Chapter 16
Particle Swarm Optimization for Model Predictive Control in Reinforcement Learning Environments

Daniel Hein
Technische Universität München, Germany

Alexander Hentschel
AxiomZen, Canada

Thomas A. Runkler
Siemens AG, Germany

Steffen Udluft
Siemens AG, Germany

ABSTRACT

This chapter introduces a model-based reinforcement learning (RL) approach for continuous state and action spaces. While most RL methods try to find closed-form policies, the approach taken here employs numerical online optimization of control action sequences following the strategy of nonlinear model predictive control. First, a general method for reformulating RL problems as optimization tasks is provided. Subsequently, particle swarm optimization (PSO) is applied to search for optimal solutions. This PSO policy (PSO-P) is effective for high dimensional state spaces and does not require a priori assumptions about adequate policy representations. Furthermore, by translating RL problems into optimization tasks, the rich collection of real-world-inspired RL benchmarks is made available for benchmarking numerical optimization techniques. The effectiveness of PSO-P is demonstrated on two standard benchmarks mountain car and cart-pole swing-up and a new industry-inspired benchmark, the so-called industrial benchmark.

DOI: 10.4018/978-1-5225-5134-8.ch016

INTRODUCTION

This chapter focuses on a general reinforcement learning (RL) setting with continuous state and action spaces. In this domain, the policy performance often strongly depends on the algorithms for policy generation and the chosen policy representation (Sutton & Barto, 1998). In the authors' experience, tuning the policy learning process is generally challenging for industrial RL problems. Specifically, it is hard to assess whether a trained policy has unsatisfactory performance due to inadequate training data, unsuitable policy representation, or an unfitting training algorithm. Determining the best problem-specific RL approach often requires time-intensive trials with various policy configurations and training algorithms. In contrast, it is often significantly easier to train a well-performing system model from observational data, compared to directly learning a policy and assessing its performance.

The main purpose of the present contribution is to provide a heuristic for solving RL problems which employs numerical online optimization of control action sequences. As an initial step, a neural system model is trained from observational data with standard methods. However, the presented method also works with any other model type, e.g., Gaussian process or first principal models. The resulting problem of finding optimal control action sequences based on model predictions is solved with particle swarm optimization (PSO), because PSO is an established algorithm for non-convex optimization. Specifically, the presented heuristic iterates over the following steps. (1) PSO is employed to search for an action sequence that maximizes the expected return when applied to the current system state by simulating its effects using the system model. (2) The first action of the sequence with the highest expected return is applied to the real-world system. (3) The system transitions to the subsequent state and the optimization process are repeated based on the new state (go to step 1).

As this approach can generate control actions for any system state, it formally constitutes an RL policy. This PSO policy (PSO-P) deviates fundamentally from common RL approaches. Most methods for solving RL problems try to learn a closed-form policy (Sutton & Barto, 1998). The most significant advantages of PSO-P are the following. (1) Closed-form policy learners generally select a policy from a user-parameterized (potentially infinite) set of candidate policies. For example, when learning an RL policy based on tile coding (Sutton, 1996), the user must specify partitions of the state space. The partition's characteristics directly influence how well the resulting policy can differentiate the effect of different actions. For complex RL problems, policy performances usually vary drastically depending on the chosen partitions. In contrast, PSO-P does not require a priori assumptions about problem-specific policy representations, because it directly optimizes action sequences. (2) Closed-form RL policies operate on the state space and are generally affected by the *curse of dimensionality* (Bellman, Adaptive Control Processes: A Guided Tour, 1962). Simply put, the number of data points required for a representative coverage of the state space grows exponentially with the state space's dimensionality. Common RL methods, such as tile coding, quickly become computationally intractable with increasing dimensionality. Moreover, for industrial RL problems it is often very expensive to obtain adequate training data prohibiting data-intensive RL methods. In comparison, PSO-P is not affected by the state space dimensionality because it operates in the space of action sequences.

From a strictly mathematical standpoint, PSO-P follows a known strategy from nonlinear model predictive control (MPC): employing online numerical optimization in search for the best action sequences. While MPC and RL target almost the same class of control optimization problems with different methods, the mathematical formalisms in both communities are drastically different. Particularly, the authors find that the presented approach is rarely considered in the RL community. The main contribution of

this chapter is to provide a hands-on guide for employing online optimization of action sequences in the mathematical RL framework and demonstrate its effectiveness for solving RL problems. On the one hand, PSO-P generally requires significantly more computation time to determine an action for a given system state compared to closed-form RL policies. On the other hand, the authors found PSO-P particularly useful for determining the optimization potential of various industrial control optimization problems and for benchmarking other RL methods.

In the Sections 'Formulation of Reinforcement Learning as Optimization Problem' and 'The PSO-Policy Framework', the methodology is developed, starting by formulating RL as a non-convex optimization problem and subsequently employing PSO as a solver. The results of the conducted benchmark experiments are presented in the Section 'Experiments, Results, and Analysis'. Future research opportunities are proposed in Section 'Future Research Directions' followed by the discussion of the experimental results and current limitations of PSO-P in the final Section 'Conclusion'.

BACKGROUND

RL is an area of machine learning inspired by biological learning. Formally, a software agent interacts with a system in discrete time steps. At each time step, the agent observes the system's state s and applies an action a. Depending on s and a, the system transitions into a new state and the agent receives a real-valued reward $r \in \mathbb{R}$. The agent's goal is to maximize its expected cumulative reward, called return $\mathcal{R}$. The solution to an RL problem is a policy, i.e., a map that generates an action for any given state. (Sutton & Barto, Reinforcement learning: An introduction, 1998)

To bypass the challenges of learning a closed-form RL policy, the authors adapted an approach from MPC (Rawlings & Mayne, 2009; Camacho & Alba, 2007), which employs only a system model. The general idea behind MPC is deceptively simple: given a reliable system model, one can predict the future evolution of the system and determine a control strategy that results in the desired system behavior. However, complex industry systems and plants commonly exhibit nonlinear system dynamics (Schaefer, Schneegass, Sterzing, & Udluft, 2007; Piche, et al., 2000). In such cases, closed-form solutions to the optimal control problem often do not exist or are computationally hard to find (Findeisen & Allgoewer, 2002; Magni & Scattolini, 2004). Therefore, MPC tasks for nonlinear systems are typically solved by numerical online optimization of sequences of control actions (Gruene & Pannek, 2011). Unfortunately, the resulting optimization problems are generally non-convex (Johansen, 2011) and no universal method for tackling nonlinear MPC tasks has yet been found (Findeisen, Allgoewer, & Biegler, 2007; Rawlings, Tutorial overview of model predictive control, 2000). Moreover, one might argue, based on theoretical considerations, that such a universal optimization algorithm does not exist (Wolpert & Macready, 1997).

PSO and evolutionary algorithms are established heuristics for solving non-convex optimization problems. Both have been applied in the context of RL, however, almost exclusively to optimize policies directly. Moriarty, Schultz, & Grefenstette (1999) give a comprehensive overview of the various approaches, using evolutionary algorithms to tackle RL problems. Methods, which apply PSO to generate policies for specific system control problems, were studied in (Feng, 2005), (Solihin & Akmeliawati, 2010), and (Montazeri-Gh, Jafari, & Ilkhani, 2012).

Recently, several combinations of swarm optimization and MPC have been proposed in the literature. In (Van Heerden, Fujimoto, & Kawamura, 2014) the nonlinear and underactuated Acrobot problem was solved by adapting PSO to run in parallel on graphics hardware, yielding a real-time MPC controller. Ou,

Kang, Kim, & Julius (2015) investigated the use of a single control signal and a PSO-MPC algorithm for controlling the movement of multiple magnetized cells while avoiding obstacles. In (Xu, Chen, Gong, & Mei, 2016) the authors tackled the problem of real-time application of nonlinear MPC by implementing it on a field-programmable gate array that employs a PSO algorithm. By using a parallelized PSO implementation, good computational performance and satisfactory control performance were achieved. Lee & Myung (2015) significantly reduced the computational cost of collision avoidance for a class of mobile robots. By applying PSO instead of traditional optimization techniques, such as sequential quadratic programming, they achieved a significant speedup during the optimization phase. They also verified the effectiveness of the proposed RHPSO-based formation control by means of numerical simulations.

However, none of the reviewed approaches generalizes to RL, as expert-designed objective functions, that already contain detailed knowledge about the optimal solution to the respective control problem, are used. In contrast, in the present chapter, the general RL problem is reformulated as an optimization problem. This representation allows searching for optimal action sequences on a system model, even if no expert knowledge about the underlying problem dynamics is available.

FORMULATION OF REINFORCEMENT LEARNING AS OPTIMIZATION PROBLEM

In this chapter, the problem of optimizing the behavior of a physical system, that is observed in discrete, equally spaced time steps $t \in \mathbb{Z}$, is considered. The current time is denoted as $t = 0$. Hence, $t = 1$ and $t = -1$ represent one step into the future and one step into the past, respectively. At each time step t, the system is described by its Markovian state $s_t \in \mathcal{S}$, from the state space $\mathcal{S}$. The agent's action a_t is represented by a vector of I different control parameters, i.e., $a_t \in \mathcal{A} \subset \mathbb{R}^I$. Based on the system's state and the applied action, the system transitions into the state s_{t+1} and the agent receives the reward r_t.

In the following, deterministic systems, which are described by a state transition function $m : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S} \times \mathbb{R}$ with $m(s_t, a_t) = (s_{t+1}, r_t)$, are considered.

The goal is to find an action sequence $\mathbf{x} = (a_t, a_{t+1}, \ldots, a_{t+T-1})$ that maximizes the expected return $\mathcal{R}$. The search space is bounded by $\mathbf{x}_{\min}$ and $\mathbf{x}_{\max}$ which are defined as:

$$\mathbf{x}_{\min_j} = a_{\min_{(j \mod I)}} \forall j = 0, \ldots, I \cdot T - 1 \tag{1}$$

and

$$\mathbf{x}_{\max_j} = a_{\max_{(j \mod I)}} \forall j = 0, \ldots, I \cdot T - 1, \tag{2}$$

where $a_{\min}$ ($a_{\max}$) are the lower (upper) bounds of the control parameters.

To incorporate the increasing uncertainty when planning actions further and further into the future, the simulated reward r_{t+k} for k time steps into the future is weighted by γ^k, where $\gamma \in [0,1]$ is referred to as the discount factor.

A common strategy is to simulate the system evolution only for a finite number of $T \geq 1$ steps. The return is (Sutton & Barto, 1998)

$$\mathcal{R}(s_t, \mathbf{x}) = \sum_{k=0}^{T-1} \gamma^k r_{t+k}, \quad \text{with } (s_{t+k+1}, r_{t+k}) = m(s_{t+k}, a_{t+k}). \tag{3}$$

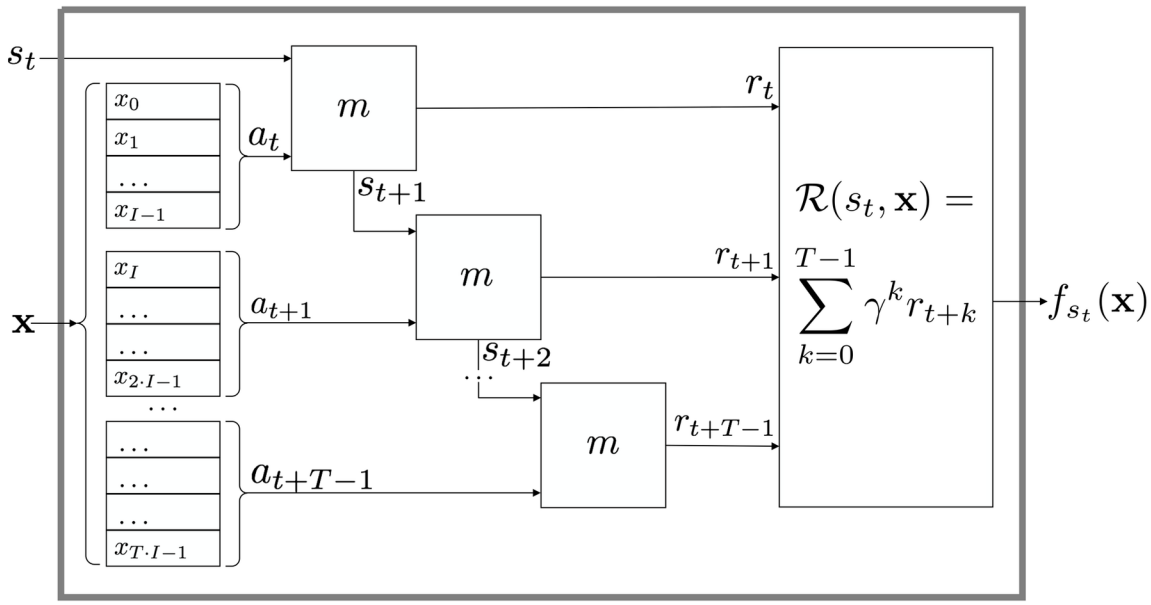
The authors chose γ such that at the end of the time horizon T, the last reward accounted for is weighted by the user-defined constant $q \in [0,1]$, which implies $\gamma = q^{1/(T-1)}$.

Solving the RL problem corresponds to finding the optimal action sequence $\hat{\mathbf{x}}$ by maximizing

$$\hat{\mathbf{x}} \in \underset{\mathbf{x} \in \mathcal{A}^T}{\operatorname{argmax}} f_{s_t}(\mathbf{x}), \tag{4}$$

with respect to the fitness function $f_{s_t} : \mathbb{R}^{I \cdot T} \to \mathbb{R}$ with $f_{s_t}(\mathbf{x}) = \mathcal{R}(s_t, \mathbf{x})$. Figure 1 illustrates the process of computing $f_{s_t}(\mathbf{x})$.

Figure 1. Model-based computation of the fitness function, i.e., return function, from the system's current state s_t and an action sequence $\mathbf{x}$. The accumulated rewards, predicted by the model, yield the fitness value f_{s_t}, which is then used to drive the optimization.



THE PSO-POLICY FRAMEWORK

The PSO algorithm is a population-based, stochastic optimization heuristic for solving non-convex optimization problems (Kennedy & Eberhart, 1995). Generally, PSO can operate on any search space that is a bounded sub-space of a finite-dimensional vector space (Engelbrecht, 2005). The PSO algorithm performs a search using a population (swarm) of individuals (particles) that are updated from iteration to iteration.

In this chapter, PSO is used to solve Eq. (4), i.e., the particles move through the search space of action sequences $\mathcal{A}^T$. Consequently, a particle's position represents a candidate action sequence $\mathbf{x} = (a_t, a_{t+1}, \ldots, a_{t+T-1})$, which is initially chosen at random.

At each iteration, particle i remembers its local best position $\mathbf{y}_i$ that it has visited so far (including its current position). Furthermore, particle i also knows the neighborhood best position

$$\hat{\mathbf{y}}_i(p+1) \in \underset{\mathbf{z} \in \{\mathbf{y}_j(p) \nexists\, j \in \mathcal{N}_i\}}{\operatorname{argmax}} f(\mathbf{z}), \tag{5}$$

found so far by any particle in its neighborhood $\mathcal{N}_i$ (including itself). The neighborhood relations between particles are determined by the swarm's population topology and are generally fixed, irrespective of the particles' positions.

In the experiments presented in Section 'Experiments, Results, and Analysis' the authors use the ring topology (Eberhart, Simpson, & Dobbins, 1996).

From iteration p to $p+1$ the particle position update rule is

$$\mathbf{x}_i(p+1) = \mathbf{x}_i(p) + \mathbf{v}_i(p+1). \tag{6}$$

The components of the velocity vector $\mathbf{v}$ are calculated as

$$v_{ij}(p+1) = w v_{ij}(p) + \underbrace{c_1 r_{1j}(p)[y_{ij}(p) - x_{ij}(p)]}_{\text{cognitive component}} + \underbrace{c_2 r_{2j}(p)[\hat{y}_{ij}(p) - x_{ij}(p)]}_{\text{social component}}, \tag{7}$$

where w is the inertia weight factor, $v_{ij}(p)$ and $x_{ij}(p)$ are the velocity and the position of particle i in dimension j, c_1 and c_2 are positive acceleration constants used to scale the contribution of the cognitive and the social components $y_{ij}(p)$ and $\hat{y}_{ij}(p)$, respectively. The factors $r_{1j}(p)$, $r_{2j}(p) \sim U(0,1)$ are random values, sampled from a uniform distribution to introduce a stochastic element to the algorithm. Shi and Eberhart (2000) proposed to set the values to $w = 0.7298$ and $c1 = c2 = 1.49618$.

Even though a sequence of T actions is optimized, only the first action is applied to the real-world system and an optimization of a new action sequence is performed for the subsequent system state s_{t+1}. This approach follows the widely applied control theory methods known as MPC, receding horizon control, or moving horizon method (Kwon, Bruckstein, & Kailath, 1983; Rawlings & Mayne, Model predictive control theory and design, 2009; Camacho & Alba, 2007). Most often the dynamic models in MPC are realized by empirical models obtained by system identification. Thereby, mathematical

models $\tilde{m}$ are learned by measured data from the real dynamical system m. Since this data is already available in batch RL problems considered herein, applying an MPC-like approach like PSO-P appears likely to succeed for such problems, too.

Despite the fact that empirical models $\tilde{m}$ are likely to be inaccurate in their predictions, i.e., $\tilde{m}\left(s_t, a_t\right) = \left(\tilde{s}_{t+1}, \tilde{r}_t\right) \neq \left(s_{t+1}, r_t\right) = m\left(s_t, a_t\right)$ in Eq. (3), the experiments presented in Section 'Experiments, Results, and Analysis' verify that very stable control results can still be achieved. The reason for this advantageous behavior lies in the fact, that applying only the first action of the optimized action trajectory to the system, and subsequently initializing PSO-P with the resulting real system state s_{t+1}, resets the agent to the underlying true environmental conditions after each time step. Subsequently, the optimization starts with the correct initialization from scratch.

Implementation details can be found in Appendix 2.

EXPERIMENTS, RESULTS, AND ANALYSIS

The authors applied the PSO-P framework to three different RL problems. Two standard problems are the mountain car (MC) (Sutton & Barto, 1998) and the cart-pole (CP) swing-up benchmark (Fantoni & Lozano, 2002), which are used to illustrate the framework's capability of solving RL problems in general. The third problem is an industry inspired benchmark, the so-called industrial benchmarks (IB) (Hein, et al., 2017b), which evaluates the framework's performance on high-dimensional and stochastic dynamics.

For each benchmark, a neural network (NN) has been trained as the system model m using standard techniques (Montavon, Orr, & Müller, 2012). NNs are well suited for data-driven black-box models as they are universal approximators (Hornik, Stinchcombe, & White, 1989). In addition, the authors found that the resulting models generalize well to new data in many real-world applications.

The authors have chosen this approach instead of working directly with benchmark simulations because in many real-world scenarios physical simulations are either unavailable or strongly idealized. However, PSO-P works with other model types as well, such as first principle or Gaussian process models (Rasmussen & Williams, 2006).

Mountain Car Benchmark

In the MC benchmark an underpowered car is driven up to the top of a hill (Figure 2). This is done by building up momentum with the help of driving in the opposite direction to gain enough potential energy.

In the present implementation, the hill landscape is equivalent to $\sin(3\rho)$. The task for the RL agent is to find a sequence of force actions $a_t, a_{t+1}, a_{t+2}, \ldots \in [-1,1]$ that drive the car up the hill, which is achieved when reaching a position $\rho > \pi/6$.

At the start of each episode the car's state is $(\rho, \dot{\rho}) = (-\pi \,/\, 6, 0.0)$. The agent receives a reward of $r(\rho) = \sin(3\rho) - 1$ after every action-state update. When the car reaches the goal position, the car's position is fixed, and the agent receives the maximum reward in every following time step, regardless of the applied actions.

Using the parameters given in Table 1, PSO-P is able to solve this RL problem. Details of the algorithm and the determination of suitable algorithmic parameters are summarized in Appendix 1.

Figure 2. Mountain car task. The system can be described completely by its Markov state variables ρ and $\dot{\rho}$, which represent the car's position and velocity, respectively (Sutton & Barto, 1998)

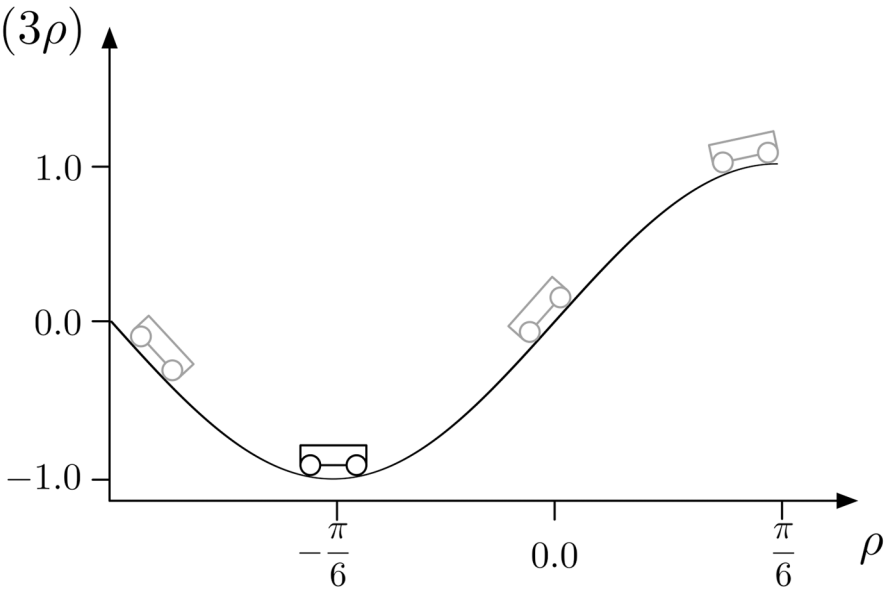


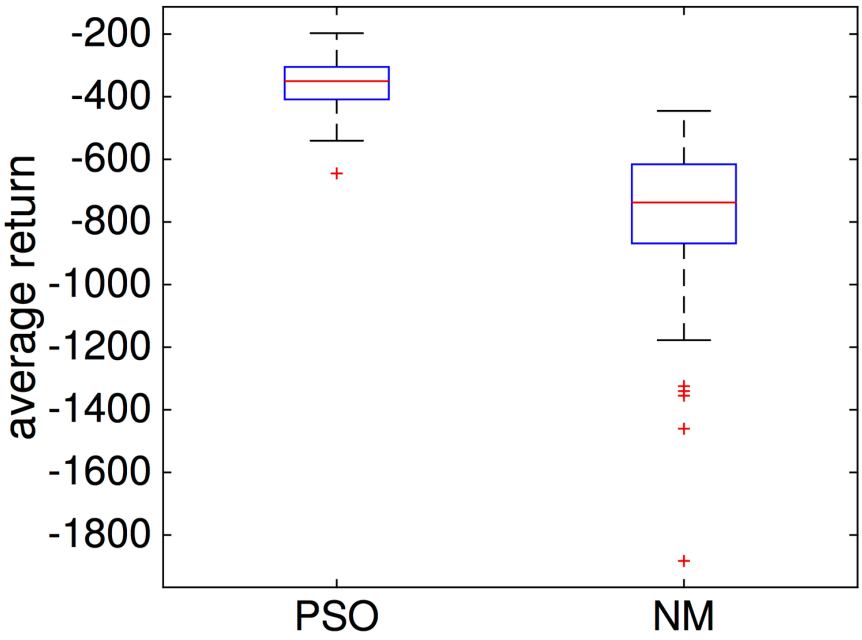
Table 1. PSO-P setup parameters and achieved experiment results for the MC benchmark

Particles	100
Iterations	100
Topology	(each particle with 5 neighbors, including itself)
(w, c_1, c_2)	(0.72981, 1.49618, 1.49618)
(T, q)	(100, 0.05)
Model approximation m	RNN trained with 10,000 randomly generated state transitions
Benchmark start states	100 times $s = (\rho, \dot{\rho}) = (-\pi \,/\, 6, 0.0)$
Return (1,000 steps)	median: -350, min: -644, max: -197

To confirm that finding the optimal way of driving the car up the mountain is represented as a non-convex optimization problem, the performance of PSO has been compared with a standard simplex algorithm (NM) published by (Nelder & Mead, 1965) applied to Eq. (4). The NM algorithm was allowed to utilize the exact same number of fitness evaluations during the optimization as the PSO (100 particles and 100 PSO iterations corresponding to 10,000 fitness evaluations).

The results presented in Figure 3 show that on average PSO yields a significantly better control performance than NM. This result was expected, since the problem is assumed to be highly non-convex and NM is likely to get stuck in local optima. Nevertheless, the majority of runs using NM managed to drive the car up the hill in less than 1,000 time steps, even though it took NM significantly more time steps on average (Figure 4).

Figure 3. Visualization of the average return of PSO and NM computed from 100 experiments per setup. Applying the exact same number of fitness function evaluations, PSO outperforms NM on the MC benchmark. On each box, the central mark is the median, the box edges are the 25th and 75th percentiles, and the whiskers extend to the most extreme data points not considered outliers. An average return point g is categorized as an outlier (+) if $g > q_3 + 1.5(q_3 - q_1)$ or $g < q_1 - 1.5(q_3 - q_1)$, for q_1 the 25th and q_3 the 75th percentile.



Cart-Pole Swing-Up Benchmark

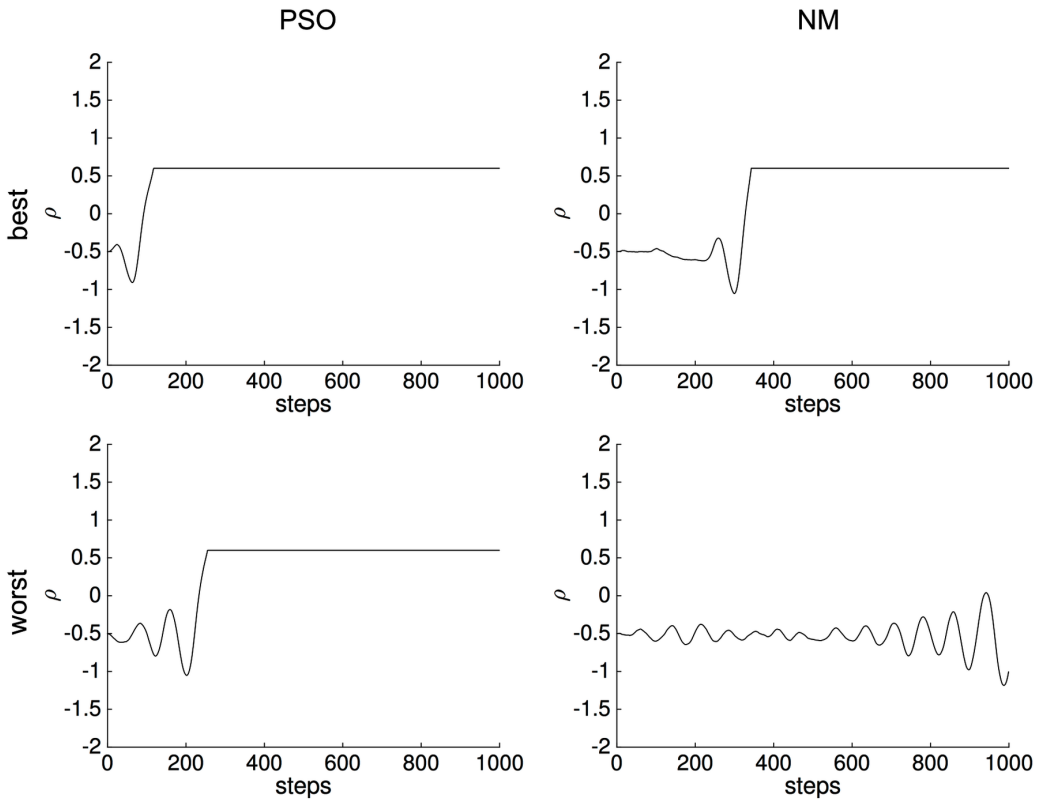
The objective of the CP benchmark is to apply forces to a cart moving on a one-dimensional track to bring a pole hinged to the cart in an upright position (Fantoni & Lozano, 2002). The four Markov state variables are the pole's angle θ, the pole's angular velocity $\dot{\theta}$, the cart's position ρ, and the cart's velocity $\dot{\rho}$, as illustrated in Figure 5.

The start settings for the experiments are: $\theta = \pi$, $\dot{\theta} = 0$, $\rho = 0$, and $\dot{\rho} = 0$, i.e., the pole is hanging down with the cart at rest. The goal is to find force actions $a_t, a_{t+1}, a_{t+2}, \ldots \in [-1,1]$, that swing the pole up and subsequently prevent the pole from falling over while keeping the cart close to $\rho = 0$ for a possibly infinite period of time. The closer the CP gets to the desired position ($\theta = 0, \rho = 0$) the higher are the rewards $r(\rho,\theta) = -\sqrt{(\rho \,/\, 1.4)^2 + (\theta \,/\, 0.3)^2}$ for the corresponding transitions.

Using the parameters given in Table 2, the authors show that PSO-P is able to solve this RL problem. To find the best setting for the user-defined parameters, the authors again followed their recipe from Appendix 1.

Similar to the MC benchmark, the authors compared the resulting performance of PSO to NM when solving Eq. (4). While the MC optimization problem is simple enough for NM to solve it in less than 1,000 time steps, NM completely failed to stabilize the cart's pole in 1,000 time steps. In Figure 6 the

Figure 4. Mountain car experiments. Applying PSO always resulted in well-performing and successful action trajectories, while using NM sometimes did not generate action sequences driving the car up the hill in less than 1,000 time steps.



average return values of PSO and NM are compared. The best and worst out of 100 generated trajectories with PSO and NM optimizations are compared in Figure 7. It is evident that PSO significantly outperforms NM, which indicates that solving CP is a hard optimization problem for deterministic algorithms such as NM.

Industrial Benchmark

The (Hein, et al., 2017a; Hein, et al., 2017b) (source code available at http://github.com/siemens/industrialbenchmark) was designed to emulate several challenging aspects eminent in many industrial applications. It is not designed to be an approximation of any specific real-world system, but to pose a comparable hardness and complexity found in many industrial applications.

State and action spaces are continuous. Moreover, the state space is high-dimensional and only partially observable. The actions consist of three continuous components and affect three control inputs. Moreover, the IB includes stochastic and delayed effects. The optimization task is multi-criterial in the sense that there are two reward components that show opposite dependencies on the actions. The dynamical behavior is heteroscedastic with state-dependent observation noise and state-dependent probability

Figure 5. Cart-pole system

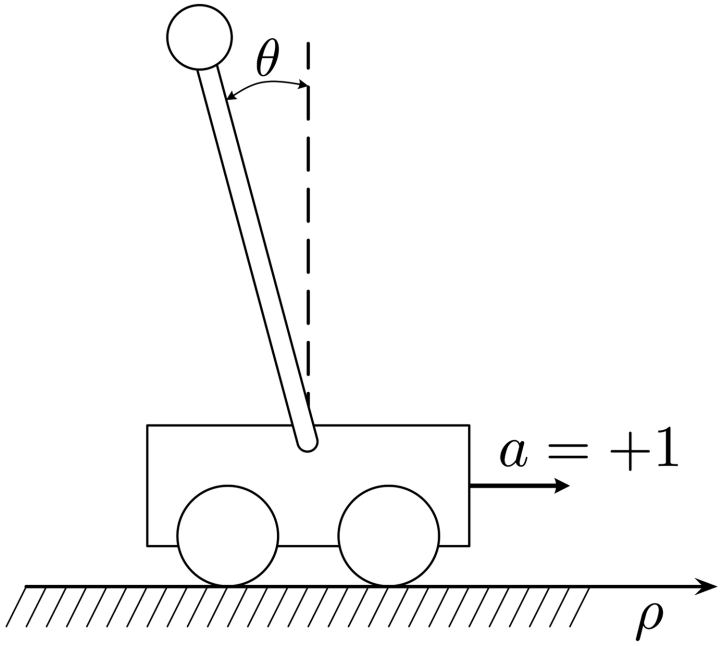


Table 2.PSO-P setup parameters and achieved experiment results for the CP benchmark

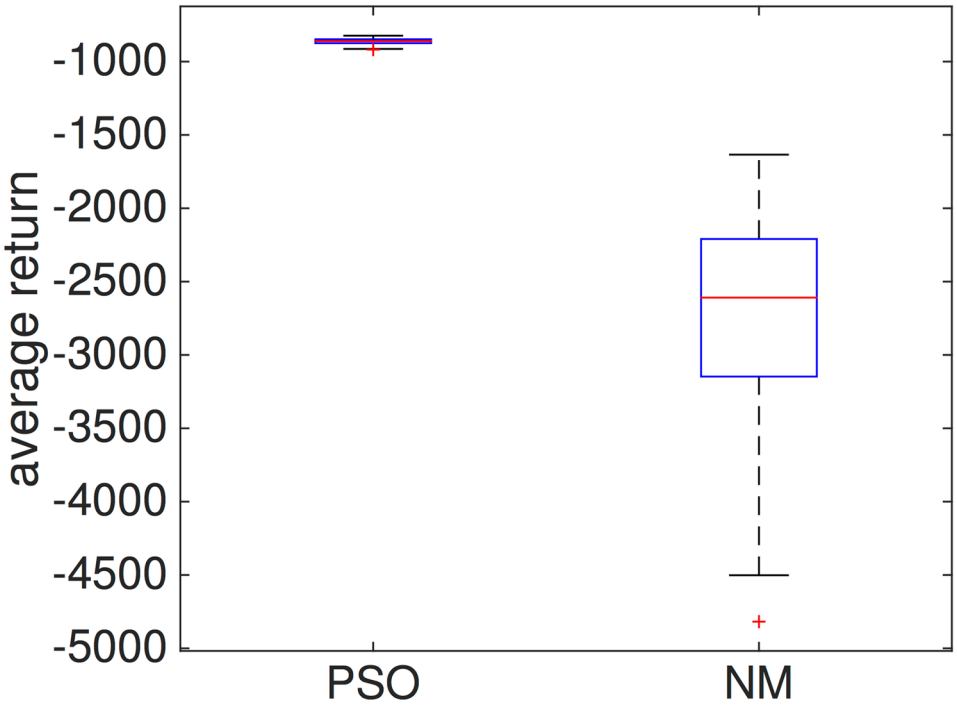
Particles	100
Iterations	100
Topology	(each particle with 5 neighbors, including itself)
(w, c_1, c_2)	(0.72981, 1.49618, 1.49618)
(T, q)	(150, 0.05)
Model approximation m	RNN trained with 10,000 randomly generated state transitions
Benchmark start states	100 times $s = (\theta, \dot{\theta}, \rho, \dot{\rho}) = (\pi, 0, 0, 0)$
Return (1,000 steps)	median: -860, min: -918, max: -823

distributions, based on latent variables. Furthermore, it depends on an external driver that cannot be influenced by the actions.

The IB is designed such that the optimal policy will not approach a fixed operation point in the three control inputs, i.e., constantly changing the control inputs with regard to past observations, resulting in significantly higher return. Note that any specific design choice is driven by experience with industrial challenges.

At any time step t the RL agent can influence the IB via actions a_t that are three dimensional vectors in $\left[-1,1\right]^3$. Each action can be interpreted as three proposed changes to three observable state

Figure 6. Visualization of the average return of PSO and NM computed from 100 experiments per setup. Applying the exact same number of fitness function evaluations, PSO outperforms NM on the CP benchmark. On each box, the central mark is the median, the box edges are the 25th and 75th percentiles, and the whiskers extend to the most extreme data points not considered outliers.



control variables. Those variables are: velocity v, gain g, and shift h. Each variable is limited to $[0,100]$ and calculated as follows:

$$a_t = \left(\Delta v_t, \Delta g_t, \Delta h_t\right), \tag{8}$$

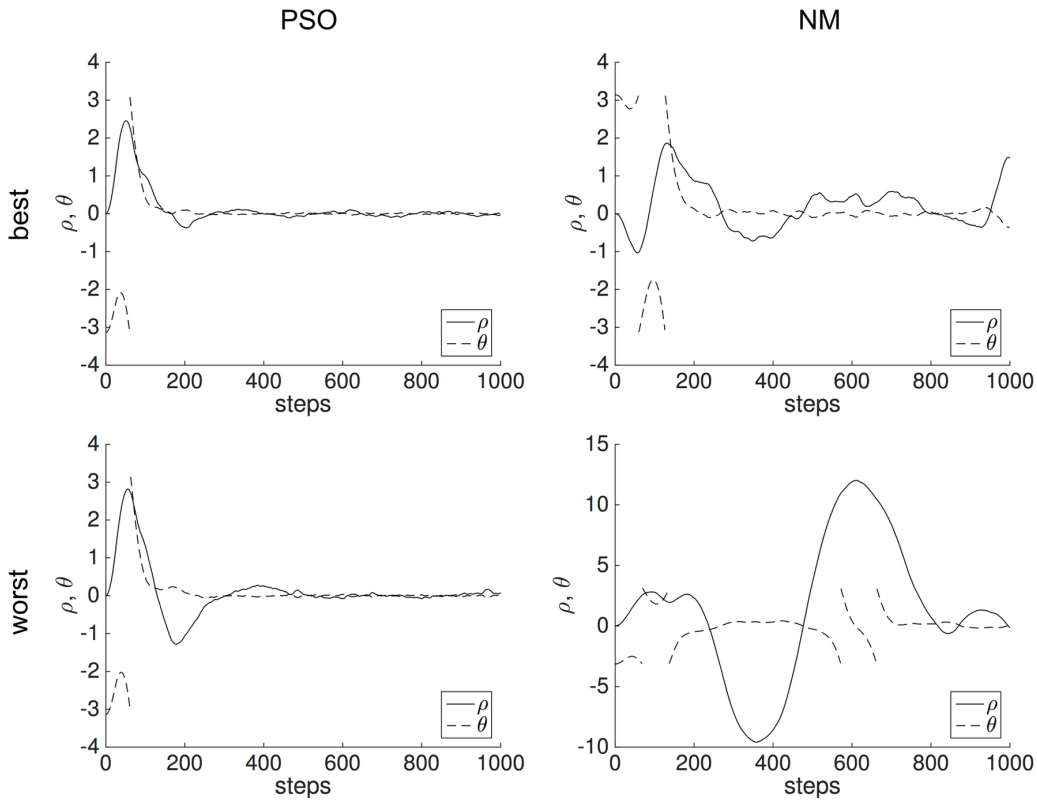
$$v_{t+1} = \max\left(0, \min\left(100, v_t + d^{\mathrm{v}}\Delta v_t\right)\right), \tag{9}$$

$$g_{t+1} = \max\left(0, \min\left(100, g_t + d^{\mathrm{g}}\Delta g_t\right)\right), \tag{10}$$

$$h_{t+1} = \max\left(0, \min\left(100, h_t + d^{\mathrm{h}}\Delta h_t\right)\right), \tag{11}$$

with scaling factors $d^{\mathrm{v}} = 1$, $d^{\mathrm{g}} = 10$, and $d^{\mathrm{h}} = 5.75$.

Figure 7. Cart-pole experiments. Even the worst PSO runs produced action sequences, capable of swinging up the pole and balancing it upright. In contrary the best NM sequences still yielded an overall unstable system control policy.



After applying the action a_t, the environment transitions to the next time step $t+1$, yielding the internal state s_{t+1}. State s_t and successor state s_{t+1} are the Markovian states of the environment, which are only partially observable by the agent. In addition to the three control variables velocity v, gain g, and shift h, an operator defined load p_t is applied to the system. Load p_t simulates an external force like the demanded load in a power plant or the wind speed actuating a wind turbine, which cannot be controlled by the agent, but still has a major influence on the system's dynamics. Depending on load p_t and the control values a_t, the system suffers from detrimental fatigue f_t and consumes resources such as power, fuel, etc., represented by consumption c_t. Both, p_t and a_t, are external drivers for the IB dynamics. In response, the IB generates output values for c_{t+1} and f_{t+1}, which are part of the internal state s_{t+1}. The reward is solely determined by s_{t+1} as follows:

$$r_t = -c_{t+1} - 3f_{t+1} \tag{12}$$

In the real-world tasks that motivated the design of the IB, the reward function has always been known explicitly. Therefore, it is assumed that the reward function of the IB is also known and consumption

and fatigue are observable. However, except for the values of the steerings, the remaining part of the Markov state s_t remains unobservable. This yields an observation vector $o_t \subset s_t$ consisting of:

- The current control variables: velocity v_t, gain g_t, and shift h_t,
- The external driver: set point p_t,
- And the reward relevant variables: consumption c_t and fatigue f_t.

In Section 'Formulation of Reinforcement Learning as Optimization Problem' the optimization task, which is solved during PSO-P runtime, is described as working on the Markovian state s of the system dynamics. Since this state is not observable in the IB environment s_t is approximated by a sufficient amount of historic observations $\left(o_{t-H,}o_{t-H+1},\ldots,o_t\right)$ with time horizon H. Given a system model $m(o_{t-H},o_{t-H+1},\ldots,o_t,a_t)=\left(o_{t+1},r_t\right)$ with $H=30$ an adequate prediction performance could be achieved during IB experiments. Note that observation size $\left|o\right|=6$ in combination with time horizon $H=30$ results in a 180-dimensional approximation vector of the Markovian state. Since the size of the solution space of an RL problem grows exponentially with each additional feature describing the state (Kaelbling, Littman, & Moore, 1996), finding closed-form policies is rather difficult for common RL approaches. Belman (1957) described this problem as *curse of dimensionality*. With PSO-P no closed-form RL policy is trained, instead the complexity of learning state-action dependencies is transferred to the supervised system identification yielding system model m. Recent research has shown that this approach can result in significantly better system control performance for problems with high-dimensional state space compared to standard close-form RL methods (Hein, et al., 2017a).

The task for the PSO-P RL agent is to find a sequence of actions $\mathbf{x}=(\Delta v_t,\Delta g_t,\Delta h_t,\Delta v_{t+1},\Delta g_{t+1},\Delta h_{t+1},\ldots,\Delta v_{t+T-1},\Delta g_{t+T-1},\Delta h_{t+T-1})$ which changes the control variables in a way that return $\mathcal{R}$ is as high as possible for a given time horizon T.

Using the parameters given in Table 3, the authors were able to produce excellent control results on the IB task in terms of average per-step-rewards. Moreover, PSO-P has shown that the policy performance is robust even against highly stochastic benchmark dynamics as present in the IB. Figure 8 compares the average per-step-reward-values of 10 independent IB runs on set point $p=100$. Even though NM performed the exact same number of function evaluations during the optimization phase, it produced a far less satisfying average performance. In Figure 9 the best and worst trajectories of PSO and NM optimization runs are depicted.

FUTURE RESEARCH DIRECTIONS

The experiments with the IB demonstrate that there are stochastic systems which PSO-P can successfully control using a deterministic system model. It is expected that this is not possible in general for stochastic RL environments. To overcome this problem, modeling techniques that can provide a measure of uncertainty along with their predictions are a very promising area of research. Possible modeling candidates are Bayesian NNs, which are approximators with prior distributions on their network weights (Depeweg, Hernández-Lobato, Doshi-Velez, & Udluft, 2016; Neal, 1996), or Gaussian processes (Ras-

Table 3. PSO-P setup parameters and achieved experiment results for the IB benchmark

Particles	100
Iterations	100
Topology	(each particle with 5 neighbors, including itself)
(w, c_1, c_2)	(0.72981, 1.49618, 1.49618)
(T, q)	(100, 0.05)
Model approximation m	RNN trained with 100,000 state transitions generated by random trajectories
Benchmark start states	10 times IB initialized with set point $p = 100$
Return (1,000 steps)	median: -223, min: -224, max: -222

Figure 8. Visualization of the average reward of PSO and NM computed from 10 experiments per setup. Applying the exact same number of fitness function evaluations, PSO outperforms NM on the IB benchmark. On each box, the central mark is the median, the box edges are the 25th and 75th percentiles, and the whiskers extend to the most extreme data points not considered outliers.

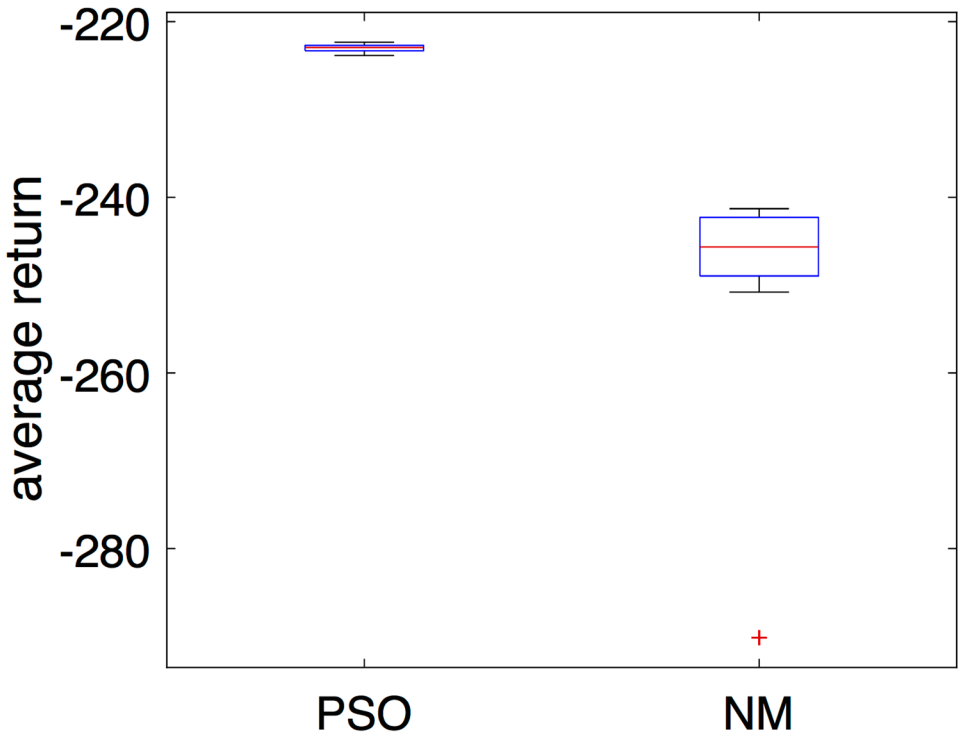
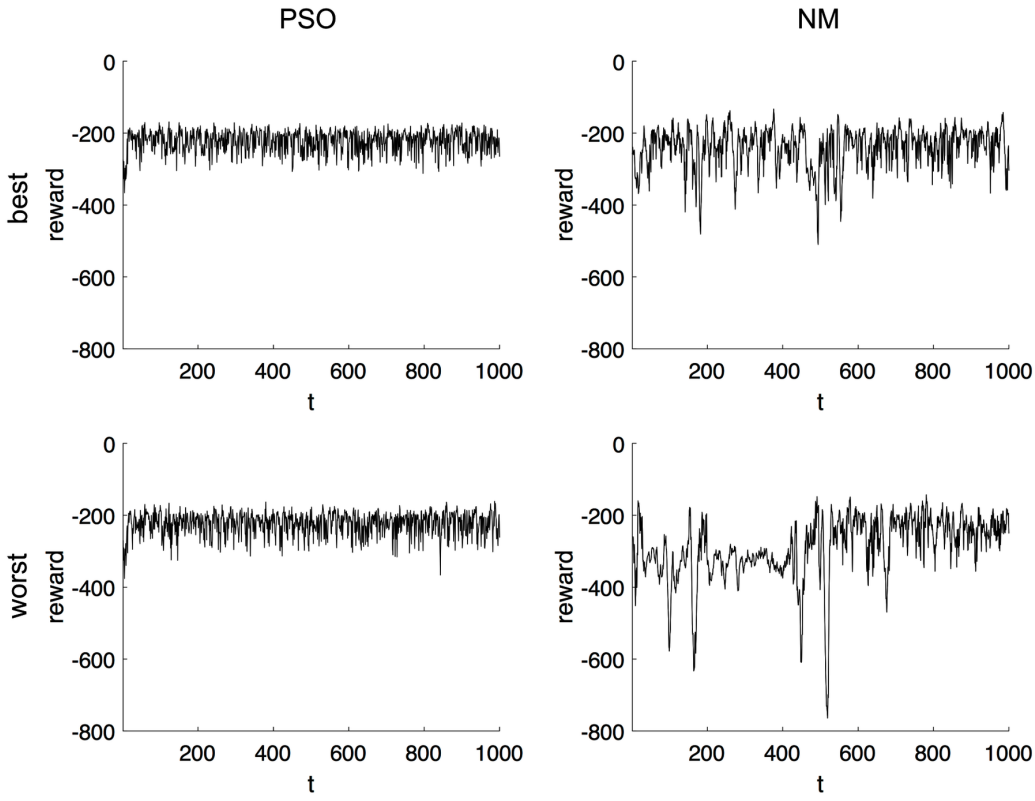


Figure 9. Industrial benchmark experiments. Depicted are the best and the worst performing trajectories for PSO and NM. Note that the performance can easily break down in the IB if suboptimal actions are applied. On the other hand, applying PSO shows clearly that it is possible to find well performing actions even under the presence of latent stochastic effects and state-dependent observation noise.



mussen & Williams, 2006; Damianou, Titsias, & Lawrence, 2016), where every point in a continuous input space is associated with a normally distributed random variable. Together with noise-resistant PSO (Bartz-Beielstein, Blum, & Branke, 2007) the evaluation of these techniques is a promising future research direction for PSO-P application in stochastic environments.

Recent breakthroughs in the area of applying deep NNs on image, video, speech or text data (Lecun, Bengio, & Hinton, 2015) could bring model-based RL methods like PSO-P into new domains of application. Modeling environments in these domains is an emerging trend, promising human-level control through deep RL (Mnih, et al., 2015).

Another open research topic is the real-time application of PSO-P on systems with time-critical constraints. The implementation of a parallelized PSO and the system model directly on hardware could enable achieving this goal for a broad set of different industry relevant applications (Van Heerden, Fujimoto, & Kawamura, 2014; Xu, Chen, Gong, & Mei, 2016).

CONCLUSION

The presented results show that PSO-P is capable of providing RL agents with high-quality state-to-action mappings. In essence, PSO-P performs an online optimization of an action sequence, each time an action for a given system state is requested. Compared to learning a functional policy representation, whose actions are recalled later on, PSO-P has the following advantages:

- PSO-P does not require a priori assumptions about adequate policy representations. Hence, no bias with respect to a specific policy behavior is introduced.
- PSO-P is effective for high-dimensional state spaces, as the optimization runs in the space of action sequences, which are independent of the state space's dimensionality.
- The reward function can be changed after each system transition, as the optimization process starts from scratch for each new system state.

The drawback compared to closed-form policies is the significantly higher computational load for computing actions using PSO-P. Implementing parallelized PSO on hardware or using cloud-based computational resources will enable MPC policy solutions like PSO-P to become feasible for more and more applications. Furthermore, in many real-world industrial applications high-level system control is implemented by changing control parameters in terms of seconds or minutes which, in many cases, is a sufficient amount of time to compute the next action using PSO-P.

PSO-P is a complementary approach for solving RL because it searches in the action space, while established RL methods generally work in the value function space or the policy space (Sutton & Barto, 1998). Therefore, a promising application is to use PSO-P for benchmarking other RL methods. Moreover, PSO-P can be used for reward function design or tuning, i.e., for the process of designing a reward function that induces a desired policy behavior.

Furthermore, the presented method for formulating RL problems as optimization tasks makes the rich class of real-world inspired RL benchmarks accessible for benchmarking gradient-free optimization algorithms. The fitness landscapes of RL problems are generally non-convex and high-dimensional. Since each point in this space corresponds to an action trajectory, the optimization process can be visualized as a sequence of such action trajectories, which may be used to interpret the behavior of different optimization algorithms.

ACKNOWLEDGMENT

The project this report is based on was supported with funds from the German Federal Ministry of Education and Research under project number 01IB15001. The sole responsibility for the report's contents lies with the authors.

REFERENCES

Bartz-Beielstein, T., Blum, D., & Branke, J. (2007). Particle swarm optimization and sequential sampling in noisy environments. In K. F. Doerner, M. Gendreau, P. Greistorfer, W. Gutjahr, R. F. Hartl, & M. Reimann (Eds.), *Metaheuristics: Progress in Complex Systems Optimization* (pp. 261–273). Boston, MA: Springer US. doi:10.1007/978-0-387-71921-4_14

Bellman, R. E. (1957). *Dynamic Programming*. Princeton University Press.

Bellman, R. E. (1962). *Adaptive Control Processes: A Guided Tour*. Princeton University Press.

Camacho, F., & Alba, C. (2007). *Model predictive control*. London: Springer. doi:10.1007/978-0-85729-398-5

Damianou, A. C., Titsias, M. K., & Lawrence, N. D. (2016). Variational inference for latent variables and uncertain inputs in Gaussian processes. *Journal of Machine Learning Research*, *17*(1), 1425–1486.

Depeweg, S., Hernández-Lobato, J. M., Doshi-Velez, F., & Udluft, S. (2016). *Learning and policy search in stochastic dynamical systems with Bayesian neural networks.* arXiv preprint arXiv:1605.07127

Eberhart, R., & Shi, Y. (2000). Comparing inertia weigths and constriction factors in particle swarm optimization. *Proceedings of the IEEE Congress on Evolutionary Computation,* (1), 84–88.

Eberhart, R., Simpson, P., & Dobbins, R. (1996). *Computational intelligence PC tools*. San Diego, CA: Academic Press Professional, Inc.

Engelbrecht, A. (2005). *Fundamentals of computational swarm intelligence*. Wiley.

Fantoni, I., & Lozano, R. (2002). *Non-linear control for underactuated mechanical systems*. London: Springer. doi:10.1007/978-1-4471-0177-2

Feng, H.-M. (2005). Particle swarm optimization learning fuzzy systems design. *Third International Conference on Information Technology and Applications,* 1, 363-366. doi:10.1109/ICITA.2005.206

Findeisen, R., & Allgoewer, F. (2002). An introduction to nonlinear model predictive control. *21st Benelux Meeting on Systems and Control*, 1-23.

Findeisen, R., Allgoewer, F., & Biegler, L. (2007). *Assessment and future directions of nonlinear model predictive control*. Berlin: Springer-Verlag. doi:10.1007/978-3-540-72699-9

Gruene, L., & Pannek, J. (2011). *Nonlinear model predictive control*. London: Springer. doi:10.1007/978-0-85729-501-9

Hein, D., Depeweg, S., Tokic, M., Udluft, S., Hentschel, A., Runkler, T. A., & Sterzing, V. (2017b). *A benchmark environment motivated by industrial control problems.* arXiv preprint arXiv:1709.09480

Hein, D., Udluft, S., Tokic, M., Hentschel, A., Runkler, T. A., & Sterzing, V. (2017a). Batch reinforcement learning on the industrial benchmark: First experiences. *Proceedings of the IEEE International Joint Conference on Neural Networks*, 4214-4221. doi:10.1109/IJCNN.2017.7966389

Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, *2*(5), 359–366. doi:10.1016/0893-6080(89)90020-8

Johansen, T. (2011). Introduction to nonlinear model predictive control and moving horizon estimation. In *Selected Topics on Constrained and Nonlinear Control*. Bratislava: STU Bratislava/NTNU Trondheim.

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research*, *4*(1), 237–285.

Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. *Proceedings of IEEE International Conference on Neural Networks*, 1942–1948. doi:10.1109/ICNN.1995.488968

Kwon, W. H., Bruckstein, A. M., & Kailath, T. (1983). Stabilizing state-feedback design via the moving horizon method. *International Journal of Control*, *37*(3), 631–643. doi:10.1080/00207178308932998

Lecun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436–444. doi:10.1038/nature14539 PMID:26017442

Lee, S.-M., & Myung, H. (2015). Receding horizon particle swarm optimisation-based formation control with collision avoidance for non-holonomic mobile robots. *IET Control Theory & Applications*, *9*(14), 2075–2083. doi:10.1049/iet-cta.2015.0071

Magni, L., & Scattolini, R. (2004). Stabilizing model predictive control of nonlinear continuous time systems. *Annual Reviews in Control*, *28*(1), 1–11. doi:10.1016/j.arcontrol.2004.01.001

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., & Hassabis, D. et al. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529–533. doi:10.1038/nature14236 PMID:25719670

Montavon, G., Orr, G., & Müller, K. (2012). *Neural networks: Tricks of the trade*. Berlin: Springer. doi:10.1007/978-3-642-35289-8

Montazeri-Gh, M., Jafari, S., & Ilkhani, M. (2012). Application of particle swarm optimization in gas turbine engine fuel controller gain tuning. *Engineering Optimization*, *44*(2), 225–240. doi:10.1080/0305215X.2011.576760

Moriarty, D., Schultz, A., & Grefenstette, J. (1999). Evolutionary algorithms for reinforcement learning. *Journal of Artificial Intelligence Research*, *11*, 241–276.

Neal, R. M. (1996). *Bayesian learning for neural networks* (Vol. 118). New York: Springer-Verlag. doi:10.1007/978-1-4612-0745-0

Nelder, J., & Mead, R. (1965). A simplex method for function minimization. *The Computer Journal*, *7*(4), 308–313. doi:10.1093/comjnl/7.4.308

Ou, Y., Kang, P., Jun, K. M., & Julius, A. A. (2015). Algorithms for simultaneous motion control of multiple T. pyriformis cells: Model predictive control and particle swarm optimization. *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 3507-3512. doi:10.1109/ICRA.2015.7139684

Piche, S., Keeler, J., Martin, G., Boe, G., Johnson, D., & Gerules, M. (2000). Neural network based model predictive control. *Advances in Neural Information Processing Systems*, 1029–1035.

Rasmussen, C., & Williams, C. (2006). *Gaussian processes for machine learning*. MIT Press.

Rawlings, J. (2000). Tutorial overview of model predictive control. *IEEE Control Systems Magazine*, *20*(3), 38–52. doi:10.1109/37.845037

Rawlings, J., & Mayne, D. (2009). *Model predictive control theory and design*. Nob Hill Publishing.

Schaefer, A., Schneegass, D., Sterzing, V., & Udluft, S. (2007). A neural reinforcement learning approach to gas turbine control. *IEEE International Conference on Neural Networks - Conference Proceedings*, 1691-1696. doi:10.1109/IJCNN.2007.4371212

Solihin, M., & Akmeliawati, R. (2010). Particle swam optimization for stabilizing controller of a self-erecting linear inverted pendulum. *International Journal of Electrical and Electronic Systems Research*, *2*, 13–23.

Sutton, R. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Advances in Neural Information Processing Systems*, *8*, 1038–1044.

Sutton, R., & Barto, A. (1998). *Reinforcement learning: An introduction*. Cambridge, MA: MIT Press.

Van Heerden, K., Fujimoto, Y., & Kawamura, A. (2014). A combination of particle swarm optimization and model predictive control on graphics hardware for real-time trajectory planning of the under-actuated nonlinear acrobot. *2014 IEEE 13th International Workshop on Advanced Motion Control (AMC)*, 464-469.

Wolpert, D., & Macready, W. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, *1*(1), 67–82. doi:10.1109/4235.585893

Xu, F., Chen, H., Gong, X., & Mei, Q. (2016). Fast nonlinear model predictive control on FPGA using particle swarm optimization. *IEEE Transactions on Industrial Electronics*, *63*(1), 310–321. doi:10.1109/TIE.2015.2464171

KEY TERMS AND DEFINITIONS

Benchmark: A computer program used to assess the performance of different methods.

Model Predictive Control: A method of process control using a system model with finite time-horizon, where at each time step only the next control action is applied to the real system.

Neural Network: A technical computing system inspired by biological brains. It consists of connected nodes (neurons) arranged in layers, where the output of each neuron is computed from the inputs using activation functions.

Policy: A mapping from state to action space, which is the result of a reinforcement learning training.

Reinforcement Learning: Software agents are trained to take optimal actions in a given environment in order to maximize a cumulative reward.

System Model: An approximation of the input-to-output behavior of a real system trained from observational data by supervised machine learning. It may be used for policy evaluation or selection.

Trajectory: A time-ordered set of states or actions.

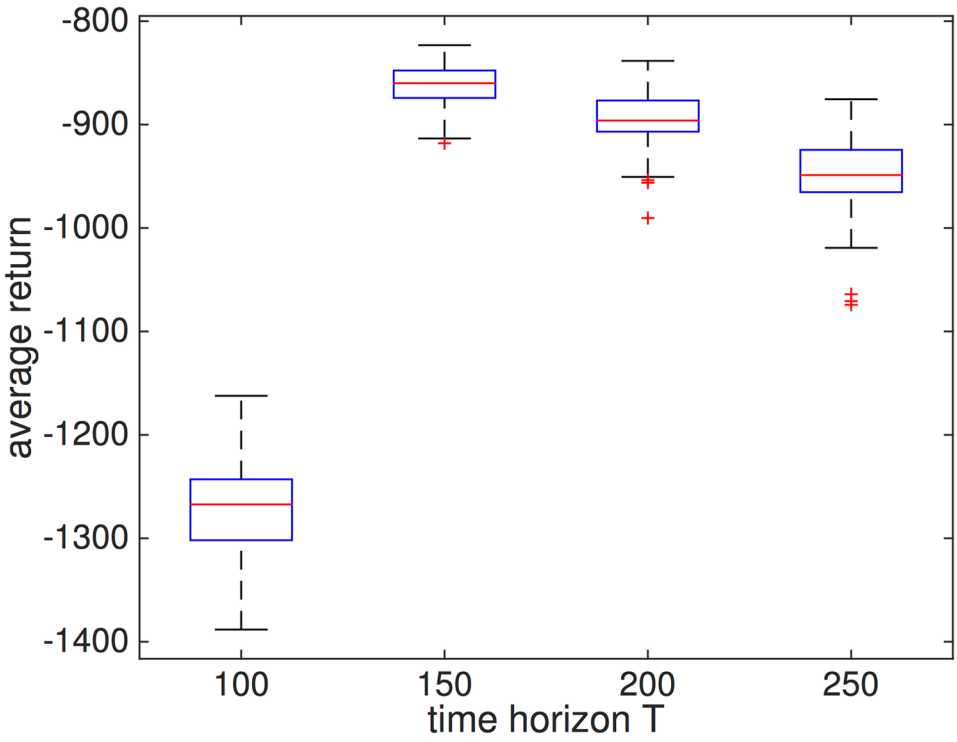
APPENDIX 1

Given a sufficiently trained model of the real system, the conducted experiments show that the following recipe successfully finds appropriate parameters for the PSO-P:

0. Start with the ring topology and an initial guess of the swarm size, depending on the intended computational effort.
1. Evaluate the problem dependent *time horizon* T .
2. Compare different *topologies* for both convergence properties; speed and quality of the found solutions.
3. Determine the *number of particles* which leads to the best rewards, given a fixed level of computational effort.

In the following, an exemplary PSO-P parameter evaluation for the CP benchmark is described. The first step is to find a suitable time horizon for the RL problem. On the one hand, this horizon should be as short as possible to keep computational effort low. On the other hand, it has to be long enough to recognize all possible future effects of the current action. Figure 10 shows the results for time horizons of length 100, 150, 200 and 250 time steps. A time horizon length of 100 yields a relatively low average return compared to the horizon lengths 150, 200 and 250. The reason for this is that it is much harder for the

Figure 10. The data has been produced evaluating 100 independent trial runs with the goal of swinging up and stabilizing the cart-pole. Each trial contains 1,000 applied actions



PSO-P to determine whether an action sequence leads to constantly good results in the future if the time horizon is below 150 in the CP benchmark. The increase of the horizon above 150 did not yield better results, so the horizon of 150 seems to be a good compromise between stable results and fast computing. In the second step, the influence of the PSO topology on the CP task is evaluated. Three topologies have been tested: star (global PSO), ring with three neighbors, and ring with five neighbors (including the particle itself). While the two ring topologies produced similar results, the global PSO performed slightly worse. Probably, the swarm prematurely collapses to suboptimal solutions. In Figure 11 - Figure 12 the average performance of 100 PSO optimizations on the CP start state is shown in relation to the number of function calls. A smaller neighborhood seems to be a much better approach to the CP task, which in essence limits the communication of good positions through the swarm. Thereby, a smaller neighborhood size favors exploration over exploitation. Since the topology ring with five neighbors produced the best median result the experiments are continued using this topology.
In the last step, the influences of the number of particles and PSO iterations are investigated. In the experiments, the runtime of the optimization has been fixed by limiting the PSO to a total of 10,000 fitness evaluations. Consequently, a swarm of 200 particles can run 50 PSO iterations, while a swarm of 100 Particles can perform 100 iterations using the same computation time. The results in Figure 13 - Figure 14 show that a swarm of size 100 particles finds better solutions in 100 PSO iteration steps than 50 particles in 200 iterations, or 200 particles in 50 iterations. However, if the time frame allowed only 5000 fitness function calls to compute the next action, it would be significantly better to use the combination of 50 particles in 200 PSO iterations than any other ratio evaluated in the experiment.

Figure 11. Results of the comparison of the three PSO topologies, ring with three neighbors for each particle, ring with five neighbors for each particle, and the star topology. Illustrated are the average convergence speeds of 100 PSO runs searching for an optimal action sequence for the initial state.

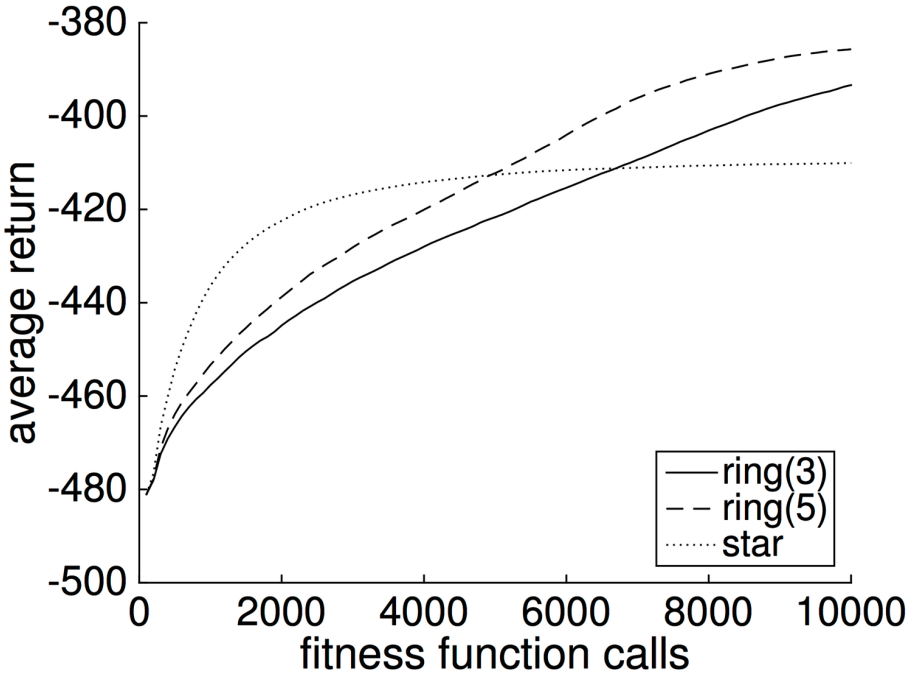


Figure 12. Depicted are the average results on a complete PSO-P run of 1,000 steps

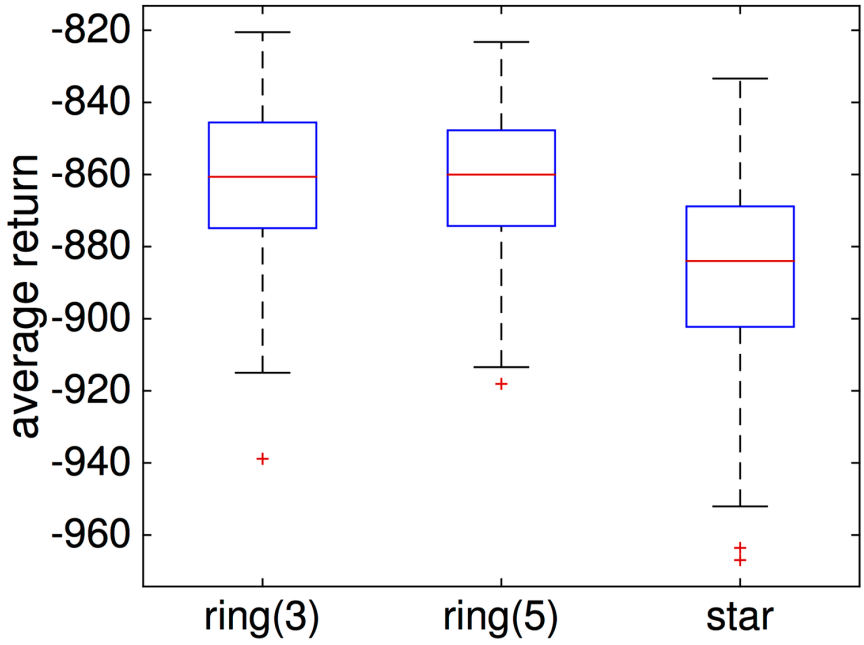


Figure 13. Results of the comparison of three numbers of particles to PSO iteration ratios. The graphs illustrate the average convergence speed of 100 PSO runs searching for an optimal action sequence for the initial state

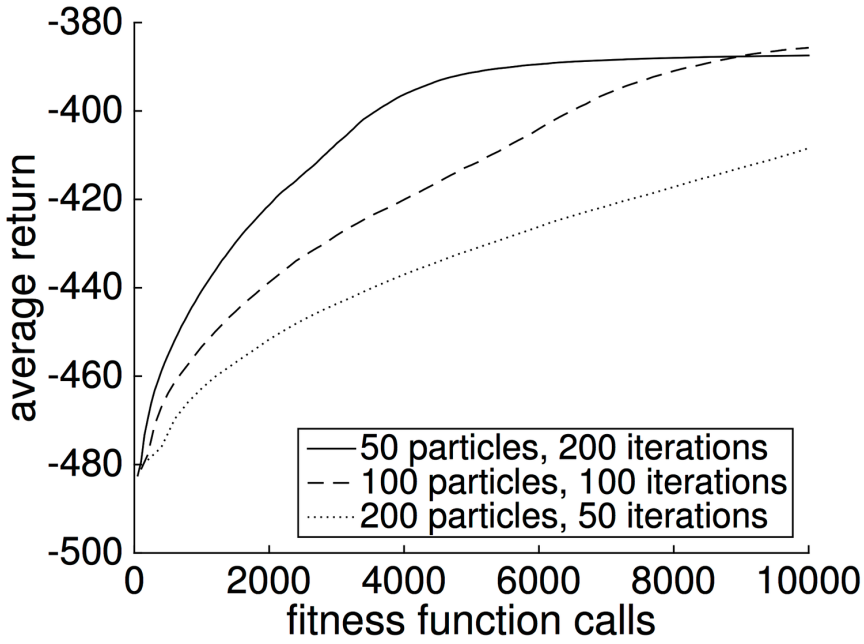
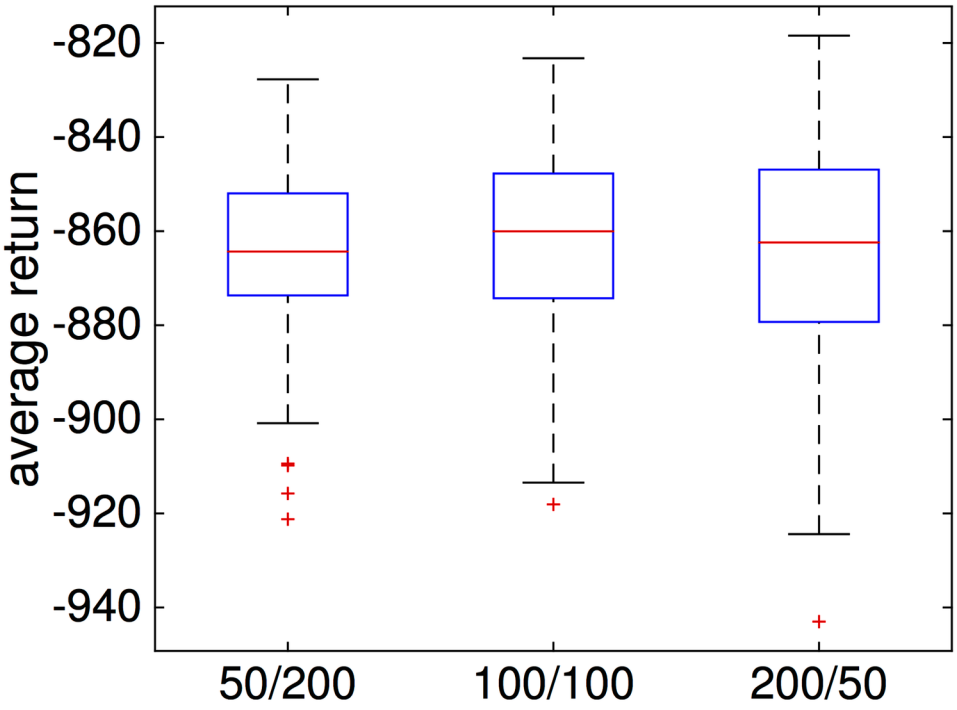


Figure 14. Depicted are the average results on a complete PSO-P run of 1,000 steps



APPENDIX 2

Table 4.

Policy Framework

// s_0 benchmark start state
// $g(s,a)$ real-world system
// I action dimensionality
Begin
$s \leftarrow s_0$
Repeat
$\hat{\mathbf{x}} \leftarrow$ **PSO-P**(s) // call PSO-P procedure and determine best
// action sequence
$a \leftarrow \hat{\mathbf{x}}[0,\cdots,I-1]$ // extract first action vector of the sequence
$(s,r) \leftarrow g(s,a)$ // apply action on the real system
Until (termination conditions achieved)
End

Table 5.

PSO-P

// i particle index
// j search space dimension
// $\mathbf{x}_i$ position vector
// $\mathbf{v}_i$ velocity vector
// $\mathbf{y}_i$ best position
// $\hat{\mathbf{y}}_i$ best position in particle i's neighborhood
// $\mathbf{x}_{\min}$ minimum position (Eq. (1))
// $\mathbf{x}_{\max}$ maximum position (Eq. (2))
// $\mathbf{v}_{\min}$ minimum position, $\mathbf{v}_{\min_j} = -0.1 \cdot (\mathbf{x}_{\max_j} - \mathbf{x}_{\min_j})$
// $\mathbf{v}_{\max}$ maximum position, $\mathbf{v}_{\max_j} = 0.1 \cdot (\mathbf{x}_{\max_j} - \mathbf{x}_{\min_j})$
// P applied PSO iterations
// Input:
// s optimization start state
// Output:
// $\hat{\mathbf{x}}$ optimized action sequence
Begin
// Initialization
$p \leftarrow 0$
$\mathbf{x}_i(p) \sim U(\mathbf{x}_{\min}, \mathbf{x}_{\max})$ // set random positions
$\mathbf{v}_i(p) \sim U(\mathbf{v}_{\min}, \mathbf{v}_{\max})$ // set random velocities
// Iteration
For $p < P$
$f(\mathbf{x}_i(p)) \leftarrow \mathbf{mbc}(s, \mathbf{x}_i(p))$ // compute fitness of all particles
Update best positions $\mathbf{y}_i(p)$
Update best neighborhood positions $\hat{\mathbf{y}}_i(p)$ // Eq. (5)
Update velocity vectors $\mathbf{v}_i(p+1)$ // Eq. (7)
Bound velocity vectors in between $\mathbf{v}_{\min}$ and $\mathbf{v}_{\min}$
Update positions $\mathbf{x}_i(p+1)$ // Eq. (6)
Bound positions in between $\mathbf{x}_{\min}$ and $\mathbf{x}_{\max}$
$p \leftarrow p + 1$
End
$\hat{\mathbf{x}} \leftarrow$ best overall particle position // Eq. (4)
End

Table 6.

mbc - Model-Based Computation

// $m(s,a)$ model approximation of the real-world system $g(s,a)$
// γ discount factor
// I action dimensionality
// Input:
// s model start state
// $\mathbf{x}$ action sequence
// Output:
// $\mathcal{R}$ return prediction
Begin
$\mathcal{R} \leftarrow 0$
$s \leftarrow s_t$
$k \leftarrow 0$
For $k < T$
$a \leftarrow \mathbf{x}[k \cdot I, \cdots, k \cdot I + I - 1]$ // extract action
$(s, r) \leftarrow m(s, a)$ // perform one step on the model
$\mathcal{R} \leftarrow \mathcal{R} + \gamma^k \cdot r$ // discount reward and accumulate
$k \leftarrow k + 1$
End
End