Journal of Machine Learning Research 21 (2020) 1-5 Submitted 6/18; Revised 3/20; Published 5/20

AI-Toolbox: A C++ library for Reinforcement Learning and Planning (with Python Bindings)

Eugenio Bargiacchi SVALORZEN@GMAIL.COM
Department of Computer Science, Vrije Universiteit Brussel
Brussels, Belgium

Diederik M. Roijers DIEDERIK.YAMAMOTO-ROIJERS@HU.NL
Microsystems Technology, HU University of Applied Sciences
Utrecht, the Netherlands &
Department of Computer Science, Vrije Universiteit
Brussel, Belgium

Ann Nowé ANN.NOWE@AI.VUB.AC.BE
Department of Computer Science, Vrije Universiteit Brussel
Brussels, Belgium

Editor: Balázs Kégl

Abstract

This paper describes AI-Toolbox, a C++ software library that contains reinforcement learning and planning algorithms, and supports both single and multi agent problems, as well as partial observability. It is designed for simplicity and clarity, and contains extensive documentation of its API and code. It supports Python to enable users not comfortable with C++ to take advantage of the library's speed and functionality. AI-Toolbox is free software, and is hosted online at https://github.com/Svalorzen/AI-Toolbox.

Keywords: MDP, POMDP, multiagent, reinforcement learning, software, open-source

1. Introduction

Decision theory is the branch of artificial intelligence that deals with action selection. Its applications span from robotics, to information gathering, to video games. The field tackles the problem of finding the optimal action-selection policy to perform in a given environment, and is roughly divided into planning and reinforcement learning. In the former case a model of the environment is known in advance (Boutilier et al., 1995), while in the latter case the agent must learn from its interactions with the world (Sutton and Barto, 1998).

When new decision-theoretic algorithms are proposed, they need to be empirically compared to existing methods to prove their worth. However, this comparative work often requires re-implementing existing methods, which takes considerable effort. Furthermore, each new line of code increases the risk of bugs, and thus makes comparisons across implementations harder, if not impossible. It also diverts resources from research towards software maintenance. There is thus a great need in the research community for reusable, proven software that implements existing planning and reinforcement learning algorithms.

In this article, we present AI-Toolbox, a C++ framework containing implementations of reinforcement learning and planning algorithms for Markov decision process (MDPs) and its partially observable (POMDP) and multi-agent (MMDP) extensions.

2. Related Work

Several libraries with partially overlapping functionality to AI-Toolbox exist.

MADP (Oliehoek et al., 2017) is one of the best known toolboxes. It is written in C++, and geared towards multi-agent partially observable models, and offers a wide variety of algorithms. MADP is object oriented, which leads to a large hierarchy of classes, while AI-Toolbox has a more compact design. In addition, MADP does not have Python bindings.

BURLAP Is an extensive JAVA library for reinforcement learning and planning. It includes code to visualize environments and can be used with the ROS framework (Quigley et al., 2009). It is mostly focused on fully-observable environments, while AI-Toolbox contains multiple state-of-the-art POMDP algorithms.

pomdp-solve is a C library by Anthony Cassandra, which contains relatively old POMDP algorithms (the most recent was published in 2004). It additionally requires the commercially licensed CPLEX linear programming solver.

MDPToolbox (Chades et al., 2014) is a MATLAB toolbox for single agent MDP algorithms. In contrast, AI-Toolbox also supports bandits, POMDPs and MMDPs algorithms.

Other toolboxes exist, such as PyMDPToolbox, JuliaPOMDP (Egorov et al., 2017), ZMDP and APPL, but they are much smaller in scope than AI-Toolbox.

3. Description

AI-Toolbox is written in C++17, taking advantage of all features of the language, and is built following modern standard practices, i.e. unit tests, continuous integration, separate concerns, etc. The goals of this framework are, in descending order of importance: usability and documentation, ease of modification, clarity and performance.

3.1. Features

Documentation. All public functions, classes and interfaces in the library are fully documented, and the implementation code is extensively commented (nearly 45% of the lines of code in the library are comments). The library's repository contains tutorials on how to use the library and examples, both in C++ and Python. The included unit tests can also be used as example implementations to understand how to use the library.

Architecture. In AI-Toolbox classes are organized in a template-based, mostly-flat hierarchy, which limits the amount of code that a user has to read in order to understand how a method works. Furthermore, extending the library only requires respecting few template interfaces. The only exception is for policy classes, which do not use templates but inheritance, as policies are designed to be composable. Thus, it may be necessary for custom policies to inherit from provided interfaces to leverage virtual dispatching.

The library is designed to simplify customization. We believe it is more important that users find easy to tune the code for their specific settings, rather than to provide a large number of options natively. To reach this goal, the API only offers parameters that do not

increase the branching factor of the code. For example, an ϵ parameter to tune an epsilon-greedy policy is allowed, while a parameter that selectively enables an optimization is not. This choice allows users to customize the library for their particular usage more easily.

Classes are organized in separate folders and files, depending on their functions: algorithms reside in an `Algorithms` folder, policies in a `Policies` folder, and so on. Utility functions reside in `Utils` files and folders. Internal classes are hidden where possible in `Impl` namespaces and folders, so that users can safely ignore them.

`AI-Toolbox` is one of the largest libraries available in the field, currently containing over 40 learning and planning algorithms, 5 different policy types, and a large number of independent helper functions and classes. All algorithms are listed in the supplement.

Python Bindings. The framework offers Python bindings, to allow researchers who are not familiar with the C++ language to still benefit from its performance. The Python interface of the library is extremely close to its C++ counterpart, which results in a near line by line match between user code in Python and C++. The project's examples help to show the similarities, as they are implemented in both languages.

Performance. The C++ language allows for high computational performance compared to algorithms implemented purely in Python or MATLAB. `AI-Toolbox` has support for sparse matrices, yielding performance improvements on models with sparse transition, reward and observation tables. As an example of computational performance, our implementation of `GapMin` (Poupart et al., 2011) takes only `0.8s` to reach results for a POMDP that in the original MATLAB implementation took `15s`—an improvement of nearly `19x`.

Community. The library has already been used in multiple projects, demos and papers, and has thus shown to be useful in practical applications and research. The library is hosted on GitHub, where it has over 420 stars and has been forked 60 times by individuals and organizations. GitHub supports both an issue tracker, as well as pull request handling for contributing to the main codebase.

3.2. Considerations

Language Concerns. Nowadays, C++ is not so common in the research community anymore, due to its complexity. To mitigate this, we kept non-utility code as simple and accessible as possible in C++, and provide extensive documentation which does not require reading the code directly. In addition, Python bindings are provided, so that C++ does not need to be used at all, while still providing fast execution.

Non-Linear Function Approximation. Recently, there has been an incredible development in the field of AI regarding non-linear function approximation, e.g. deep learning. As of present, `AI-Toolbox` only supports linear function approximation in factored domains. However, more developments in this direction are currently planned.

Backwards Compatibility. As a software project evolves, there are often concerns of portability with newer versions. As additional methods are added to the library, sometimes we must make a choice on whether to keep old code as-is, or to refactor the library to improve the overall interface. Since one of the main goals of the library is clarity, we generally opt for refactoring, which may require users to update their code if they want to keep using the newest version. However, the documentation is always kept up to date, and all older versions are always accessible via version control.

4. Technical Details

The framework is easily built with CMake, a cross-platform tool that supports many operating systems. It additionally requires the `Boost` library, the `Eigen` matrix library and, for POMDPs and MMDPs, the `lp-solve` linear programming library. All dependencies are free and open source, to avoid having to buy expensive software licenses.

`AI-Toolbox` is released through the GPL 3.0 license, making it Free and Libre Open Source Software. It is versioned using the Git version control system, and it is hosted online on GitHub. It has been developed on an Ubuntu machine, but has also been successfully built on both Mac and Windows.

The library contains more than 70 unit tests, each of which is composed of one or more tests. Adding and extending tests is easy, which allows to guarantee correctness even as the library is further developed. All tests are run using continuous integration, which alerts if any given commit fails to compile or breaks existing functionality.

5. Example

In this section we go over a brief example on how to use the incremental pruning POMDP algorithm on the tiger problem (Cassandra et al., 1994) using `AI-Toolbox`. This example is in C++, but the (nearly identical) equivalent example in Python is provided in the repository.

```
// The model can be any custom class that respects a 10-method interface.
auto model = makeTigerProblem();
unsigned horizon = 10; // The horizon of the solution.

// The 0.0 is the convergence parameter. It gives a way to stop the
// computation if the policy has converged before the horizon.
AIToolbox::POMDP::IncrementalPruning solver(horizon, 0.0);

// Solve the model and obtain the optimal value function.
auto [bound, valueFunction] = solver(model);

// We create a policy from the solution to compute the agent's actions.
// The parameters are the size of the model (SxAxO), and the value function.
AIToolbox::POMDP::Policy policy(2, 3, 2, valueFunction);

// We begin a simulation with a uniform belief. We sample from the belief
// in order to get a "real" state for the world, since this code has to
// both emulate the environment and control the agent.
AIToolbox::POMDP::Belief b(2); b << 0.5, 0.5;
auto s = AIToolbox::sampleProbability(b.size(), b, rand);

// We sample the first action. The id is to follow the policy tree later.
auto [a, id] = policy.sampleAction(b, horizon);

double totalReward = 0.0;// As an example, we store the overall reward.
for (int t = horizon - 1; t >= 0; --t) {
    // We advance the world one step.
    auto [s1, o, r] = model.sampleSOR(s, a);
    totalReward += r;

    // We select our next action from the observation we got.
    std::tie(a, id) = policy.sampleAction(id, o, t);

    s = s1; // Finally we update the world for the next timestep.
}
```

Acknowledgments

This research was supported by funding from the Fonds voor Wetenschappelijk Onderzoek (FWO) through the grant of Eugenio Bargiacchi (#1SA2820N), and by funding from the Flemish Government under the âĂIJOnderzoeksprogramma ArtificiÃńle Intelligentie (AI) VlaanderenâĂİ for Diederik M. Roijers.

References

C. Boutilier, T. Dean, and S. Hanks. Planning under uncertainty: Structural assumptions and computational leverage. In *Proceedings of the Second European Workshop on Planning*, pages 157–171, 1995.

A.R. Cassandra, L.P. Kaelbling, and M.L. Littman. Acting optimally in partially observable stochastic domains. In *AAAI Conference on Artificial Intelligence*, Seattle, WA, 1994.

I. Chades, G. Chapron, MJ. Cros, F. Garcia, and R. Sabbadin. MDPtoolbox: a multi-platform toolbox to solve stochastic dynamic programming problems. In *Ecography*, volume 37, pages 916–920, 2014.

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

F. Oliehoek, M. T. J. Spaan, B. Terwijn, P. Robbel, and J. V. Messias. The MADP toolbox: An open-source library for planning and learning in (multi-)agent systems. *Journal of Machine Learning Research*, 18(1):3112–3116, 2017.

P. Poupart, K. Kim, and D. Kim. Closing the gap: Improved bounds on optimal POMDP solutions. In *International Conference on Automated Planning and Scheduling*, 2011.

M. Quigley, K. Conley, B.P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A.Y. Ng. Ros: an open-source robot operating system. In *International Conference on Robotics and Automation Workshop on Open Source Software*, 2009.

R.S. Sutton and A.G. Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

Appendix A. Implemented Algorithms

In this section we list all the algorithms currently implemented by `AI-Toolbox` at the time of writing. The algorithms are divided by the type of model where they are applicable. Note that this list does not include the large number of utility function that `AI-Toolbox` provides.

In the following lists two types of structures are listed: value-based algorithms and policies (exploration strategies and actor-critic). The library considers as value-based algorithms all classes and functions that relate to planning and learning through value functions. The library considers policies all classes and functions that use the learned values in order to produce a policy, or that learn a policy directly. This division is approximate as each algorithm is unique, but it results in a rough split where algorithms in the same group tend to have similar API.

Note that different classes of models may list the same type of policy, as policies that apply to different models must have different API from each other.

The library offers detailed documentation and brief descriptive summary on each method in the online documentation.

A.1. Bandit and Normal Games

Policies:

- Q-Greedy policy (Russo et al., 2017)
- Softmax policy
- Thompson sampling (normal distribution) (Thompson, 1933; Korda et al., 2013)
- Linear reward penalty (Unsal, 1993)
- Exploring selfish reinforcement learning (ESRL) (Verbeeck et al., 2007)

A.2. Single Agent MDP and Stochastic Games

Value-based algorithms:

- Dyna-Q (Sutton, 1990)
- Dyna2 (Silver et al., 2008)
- Expected SARSA (Van Seijen et al., 2009)
- Hysteretic Q-Learning (Matignon et al., 2007)
- Importance Sampling (Precup, 2000)
- Linear programming (Schweitzer and Seidmann, 1985)
- Monte carlo tree search (MCTS) (Coulom, 2006)
- Policy evaluation (Howard, 1960)

- Policy iteration (Howard, 1960)
- Prioritized sweeping (Moore and Atkeson, 1993)
- Q-learning (Watkins, 1989)
- Q(λ) (Harutyunyan et al., 2016)
- R-Learning (Schwartz, 1993)
- SARSA (Rummery and Niranjan, 1994)
- SARSA(λ) (Rummery and Niranjan, 1994)
- Retrace(λ) (Munos et al., 2016)
- Tree backup(λ) (Precup, 2000)
- Value iteration (Bellman, 1957)

Policies:

- Policy from value function
- Q-greedy policy
- Epsilon-greedy policy
- Softmax policy
- PGA-APP (Zhang and Lesser, 2010)
- Win or learn fast policy iteration (WoLF) (Bowling and Veloso, 2001)

A.3. Single Agent POMDP

Value-based algorithms:

- Augmented MDP (AMDP) (Roy and Thrun, 2000; Thrun, 2002)
- Blind strategies (Hauskrecht, 1997)
- Fast informed bound (Hauskrecht, 2000)
- GapMin (Poupart et al., 2011)
- Incremental pruning (Cassandra et al., 1997)
- Linear support (Cheng, 1988)
- PERSEUS (Spaan and Vlassis, 2005)
- Point based value iteration (PBVI) (Pineau et al., 2003)

- POMCP with UCB1 (Silver and Veness, 2010)
- QMDP (Littman et al., 1995; Thrun, 2002)
- rPOMCP (Bargiacchi, 2016)
- Real-time belief state search (RTBSS) (Paquet et al., 2005)
- SARSOP (Kurniawati et al., 2008)
- Witness (Kaelbling et al., 1998)

Policies:

- Policy from value function

A.4. Factored/Joint Multi-Agent Bandits and Coordination Graphs

Value-based algorithms:

- Variable elimination (Rosenthal, 1977; Dechter, 1998; Guestrin et al., 2002)
- Multi-agent Thompson sampling (MATS) (Verstraeten et al., 2020)
- Multi-agent upper confidence exploration (MAUCE) (Bargiacchi et al., 2018)
- Multi-objective variable elimination (Roijers et al., 2013, 2015)
- Learning with linear rewards (LLR) (Gai et al., 2012)
- Upper confidence variable elimination (UCVE) (Bargiacchi et al., 2018)

Policies:

- Q-greedy policy

A.5. Factored/Joint Multi-Agent MDP and Stochastic Games

Value-based algorithms:

- Cooperative Prioritized Sweeping (Bargiacchi et al., 2020)
- Factored LP (Guestrin et al., 2002)
- Multi-agent linear programming (Guestrin et al., 2002)
- Joint action learners (Claus and Boutilier, 1998)
- Sparse cooperative Q-learning (Kok and Vlassis, 2004)

Policies:

- Naive single action policy
- Epsilon-greedy policy
- Q-greedy policy

References

E. Bargiacchi. Dynamic resource allocation for multi-camera systems. Master's thesis, University of Amsterdam, 2016.

E. Bargiacchi, T. Verstraeten, D.M. Roijers, A. Nowé, and H. van Hasselt. Learning to coordinate with coordination graphs in repeated single-stage multi-agent decision problems. In *International Conference on Machine Learning*, volume 80, pages 482–490. PMLR, 10–15 Jul 2018.

E. Bargiacchi, T. Verstraeten, D. M. Roijers, and A. Nowé. Model-based multi-agent reinforcement learning with cooperative prioritized sweeping, 2020. arXiv:2001.07527.

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

M. Bowling and M. Veloso. Rational and convergent learning in stochastic games. In *International Joint Conference on Artificial Intelligence*, volume 17, pages 1021–1026. Lawrence Erlbaum Associates Ltd, 2001.

A. Cassandra, M.L. Littman, and N.L. Zhang. Incremental pruning: A simple, fast, exact method for partially observable Markov decision processes. In *Conference on Uncertainty in Artificial Intelligence*, pages 54–61. Morgan Kaufmann Publishers Inc., 1997.

H. Cheng. *Algorithms for partially observable Markov decision processes*. PhD thesis, University of British Columbia, 1988.

C. Claus and C. Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. 1998:746–752, 1998.

R. Coulom. Efficient selectivity and backup operators in Monte-Carlo tree search. In *International Conference on Computers and Games*, pages 72–83. Springer, 2006.

R. Dechter. Bucket elimination: A unifying framework for probabilistic inference. In *Learning in graphical models*, pages 75–104. Springer, 1998.

Y. Gai, B. Krishnamachari, and R. Jain. Combinatorial network optimization with unknown variables: Multi-armed bandits with linear rewards and individual observations. *IEEE/ACM Transactions on Networking (TON)*, 20(5):1466–1478, 2012.

C. Guestrin, D. Koller, and R. Parr. Multiagent planning with factored MDPs. In *Advances in neural information processing systems*, pages 1523–1530, 2002.

A. Harutyunyan, M.G. Bellemare, T. Stepleton, and R. Munos. Q(λ) with off-policy corrections. In *International Conference on Algorithmic Learning Theory*, pages 305–320. Springer, 2016.

M. Hauskrecht. Incremental methods for computing bounds in partially observable Markov decision processes. In *AAAI Conference on Artificial Intelligence*, pages 734–739. Citeseer, 1997.

M. Hauskrecht. Value-function approximations for partially observable Markov decision processes. *Journal of artificial intelligence research*, 13:33–94, 2000.

R.A. Howard. *Dynamic Programming and Markov Processes*. MIT Press, 1960.

L.P. Kaelbling, M.L. Littman, and A.R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998.

J.R. Kok and N. Vlassis. Sparse cooperative q-learning. In *International Conference on Machine Learning*, page 61. ACM, 2004.

N. Korda, E. Kaufmann, and R. Munos. Thompson sampling for 1-dimensional exponential family bandits. In *Advances in Neural Information Processing Systems*, pages 1448–1456, 2013.

H. Kurniawati, D. Hsu, and W. S. Lee. SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *In Proc. Robotics: Science and Systems*, 2008.

M.L. Littman, A.R. Cassandra, and L.P. Kaelbling. Learning policies for partially observable environments: Scaling up. In *Machine Learning Proceedings 1995*, pages 362–370. Elsevier, 1995.

L. Matignon, G.J. Laurent, and N. Le Fort-Piat. Hysteretic q-learning: an algorithm for decentralized reinforcement learning in cooperative multi-agent teams. In *International Conference on Intelligent Robots and Systems*, pages 64–69. IEEE, 2007.

A.W. Moore and C.G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine learning*, 13(1):103–130, 1993.

R. Munos, T. Stepleton, A. Harutyunyan, and M. Bellemare. Safe and efficient off-policy reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 1054–1062, 2016.

S. Paquet, L. Tobin, and B. Chaib-draa. Real-time decision making for large POMDPs. In *Conference of the Canadian Society for Computational Studies of Intelligence*, pages 450–455. Springer, 2005.

J. Pineau, G. Gordon, S. Thrun, et al. Point-based value iteration: An anytime algorithm for POMDPs. In *International Joint Conference on Artificial Intelligence*, volume 3, pages 1025–1032, 2003.

P. Poupart, K. Kim, and D. Kim. Closing the gap: Improved bounds on optimal POMDP solutions. In *International Conference on Automated Planning and Scheduling*, 2011.

D. Precup. Eligibility traces for off-policy policy evaluation. *Computer Science Department Faculty Publication Series*, page 80, 2000.

D.M. Roijers, S. Whiteson, and F.A. Oliehoek. Computing convex coverage sets for multi-objective coordination graphs. In *International Conference on Algorithmic Decision Theory*, pages 309–323, November 2013.

D.M. Roijers, S. Whiteson, and F.A. Oliehoek. Computing convex coverage sets for faster multi-objective coordination. *Journal of Artificial Intelligence Research*, 52:399–443, 2015.

A. Rosenthal. Nonserial dynamic programming is optimal. In *ACM Symposium on Theory of Computing*, pages 98–105. ACM, 1977.

N. Roy and S. Thrun. Coastal navigation with mobile robots. In *Advances in Neural Information Processing Systems*, pages 1043–1049, 2000.

G.A. Rummery and M. Niranjan. *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering, 1994.

D. Russo, B. Van Roy, A. Kazerouni, and I. Osband. A tutorial on Thompson sampling, 2017. arXiv:1707.02038.

A. Schwartz. A reinforcement learning method for maximizing undiscounted rewards. In *International Conference on Machine Learning*, volume 298, pages 298–305, 1993.

P.J. Schweitzer and A. Seidmann. Generalized polynomial approximations in Markovian decision processes. *Journal of mathematical analysis and applications*, 110(2):568–582, 1985.

D. Silver and J. Veness. Monte-Carlo planning in large POMDPs. In *Advances in neural information processing systems*, pages 2164–2172, 2010.

D. Silver, R.S. Sutton, and M. Müller. Sample-based learning and search with permanent and transient memories. In *International Conference on Machine Learning*, pages 968–975. ACM, 2008.

M.T.J. Spaan and N. Vlassis. Perseus: Randomized point-based value iteration for POMDPs. *Journal of artificial intelligence research*, 24:195–220, 2005.

R.S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Machine Learning Proceedings 1990*, pages 216–224. Elsevier, 1990.

W.R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933.

S. Thrun. Probabilistic robotics. *Communications of the ACM*, 45(3):52–57, 2002.

C. Unsal. *Self-organization in large populations of mobile robots*. PhD thesis, Virginia Tech, 1993.

H. Van Seijen, H. Van Hasselt, S. Whiteson, and M. Wiering. A theoretical and empirical analysis of expected sarsa. In *Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, pages 177–184. IEEE, 2009.

K. Verbeeck, A. Nowé, J. Parent, and K. Tuyls. Exploring selfish reinforcement learning in repeated games with stochastic rewards. *Autonomous Agents and Multi-Agent Systems*, 14(3):239–269, 2007.

T. Verstraeten, E. Bargiacchi, P. Libin, J. Helsen, D. M. Roijers, and A. Nowé. Multi-agent Thompson sampling for bandit applications with sparse neighbourhood structures. *Nature Scientific Reports*, 10(1):6728, 2020.

C. Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, 1989.

C. Zhang and V.R. Lesser. Multi-agent learning with policy prediction. In *AAAI Conference on Artificial Intelligence*, 2010.