MDPax: GPU-accelerated MDP solvers in Python with JAX

Joseph Farrington [1¶], **Wai Keong Wong** [1,2,3], **Kezhi Li** [1], **and Martin Utley** [4]

1 Institute of Health Informatics, University College London, United Kingdom **2** NIHR University College London Hospitals Biomedical Research Centre, United Kingdom **3** Cambridge University Hospitals NHS Foundation Trust, United Kingdom **4** Clinical Operational Research Unit, University College London, United Kingdom ¶ Corresponding author

DOI: 10.21105/joss.08897

Software

- Review
- Repository
- Archive

Editor: Patrick Diehl

Reviewers:

- @arrrunn
- @thchang

Submitted: 01 August 2025
Published: 31 October 2025

Summary

MDPax is a Python library for solving large-scale Markov decision processes (MDPs), leveraging JAX's (Bradbury et al., 2022) support for vectorization, parallelization, and just-in-time (JIT) compilation on graphics processing units (GPUs). It includes GPU-accelerated implementations of standard algorithms including value iteration and policy iteration (Sutton & Barto, 2018).

MDPs describe sequential decision-making problems in which, at each timestep, an agent observes the current state of its environment, selects an action, transitions to a new state by taking the selected action, and receives a reward. The goal is to find a policy (a mapping from observed states to actions) that maximizes the expected long-term reward, accounting for both the immediate and future consequences of actions. MDPs have been used to model a wide range of problems, including medical treatment planning (Schaefer et al., 2004), traffic light control (Haijema et al., 2017), financial portfolio management (Bäuerle & Rieder, 2011), and conservation policy (Nicol et al., 2010).

Exact solution methods based on dynamic programming scale poorly due to the curse of dimensionality (Bellman, 1957): the number of states can grow exponentially with problem parameters, quickly making computation very challenging. Deep reinforcement learning has proven to be very effective at finding approximate solutions for large problems (Arulkumaran et al., 2017), but exact solutions remain valuable both in their own right and for benchmarking approximate methods.

The exact algorithms are well suited to parallel execution, and modern GPUs have thousands of processing cores over which updates can be effectively distributed. By building on JAX, MDPax makes it easy to take advantage of this hardware through a high-level Python API, enabling researchers and practitioners to solve problems with millions of states without needing expertise in GPU programming.

Statement of need

MDPax was originally developed to support our work on perishable inventory management. Solving these problems exactly requires accounting not just for total inventory levels, but also for the age profile of the stock. As a result, the state space grows exponentially with the product's maximum useful life, making realistic problem instances extremely large. Although many MDP solvers exist, exact methods are widely considered impractical or infeasible for realistically sized perishable inventory problems (Abouee-Mehrizi et al., 2025; De Moor et al., 2022; Hendrix et al., 2019; Nahmias, 1982).

Implementations of exact solution methods face two main challenges when applied to large MDPs:

- Memory requirements: The full transition matrix describing the dynamics of the problem grows quadratically with the number of states and linearly with the number of actions, making it infeasible to store and process for large problems.
- Computational complexity: The core operations involve nested loops over states, actions, and successor states, which become prohibitively expensive as the state space grows.

Most existing MDP libraries run exclusively on CPUs. MDPtoolbox (Chadès et al., 2014) provides exact solution algorithms across Python (Cordwell, 2015), MATLAB (Cros, 2015), and R (Chadès et al., 2017), but offers no support for parallelism or GPU acceleration. Two more recent Python libraries, MDPSolver (Andersen & Andersen, 2025) and madupite (Gargiani et al., 2025), aim to improve performance on large problems by implementing their solvers in C++ with CPU-based parallelism. In addition, madupite provides a broader set of inexact policy iteration methods to support solving larger problems. For Julia, POMDPs.jl (Egorov et al., 2017) provides a flexible interface for specifying MDPs and supports a wide range of CPU-based solvers.

The benefits of GPU-acceleration for exact methods have been demonstrated in the literature (Jóhannsson, 2009; Ortega et al., 2019; Sargent & Stachurski, 2025), but remain uncommon in practice. We suggest this is due to the perceived complexity of GPU programming and the limited availability of researcher-friendly software that provides GPU-accelerated solvers. The VFI Toolkit for MATLAB (Kirkby, 2017) supports GPU-acceleration but requires users to provide the full transition matrix, which becomes infeasible for large problems due to memory constraints.

MDPax addresses the memory challenge by requiring users to provide a deterministic transition function instead of the full transition matrix (similar to the approach used in POMDPs.jl). This function maps a state, action, and random event to the resulting next state and reward, and is used to dynamically compute the next state and reward on demand. MDPax uses JAX to exploit the massive parallel processing capabilities of modern GPUs, significantly reducing the runtime required for solving large MDPs by calculating value updates for batches of states in parallel.

MDPax has been developed to solve large MDPs with millions of states. For small to medium-sized MDPs, MDPax may be slower than existing CPU-based packages due to the overheads introduced by the use of JAX and GPUs, including JIT compilation and data transfer between the host and GPU(s). For large problems, these overheads are outweighed by substantial performance gains.

An early version of MDPax was used in our work to solve large instances of three perishable inventory problems that had previously been described as infeasible or impractical to solve exactly (Farrington et al., 2025). In one case, the original study reported that value iteration using a CPU-based MATLAB implementation failed to converge within a week on an MDP with over 16 million states (Hendrix et al., 2019). Using MDPax, the same algorithm consistently converged in under 3.5 hours on a consumer-grade GPU (based on 10 runs using an Nvidia GeForce RTX 3060 GPU, Python 3.12.4 and JAX 0.5.0). The runtime can be further reduced without any code changes using multiple data-centre-grade GPUs (Farrington et al., 2025).

Features and design

MDPax is structured around two core classes: the Problem and the Solver.

The Problem class represents an MDP and is intended to be subclassed by users. To define a custom problem, users implement methods that specify the sets of states and actions, random events and their probabilities, and a deterministic transition function that maps a (state, action, random event) triple to the next state and corresponding reward. MDPax includes four example Problems: a forest management problem adapted from pymdptoolbox (Cordwell, 2015) and three perishable inventory problems from the literature (Abouee-Mehrizi et al., 2025; De Moor et al., 2022; Hendrix et al., 2019).

The Solver class defines a common framework for implementing dynamic programming methods to solve MDPs. MDPax currently includes implementations of three standard algorithms: value iteration, relative value iteration (to optimize the average reward), and policy iteration. It also provides a variant of value iteration for MDPs with periodic dynamics (e.g., when demand depends on the day of the week), and a semi-asynchronous version in which updated values for each batch of states are made available for subsequent batch updates within the same iteration on the same device.

For large problems, solving an MDP can still be time-consuming. MDPax therefore includes checkpointing functionality using Orbax (Gaffney et al., 2025), enabling users to save and restore the state of the Solver and resume optimization after an interruption.

Acknowledgements

JF was funded by UKRI training grant EP/S021612/1, the CDT in AI-enabled Healthcare Systems, and the Clinical and Research Informatics Unit at the NIHR University College London Hospitals Biomedical Research Centre.

References

Abouee-Mehrizi, H., Mirjalili, M., & Sarhangian, V. (2025). Platelet inventory management with approximate dynamic programming. *INFORMS Journal on Computing*. https://doi.org/10.1287/ijoc.2023.0245

Andersen, A. R., & Andersen, J. F. (2025). MDPSolver: An efficient solver for Markov decision processes. *Journal of Open Source Software*, *10*(109), 7544. https://doi.org/10.21105/joss.07544

Arulkumaran, K., Deisenroth, M. P., Brundage, M., & Bharath, A. A. (2017). Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, *34*(6), 26–38. https://doi.org/10.1109/MSP.2017.2743240

Bäuerle, N., & Rieder, U. (2011). *Markov decision processes with applications to finance*. Springer. ISBN: 978-3-642-18324-9

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

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., & Zhang, Q. (2022). JAX: Composable transformations of Python+NumPy programs. In *GitHub repository*. http://github.com/google/jax

Chadès, I., Chapron, G., Cros, M., Garcia, F., & Sabbadin, R. (2014). MDPtoolbox: A multi-platform toolbox to solve stochastic dynamic programming problems. *Ecography*, *37*(9), 916–920. https://doi.org/10.1111/ecog.00888

Chadès, I., Chapron, G., Cros, M., Garcia, F., & Sabbadin, R. (2017). MDPtoolbox: Markov decision processes toolbox. In *The Comprehensive R Archive Network (CRAN)*. https://doi.org/10.32614/cran.package.mdptoolbox

Cordwell, S. (2015). Pymdptoolbox. In *GitHub repository*. https://github.com/sawcordwell/pymdptoolbox

Cros, M.-J. (2015). Markov decision processes (MDP) toolbox. In *MATLAB*

Central File Exchange. https://uk.mathworks.com/matlabcentral/fileexchange/25786-markov-decision-processes-mdp-toolbox

De Moor, B. J., Gijsbrechts, J., & Boute, R. N. (2022). Reward shaping to improve the performance of deep reinforcement learning in perishable inventory management. *European Journal of Operational Research*, *301*(2), 535–545. https://doi.org/10.1016/j.ejor.2021.10.045

Egorov, M., Sunberg, Z. N., Balaban, E., Wheeler, T. A., Gupta, J. K., & Kochenderfer, M. J. (2017). POMDPs.jl: A framework for sequential decision making under uncertainty. *Journal of Machine Learning Research*, *18*(26), 1–5. http://jmlr.org/papers/v18/16-300.html

Farrington, J., Wong, W. K., Li, K., & Utley, M. (2025). Going faster to see further: Graphics processing unit-accelerated value iteration and simulation for perishable inventory control using JAX. *Annals of Operations Research*. https://doi.org/10.1007/s10479-025-06551-6

Gaffney, C., Li, D., Zhang, J., Sang, R., Jain, A., & Hu, H. (2025). Orbax. In *GitHub repository*. https://github.com/google/orbax

Gargiani, M., Pawlowsky, P., Sieber, R., Hapla, V., & Lygeros, J. (2025). Madupite: A high-performance distributed solver for large-scale Markov decision processes. *Journal of Open Source Software*, *10*(108), 7411. https://doi.org/10.21105/joss.07411

Haijema, R., Hendrix, E. M. T., & Wal, J. van der. (2017). Dynamic control of traffic lights. In R. J. Boucherie & N. M. van Dijk (Eds.), *Markov Decision Processes in Practice* (pp. 371–386). Springer. https://doi.org/10.1007/978-3-319-47766-4_13

Hendrix, E. M., Ortega, G., Haijema, R., Buisman, M. E., & García, I. (2019). On computing optimal policies in perishable inventory control using value iteration. *Computational and Mathematical Methods*, *1*(4), e1027. https://doi.org/10.1002/cmm4.1027

Jóhannsson, A. P. (2009). *GPU-based Markov decision process solver* [M.Sc. dissertation, Reykjavík University]. https://en.ru.is/media/skjol-td/MSThesis_ArsaellThorJohannsson.pdf

Kirkby, R. (2017). A toolkit for value function iteration. *Computational Economics*, *49*(1). https://doi.org/10.1007/s10614-015-9544-1

Nahmias, S. (1982). Perishable inventory theory: A review. *Operations Research*, *30*(4), 680–708. https://doi.org/10.1287/opre.30.4.680

Nicol, S. C., Chadès, I., Linke, S., & Possingham, H. P. (2010). Conservation decision-making in large state spaces. *Ecological Modelling*, *221*(21), 2531–2536. https://doi.org/10.1016/j.ecolmodel.2010.02.009

Ortega, G., Hendrix, E. M., & García, I. (2019). A CUDA approach to compute perishable inventory control policies using value iteration. *The Journal of Supercomputing*, *75*(3), 1580–1593. https://doi.org/10.1007/s11227-018-2692-z

Sargent, T. J., & Stachurski, J. (2025). Quantitative economics with JAX. In *QuantEcon*. https://jax.quantecon.org/

Schaefer, A. J., Bailey, M. D., Shechter, S. M., & Roberts, M. S. (2004). Modeling medical treatment using Markov decision processes. In M. L. Brandeau, F. Sainfort, & W. P. Pierskalla (Eds.), *Operations Research and Health Care: A Handbook of Methods and Applications* (pp. 593–612). Springer. https://doi.org/10.1007/1-4020-8066-2_23

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction* (2nd ed.). The MIT Press. ISBN: 978-0-262-03924-6