Cache-Efficient Dynamic Programming MDP Solver

Leveraging modern computer memory architecture

Jaël Champagne Gareau
Guillaume Gosset
Éric Beaudry
Vladimir Makarenkov

Computer Science Department
Université du Québec à Montréal

30 September – 4 October 2023

Outline

Probabilistic planning

- **Markov Decision Processes** (MDPs) are used to model problems of decision-making under uncertainty.
- We focus this research on the **Stochastic Shortest-Path** (SSP-MDP) problem, where the objective is to find a policy $\pi : S \rightarrow A$ that minimizes the expected cost $V^\pi : S \rightarrow \mathbb{R}$ to reach a goal state.
- Many real-world MDP problems require a large number of state variables.
- Curse of dimensionality : the number of states is exponential in the number of state variables.
- Often we are limited in time to find the problem's solution.

Objective of this research

- Find new ways to accelerate MDP computations of an optimal policy.

Classical dynamic programming algorithms

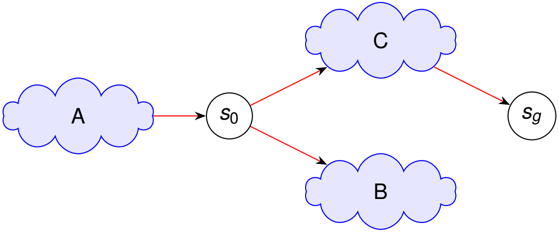
- **Policy Iteration** (PI) :
  - $\pi_0 \xrightarrow{\text{E}} V^{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} V^{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi^\star \xrightarrow{\text{E}} V^\star.$
- **Value Iteration** (VI) :
  - $V_0 \xrightarrow{\text{S}} V_1 \xrightarrow{\text{S}} V_2 \xrightarrow{\text{S}} \cdots \xrightarrow{\text{S}} V^\star.$
  - VI uses the Bellman optimality equations :

$$\forall s \in S, V(s) = \begin{cases} 0 & \text{if } s \in G, \\ \min_{a \in A} \left[C(s,a) + \sum_{s' \in S} T(s,a,s') V(s') \right] & \text{otherwise.} \end{cases}$$

- In both cases, many sweeps over the state-space (in arbitrary order) are necessary.

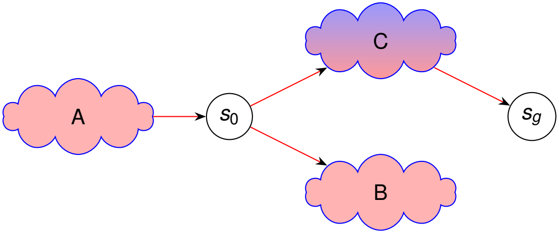
Heuristic search algorithms

- These methods assume that we have two additionnal elements :
  - A an initial state known a priori ;
  - B 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.).
- They allow to prune parts of the state space.



Heuristic search algorithms

- These methods assume that we have two additionnal elements :
  - A an initial state known a priori ;
  - B 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.).
- They allow to prune parts of the state space.
  - A States not reachable from s_0 can be pruned.
  - B States that cannot reach s_g can be pruned.
  - C States that cannot be reached from s_0 when following the policy $\pi^\star$ can be pruned.

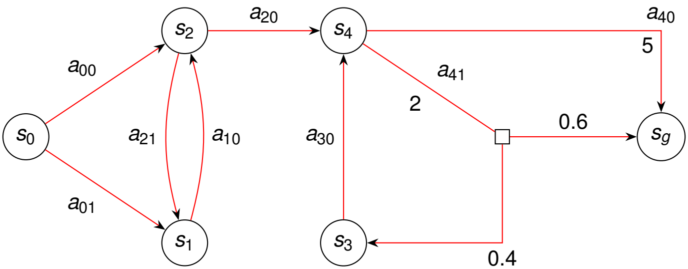


Prioritized methods

- The order of states' value update can 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.
- Many priority criteria can be used with PVI :
  - Prioritize states close to a goal (for a more efficient back-propagation of states' value) ;
  - Prioritize states with a large residual (furthest from convergence) ;
- Some methods partition the MDP and assign priority to them rather than states.
  - E.g., P-EVA, P3VI, etc.

Topological Value Iteration

- Instead of assigning an explicit priority to the states or partitions, we can instead find an order given implicitly by the graph topology of the determinized MDP, e.g., its strongly connected components (SCCs).
  - E.g., **Topological Value Iteration** (TVI), FTVI, etc.



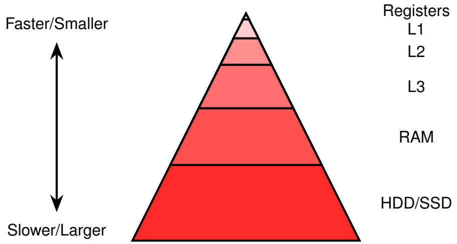
$$\begin{tikzcd} s_0 \arrow[r] & \boxed{s_1, s_2} \arrow[r] \arrow[loop below] & \boxed{s_3, s_4} \arrow[r] \arrow[loop below] & s_g \end{tikzcd}$$

Modern computer architecture

- Another way of improving speed is to consider the architecture of modern computers, e.g. :
  - **Memory hierarchy** ;
  - Thread level parallelism ;
  - Data level parallelism (e.g., SIMD operations) ;
  - GPU implementation.
- Many considerable speedups have been obtained in other domains.
- E.g., in ML, many researches considered efficient implementations of ML techniques (specialized data structures, specialized CPU datatype (bfloat), consideration of cache, etc.), leading to a speedup of many orders of magnitude.
- In MDP planning, these elements have been much less considered.

Memory Hierarchy

Type	Size	Transfer Rate (GB/s)	Latency (ns)
L1	64 Ko/cores	2000	1
L2	1 Mo/cores	1000	3
L3	32 Mo	600	12
DDR5	8-128 Go	50	90
SSD (NVMe)	250 Go-4 To	7	80 000



Cache-efficient with clustering (CEC) algorithm

- The **Cache-efficient with clustering**[1] algorithm is based on FTVI.
- It subdivides each SCC into partitions using a clustering algorithm :
  - take a random state *s* inside the currently considered SCC ;
  - do a BFS from *s* and add visited state to the current partition ;
  - stop when the size of the partition reaches the L3 cache size.
- For each partitionned SCC, CEC cyclically do one Bellman-sweep on each partition one-by-one until the SCC converges.

1. Jain, A., & Sahni, S. (2020). Cache efficient Value Iteration using clustering and annealing. Computer Communications, 159, 186-197.

Cache-efficient MDP memory representation

- CSR-MDP [2] is inspired by the **Compressed Sparse Row** repr. of graphs.
- 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.
- Most solving algorithms can be used with MDPs stored in CSR-MDP format.

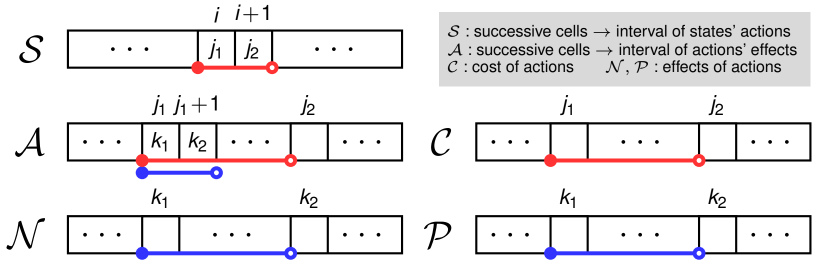
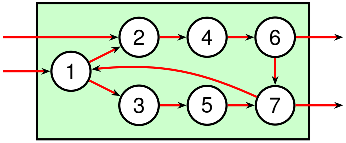

i
$i+1$
$\mathcal{S}$
j_1
j_2
$\mathcal{S}$: successive cells → interval of states' actions
$\mathcal{A}$: successive cells → interval of actions' effects
$\mathcal{C}$: cost of actions
$\mathcal{N}$, $\mathcal{P}$: effects of actions
j_1 j_1+1 j_2
$\mathcal{A}$
k_1 k_2
$\mathcal{C}$
j_1 j_2
k_1 k_2
$\mathcal{N}$
$\mathcal{P}$
k_1 k_2

Figure – CSR-MDP memory representation scheme

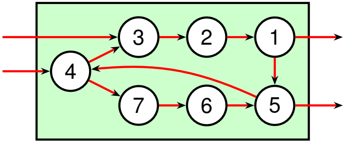
2. Champagne Gareau, J., Beaudry, É., & Makarenkov, V. (2022). Cache-Efficient Memory Representation of Markov Decision Processes. Proceedings of the Canadian Conference on Artificial Intelligence.

Performance analysis of Topological Value Iteration

- TVI can improve performance compared to VI in three ways :
  - A it maximizes the effectiveness of every state backup by ensuring only converged state values are propagated from one SCC to the other ;
  - B the backup order is given by a postorder DFS, which will most likely improve the information flow compared to the arbitrary order used by VI ;
  - C since there are fewer states considered in an SCC sweep, there are less cache misses.



VI : arbitrary order



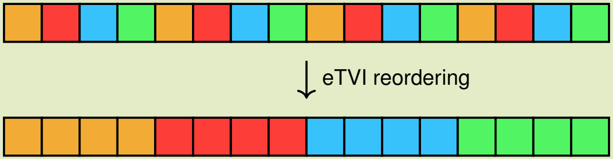
TVI : DFS postorder

eTVI : Reordering according to the order external to the SCCs

- One way to improve cache performance is to reorder the MDP in memory such that the data relative to each SCC is contiguous.
- Our first proposed algorithm, **eTVI**, uses this idea and reorders the states according to their *external* position (the position among the SCCs).

eTVI example : reordering according to an order external to the SCCs

- We assume each state takes 16 bytes.
- Each SCC contains four states.
- Before : each SCC is spreaded across four cache lines.
- After : each SCC is contained in a single cache line.

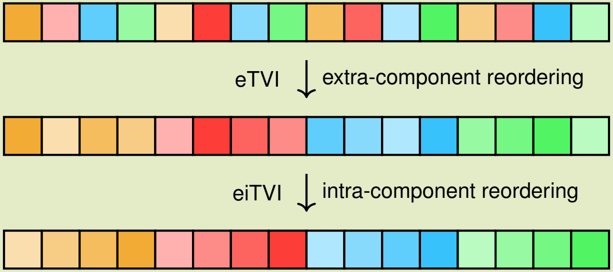


eiTVI : Reordering according to the order internal to the SCCs

- eTVI only considers the order of states with respect to the SCC they're part of.
- What about the order of states **inside** an SCC ?
- States should be stored in the same order as the Bellman sweeps consider them.

eiTVI example : reordering according to an order internal to the SCCs

- Assume the shade of a color represent the order of states considered inside an SCC during a sweep (lighter shades are considered before darker shades).

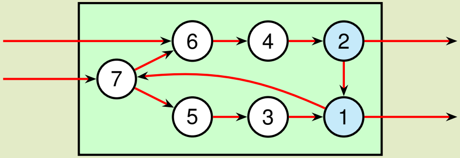


eiTVI : What is the best order internal to an SCC ?

- TVI uses the states' discover order while searching the SCCs (a postorder DFS).
- We should use an order that maximizes state-values propagation
- One possible solution : find a dynamic order in a way similar to, e.g., PVI.
  - It requires a priority queue, and has a large overhead.
- Instead, we propose a static backup order given by a reversed BFS started at the outward border states of the SCC.

Example of the proposed order

- States in blue are the *outward border states* of the SCC.
- States are numbered by their optimal propagation order.



Methodology

- We compare the performance of eTVI and eiTVI to the performance of :
  - VI (the asynchroneous round-robin variant) ;
  - LRTDP (with the admissible and domain independent h_{min} heuristic) ;
  - ILAO* ;
  - TVI.
- We implemented the proposed algorithms in C++.
- We used the GNU g++ compiler (version 11.2) with level 3 optimizations.
- The tests were carried out on a computer equipped with an Intel Core i5 7600k processor.
- On every instance, the planner never used more than 2 GB of RAM, therefore it was not a relevant metric with regard to the capacity of modern computers.
- 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 compared the performance of the algorithms on three 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

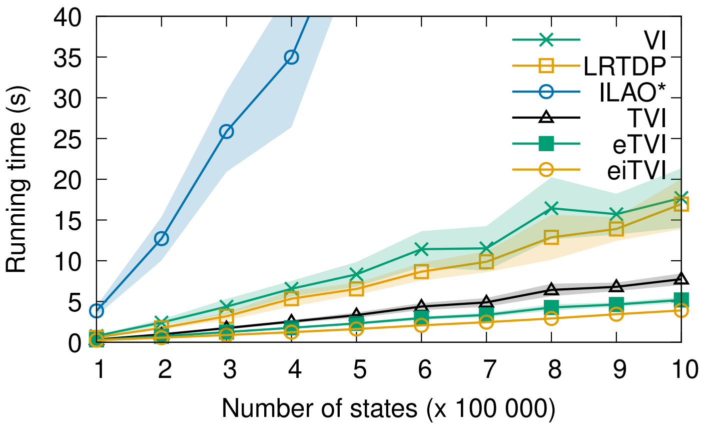
- Two-dimensional minimum-time optimal control problem.
- Two possible actions at each state : apply a positive or negative torque.
- Objective : Balance the pendulum to the top.

Wetfloor domain

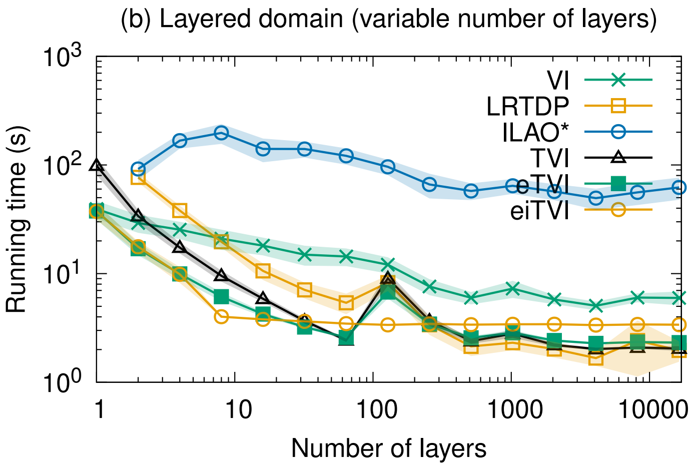
- Many rooms (square navigation grid) are connected to each other.
- Each cell in a room can be slightly wet, heavily wet or dry.
- The goal is to find a shortest path between two positions.

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

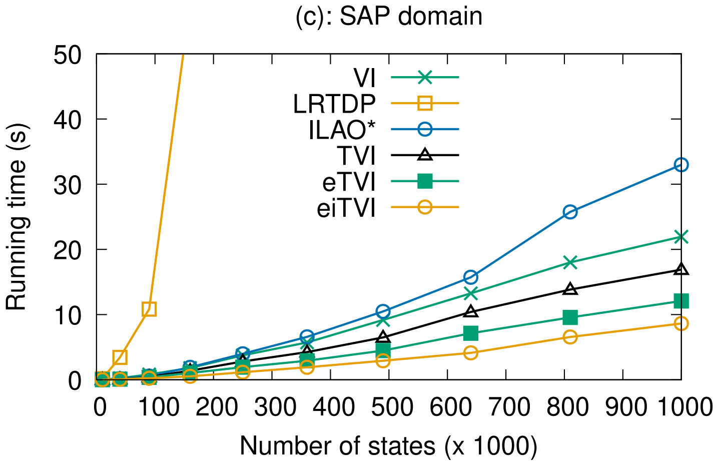
(a) Layered domain (variable number of states)



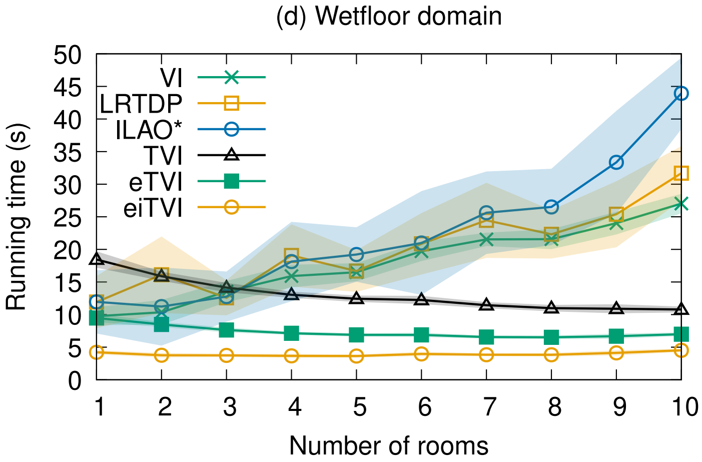
Layered domain when varying the number of layers (1M states)



Single-Armed Pendulum (SAP) domain



Wetfloor domain (500k states)



Results : Average speedup factors obtained on each tested domain

Domain	TVI vs VI	eTVI vs TVI	eiTVI vs eTVI	eiTVI vs TVI
Layered (var. states)	2.4988	1.4306	1.3955	1.9965
Layered (var. layers)	1.8054	1.4549	0.9774	1.4220
SAP	1.3999	1.3725	1.7440	2.3937
Wetfloor	1.3810	1.7788	1.8635	3.3147
Average	1.6285	1.6018	1.3119	2.1014

Results : Cache metrics obtained

- Layered domain (instance with 1M states, 10 layers).

Solver	Cache-Refs	Cache-Misses	Miss Percent
TVI	2.87G	0.860G	29.96
eTVI	2.39G	0.413G	17.28
eiTVI	1.59G	0.328G	20.62

Conclusion

- We proposed two ways of improving cache performance of MDP solvers, both based on TVI and leveraging the CSR-MDP memory representation :
  - eTVI : reorders states in memory so that data inside each SCC is contiguous ;
  - eiTVI : reorders states in memory so that the order in memory matches the order in which states are considered during Bellman sweeps.
- Both methods combined led to solvers more than twice as fast.
- Combined with the speedup factor of 6 provided by CSR-MDP, the resulting planner is thus on average more than 12 times as fast.
- As future work, we plan to :
  - develop and test a subdivision of SCCs into smaller subcomponents with minimal dependencies between them using, e.g., Louvain's algorithm or by finding provably suboptimal strong bridges.

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).