The Journal of Open Source Software

madupite: A High-Performance Distributed Solver for Large-Scale Markov Decision Processes

Matilde Gargiani [1*¶], **Philip Pawlowsky** [1*¶], **Robin Sieber** [1*¶], **Václav Hapla** [2,3], **and John Lygeros** [1¶]

1 Automatic Control Laboratory (IfA), ETH Zurich, 8092 Zurich, Switzerland **2** Department of Earth and Planetary Sciences, ETH Zurich, 8092 Zurich, Switzerland **3** Department of Applied Mathematics, FEECS at VSB-TU Ostrava, Czechia ¶ Corresponding author * These authors contributed equally.

DOI: 10.21105/joss.07411

Software

- Review
- Repository
- Archive

Editor: Tristan Miller

Reviewers:

- @lukeolson
- @victorapm
- @bhttchr6

Submitted: 19 September 2024
Published: 02 April 2025

License
Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License (CC BY 4.0).

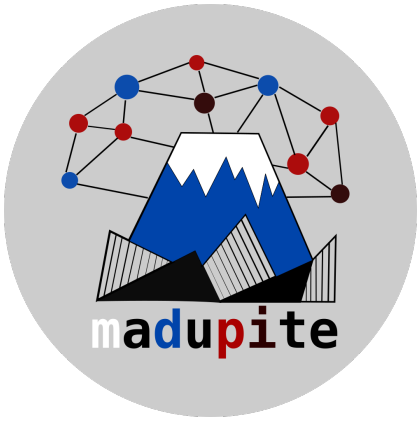

Figure 1: Logo of `madupite`.

Summary

In this paper we describe `madupite`, a distributed high-performance solver for Markov Decision Processes (MDPs). MDPs are a powerful mathematical tool to model a variety of problems arising in different fields (Feinberg & Shwartz, 2002; White, 1985), from finance (Bäuerle & Rieder, 2011) to epidemiology (Steimle & Denton, 2017) and traffic control (Xu et al., 2016). In general terms, MDPs are used to mathematically characterize dynamical systems whose state is evolving in time as a consequence of actions that we play on the system and disturbances that are acting on it. The goal is generally to select actions in order to minimize in expectation a certain cumulative discounted metric over time, e.g., deviations from a reference state and/or costs incurred by selecting certain actions in specific states of the system (Bellman, 1957; Bertsekas, 2007).

MDPs arising from real-world applications tend to be extremely high-dimensional, and in some cases, the number of states in the system grows exponentially with the number of certain parameters (Powell, 2011). This phenomenon is known as the *curse-of-dimensionality* and it is generally tackled in reinforcement learning with the deployment of function approximations (Sutton & Barto, 2018). The deployment of the latter leads to a smaller size optimization problem since, instead of optimizing for the number of states, there is only need to optimize for the number of parameters deployed in the function approximation, which is generally much smaller than the original state space size. This comes at the price of introducing sub-optimality with respect to the original solution.

Statement of need

Modern high-performance clusters and supercomputers offer the possibility of simulating, storing, and solving large-scale MDPs. To exploit modern computational resources, high-performance software packages that can efficiently distribute computation as well as scale adequately are needed. Even though there are a number of toolboxes to solve MDPs, such as `pymdptoolbox` (Chadès et al., 2014) and the recent `mdpsolver` (Reenberg Andersen & Fink Andersen, 2024), to the best of our knowledge there is no existing solver that combines high-performance distributed computing with the possibility of selecting a solution method that is tailored to the application at hand. `pymdptoolbox` is coded in plain Python, which is limited in scalability due to the lack of parallel support. In addition, it does not support parallel and distributed computing. `mdpsolver` is instead written in C++, supports parallel computing, and comes with a user-friendly Python API. Even so, `mdpsolver` is not fully featured. The solution methods available are limited to modified policy iteration, a dynamic programming (DP) method that has shown poor performance for a significant class of problems of practical interest (Gargiani et al., 2023, 2024). In addition, `mdpsolver` makes certain implementation choices that limit its applicability; e.g., matrices with values and indices being stored in nested `std::vector` independently of their sparsity degree and thus precluding the use of available optimized linear algebra routines.

Our solver aims to enable the solution of large-scale MDPs with more than a million states through efficient utilization of modern computational resources, while abstracting away the complexity of distributed computing to provide a user-friendly experience. `madupite` is a high-performance distributed solver that is capable of efficiently distributing the memory load and computation, and that comes with a wide range of choices for solution methods enabling the user to select the one that is best tailored to its specific application. Furthermore, madupite's core is in C++, but it is equipped with a user-friendly Python API to enable its deployment to a broader range of users. The combination of distributed computing, user-friendly Python API, and support for a wide range of methods in `madupite` will enable researchers and engineers to solve exactly gigantic scale MDPs which previously could only be tackled via function approximations.

Problem Setting and Solution Methods

`madupite` solves infinite-horizon discounted MDPs with finite state and action spaces. This problem class can be efficiently tackled with *inexact policy iteration methods* (iPI) (Gargiani et al., 2023, 2024). These methods are a variant of policy iteration (Bertsekas, 2007), where inexactness is introduced at the policy evaluation step for scalability reasons. iPI methods are general enough to embrace standard DP methods, such as value iteration and modified policy iteration, which are the main solution methods of `mdpsolver`. Interested readers should refer to Gargiani et al. (2024) for an in-depth description of the problem setting and a thorough mathematical analysis of iPI methods. The versatility of iPI methods makes them particularly suited to efficiently solve different large-scale instances, while their structure is also favorable for distributed implementations, which are needed to exploit high-performance computing clusters.

The great flexibility of `madupite` is that on the algorithmic side it relies on the possibility of customizing the iPI method deployed for the specific problem at hand by tuning the level of inexactness and selecting among a wide range of inner solvers for the approximate policy evaluation step (step 8 of Algorithm 3 in Gargiani et al. (2024)). It is indeed empirically and theoretically demonstrated that, depending on the specific structure of the problem, different inner solvers may enhance the convergence performance (Gargiani et al., 2023, 2024).

Implementation

The core of `madupite` is written in C++ and relies on PETSc (Portable, Extensible Toolkit for Scientific Computation) for the distributed implementation of iPI methods. PETSc (Balay et al., 1997, 2024a, 2024b) is an open-source high-performance C library and comes with a wide range of highly-optimized linear system solvers and memory-efficient sparse linear algebra data types and routines that can be used for a variety of problems, including DP. We rely on PETSc also as it natively adopts a distributed memory parallelism using the MPI standard to enable scalability beyond one CPU. MPI allows users to abstract the parallelism away from the underlying hardware and enables them to run the same code in parallel on their personal device or even on multiple nodes of a high-performance computing cluster. `madupite` itself is a fully-featured C++20 library and, by leveraging the `nanobind` binding library (Jakob, 2022), offers an equivalent API in Python that can be installed as a package using `pip`.

Finally, `madupite` allows the user to create an MDP by loading offline data collected from previously run experiments as well as from online simulations, offering the possibility of carrying out both the simulations and the solution in a completely parallel/distributed fashion. More details on how to use `madupite` and all of its functionalities can be found in its documentation and in the `examples` folder, where we provide an extensive selection of code examples on how to use this library in Python and C++.

Acknowledgements

This work was supported by the European Research Council under the Horizon 2020 Advanced under Grant 787845 (OCAL) and by the SNSF through NCCR Automation (Grant Number 180545).

References

Balay, S., Abhyankar, S., Adams, M. F., Benson, S., Brown, J., Brune, P., Buschelman, K., Constantinescu, E. M., Dalcin, L., Dener, A., Eijkhout, V., Faibussowitsch, J., Gropp, W. D., Hapla, V., Isaac, T., Jolivet, P., Karpeev, D., Kaushik, D., Knepley, M. G., … Zhang, J. (2024a). *PETSc Web page*. https://petsc.org/

Balay, S., Abhyankar, S., Adams, M. F., Benson, S., Brown, J., Brune, P., Buschelman, K., Constantinescu, E., Dalcin, L., Dener, A., Eijkhout, V., Faibussowitsch, J., Gropp, W. D., Hapla, V., Isaac, T., Jolivet, P., Karpeev, D., Kaushik, D., Knepley, M. G., … Zhang, J. (2024b). *PETSc/TAO users manual* (ANL-21/39 - Revision 3.21). Argonne National Laboratory. https://doi.org/10.2172/2205494

Balay, S., Gropp, W. D., McInnes, L. C., & Smith, B. F. (1997). Efficient management of parallelism in object oriented numerical software libraries. In E. Arge, A. M. Bruaset, & H. P. Langtangen (Eds.), *Modern software tools in scientific computing* (pp. 163–202). Birkhäuser Press. https://doi.org/10.1007/978-1-4612-1986-6_8

Bäuerle, N., & Rieder, U. (2011). *Markov decision processes with applications to finance*. Springer. ISBN: 3642183239

Bellman, R. (1957). *Dynamic programming* (1st ed.). Princeton University Press.

Bertsekas, D. P. (2007). *Dynamic programming and optimal control, vol. II* (3rd ed.). Athena Scientific. ISBN: 1886529302

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

Feinberg, E. A., & Shwartz, A. (Eds.). (2002). *Handbook of Markov decision processes: Methods and applications*. Kluwer International Series.

Gargiani, M., Liao-McPherson, D., Zanelli, A., & Lygeros, J. (2023). Inexact GMRES policy iteration for large-scale Markov decision processes. *IFAC-PapersOnLine*, *56*(2), 11249–11254. https://doi.org/10.1016/j.ifacol.2023.10.316

Gargiani, M., Sieber, R., Balta, E., Liao-McPherson, D., & Lygeros, J. (2024). Inexact policy iteration methods for large-scale Markov decision processes. *arXiv Preprint arXiv:2404.06136*. https://doi.org/10.48550/arXiv.2404.06136

Jakob, W. (2022). *Nanobind: Tiny and efficient C++/Python bindings*. https://github.com/wjakob/nanobind

Powell, W. B. (2011). *Approximate dynamic programming: Solving the curses of dimensionality* (2nd ed). Wiley. https://doi.org/10.1002/9781118029176

Reenberg Andersen, A., & Fink Andersen, J. (2024). *Areenberg/MDPSolver: MDPSolver v0.9.7* (Version v0.9.7). Zenodo. https://doi.org/10.5281/zenodo.11844058

Steimle, L. N., & Denton, B. T. (2017). *Markov decision processes for screening and treatment of chronic diseases*. 189–222. https://doi.org/10.1007/978-3-319-47766-4_6

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction* (Second). The MIT Press. ISBN: 978-0262039246

White, D. J. (1985). Real applications of Markov decision processes. *Interfaces*, *15*(6), 73–83. https://doi.org/10.1287/inte.15.6.73

Xu, Y., Xi, Y., Li, D., & Zhou, Z. (2016). Traffic signal control based on Markov decision process. *IFAC-PapersOnLine*, *49*(3), 67–72. https://doi.org/10.1016/j.ifacol.2016.07.012