The University Of Sheffield.
7th International Conference on Evolutionary Multi-Criterion Optimization

Hypervolume-based Multi-Objective Reinforcement Learning

CoMo
Computational Modeling Lab

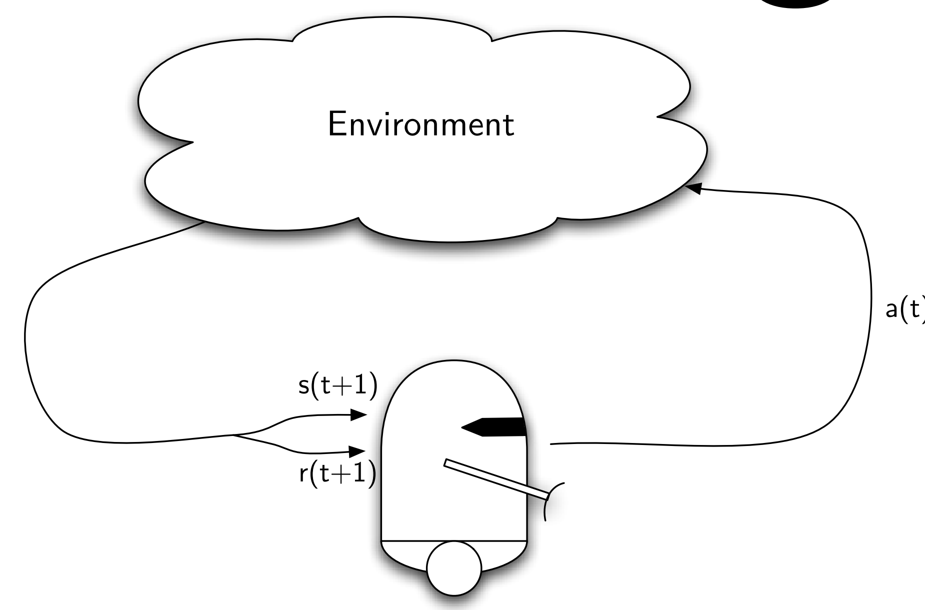
Kristof Van Moffaert
Madalina M. Drugan
Ann Nowé

Vrije Universiteit Brussel

Overview

- Single-objective reinforcement learning (RL)
- Multi-objective RL
  - State of the art
- Hypervolume-based RL
- Experiments
- Conclusions

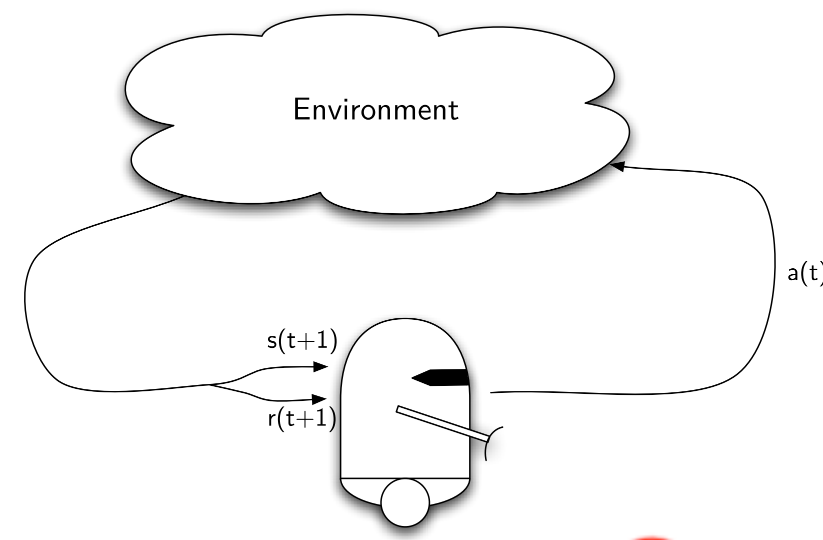
Reinforcement Learning

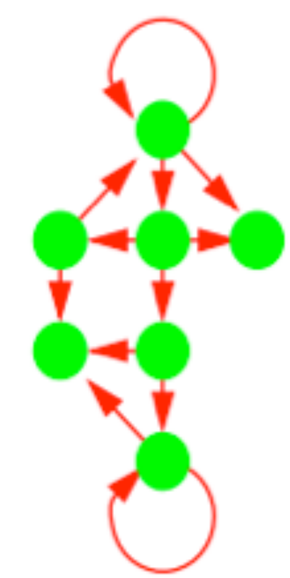


- Origin in psychology
- Learning from interaction
- Senses and acts upon its environment
- Chosen action influences the state of the environment, which determines the reward

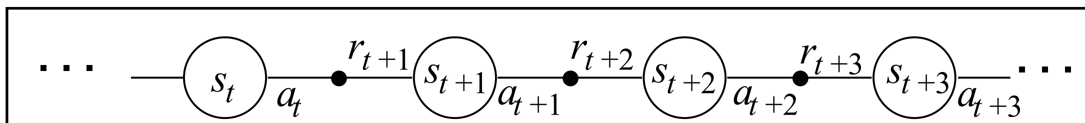
Reinforcement Learning

- Environment?
  - Markov Decision Process (MDP) contains:
    1. A set of possible states S
    2. A set of possible actions A
    3. A real-valued reward function R(s,a)
    4. A transition function T : S x A ⟶ Prob(S)





- Goal?
  - Maximize long-term reward (**R**)



$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1},$$

  - Learn policy
    - Determine (optimal) action to take in each state

Reinforcement Learning

- How?
  - *Q*-values store estimated quality of state-action pair, i.e. *Q*(s,a)
  - Update rule adapts *Q*-values into the direction of the discounted future reward

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha_t(s_t, a_t)}_{\text{learning rate}} \times \left[\overbrace{\underbrace{R_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \underbrace{\max_{a_{t+1}} Q(s_{t+1}, a_{t+1})}_{\text{max future value}}}^{\text{learned value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \right]$$

Single-objective Q-learning

Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
    Initialize s
    Repeat (for each step of episode):
        Choose a from s using policy derived from Q $\boxed{\text{(e.g., } \varepsilon\text{-greedy)}}$
        Take action a, observe r, s'
        $\boxed{Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]}$
        $s \leftarrow s'$;
    until s is terminal

Algorithm 2 ϵ-greedy action selection, $\boxed{\epsilon\text{-greedy}()}$

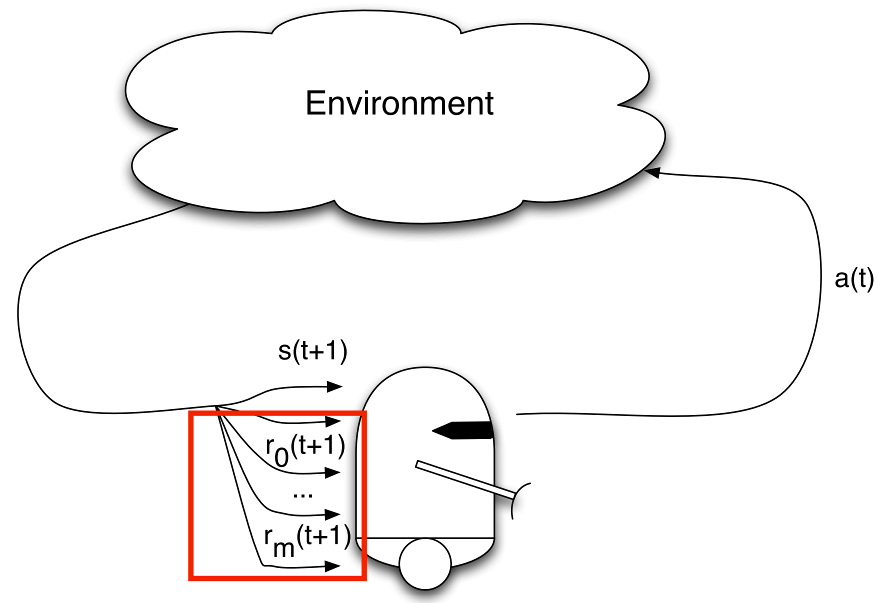
1: $r \leftarrow rnd$
2: **if** $r > \epsilon$ **then return** $\text{argmax}_a Q(s,a)$
3: **else return** $\text{random}_a Q(s,a)$
4: **end if**

take current best

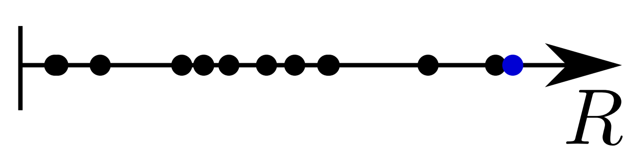
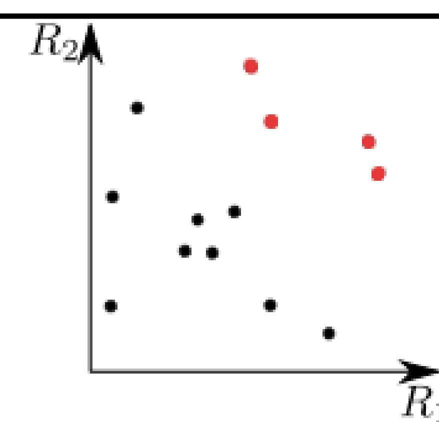
take random

Multiple objectives

- Multi-objective reinforcement learning (MORL)
  - MOMDP
  - Vector of rewards
  - Vector of *Q*-values



- Goal:

Single-objective	Multi-objective
	

State of the art MORL

- Scalarization approaches
  1. Linear scalarization MORL
     - Weighted-sum [Vamplew, 2011]
  2. Non-linear scalarization MORL
     - Chebyshev function [Van Moffaert, 2013]

Problems are similar to problems in MO

➡ Defining weights a-priori

➡ Performance heavily depends on weights used

➡ Not all solutions in Pareto front discovered

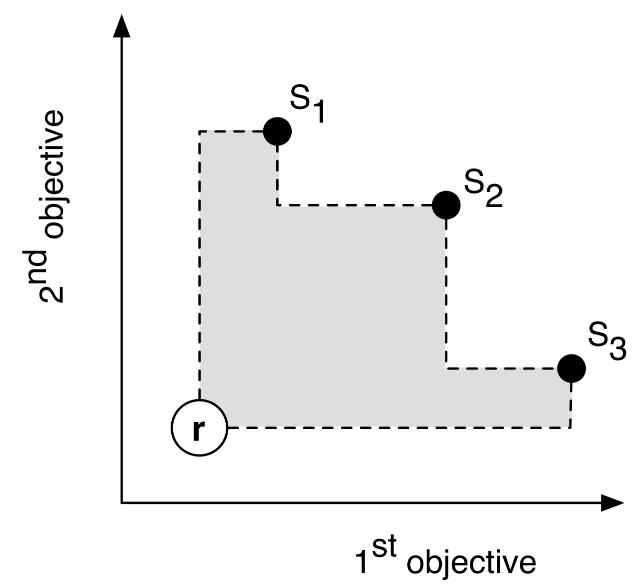
Alternative solution?

Indicator-based search!

Hypervolume unary indicator

- A unary quality indicator *I* assigns a real number to a Pareto set approx.

  $$I : \Psi \to \mathbb{R}$$



- Measures the hypervolume between *r* and s_1, s_2 *and* s_3
- Used in EMO algorithms:
  - MO-CMA-ES, HypE, SMS-EMOA, ...

Hypervolume-based MORL

Algorithm 4 Hypervolume-based Q-learning algorithm

Initialize $\boxed{Q(s,a,o)}$ arbitrarily
for each episode T **do**
  Initialize s, $\boxed{l = \{\}}$ list of previously visited Q-vectors
  repeat
    Choose a from s using policy derived from Q (e.g. ϵ-greedy HBAS(s, l))
    Take action a and observe state $s' \in S$, reward vector $\vec{r} \in \vec{\mathbb{R}}$
    $\vec{o} \leftarrow \{Q(s,a,o_1), \ldots, Q(s,a,o_m)\}$
    Add $\vec{o}$ to l ▷ Add Q-values of selected action a to l
    $max_{a'} \leftarrow$greedy HBAS(s', l) ▷ Get greedy action in s' based on new l
    for each objective o **do** ▷ Update Q-values for each objective
      $Q(s,a,o) \leftarrow Q(s,a,o) + \alpha[\vec{r}(s,a,o) + \gamma Q(s', \max_{a'}, o) - Q(s,a,o)]$
    end for
    $s \leftarrow s'$ ▷ Proceed to next state
  until s is terminal
end for

Hypervolume-based MORL

Perform action selection based on current state and l

Algorithm 4 Hypervolume-based Q-learning algorithm

Initialize $Q(s, a, o)$ arbitrarily
for each episode T **do**
  Initialize s, $l = \{\}$
  repeat
    Choose a from s using policy derived from Q (e.g. ϵ-greedy HBAS(s, l))
    Take action a and observe state $s' \in S$, reward vector $\vec{r} \in \mathbb{R}$
    $\vec{o} \leftarrow \{Q(s, a, o_1), \ldots, Q(s, a, o_m)\}$
    Add $\vec{o}$ to l ▷ Add Q-values of selected action a to l
    $max_{a'} \leftarrow$greedy HBAS(s', l) ▷ Get greedy action in s' based on new l

    for each objective o **do** ▷ Update Q-values for each objective
      $Q(s, a, o) \leftarrow Q(s, a, o) + \alpha[\vec{r}(s, a, o) + \gamma Q(s', \max_{a'}, o) - Q(s, a, o)]$
    end for

    $s \leftarrow s'$ ▷ Proceed to next state
  until s is terminal
end for

Algorithm 3 Greedy Hypervolume-based Action Selection, HBAS(s, l)

$volumes \leftarrow \{\}$ ▷ The list collects hv contributions for each action
for each action $a_i \in A$ of state s **do**
  $\vec{o} \leftarrow \{Q(s, a_i, o_1), \ldots, Q(s, a_i, o_m)\}$
  $hv \leftarrow calculate\_hv(l + \vec{o})$ ▷ Compute hv contribution of a_i to l
  Append hv to $volumes$
end for
return $\text{argmax}_a\, volumes$ ▷ Retrieve the action with the maximal contribution

Hypervolume-based MORL

Algorithm 4 Hypervolume-based Q-learning algorithm

Perform action selection based on current state and *l*

Initialize $Q(s, a, o)$ arbitrarily
for each episode T **do**
  Initialize s, $l = \{\}$
  repeat
    Choose a from s using policy derived from Q (e.g. ϵ-greedy HBAS(s, l))
    Take action a and observe state $s' \in S$, reward vector $\vec{r} \in \mathbb{R}$
    $\vec{o} \leftarrow \{Q(s, a, o_1), \ldots, Q(s, a, o_m)\}$
    Add $\vec{o}$ to l ▷ Add Q-values of selected action a to l
    $max_{a'} \leftarrow$greedy HBAS(s', l) ▷ Get greedy action in s' based on new l

    for each objective o **do** ▷ Update Q-values for each objective
      $Q(s, a, o) \leftarrow Q(s, a, o) + \alpha[\vec{r}(s, a, o) + \gamma Q(s', \max_{a'}, o) - Q(s, a, o)]$
    end for

    $s \leftarrow s'$ ▷ Proceed to next state
  until s is terminal
end for

Algorithm 3 Greedy Hypervolume-based Action Selection, HBAS(s, l)

$volumes \leftarrow \{\}$ ▷ The list collects hv contributions for each action
for each action $a_i \in A$ of state s **do**
  $\vec{o} \leftarrow \{Q(s, a_i, o_1), \ldots, Q(s, a_i, o_m)\}$
  $hv \leftarrow calculate\_hv(l + \vec{o})$
  Append hv to $volumes$
end for
return $\text{argmax}_a\ volumes$

Return action *a* with maximal contribution in HV taking into account the contents of *l*

Hypervolume-based MORL

Algorithm 4 Hypervolume-based Q-learning algorithm

Initialize $Q(s, a, o)$ arbitrarily
for each episode T **do**
  Initialize s, $l = \{\}$
  repeat
    Choose a from s using policy derived from Q (e.g. ϵ-greedy HBAS(s, l))
    Take action a and observe state $s' \in S$, reward vector $\vec{r} \in \vec{\mathbb{R}}$
    $\vec{o} \leftarrow \{Q(s, a, o_1), \ldots, Q(s, a, o_m)\}$
    Add $\vec{o}$ to l ▷ Add Q-values of selected action a to l
    $max_{a'} \leftarrow$greedy HBAS(s', l) ▷ Get greedy action in s' based on new l
    for each objective o **do** ▷ Update Q-values for each objective
      $Q(s, a, o) \leftarrow Q(s, a, o) + \alpha[\vec{r}(s, a, o) + \gamma Q(s', \max_{a'}, o) - Q(s, a, o)]$
    end for
    $s \leftarrow s'$ ▷ Proceed to next state
  until s is terminal
end for

Add current Q-vector to l

Update Q-value for each objective individually

Algorithm 3 Greedy Hypervolume-based Action Selection, HBAS(s, l)

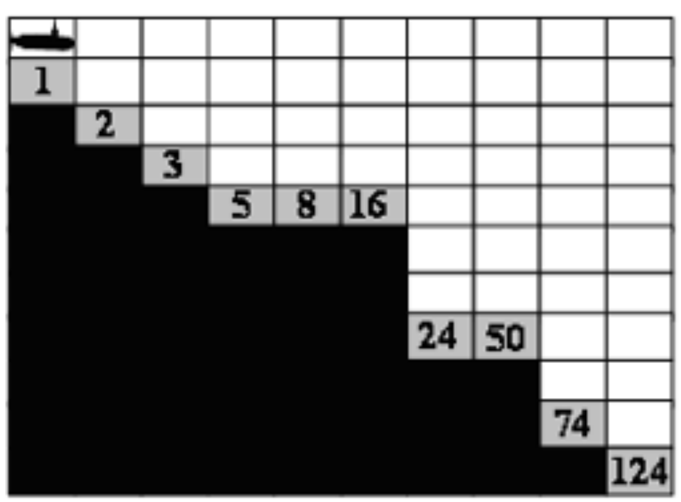
$volumes \leftarrow \{\}$ ▷ The list collects hv contributions for each action
for each action $a_i \in A$ of state s **do**
  $\vec{o} \leftarrow \{Q(s, a_i, o_1), \ldots, Q(s, a_i, o_m)\}$
  $hv \leftarrow calculate\_hv(l + \vec{o})$ ▷ Compute hv contribution of a_i to l
  Append hv to $volumes$
end for
return $\text{argmax}_a\, volumes$ ▷ Retrieve the action with the maximal contribution

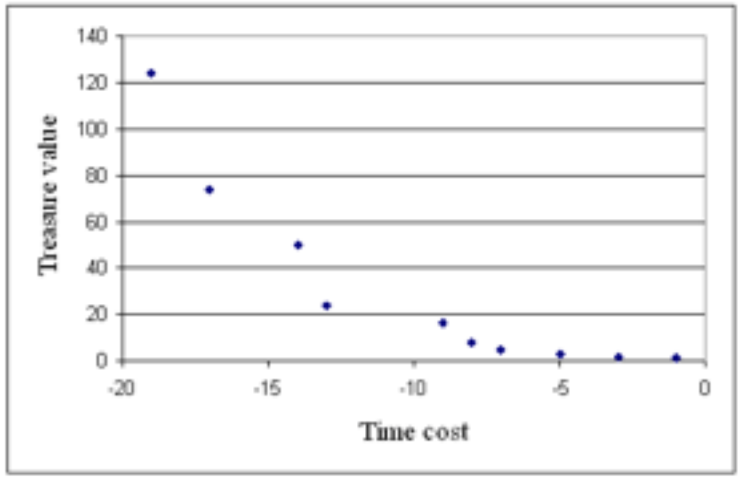
Benchmark 1

- Benchmark instances [Vamplew, 2011]

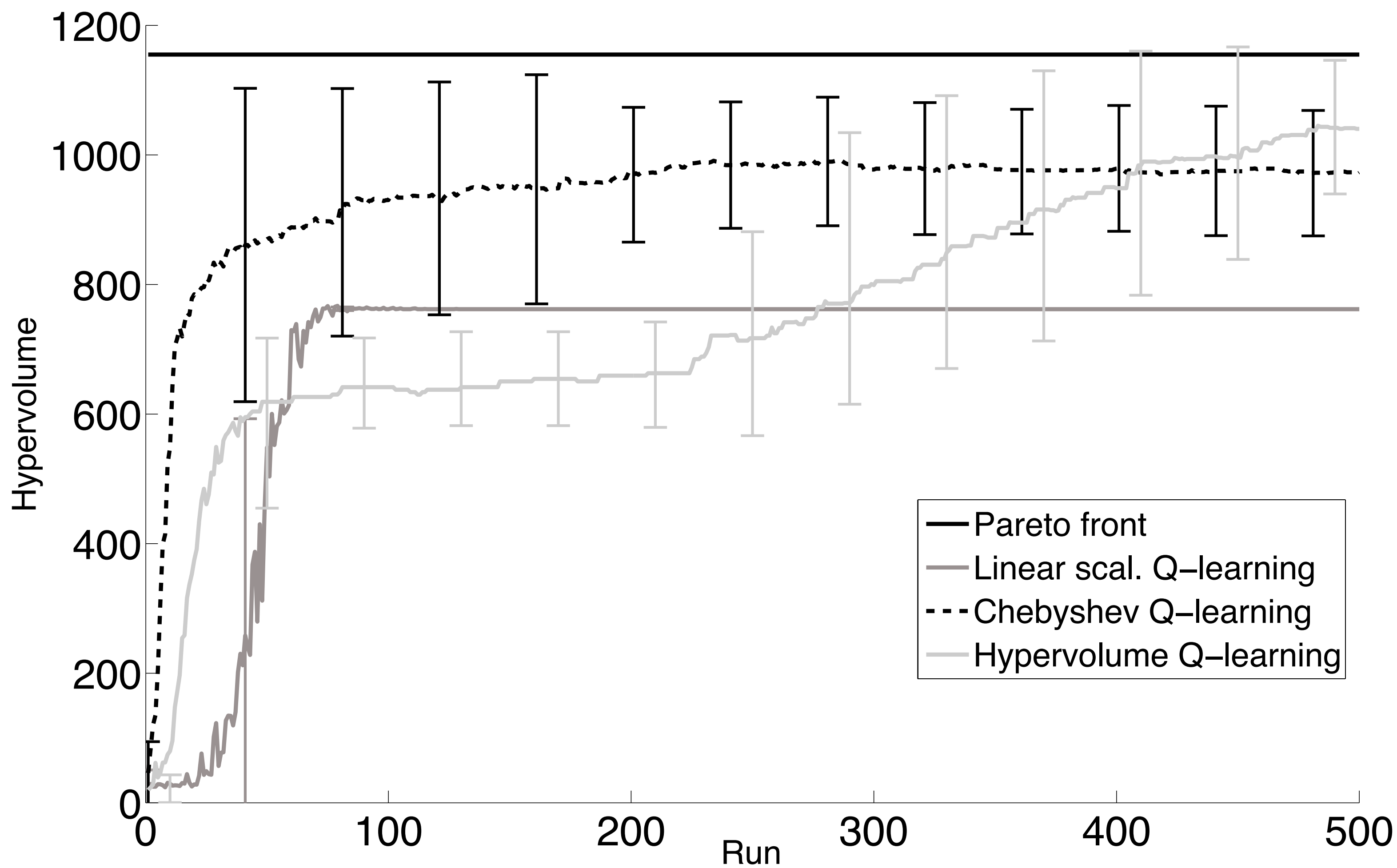
  Deep Sea Treasure world

  - Minimize time and maximize treasure value
  - Transformed into full maximization problem
    - Time objective x -1
  - 10 Pareto optimal policies
  - Represent non-convex Pareto front



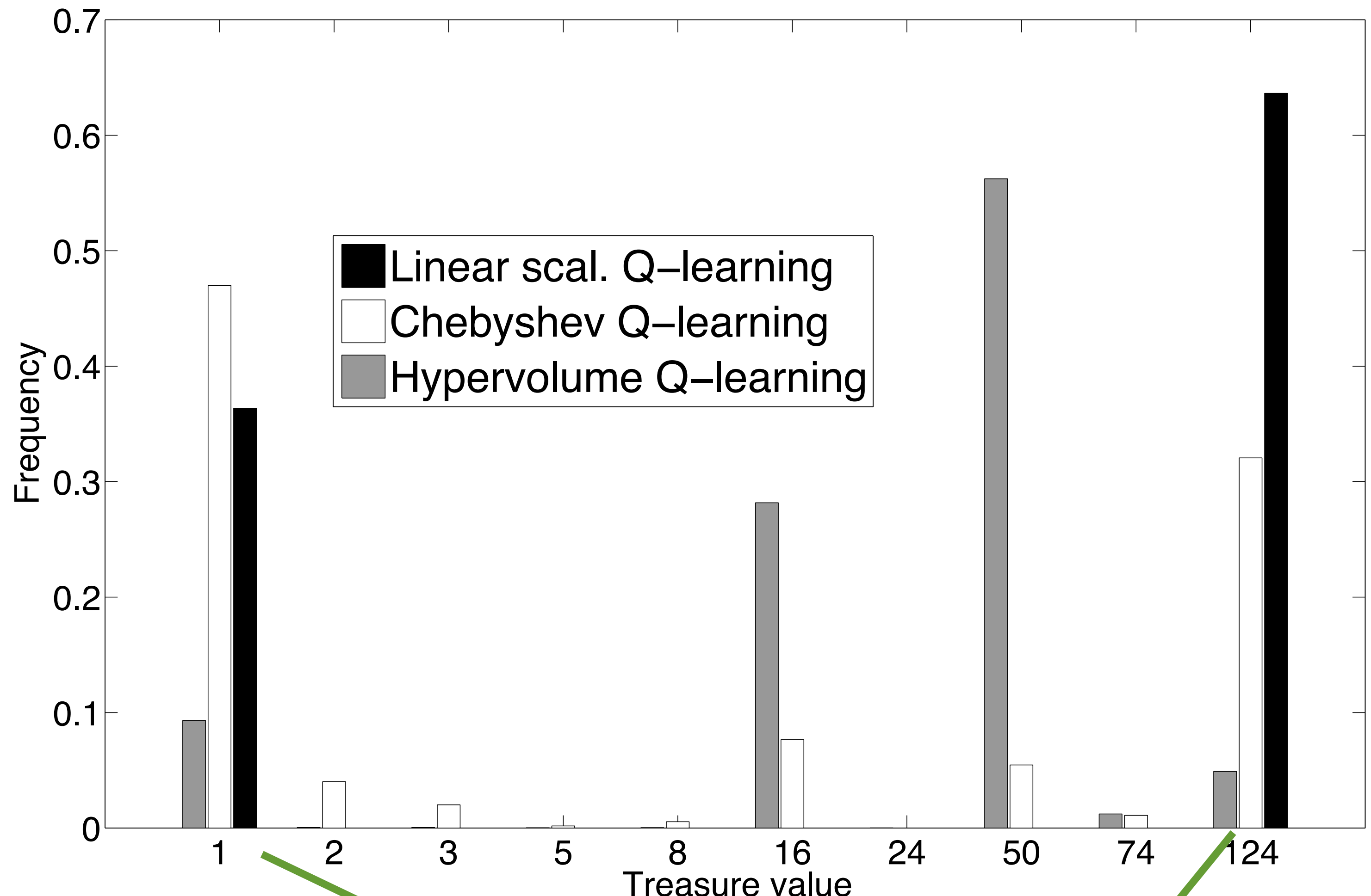


Learning curve



(a) Deep Sea Treasure world

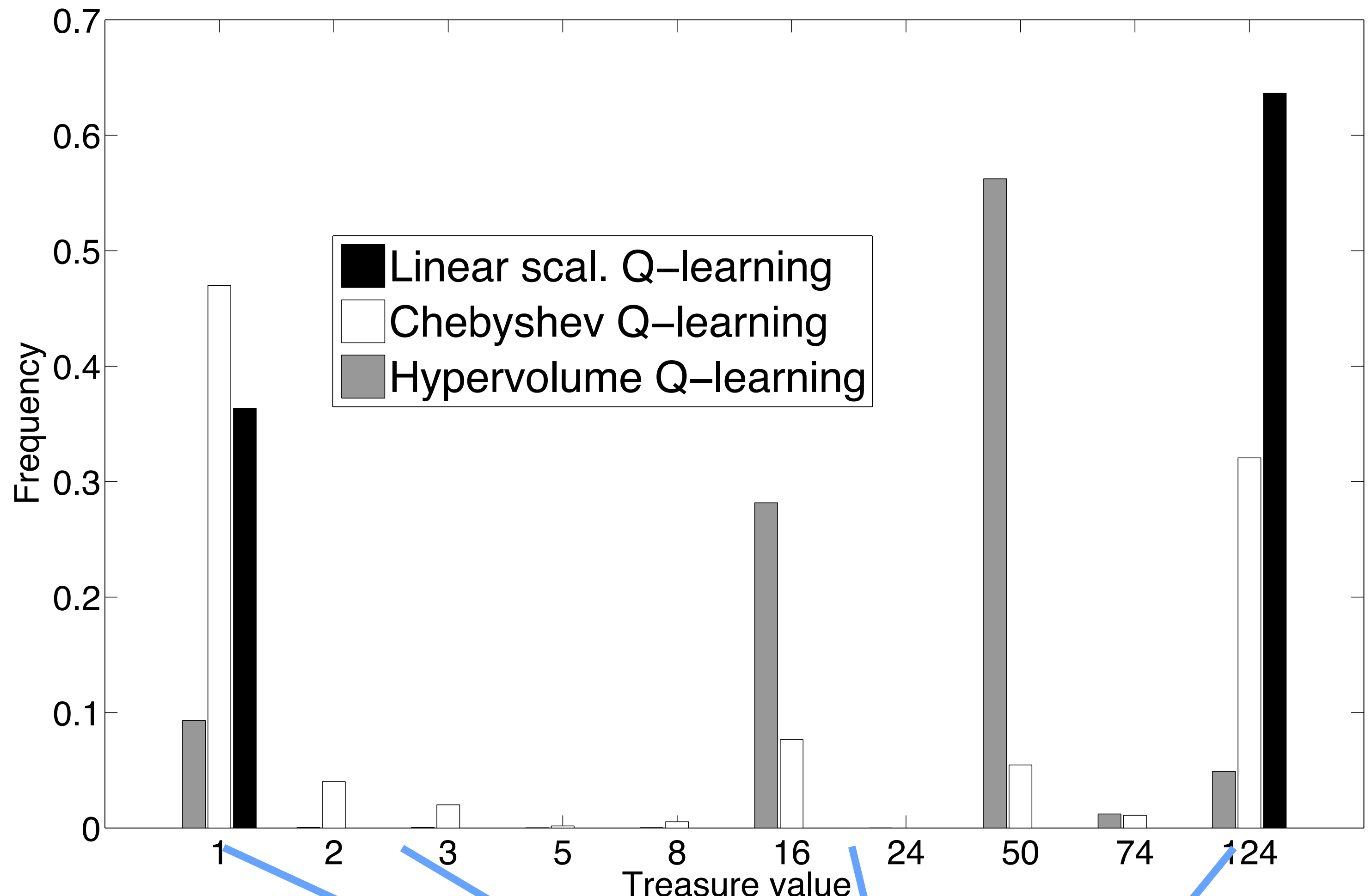
Pareto optimal policies learned



(c) Frequency of goals in the Deep Sea world

As expected, the linear scalarization learner was ineffective in the non-convex environment

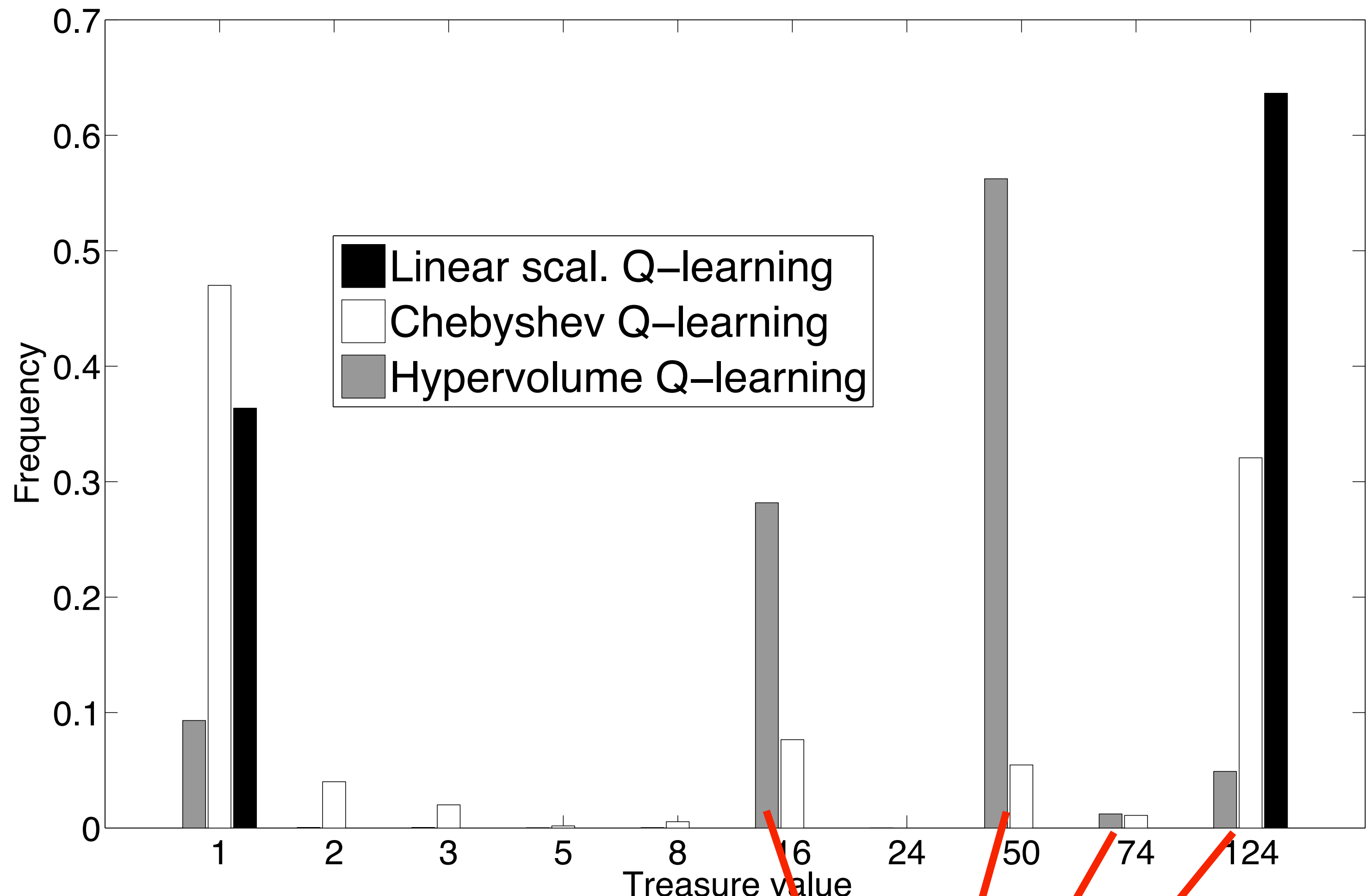
Pareto optimal policies learned



(c) Frequency of goals in the Deep Sea world

The Chebyshev learner obtained the best spread,
but not all the time (cfr. learning graph)

Pareto optimal policies learned



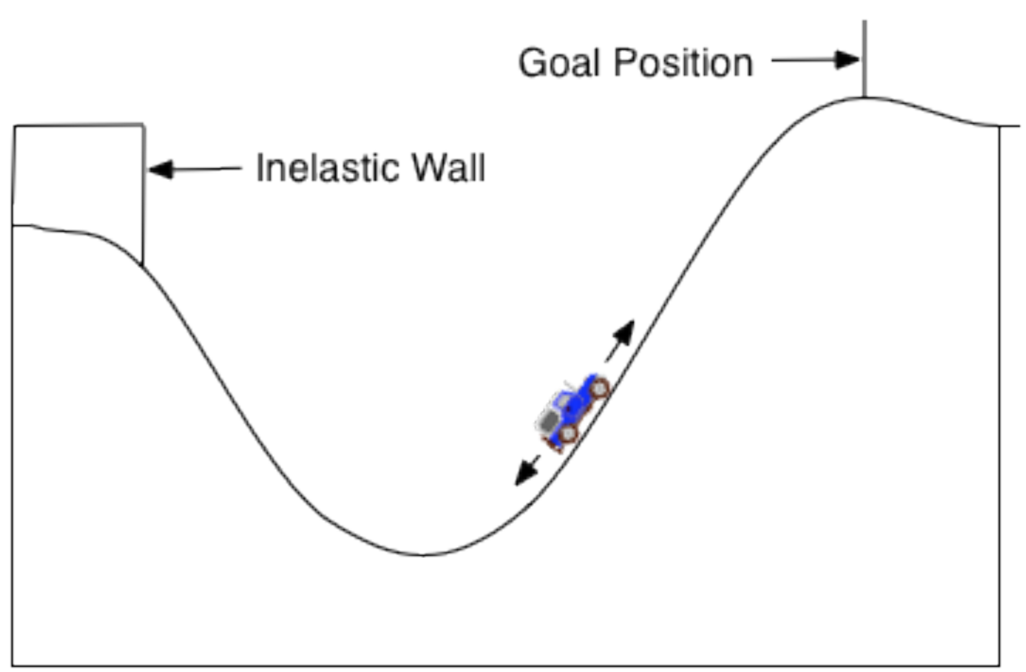
(c) Frequency of goals in the Deep Sea world

HB-MORL focusses on policies that maximize
the hypervolume, given a particular reference point

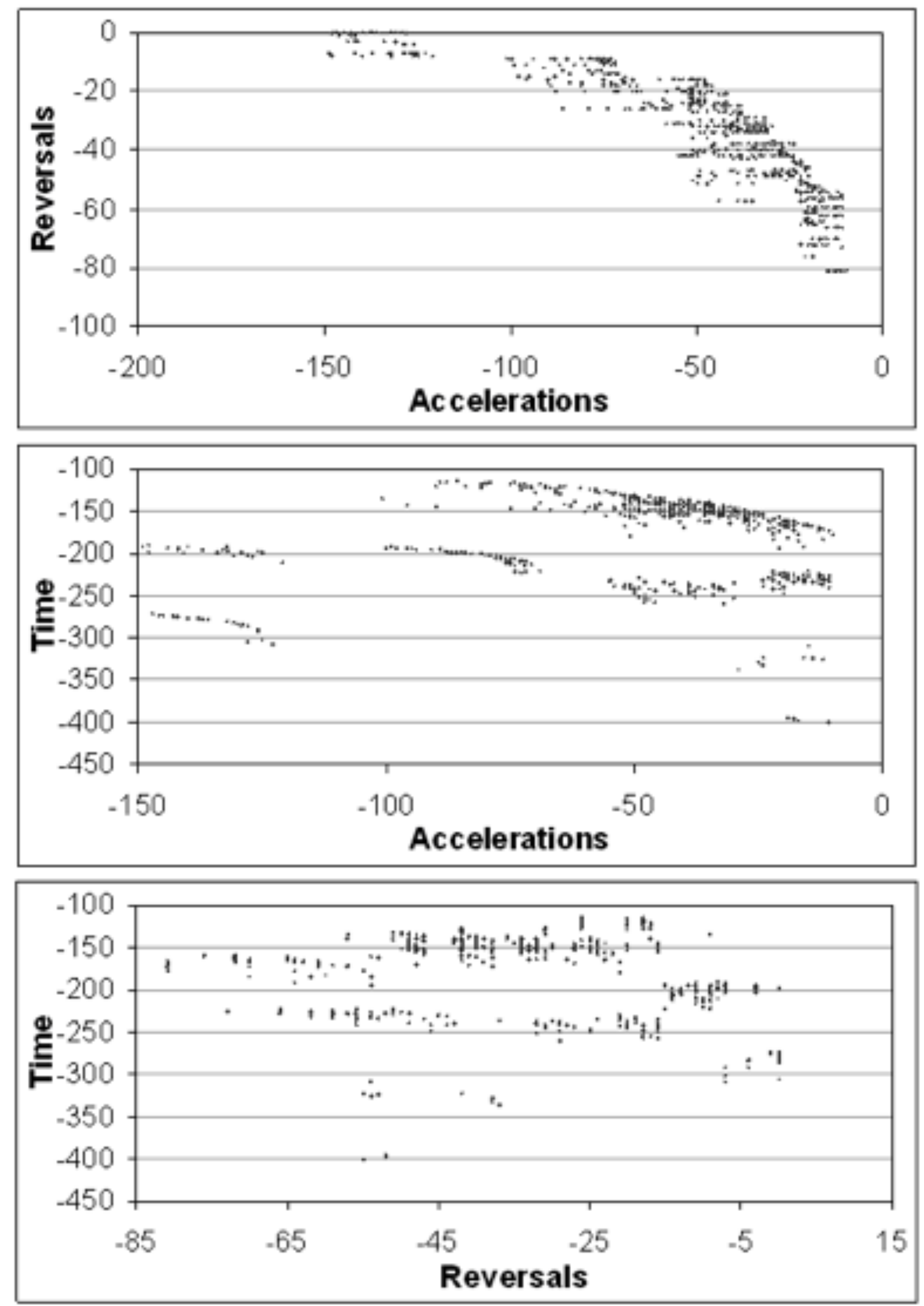
Benchmark 2

MO Mountain Car world

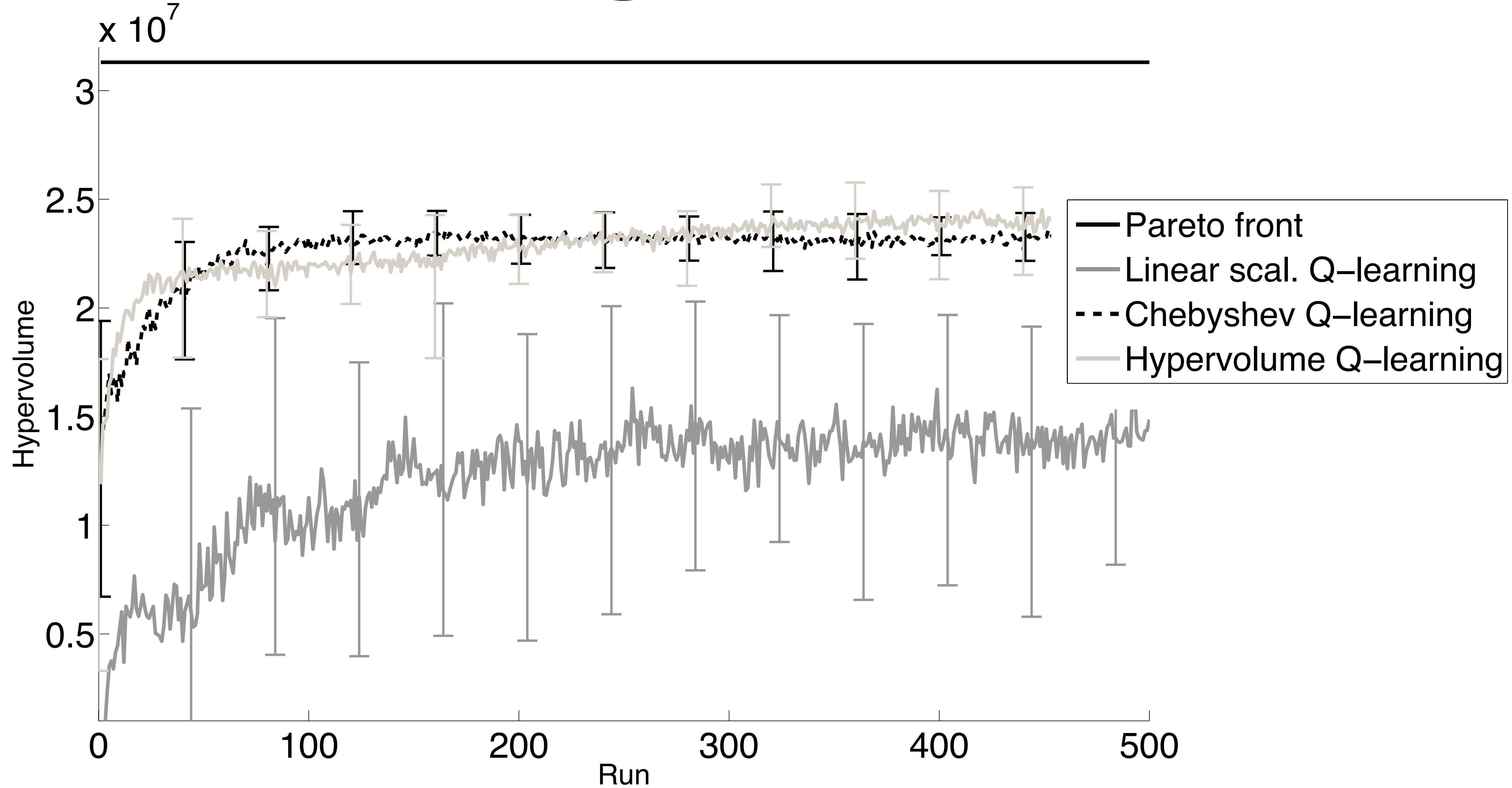
- 3-objective
- minimize time, number of reversal and acceleration actions
- Transformed into maximization problem
- 470 elements in Pareto front



Pareto front



Learning curve



(b) MO Mountain Car world

Quality indicator comparison

		Linear	Chebyshev	HB-MORL
Inverted Generational distance	**DS**	0.128	**0.0342**	0.0371
	MC	0.012	0.010	**0.005**
Generalized spread	**DS**	$\mathbf{3.14e^{-16}}$	0.743	0.226
	MC	**0.683**	0.808	0.701
Generational distance	**DS**	0	0	0
	MC	0.0427	0.013824	**0.013817**
Hypervolume	**DS**	762	959.26	**1040.2**
	MC	15727946	23028392	**23984880**
Cardinality	**DS**	2	**8**	5
	MC	15	**38**	37

Conclusions

- We have combined EMO principles with RL to design a hybrid MORL algorithm
- HB-MORL uses the hypervolume measure to guide the action selection
- Results
  - Linear scalarization learner is not generally applicable
  - Chebyshev learns more spread results, but not robust all the time
  - Scalarization methods and their performance depend on weight tuples used
  - ➡ HB-MORL focuses on policies that maximize HV and finds them nearly always

Thank you

Kristof Van Moffaert
Madalina M. Drugan
Ann Nowé

CoMo Computational Modeling Lab

Vrije Universiteit Brussel