Optimizing Sensor Redundancy in Sequential Decision-Making Problems

Jonas Nüßlein, Maximilian Zorn, Fabian Ritz, Jonas Stein, Gerhard Stenzel, Julian Schönberger,
Thomas Gabor and Claudia Linnhoff-Popien

Institute of Computer Science, LMU Munich, Germany

fi

Keywords: Reinforcement Learning, Sensor Redundancy, Robustness, Optimization.

Abstract: Reinforcement Learning (RL) policies are designed to predict actions based on current observations to maximize cumulative future rewards. In real-world, i.e. not simulated, environments, sensors are essential for measuring the current state and providing the observations on which RL policies rely to make decisions. A significant challenge in deploying RL policies in real-world scenarios is handling sensor dropouts, which can result from hardware malfunctions, physical damage, or environmental factors like dust on a camera lens. A common strategy to mitigate this issue is to use backup sensors, though this comes with added costs. This paper explores the optimization of backup sensor configurations to maximize expected returns while keeping costs below a specified threshold, C. Our approach uses a second-order approximation of expected returns and includes penalties for exceeding cost constraints. The approach is evaluated across eight OpenAI Gym environments and a custom Unity-based robotic environment (*RobotArmGrasping*). Empirical results demonstrate that our quadratic program effectively approximates real expected returns, facilitating the identification of optimal sensor configurations.

1 INTRODUCTION

Reinforcement Learning (RL) has emerged as a prominent technique for solving sequential decision-making problems, paving the way for highly autonomous systems across diverse fields. From controlling complex physical systems like tokamak plasma (Degrave et al., 2022) to mastering strategic games such as Go (Silver et al., 2018), RL has demonstrated its potential to revolutionize various domains. However, when transitioning from controlled environments to real-world applications, RL faces substantial challenges, particularly in dealing with sensor dropouts. In critical scenarios, such as healthcare or autonomous driving, the failure of sensors can result in suboptimal decisions or even catastrophic outcomes.

The core of RL decision-making lies in its reliance on continuous, accurate observations of the environment, often captured through sensors. When these sensors fail to provide reliable data — whether due to hardware malfunctions, environmental interference, or physical damage — the performance of RL policies can degrade significantly (Dulac-Arnold et al., 2019). In domains such as aerospace, healthcare, nuclear energy, and autonomous vehicles, the consequences of sensor failures are particularly severe. This vulnerability highlights the importance of addressing sensor dropout to ensure the robustness of RL systems in real-world applications. Thus, improving the resilience of RL systems to sensor dropouts is not just desirable—it is critical for ensuring their safe and reliable deployment in mission-critical applications.

One common approach to mitigate the risks of sensor dropouts is the implementation of redundant backup sensors. These backup systems provide an additional layer of security, stepping in when primary sensors fail, thereby maintaining the availability of crucial data. While redundancy can enhance system resilience, it also introduces significant costs, and not all sensor dropouts result in performance degradation severe enough to justify the investment in backups. This paper presents a novel approach to optimizing backup sensor configurations in RL-based systems. Our method focuses on balancing system performance — quantified by expected returns in a Markov Decision Process (MDP) — and the costs associated with redundant sensors. [1]

---

[1] This publication was created as part of the Q-Grid project (13N16179) under the "quantum technologies – from basic research to market" funding program, supported by the German Federal Ministry of Education and Research.

2 BACKGROUND

2.1 Markov Decision Processes (MDP)

Sequential Decision-Making problems are frequently modeled as Markov Decision Processes (MDPs). An MDP is defined by the tuple $E = \langle S, A, T, r, p_0, \gamma \rangle$, where S represents the set of states, A is the set of actions, and $T(s_{t+1} \mid s_t, a_t)$ is the probability density function (pdf) that governs the transition to the next state s_{t+1} after taking action a_t in state s_t. The process is considered Markovian because the transition probability depends only on the current state s_t and the action a_t, and not on any prior states $s_{\tau < t}$.

The function $r : S \times A \to \mathbb{R}$ assigns a scalar reward to each state-action pair (s_t, a_t). The initial state is sampled from the start-state distribution p_0, and $\gamma \in [0, 1)$ is the discount factor, which applies diminishing weight to future rewards, giving higher importance to immediate rewards.

A deterministic *policy* $\pi : S \to A$ is a mapping that assigns an action to each state. The *return* $R = \sum_{t=0}^{\infty} \gamma^t \cdot r(s_t, a_t)$ is the total (discounted) sum of rewards accumulated over an episode. The objective, typically addressed by Reinforcement Learning (RL), is to find an optimal policy π^* that maximizes the expected cumulative return:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{p_0} \left[\sum_{t=0}^{\infty} \gamma^t \cdot r(s_t, a_t) \mid \pi \right]$$

Actions a_t are selected according to the policy π. In Deep Reinforcement Learning (DRL), where state and action spaces can be large and continuous, the policy π is often represented by a neural network $\hat{f}_\phi(s)$ with parameters ϕ, which are learned through training (Sutton and Barto, 2018).

2.2 Quadratic Unconstrained Binary Optimization (QUBO)

Quadratic Unconstrained Binary Optimization (QUBO) is a combinatorial optimization problem defined by a symmetric, real-valued $(m \times m)$ matrix Q, and a binary vector $x \in \mathbb{B}^m$. The objective of a QUBO problem is to minimize the following quadratic function:

$$x^* = \arg\min_x H(x) = \arg\min_x \sum_{i=1}^{m} \sum_{j=i}^{m} x_i x_j Q_{ij} \quad (1)$$

The function $H(x)$ is commonly referred to as the *Hamiltonian*, and in this paper, we refer to the matrix Q as the "QUBO matrix".

The goal is to find the optimal binary vector x^* that minimizes the Hamiltonian. This task is known to be *NP*-hard, making it computationally intractable for large instances without specialized techniques. QUBO is a significant problem class in combinatorial optimization, as it can represent a wide range of problems. Moreover, several specialized algorithms and hardware platforms, such as quantum annealers and classical heuristics, have been designed to solve QUBO problems efficiently (Morita and Nishimori, 2008; Farhi and Harrow, 2016; Nüßlein et al., 2023b; Nüßlein et al., 2023a).

Many well-known combinatorial optimization problems, such as Boolean satisfiability (SAT), the knapsack problem, graph coloring, the traveling salesman problem (TSP), and the maximum clique problem, have been successfully reformulated as QUBO problems (Choi, 2011; Lodewijks, 2020; Choi, 2010; Glover et al., 2019; Lucas, 2014). This versatility makes QUBO a powerful tool for solving a wide variety of optimization tasks, including the one addressed in this paper.

3 ALGORITHM

3.1 Problem Definition

Let π be a trained agent operating within an MDP E. At each timestep, the agent receives an observation $o \in \mathbb{R}^u$, which is collected using n sensors $\{s_i\}_{1 \le i \le n}$. Each sensor s_i produces a vector o^i, and the full observation o is formed by concatenating these vectors: $o = [o^1, o^2, \ldots, o^n]$. The complete observation o has dimension $|o| = \sum_i |o^i|$.

At the start of an episode, each sensor s_i may drop out with probability $d_i \in [0, 1]$, meaning it fails to provide any meaningful data for the entire episode. In the event of a dropout, the sensor's output is set to $o^i = \mathbf{0}$ for the rest of the episode.

If π is represented as a neural network, it is evident that the performance of π, quantified as the expected return, will degrade as more sensors drop out. To mitigate this, we can add backup sensors. However, incorporating a backup sensor s_i incurs a cost $c_i \in \mathbb{N}$, representing the expense associated with adding that redundancy.

The task is to find the optimal backup sensor configuration that maximizes the expected return while keeping the total cost of the backups within a predefined budget $C \in \mathbb{N}$. Let $x \in \mathbb{B}^n$ be a vector of binary decision variables, where $x_i = 1$ indicates the inclusion of a backup for sensor s_i. If $\mathbb{E}_{d,\pi,x}[R]$ represents the expected return while using backup configuration

x, the optimization problem can be formulated with both a soft and hard constraint:

$$x^* = \arg\max_x \mathbb{E}_{d,\pi,x}[R] \quad \text{s.t.} \quad \sum_{i=1}^{n} x_i c_i \leq C \quad (2)$$

An illustration of this problem is provided in *Figure 1*. In the next section, we describe a method to transform this problem into a QUBO representation.

3.2 SensorOpt

There are potentially 2^n possible combinations of sensor dropout configurations, making it computationally expensive to evaluate all of them directly. Since we only have a limited number of episodes $B \in \mathbb{N}$ to sample from the environment E, we opted for a second-order approximation of $\mathbb{E}_{d,\pi,x}[R]$ to reduce the computational burden while still capturing meaningful interactions between sensor dropouts.

To compute this approximation, we first calculate the probability $q(d)$ that at most two sensors drop out in an episode:

$$q(d) = \prod_{i=1}^{n}(1-d_i) + \sum_{i=1}^{n} d_i \cdot \prod_{j \neq i}(1-d_j)$$
$$+ \sum_{i=1}^{n}\sum_{j<i} d_i d_j \cdot \prod_{l \neq i,j}(1-d_l)$$

Next, we estimate the expected return $\hat{R}_{(i,j)}$ for each pair of sensors (s_i, s_j) dropping out. This can be efficiently achieved using **Algorithm 1**, which samples the episode return $E^{(i,j)}(\pi)$ using policy π with sensors s_i and s_j removed. The algorithm begins by calculating two return values for each sensor pair. It then iteratively selects the sensor pair with the highest *momentum* of the mean return, defined as $|\overline{R_{(i,j)}[:-1]} - \overline{R_{(i,j)}}|$. Here, $\overline{R_{(i,j)}[:-1]}$ denotes the mean of all previously sampled returns for the sensor pair (s_i, s_j), excluding the most recent return, while $\overline{R_{(i,j)}}$ represents the mean including the most recent return. The difference between these two values measures the momentum, capturing how much the expected return is changing as more episodes are sampled.

We base the selection of sensor pairs on momentum rather than return variance to differentiate between aleatoric and epistemic uncertainty (Valdenegro-Toro and Mori, 2022). Momentum is less sensitive to aleatoric uncertainty, making it a more reliable indicator in this context.

A simpler, baseline approach to estimate $\hat{R}_{(i,j)}$ would be to divide the budget B of episodes evenly across all sensor pairs (s_i, s_j). In this *Round Robin* approach, we sample $k = \frac{B}{n(n+1)/2}$ episodes for each sensor pair and estimate $\hat{R}_{(i,j)}$ as the empirical mean. In the *Experiments* section, we compare our momentum-based approach from *Algorithm 1* against this naive Round Robin method to highlight its efficiency and accuracy.

Once we have estimated all the values of $\hat{R}_{(i,j)}$, we can compute the overall expected return $\hat{R}(d)$ for a given dropout probability vector d, again using a second-order approximation:

$$\hat{R}(d) = \frac{1}{q(d)}\left[\hat{R}_0\prod_{i=1}^{n}(1-d_i) + \sum_{i=1}^{n}\hat{R}_{(i,i)}d_i\prod_{j\neq i}^{n}(1-d_j)\right.$$
$$\left. + \sum_{i=1}^{n}\sum_{j<i}^{n}\hat{R}_{(i,j)}d_i d_j\prod_{l\neq i,j}^{n}(1-d_l)\right]$$

Now we can calculate the advantage of using a backup for sensor s_i vs. using no backup sensor for s_i:

$$\Delta\hat{R}_{(i,i)}(d) = \hat{R}(d^{\{i\}}) - \hat{R}(d)$$

with

$$d^A = \begin{cases} d_i^2 & \text{if } i \in A, \\ d_i & \text{if } i \notin A. \end{cases}$$

When a backup sensor is used for sensor s_i, the new dropout probability for s_i becomes $d_i \leftarrow d_i^2$, since the system will only fail to provide an observation if both the primary and backup sensors drop out. We adopt the notation d^A to represent the updated dropout probabilities when backup sensors from set A are in use. For example, if the dropout probability vector is $d = (0.2, 0.5, 0.1)$ and a backup is used for sensor 1, the updated vector becomes $d^1 = (0.2, 0.25, 0.1)$, as the backup reduces the dropout probability for sensor 1 while sensors 0 and 2 remain unchanged.

We can now compute the joint advantage of using backups for both sensors s_i and s_j. This joint advantage is not simply the sum of the individual advantages of using backups for each sensor in isolation:

$$\Delta\hat{R}_{(i,j)}(d) = \hat{R}(d^{\{i,j\}}) - \hat{R}(d) - \Delta\hat{R}_{(i,i)}(d) - \Delta\hat{R}_{(j,j)}(d)$$

This formula accounts for the interaction between the two sensors, reflecting the fact that their joint contribution to the expected return is influenced by how the presence of both backup sensors affects the system as a whole, beyond just the sum of their individual contributions. This interaction term is crucial when optimizing backup sensor configurations, as it helps
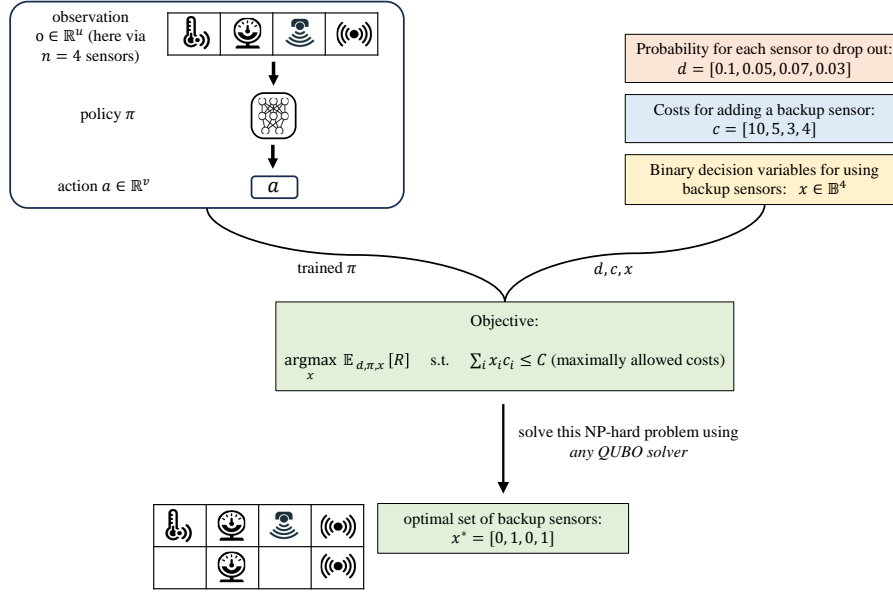
Figure 1: An illustration of our approach for optimizing the backup sensor configuration.

Algorithm 1: Estimating $\hat{R}_{(i,j)}$.

Input: budget of episodes to run $B \in \mathbb{N}$
$\qquad$ # sensors $n \in \mathbb{N}$
$\qquad$ MDP E
$\qquad$ trained policy π

$R_{(i,j)} = [E^{(i,j)}(\pi), E^{(i,j)}(\pi)] \quad \forall i < j, (i,j) \in [1,n]^2$

for $e = 1$ **to** $B - n(n+1)$ **do**
$\qquad (i,j) = \underset{(i,j)}{\arg\max} \; | \overline{R_{(i,j)}[:-1]} - \overline{R_{(i,j)}} |$
$\qquad R_{(i,j)}.\text{append}(E^{(i,j)}(\pi))$
end for
$\hat{R}_{(i,j)} = \overline{R_{(i,j)}} \qquad$ // expected return = mean
$\qquad\qquad\qquad\qquad\qquad$ empirical return

return all $\hat{R}_{(i,j)}$

---

capture the diminishing or synergistic returns that can occur when multiple sensors are backed up together.

With these components in hand, we can now formulate the QUBO problem for the sensor optimization task defined in (2). The soft constraint H^{soft} captures the optimization of the expected return, approximated by the change in return $\Delta\hat{R}_{(i,i)}(d)$ for individual sensors and $\Delta\hat{R}_{(i,j)}(d)$ for sensor pairs:

$$H^{soft} = \sum_{i=1}^{n} x_i \cdot \Delta\hat{R}_{(i,i)}(d) + \sum_{i<j}^{n} x_i x_j \cdot \Delta\hat{R}_{(i,j)}(d)$$

Here, x_i are binary decision variables where $x_i = 1$ indicates that a backup sensor is added for sensor s_i.

This formulation ensures that the algorithm optimizes the expected return based on the specific backup configuration.

To enforce the cost constraint, we introduce a hard constraint H^{hard}, which penalizes configurations where the total cost exceeds the specified budget C:

$$H^{hard} = \left(\sum_{i=1}^{n} x_i c_i + \sum_{i=n+1}^{n+1+\lceil \log_2(C) \rceil} x_i 2^{(i-n-1)} - C \right)^2$$

The expression $\sum_{i=1}^{n} x_i c_i$ represents the total cost of the selected backup sensors, and the second term accounts for the binary encoding of the cost constraint, ensuring that no configuration exceeds the budget C.

Finally, we introduce a scaling factor α to balance the contributions of the soft and hard constraints:

$$\alpha = \sum_{i=1}^{n} |\Delta\hat{R}_{(i,i)}(d)| + \sum_{i<j}^{n} |\Delta\hat{R}_{(i,j)}(d)|$$

The overall QUBO objective function is then given by:

$$H = -H^{soft} + \beta \cdot \alpha \cdot H^{hard} \qquad (3)$$

Here, H^{soft} approximates the expected return $\mathbb{E}_{d,\pi,x}[R]$, while H^{hard} ensures that the total cost remains within the allowable budget. The parameter β is a hyperparameter that controls the trade-off between maximizing the expected return and enforcing the cost constraint.

Algorithm **SensorOpt** summarizes our approach for optimizing sensor redundancy configurations.

This algorithm uses the QUBO formulation to find the optimal backup sensor configuration, balancing performance improvements with budget constraints.

Algorithm 2: - SensorOpt: Calculating best sensor backup configuration.

Input: # sensors $n \in \mathbb{N}$

dropout probability for each sensor $d \in [0,1]^n$

costs for each backup sensor $c \in \mathbb{N}$
maximally allowed costs $C \in \mathbb{N}$
budget of episodes to run $B \in \mathbb{N}$
MDP E
trained policy π

$\hat{R}_0 \leftarrow E(\pi)$ // expected return with no sensor dropouts

$\hat{R}_{(i,j)} \leftarrow$ use **Algorithm 1** (B,n,E,π)

$Q \leftarrow$ create QUBO-matrix using **formula (3)**

$x^* = \arg\min_x x^T Q x$ // solve using any QUBO solver,

         e.g. Tabu Search

     // $x \in \mathbb{B}^m, \quad m = n + \lceil log_2(C) \rceil$

return $x^*[:n]$ // first n bits of x^* encode the optimal

         sensor backup configuration

For a given problem instance (n,d,c,C,B,E,π), α is a constant while $\beta \in [0,1]$ is a hyperparameter we need to choose by hand. The number of boolean decision variables our approach needs is determined by $m = n + \lceil log_2(C) \rceil$.

Proposition 1. *The optimization problem described in* (2) *belongs to complexity class NP-hard.*

Proof. We can prove that the problem described in (2) belongs to *NP*-hard by reducing another problem W, that is already known to be *NP*-hard, to it. Since this would imply that, if we have a polynomial algorithm for solving (2), we could solve any problem instance $w \in W$ using this algorithm. We choose W to be Knapsack (Salkin and De Kluyver, 1975) which is known to be *NP*-hard. For a given Knapsack instance $(n^{(1)}, v^{(1)}, c^{(1)}, C^{(1)})$ consisting of $n^{(1)}$ items with values $v^{(1)}$, costs $c^{(1)}$ and maximal costs $C^{(1)}$ we can reduce this to a problem instance $(n^{(2)}, d^{(2)}, c^{(2)}, C^{(2)}, E)$ from (2) via: $n^{(2)} = n^{(1)}$, $d^{(2)} = 0$, $c^{(2)} = c^{(1)}$, $C^{(2)} = C^{(1)}$. For each possible x we define an individual *MDP* E_x: $S = \{s_0, s_{terminal}\}, A = \{a_0\}, T(s_{terminal}|s_0,a_0) = 1, r(s_0,a_0) = x \cdot v, p_0(s_0) = 1, \gamma = 1$. The expected

return $\mathbb{E}_{d,\pi,x}[R]$ will therefore be $\mathbb{E}_{d,\pi,x}[R] = x \cdot v$. $\quad\square$

4 RELATED WORK

Robust Reinforcement Learning (Moos et al., 2022; Pinto et al., 2017) is a sub-discipline within RL that tries to learn robust policies in the face of several types of perturbations in the Markov Decision Process which are (1) *Transition Robustness* due to a probabilistic transition model $T(s_{t+1}|s_t,a_t)$ (2) *Action Robustness* due to errors when executing the actions (3) *Observation Robustness* due to faulty sensor data.

There is a bulk of papers dealing with *Observation Robustness* (Lütjens et al., 2020; Mandlekar et al., 2017; Pattanaik et al., 2017). However, the vast majority is about handling noisy observations, meaning that the true state lies within an ε-ball of the perceived observation. In these scenarios, an adversarial *antagonist* is usually used in the training process that adds noise to the observation, trying to decrease the expected return, while the agent tries to maximize it (Liang et al., 2022; Lütjens et al., 2020; Mandlekar et al., 2017; Pattanaik et al., 2017). In our problem setting, however, the sensors are not noisy in the sense of ε-disturbance, but they can drop out entirely.

Another line of research is about the framework *Action-Contingent Noiselessly Observable Markov Decision Processes* (ACNO-MDP) (Nam et al., 2021; Koseoglu and Ozcelikkale, 2020). In this framework, observing (measuring) the current state s_t comes with costs. The agent therefore needs to learn when observing a state is crucially important for informed action decision making and when not. The action space is extended by two additional actions {*observe, not observe*} (Beeler et al., 2021). *Not observe*, however, affects all sensors.

Important other related work comes from *Safe Reinforcement Learning* (Gu et al., 2022; Dulac-Arnold et al., 2019). In (Dulac-Arnold et al., 2019) the problem of sensors dropping out is already mentioned and analyzed. However, they only examine the scenario in which the sensor drops out for $z \in \mathbb{N}$ timesteps. Therefore they can use Recurrent Neural Networks for representing the policy that mitigates this problem.

Our approach for optimizing the sensor redundancy configuration given a maximal cost C bears similarity to Knapsack (Salkin and De Kluyver, 1975; Martello and Toth, 1990). In (Lucas, 2014) a QUBO formulation for Knapsack was already introduced and (Quintero and Zuluaga, 2021) provides a study regarding the trade-off parameter for the hard- and soft-constraint. The major difference between the original Knapsack problem and our problem is that in Knap-
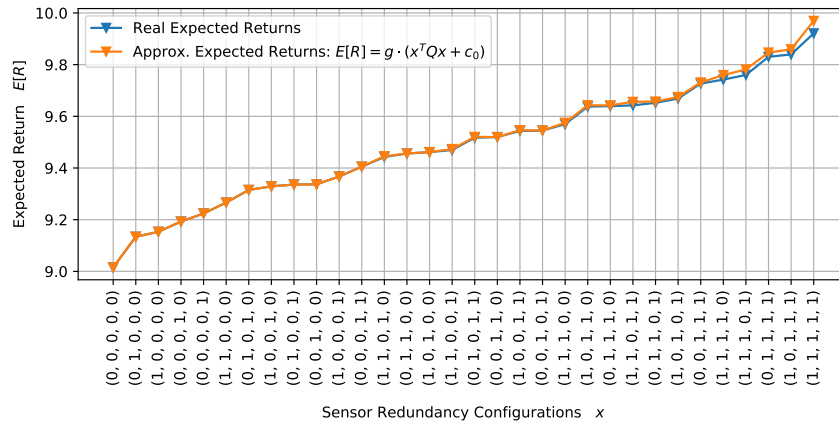
Figure 2: Proof-of-concept: this plot shows the real expected return and the approximated expected return $\mathbb{E}_{d,\pi,x}[R] \approx -x^T Q x + \hat{R}(d)$ when using a backup sensor configuration x. The configurations (x-axis) are sorted according to the real return.

sack, the value of a collection of items is the sum of all item values: $W = \sum_i w_i x_i$. In our problem formulation this, however, doesn't hold. We therefore proposed a second-order approximation for representing the soft-constraint.

To the best of our knowledge, this paper is the first to address the optimization of sensor redundancy in sequential decision-making environments.

5 EXPERIMENTS

In this *Experiments* section we want to evaluate our algorithm *SensorOpt*. Specifically, we want to examine the following hypotheses:

1. Hamiltonian H of formula (3) approximates the real expected returns.

2. Our algorithm **SensorOpt** can find optimal sensor backup configurations.

3. **Algorithm 1** approximates the expected returns $R_{(i,j)}$ faster than with *Round Robin*.

5.1 Proof-of-Concept

To test how well our QUBO formulation (a second-order approximation) approximates the real expected return for each possible sensor redundancy configuration x we created a random problem instance and determined the *Real Expected Returns* $\mathbb{E}_{d,\pi,x}[R]$ and the *Approximated Expected Returns*:

$$\mathbb{E}_{d,\pi,x}[R] \approx -x^T Q x + \hat{R}(d)$$

Note that in general a QUBO-matrix can be linearly scaled by a scalar g without altering the order of the solutions regarding solution quality. *Figure*

2 shows the two resulting graphs where the x-axis represents all possible sensor backup configurations x and the y-axis the expected return. We have sorted the configurations according to their real expected returns. The key finding in this plot is that the approximation is quite good even though it is not perfect for all x. But the relative performance is still intact, meaning especially that the best solution in our approximation is also the best solution regarding the real expected returns. We can therefore verify the first hypothesis that our Hamiltonian H approximates the real expected returns. But as with any approximation, it can contain errors.

5.2 RobotArmGrasping Environment

The *RobotArmGrasping* domain is a robotic simulation based on a digital twin of the *Niryo One*[2] which is available in the *Unity Robotics Hub*[3]. It uses Unity's built-in physics engine and models a robot arm that can grasp and lift objects.

5.3 Main Experiments

Next, we tested if *SensorOpt* can find optimal sensor backup configurations in well-known MDPs. We used 8 different OpenAI Gym environments (Brockman et al., 2016). Additionally, we created a more realistic and industry-relevant environment *RobotArmGrasping* based on Unity. In this environment, the goal of the agent is to pick up a cube and lift it to a desired location. The observation space is 20-dimensional and the continuous action space is 5-dimensional.

---

[2]https://niryo.com/niryo-one/
[3]https://github.com/Unity-Technologies/
Unity-Robotics-Hub

Table 1: Results of our algorithm *SensorOpt* on 9 different environments compared against the true optimum, determined via brute force and the baselines when using no additional backup sensors and all possible backup sensors. We solved the QUBO created in *SensorOpt* using Tabu Search. Note that *All Backups* is not a valid solution to the problem presented in formula (2), since the cost of using all backup sensors exceeds the threshold C. Due to the exponential complexity of conducting a brute force search for the optimal configuration, it was feasible to apply this approach only to the first four environments.

ENVIRONMENT	SENSOROPT	OPTIMUM*	NO BACKUPS	ALL BACKUPS
CARTPOLE-V1	**493.3**	**493.3**	424.0	498.7
ACROBOT-V1	**-86.73**	**-86.73**	-91.38	-83.01
LUNARLANDER-V2	**241.5**	**241.5**	200.1	249.5
HOPPER-V2	**2674**	**2674**	1979	3221
HALFCHEETAH-V2	**9643**	-	8119	10711
WALKER2D-V2	**3388**	-	2402	4097
BIPEDALWALKER-V3	**215.5**	-	160.2	273.0
SWIMMER-V2	**329.6**	-	272.6	341.7
ROBOTARMGRASPING	**29.41**	-	22.27	30.41

In all experiments, we sampled each dropout probability d_i, cost c_i, and maximum costs C uniformly out of a fixed set. For simplicity, we restricted the costs c_i in our experiments to be integers.

For the 8 OpenAI Gym environments we used as the trained policies SAC- (for continuous action spaces) or PPO- (for discrete action spaces) models from *Stable Baselines 3* (Raffin et al., 2021). For our *RobotArmGrasping* environment we trained a PPO agent (Schulman et al., 2017) for 5M steps. We then solved the sampled problem instances using *SensorOpt*. We optimized the QUBO in *SensorOpt* using Tabu Search, a classical meta-heuristic. Further, we calculated the expected returns when using no backup sensors (*No Backups*) and all n backup sensors (*All Backups*). For the four smallest environments *CartPole*, *Acrobot*, *LunarLander* and *Hopper* we also determined the true optimal backup sensor configuration x^* using brute force. This was however not possible for larger environments.

For each environment, we sampled 10 problem instances. Note that using all backup sensors is not a valid solution regarding the problem definition in formula (2) since the costs would exceed the maximally allowed costs $\sum_i c_i > C$. All results are reported in *Table 2*.

The results show that *SensorOpt* found the optimal sensor redundancy configuration in the first four environments for which we were computationally able to determine the optimum using brute force.

6 CONCLUSION

Sensor dropouts present a major challenge when deploying reinforcement learning (RL) policies in real-world environments. A common solution to this problem is the use of backup sensors, though this approach introduces additional costs. In this paper, we tackled the problem of optimizing backup sensor configurations to maximize expected return while ensuring the total cost of added backup sensors remains below a specified threshold, C.

Our method involved using a second-order approximation of the expected return, $\mathbb{E}_{d,\pi,x}[R] \approx -x^T Q x + \hat{R}(d)$, for any given backup sensor configuration $x \in \mathbb{B}^n$. We incorporated a penalty for configurations that exceeded the maximum allowable cost, C, and optimized the resulting QUBO matrices Q using the Tabu Search algorithm.

We evaluated our approach across eight OpenAI Gym environments, as well as a custom Unity-based robotic scenario, *RobotArmGrasping*. The results demonstrated that our quadratic approximation was sufficiently accurate to ensure that the optimal configuration derived from the approximation closely matched the true optimal sensor configuration in practice.

REFERENCES

Beeler, C., Li, X., Bellinger, C., Crowley, M., Fraser, M., and Tamblyn, I. (2021). Dynamic programming with incomplete information to overcome navigational uncertainty in a nautical environment. *arXiv preprint arXiv:2112.14657*.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym. *arXiv preprint arXiv:1606.01540*.

Choi, V. (2010). Adiabatic quantum algorithms for the NP-complete maximum-weight independent set, exact cover and 3SAT problems.

Choi, V. (2011). Different adiabatic quantum optimization algorithms for the NP-complete exact cover and 3SAT problems.

Degrave, J., Felici, F., Buchli, J., Neunert, M., Tracey, B., Carpanese, F., Ewalds, T., Hafner, R., Abdolmaleki, A., de Las Casas, D., et al. (2022). Magnetic con-

trol of tokamak plasmas through deep reinforcement learning. *Nature*, 602(7897):414–419.

Dulac-Arnold, G., Mankowitz, D., and Hester, T. (2019). Challenges of real-world reinforcement learning. *arXiv preprint arXiv:1904.12901*.

Farhi, E. and Harrow, A. W. (2016). Quantum supremacy through the quantum approximate optimization algorithm. *arXiv preprint arXiv:1602.07674*.

Glover, F., Kochenberger, G., and Du, Y. (2019). Quantum bridge analytics I: A tutorial on formulating and using QUBO models.

Gu, S., Yang, L., Du, Y., Chen, G., Walter, F., Wang, J., Yang, Y., and Knoll, A. (2022). A review of safe reinforcement learning: Methods, theory and applications. *arXiv preprint arXiv:2205.10330*.

Koseoglu, M. and Ozcelikkale, A. (2020). How to miss data? reinforcement learning for environments with high observation cost. In *ICML Workshop on the Art of Learning with Missing Values (Artemis)*.

Liang, Y., Sun, Y., Zheng, R., and Huang, F. (2022). Efficient adversarial training without attacking: Worst-case-aware robust reinforcement learning. *Advances in Neural Information Processing Systems*, 35:22547–22561.

Lodewijks, B. (2020). Mapping NP-hard and NP-complete optimisation problems to quadratic unconstrained binary optimisation problems.

Lucas, A. (2014). Ising formulations of many NP problems.

Lütjens, B., Everett, M., and How, J. P. (2020). Certified adversarial robustness for deep reinforcement learning. In *conference on Robot Learning*, pages 1328–1337. PMLR.

Mandlekar, A., Zhu, Y., Garg, A., Fei-Fei, L., and Savarese, S. (2017). Adversarially robust policy learning: Active construction of physically-plausible perturbations. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3932–3939. IEEE.

Martello, S. and Toth, P. (1990). *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons, Inc.

Moos, J., Hansel, K., Abdulsamad, H., Stark, S., Clever, D., and Peters, J. (2022). Robust reinforcement learning: A review of foundations and recent advances. *Machine Learning and Knowledge Extraction*, 4(1):276–315.

Morita, S. and Nishimori, H. (2008). Mathematical foundation of quantum annealing. *Journal of Mathematical Physics*, 49(12).

Nam, H. A., Fleming, S., and Brunskill, E. (2021). Reinforcement learning with state observation costs in action-contingent noiselessly observable markov decision processes. *Advances in Neural Information Processing Systems*, 34:15650–15666.

Nüßlein, J., Roch, C., Gabor, T., Stein, J., Linnhoff-Popien, C., and Feld, S. (2023a). Black box optimization using qubo and the cross entropy method. In *International Conference on Computational Science*, pages 48–55. Springer.

Nüßlein, J., Zielinski, S., Gabor, T., Linnhoff-Popien, C., and Feld, S. (2023b). Solving (max) 3-sat via quadratic unconstrained binary optimization. In *International Conference on Computational Science*, pages 34–47. Springer.

Pattanaik, A., Tang, Z., Liu, S., Bommannan, G., and Chowdhary, G. (2017). Robust deep reinforcement learning with adversarial attacks. *arXiv preprint arXiv:1712.03632*.

Pinto, L., Davidson, J., Sukthankar, R., and Gupta, A. (2017). Robust adversarial reinforcement learning. In *International Conference on Machine Learning*, pages 2817–2826. PMLR.

Quintero, R. A. and Zuluaga, L. F. (2021). Characterizing and benchmarking qubo reformulations of the knapsack problem. Technical report, Technical Report. Department of Industrial and Systems Engineering, Lehigh . . . .

Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N. (2021). Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8.

Salkin, H. M. and De Kluyver, C. A. (1975). The knapsack problem: a survey. *Naval Research Logistics Quarterly*, 22(1):127–144.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. (2018). A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Valdenegro-Toro, M. and Mori, D. S. (2022). A deeper look into aleatoric and epistemic uncertainty disentanglement. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1508–1516. IEEE.