Cache-Efficient Memory Representation of Markov Decision Processes

Leveraging modern computer memory architecture

Jaël Champagne Gareau
Éric Beaudry
Vladimir Makarenkov

Computer Science Department
Université du Québec à Montréal

30 May – 3 June 2022

UQÀM

Canadian AI 2022
30 May—3 June 2022, Toronto

Outline

Automated planning & scheduling

- **Automated planning & scheduling** is a branch of Artificial Intelligence.
- Its objective is to find **plans** allowing **agents** to reach **goals**.
- Some planning problems are **probabilistic** (i.e., there are **uncertainties**) :
  - endogenous uncertainties (i.e., due to the agent) ;
  - exogeneous uncertainties (i.e., due to the environment).
- **Markov Decision Processes** (MDPs) are often used to model these problems of decision-making under uncertainty.

Objective of this research

- Most interesting real-world MDP problems require a large number of state variables.
- Curse of dimensionality : Number of states is exponential in the number of state variables.
- The memory required to store the MDP modelization can be significant.
- Often we are limited in time to find the problem's solution.
- Therefore, we need to find ways to accelerate MDP computations.

Mathematical MDP representation

- There exists many variants of MDPs. The most common are :
  - **Finite-horizon MDP** ;
  - **Infinite-horizon Discounted MDP** ;
  - **Stochastic Shortest-Path MDP** (SSP-MDP).
- We focus on SSP-MDPs, since they are more general.

Stochastic Shortest-Path MDP

An **SSP-MDP** is a tuple (S, A, T, C, G) where :

- S is the finite set of **states** ;
- A is the finite set of **actions** that the agent can execute ;
- $T : S \times A \times S \to [0, 1]$ is the **transition function**, where $T(s, a, s')$ gives the probability that the agent reach state s' if it exécute action a at state s ;
- $C : S \times A \times S \to \mathbb{R}^+$ is the **cost function** where $C(s, a, s')$ gives the cost an agent must pay if it reaches state s' when executing action a at state s ;
- $G \subseteq S$ is the set of **goal states**.

Classical algorithms

Policy

A (Markovian and stationnary) **policy** is a function $\pi : S \rightarrow A$ that returns, for every state, the action an agent should execute.

Value function

A **value function** (associated to a policy π) is a function $V^{\pi} : S \rightarrow \mathbb{R}$ that maps each state s to the expected total cost of an agent starting at s that executes the actions given by π until reaching a goal.

Classical algorithms

- Policy Iteration (PI) (1960) [1]
- Value Iteration (VI) (1957) [2]

1. Howard, R. A. (1960). Dynamic Programming and Markov Processes. John Wiley.
2. Bellman, R. (1957). Dynamic Programming. Prentice Hall.

Modern approaches

Heuristic search

- These methods assume that we have two additionnal elements :
  1. an initial state known a priori ;
  2. a **heuristic function** $h \colon S \to \mathbb{R}$ estimating the expected cost to reach a goal.
- Common MDP heuristic search algorithms :
  - **LAO*** (ILAO*, RLAO*, BLAO*, etc.), **LRTDP** (BRTDP, FRTDP, etc.).

Prioritized methods

- The order of states' value update drastically influence convergence time.
- E.g., can range from $\mathcal{O}(n)$ to $\mathcal{O}(n^2)$ state updates.
- **Prioritized VI** (PVI) : a priority is assigned to every state.
- There's many variants with different priority function :
  - Prioritize states close to a goal (for a more efficient back-propagation of states' value) ;
  - Prioritize states with a large residual (farthest from convergence) ;
- Newer methods partition the MDP and assign priority to them rather than states.
- E.g., Topological Value Iteration (TVI), FTVI, P3VI, etc.

Computer architecture

- Another way of improving speed is to consider the architecture of modern computers, e.g. :
  - Cache memory hierarchy ;
  - Parallelism ;
  - Instruction set (e.g., SIMD operations) ;
  - GPU implementation.
- Many considerable speedups have been obtained in other domains.
- E.g., in ML, many researches have recently provided some efficient implementations of ML techniques (specialized data structures, specialized CPU datatype (bfloat), consideration of cache, etc.).
- In MDP planning, no such elements have ever been considered.
- Our goal with this research is to propose an MDP memory representation designed to be cache efficient (with any MDP solver).

Existing implementations

- The performance difference between MDP algorithms (VI, TVI, LRTDP, etc.) can sometimes be smaller than the performance speedup that can be obtained by a more efficient implementation.
- The data structures used to store an MDP in memory are hardly ever mentionned.
- We looked at publicly available MDP implementations to compare their memory representation (all in C++) :
  - AI Toolbox [3]
    - Uses dense or sparse matrices to store MDPs (one for transitions, and one for costs) ;
    - Some memory is wasted (dense matrices) ;
    - The cache efficiency is not optimal (sparse matrices).
  - TVI authors' implementation
    - Linked list of states ;
    - Each state contains a linked list of applicable actions ;
    - Very poor cache efficiency and some memory overhead.
  - MDP Engine Library [4] and G-Pack [5] (C++ Libraries)
    - Made resp. by authors of LRTDP and Gourmand (ICAPS2014 planning competition 2nd place) ;
    - Hash tables of structures ;
    - Two levels of indirection which prevent optimal cache usage.

---

3. E. Bargiacchi, D. M. Roijers, and A. Nowé. AI-Toolbox : A C++ library for Reinforcement Learning and Planning (with Python Bindings). Journal of Machine Learning Research (2020), pp. 1-12.

4. `https://github.com/bonetblai/mdp-engine`

5. A. Kolobov, Mausam, and D. S. Weld, "LRTDP versus UCT for online probabilistic planning," in Proc. of the Natl. Conf. on AI, 2012, vol. 3, no. 1, pp. 1786–1792.

CSR-MDP Representation

- CSR-MDP is inspired by the **Compressed Sparse Row** representation of graphs.
- This is the first MDP representation using a "structure of arrays" (SoA) memory layout instead of an "array of structures" (AoS) layout.
- Minimal wasted memory (no pointers, no need to have memory padding).
- By being packed tightly in memory, we ensure most memory inside loaded cache lines is useful for the current computation.
- This representation also simplifies an SIMD (e.g., SSE, AVX) implementation of the computations (e.g., Bellman-Backups)

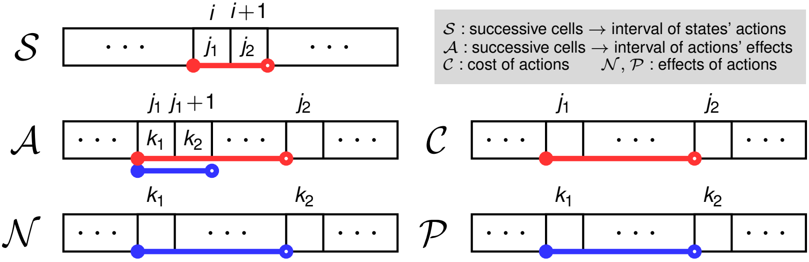
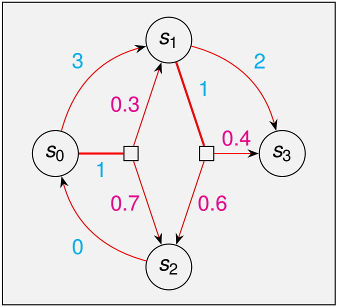


Figure – CSR-MDP memory representation scheme

CSR-MDP Example



$\mathcal{S}$

0	1	2	3	4
0	2	4	5	5

$\mathcal{A}$

0	1	2	3	4	5
0	1	3	4	6	7

$\mathcal{C}$

0	1	2	3	4
3	1	2	1	0

$\mathcal{N}$

0	1	2	3	4	5	6
1	1	2	3	3	2	0

$\mathcal{P}$

0	1	2	3	4	5	6
1	0.3	0.7	1	0.4	0.6	1

Methodology

- We compare the performance of CSR-MDP to the performance of the MDP representation in TVI's implementation by its authors on 3 differents algorithms :
  - VI (the asynchroneous round-robin variant) ;
  - LRTDP (with the admissible and domain independent h_{min} heuristic) ;
  - TVI.
- We implemented the proposed algorithms in C++.
- We used the GNU g++ compiler (version 11.2) with level 3 optimizations.
- The compiler auto-vectorized some loops using AVX instructions.
- The tests were carried out on a PC computer equipped with an Intel Core i5 7600k processor.
- The planner never used more than 2 GB, even for the largest domain instances so memory usage of our proposed planners was not an issue.
- For each parameter configurations of the tested planning domains, we randomly generated 15 instances.
- To minimize random factors, we report the median values of the obtained results.

Description of the planning domains

We evaluated the performance of VI, LRTDP and TVI on 3 different MDP domains.

Layered domain

- Domain introduced in TVI's original paper.
- Generic domain that models situations where some events are irreversible.
- E.g., board games where the number of pieces of a player can never grow.

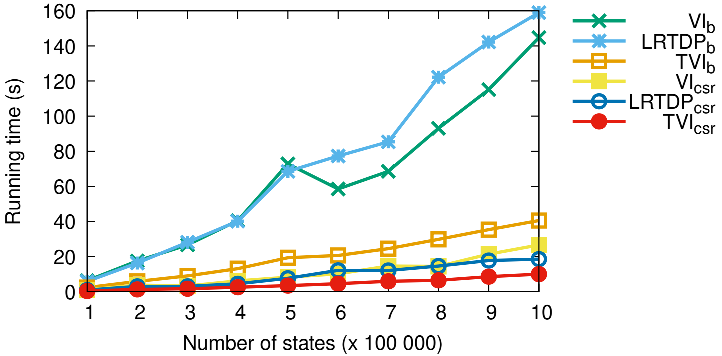
Single-Armed Pendulum (SAP) domain

- Two-dimensional minimum-time optimal control problem.
- Two possible actions at each state : apply a positive or negative torque.
- Objective : Balance the pendulum to the top.

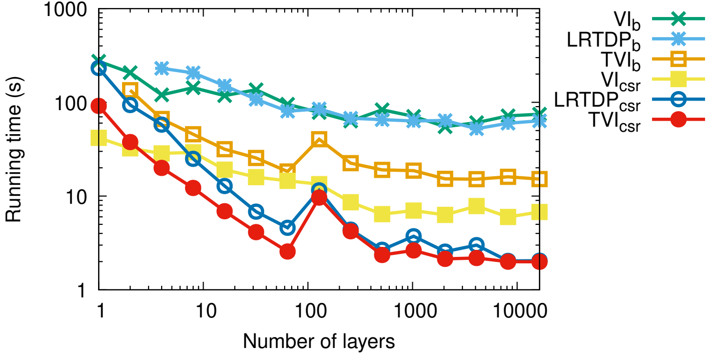
Wetfloor domain

- Many rooms (square navigation grid) are connected to each other.
- Each cell in a room can be slightly or heavily wet, or dry.
- The goal is to find a shortest path between two positions.

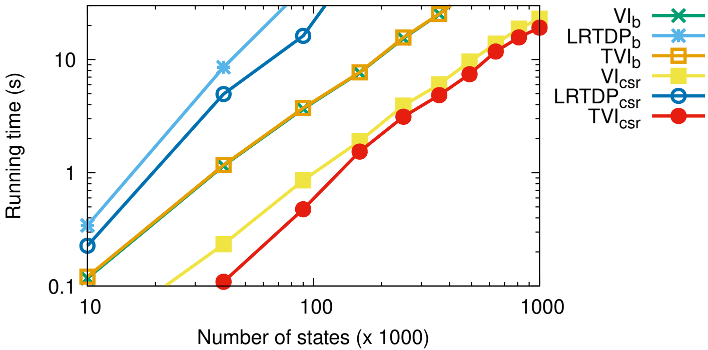
Layered domain when varying the number of states (10 layers)



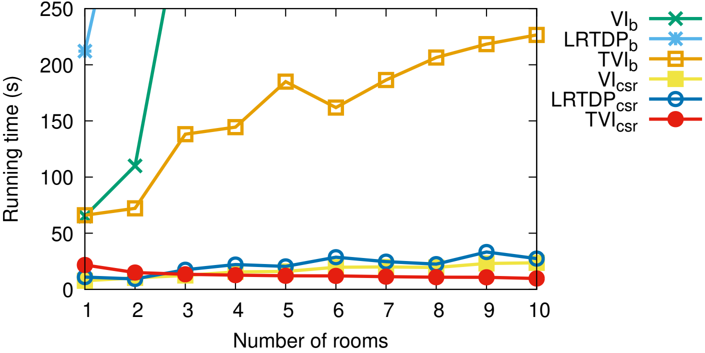
Layered domain when varying the number of layers (1M states)



Single-Armed Pendulum (SAP) domain



Wetfloor domain (500k states)



Results summary

Domain	VI	LRTDP	TVI
Layered (var. states)	5.87481	7.91771	4.46547
Layered (var. layers)	6.77031	$>$ 4.07741	$>$ 3.87843
SAP	4.36132	$>$ 1.0539	5.34032
Wetfloor	$>$ 15.3342	$>$ 13.812	12.3605
Average	$>$ 8.69197	$>$ 2.86112	$>$ 6.6481

Table – Average speedup factors obtained by every solver on every domain using the proposed CSR-MDP data structure when compared to the baseline implementation. Numbers with the '$>$' symbol are lower bounds on the true speedup factor.

Conclusion

- Finding an ϵ-optimal policy of an MDP can take an unreasonable amount of time due to the curse of dimensionality.
- We proposed a new way of storing an MDP in memory, called CSR-MDP, that :
  - minimizes memory access time when solving MDPs.
  - allows an SIMD implementation of the bellman-backup operator.
- The proposed CSR-MDP representation led to an average speedup (over all tested domains) of 8.6, 2.8 and 6.6, when using VI, LRTDP and TVI, respectively.
- As future work, we plan to :
  - develop and test a state-space decomposition method that takes into account the different levels of cache memory in modern CPUs, which could almost totally eliminate the cache misses when solving a large MDP.
  - investigate if the proposed representation can be used in GPU-based implementations.

Acknowledgments

CRSNG NSERC

Fonds de recherche Nature et technologies Québec

We acknowledge the support of the *Natural Sciences and Engineering Council of Canada* (NSERC) and the *Fonds de recherche du Québec — Nature et technologies* (FRQNT).