pcTVI: Parallel MDP Solver

Using a Decomposition Into Independent Chains

Jaël Champagne Gareau
Éric Beaudry
Vladimir Makarenkov

Computer Science Department
Université du Québec à Montréal

19 – 23 July 2022

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.
- 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 reaches state s' if it executes 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) [1]
- Value Iteration (VI) [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.

Exploiting modern computer architectures

- Another way of improving speed is to consider the architecture of modern computers, e.g. :
  - Cache memory hierarchy [3] ;
  - GPU implementation.
  - Data-level parallelism (i.e., SIMD operations) ;
  - Thread-level **Parallelism** ;
- 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, parallel decomposition, etc.).
- In MDP planning, no such elements have ever been considered.
- Our goal with this research is to propose a parallel MDP solver based on TVI.

3. Champagne Gareau, J., Beaudry, É., & Makarenkov, V. (2022). Cache-Efficient Memory Representation of Markov Decision Processes. Proceedings of the Canadian Conference on Artificial Intelligence. https ://caiac.pubpub.org/pub/pq25qiqh

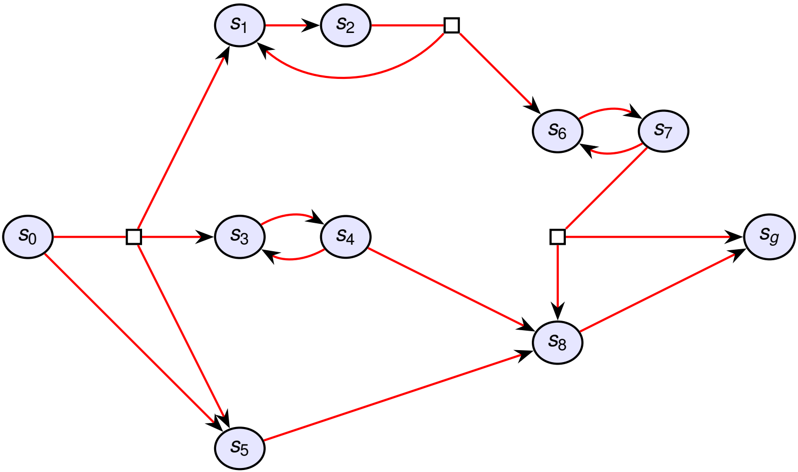
Existing parallel MDP algorithms

- P3VI : Partitionned, Prioritized, Parallel Value Iteration
  - Partitions the state-space into multiple sub-parts ;
  - Assign a priority to each part ;
  - Solve multiple parts in parallel in the order given by the priorities.
  - Disadvantages :
    - The partitionning is done on a case-by-case basis depending on the planning domain ;
    - Communication of the state values between the solving threads incurs an overhead on the computational time.
- Parallel CECA (Cache-Efficient with Clustering and Annealing)
  - CECA partitions the state-space and solves the parts one-by-one according to a simulated-annealing schedule.
  - Parallel CECA solves multiple clusters in parallel.
  - Disadvantages :
    - The final algorithm is more complex to understand/implement than other MDP algorithms ;
    - The performance improvement due to the parallelization is underwhelming (factor 2.59 speedup on a 10 cores CPU).

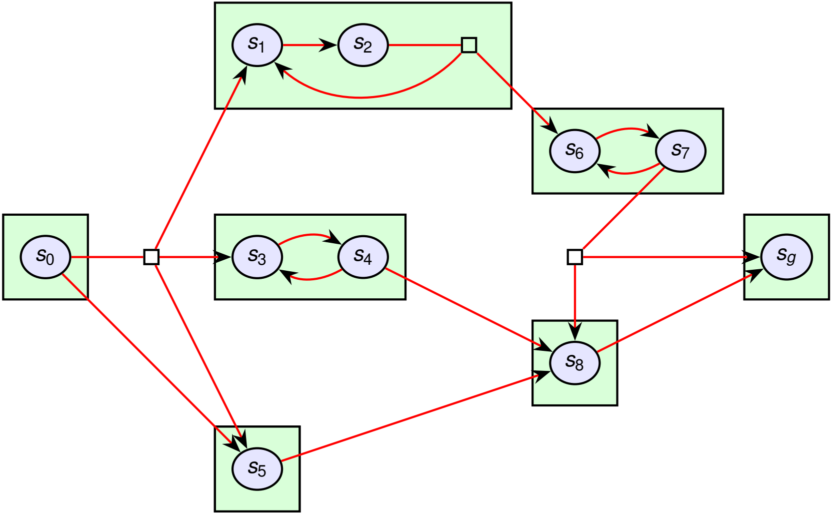
Parallel-Chained Topological Value Iteration

- Based on Topological Value Iteration ;
  - Considers the graph corresponding to the topological structure of the MDP (or, equivalently, the all-outcome determinization of the MDP) ;
  - Uses Tarjan's algorithm to decompose the graph into strongly-connected components (SCCs) ;
  - If we consider the SCCs in reverse topological order, they can be solved using a single sweep over each of them.
- Instead of choosing SCCs to solve in parallel randomly or with a priority metric, we use instead the dependencies between the SCCs.
- SCC's with no common dependencies can be computed in parallel.
- To find the dependencies : we can do a backward breadth-first search (from the goal state) in the graph condensation of the MDP (the graph containing the SCCs of the MDP structure) to find chains of independent SCCs.
- During runtime, a new parallel task is created everytime an SCC's dependencies have all been computed.

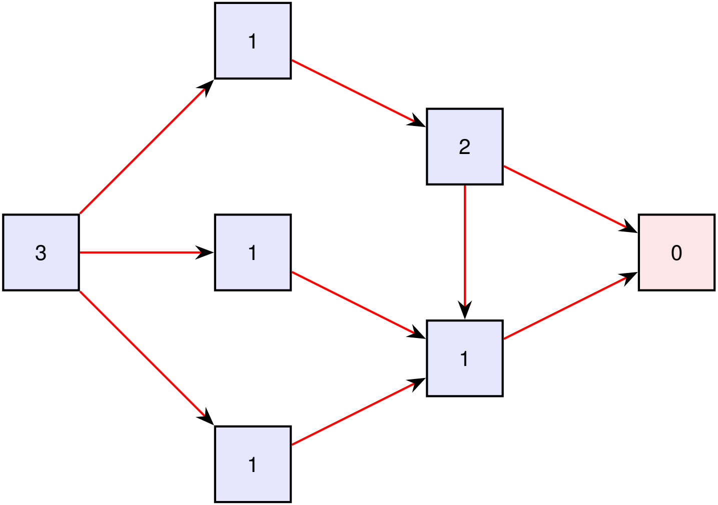
Example of pcTVI execution



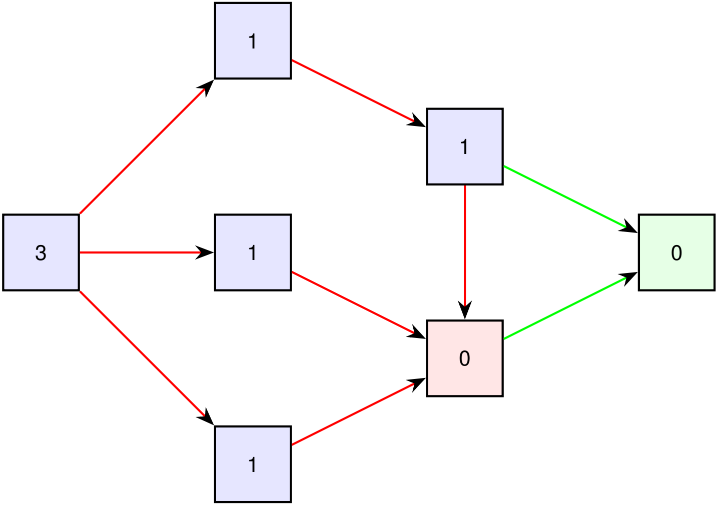
Example of pcTVI execution



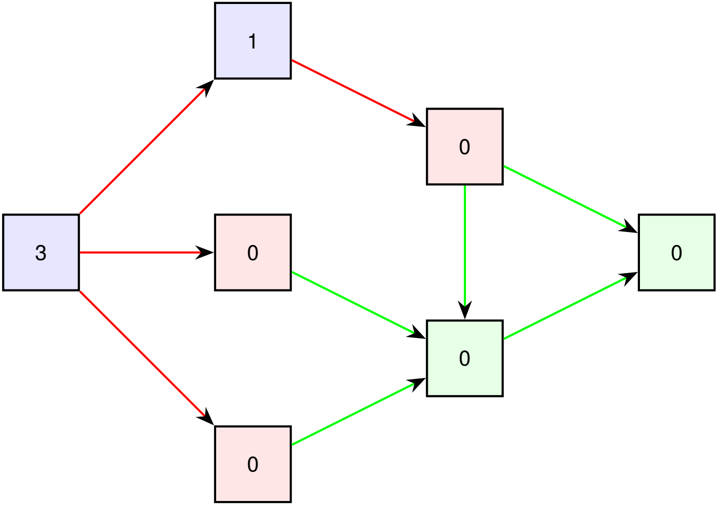
Example of pcTVI execution



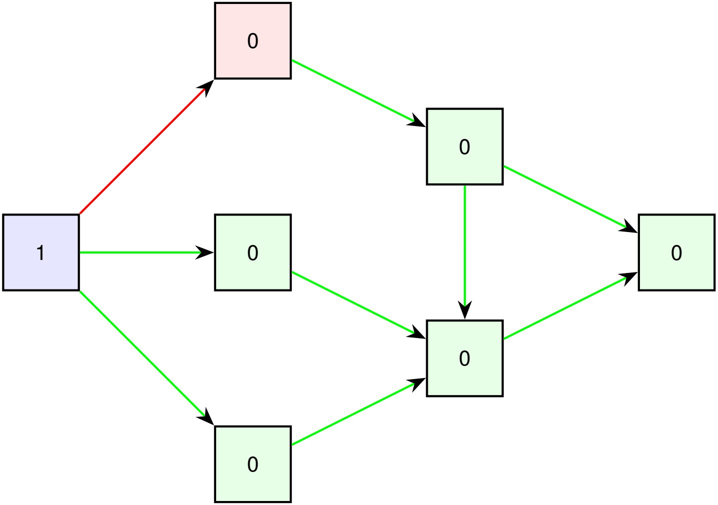
Example of pcTVI execution



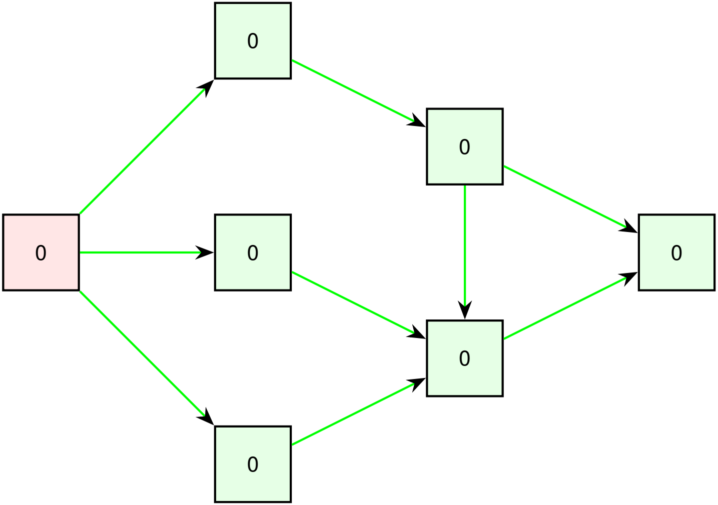
Example of pcTVI execution



Example of pcTVI execution



Example of pcTVI execution



Algorithm Parallel-Chained Topological Value Iteration

```
procedure pcTVI(M : MDP, t : Number of threads)
    ▷ Find the SCCs of M
    G ← Graph(M)                          ▷ G implicitly shares the same data structures as M
    SCCs ← Tarjan(G)                      ▷ SCCs are found in reverse topological order

    ▷ Solve in parallel independent SCCs
    G_c ← GraphCondensation(G, SCCs)
    Pool ← CreateThreadPool(t)            ▷ Create t threads
    V ← NewValueFunction()                ▷ Arbitrarily initialized ; Shared by all threads
    Q ← CreateQueue()                     ▷ Shared by all threads
    Insert(Q, Head(SCCs))                 ▷ The goal SCC is inserted in the queue
    while NotEmpty(Q) do                  ▷ Only one thread runs this loop
        scc ← ExtractNextItem(Q)
        for all neighbor ∈ Neighbors(scc) do
            Decrement NumIncomingNeighbors(neighbor)
            if NumIncomingNeighbors(neighbor) = 0 then
                AssignTaskToAvailableThread(Pool, PartialVI(M, V, scc))
                Push(Q, scc)              ▷ Neighbors of scc are ready to be considered next

    return GreedyPolicy(V)
```

Chained-MDP domain

- There was no standard probabilistic planning domain in the literature suitable to benchmark a parallel MDP solver.
- We thus propose a new parametric MDP solver, called **Chained-MDP**.
- The parameters are :
  - k : the number of independent chains $c_1, c_2, \ldots, c_k$;
  - n_{scc} : the number of SCCs $scc_{i,1}, scc_{i,2}, \ldots, scc_{i,n_{scc}}$ in every chain c_i ;
  - n_{sps} : the number of states per SCC ;
  - n_a : the number of applicable actions per state ;
  - n_e : the number of probabilistic effects per action.
- Successors of a state s in $scc_{i,j}$ can be any state in $scc_{i,j}$ or in $scc_{i+1,j}$ (or, if the latter does not exist, it can be the goal state).

Chained-MDP instance example

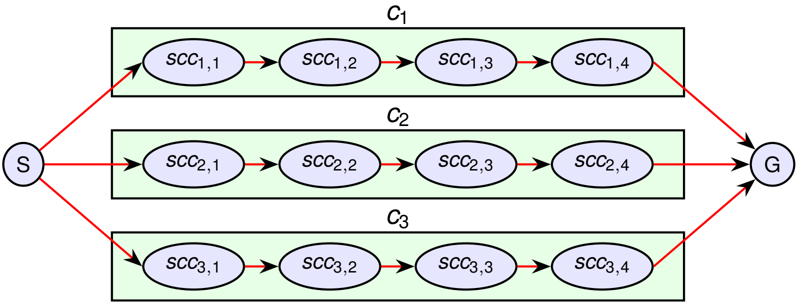
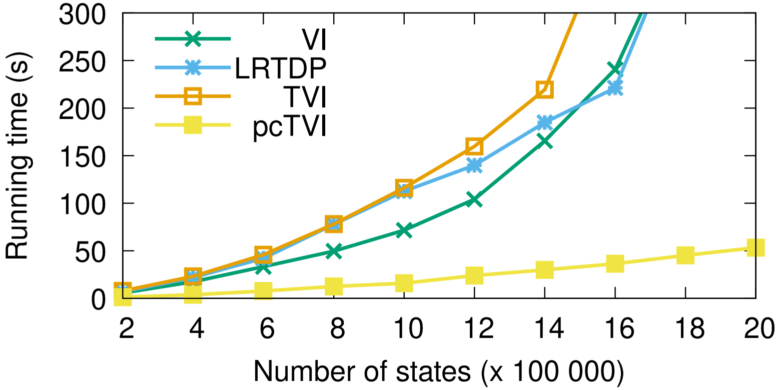


Figure – A chained-MDP instance where $n_c = 3$ and $n_{scc} = 4$. Each ellipse represents a SCC.

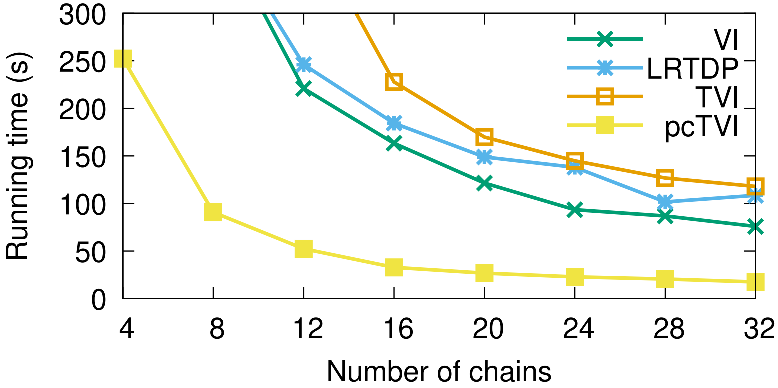
Methodology

- We compare the performance of pcTVI to the performance of :
  - 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 tests were carried out on a computer equipped with four Intel Xeon E5-2620V4 processors.
- Each of these processors have 8 cores (at 2.1 GHz), for a total of 32 cores.
- The planner never used more than 2 GB, even for the largest domain instances so memory usage of our proposed algorithm was not an issue.
- For every test instance, we measured the running time of each algorithms carried out until convergence to an ϵ-optimal value-function (we used $\epsilon = 10^{-6}$).
- For each tested parameter configurations of the parallel-chained MDP domain, we randomly generated 15 instances.
- To minimize random factors, we report the average values of the obtained results.

Chained-MDP with varying number of states and a fixed 32 chains



Chained-MDP with fixed 1M states and varying number of chains



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 domain-independent way of solving an MDP in parallel.
- We also proposed a new parametric planning domain, suitable to model any sitiation where different strategies (i.e., a chain) can reach a goal but where, once commited to a strategy, it is not possible to switch to a different one.
- The pcTVI algorithm led to an average speedup of 20, on a 32 cores computer.
- As future work, we plan to :
  - investigate ways of pruning provably suboptimal actions, which would allow more SCCs to be found ;
  - investigate on how to apply the proposed algorithm on the MDPs used in Reinforcement Learning (RL).

Acknowledgments

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).