CMA-ES [Loshchilov et al, 2011]
Generalised adaptive pursuit

[Drugan & Thierens, 2011]

• Any target operator probabilities can be considered, i.e. static or time dependent

• Assumption: Exploiting a set of related operators is beneficial for performance

• We can consider **two high value** probabilities instead of one high value probability

• We consider multiple layers of adaptive pursuit algorithms
Online multi-operator selection

[Drugan & Talbi, 2014]

• Optimise the usage of two or more operators simultaneously

• Motivated by the quadratic assignment problem:
  • Exploring large variable neighbourhoods is expensive
  • Iterated local search is efficient for QAPs

• Probability distribution of the mutation and the neighbourhood operators
  \[ P_N^{(t)} = \{ P_{11}^{(t)}, \ldots, P_{1K}^{(t)} \}, \quad P_M^{(t)} = \{ P_{21}^{(t)}, \ldots, P_{2P}^{(t)} \} \]
  \[ Q_N^{(t)} = \{ Q_{11}^{(t)}, \ldots, Q_{1K}^{(t)} \}, \quad Q_M^{(t)} = \{ Q_{21}^{(t)}, \ldots, Q_{2P}^{(t)} \} \]

• Quality distribution of the mutation and the neighbourhood operators
  \[ Q_N^{(t)} = \frac{\# \text{ improv of } v_N}{\# \text{ trials of } v_N}, \quad Q_M^{(t)} = \frac{\# \text{ improv of } v_M}{\# \text{ trials of } v_M} \]

• Update reward vectors: an improvement in the cost of the candidate solution when compared with the current solution
  \[ P_{N_i}^{(t+1)} \leftarrow P_{N_i}^{(t)} + \beta \cdot (D_{rN_i} - P_{N_i}^{(t)}), \quad \forall 1 \leq i \leq P \]
  \[ P_{M_j}^{(t+1)} \leftarrow P_{M_j}^{(t)} + \beta \cdot (D_{rM_j} - P_{M_j}^{(t)}), \quad \forall 1 \leq j \leq K \]

• Update probabilities: the probability distributions are independently updated
Generic Parameter Control with RL

[Karafotias et al, 2014]

• On fly parameter tuning of EA algorithms based on
  • Diversity (phenotypic and genotypic diversity)
  • Improvement (fitness variation and improvement and stagnation counter)

• SARSA is used by the parameter control algorithm
  • States are represented in a binary decision tree
  • Each parameter is set to a certain value
  • Each pair state-action has associated a value to determine how much impact this pair has on the estimated reward

• Evolution Strategy applied on
  • population size
  • the generation gap (the ratio of offsprings)
  • mutation step size
  • tournament size for survivor selection

• CMA-ES
II.2. Monte Carlo Tree Search (MTCS)

- [Browne et al, 2012]
- A heuristic used to solve intractable problems, i.e. huge search spaces, like playing computer Go
- MCTS builds a search tree using a search policy selecting the most probable nodes to expand
- A top down approach, i.e. root to leaves, with the following steps
  - Selection: chooses the most promising children
  - Expansion: creates new nodes using a tree policy
  - Simulation: plays at random from the current node to the end of the game
  - Back-propagation: updates the information on the explored path
Schemata bandits

[Drugan et al, 2014]

- Genetic algorithms that do not use the genetic operators to generate new individuals
  - A synergy between Schemata Theory and Monte Carlo tree Search
  - A hierarchical bandit where each arm is a schemata

- The schemata with the maximal estimated mean fitness is selected the most often using an UCB1 algorithm

- A schemata is a L – dimensional hypercube, 2L binary strings
An example of schemata net
A baseline schemata algorithm

• Initialise N random individuals
• Repeat
  • Select the root schemata
  • Select the most promising child of the current schemata using UCB1
    
    $\text{argmax}_i \bar{f}(H_i) + C \cdot \sqrt{\frac{2 \log(t)}{t_i}}$
  
  • Update counters
  • Back-propagate the information to update the value of schemata nodes

• Parameter free optimisation algorithm
• Schemata net is densely connected
  • expand only a part of the schemata net
  • hybrid between the two approaches
Bandit trees for real coded optimisation

- Monte Carlo tree search variants are used in optimisation of real-coded multi-dimensional functions

- The search space is partitioned in subdomains
- Each node in MCTS contains a multi-dimensional domain
- The search focuses on the most promising partitions, i.e. that contain the best solutions
- The other regions are explored with small probability

- Simultaneous optimistic optimisation (SOO) [Preux et al, 2014] is successfully applied on many dimensional test problems from the CEC’2014 competition on single objective real-parameter numerical optimisation.
- Hierarchical CMA-ES solver [Drugan, 2015c] uses CMA-ES solvers in each node of MCTS
II.3. Hyper-heuristics

[Burke et al, 2013]

- Ensemble of heuristics that automatically adapt a set of heuristics that optimize a specific task
  - Heuristics can be predefined or generated
  - Optimization is on the space of heuristics rather than directly in the optimization space
  - Automatically selection of heuristics’ parameters

- Uses RL to learn the optimal sequence of heuristics
- The set of heuristics greatly depends on the problem that will be applied on.
- The hyper heuristics should be independent of the type of heuristic it is applied on.
Concluding remarks on reinforcement learning for evolutionary computation

• Most EC algorithms use model free RL or multi-armed bandits techniques for parameter tuning
• Schemata theorem as initially used in association with multi-armed bandits by [Holland, 1975]
• Variants of Monte Carlo Tree Search are used to optimise multi-dimensional real world problems
• Hyper-heuristics use reinforcement learning to select the best heuristic for a given task [Ozcan et al, 2010]
• Pareto Local search is used in combination with RL for optimising in multi-objective search spaces [Inja et al, 2014]
Conclusions

• Hybrid algorithms between reinforcement learning and evolutionary computation
  • perform better than many standard settings of both algorithms
  • can represent realistic models of problems in, for example, engineering and management
    • incomplete observations
    • large stochastic and changing environments
• new methodological and theoretical challenges
  • in reinforcement learning, the convergence proofs need to take in account the multiple dimensions and the possible interactions between them
  • in evolutionary computation, some performance metrics to measure the adaptability of the algorithm needs to be considered
• potential to develop new algorithms for automatic parameter tuning
References


• [Drugan, 2015a] Madalina M. Drugan: Linear Scalarization for Pareto Front Identification in Stochastic Environments. EMO (2) 2015: 156-171


