pcTVI: Parallel MDP Solver Using a Decomposition Into Independent Chains

Jaël Champagne Gareau and Éric Beaudry and Vladimir Makarenkov

Abstract Markov Decision Processes (MDPs) are useful to solve real-world probabilistic planning problems. However, finding an optimal solution in an MDP can take an unreasonable amount of time when the number of states in the MDP is large. In this paper, we present a way to decompose an MDP into Strongly Connected Components (SCCs) and to find dependency chains for these SCCs. We then propose a variant of the Topological Value Iteration (TVI) algorithm, called *parallel chained TVI* (pcTVI), which is able to solve independent chains of SCCs in parallel leveraging modern multicore computer architectures. The performance of our algorithm was measured by comparing it to the baseline TVI algorithm on a new probabilistic planning domain introduced in this study. Our pcTVI algorithm led to a speedup factor of 20, compared to traditional TVI (on a computer having 32 cores).

Keywords: Markov Decision Process, Automated Planning, Strongly Connected Components, Dependancy Chains, Parallel Computing

1 Introduction

Automated planning is a branch of Artificial Intelligence (AI) aiming at finding optimal plans to achieve goals. One example of problems studied in automated planning is the electric vehicle path-planning problem [1]. Planning problems with non-deterministic actions are known to be much harder to

Jaël Champagne Gareau
Université du Québec à Montréal, Canada, e-mail: `champagne_gareau.jael@uqam.ca`

Éric Beaudry
Université du Québec à Montréal, Canada, e-mail: `beaudry.eric@uqam.ca`

Vladimir Makarenkov
Université du Québec à Montréal, Canada, e-mail: `makarenkov.vladimir@uqam.ca`

solve. Markov Decision Processes (MDPs) are generally used to solve such problems leading to probabilistic models of applicable actions [2].

In probabilistic planning, a solution is generally a policy, i.e., a mapping specifying which action should be executed in each observed state to achieve an objective. Usually, dynamic programming algorithms such as Value Iteration (VI) are used to find an optimal policy [3]. Since VI is time-expensive, many improvements have been proposed to find an optimal policy faster, using for example the Topological Value Iteration (TVI) algorithm [4]. However, very large domains often remain out of reach. One unexplored way to reduce the computation time of TVI is by taking advantage of the parallel architecture of modern computers and by decomposing an MDP into independent parts which could be solved concurrently.

In this paper, we show that state-of-the-art MDP planners such as TVI can run an order of magnitude faster when considering task-level parallelism of modern computers. Our main contributions are as follows:

- An improved version of the TVI algorithm, *parallel-chained TVI* (pcTVI), which decomposes MDPs into independent chains of strongly connected components and solves them concurrently.
- A new parametric planning domain, *chained-MDP*, and an evaluation of pcTVI's performance on many instances of this domain compared to the VI, LRTDP [5] and TVI algorithms.

2 Related Work

Many MDP solvers are based on the Value Iteration (VI) algorithm [3], or more precisely on asynchronous variants of VI. In asynchronous VI, MDP states can be backed up in any order and don't need to be considered the same number of times. One way to take advantage of this is by assigning a priority to every state and by considering them in priority order.

Several state-of-the-art MDP algorithms have been proposed to increase the speed of computation. Many of them are able to focus on the most promising parts of MDP through heuristic search algorithms such as LRTDP [5] or LAO* [6]. Some other MDP algorithms use partitioning methods to decompose the state-space in smaller parts. For example, the P3VI (Partitioned, Prioritized, Parallel Value Iteration) algorithm partitions the state-space, uses a priority metric to order the partitions in an approximate best solving order, and solves them in parallel [7]. The biggest disadvantage of P3VI is that the partitioning is done on a case-by-case basis depending on the planning domain, i.e., P3VI does not include a general state-space decomposition method. The inter-process communication between the solving threads also incurs an overhead on the computation time. The more recent TVI (Topological Value Iteration) algorithm [4] also decomposes the state-space, but does it by considering the topological structure of the underlying graph of the MDP, making it more general than P3VI. Unfortunately, to the best of our knowledge, no parallel version of TVI has been proposed in the literature.

3 Problem Definition

There exist different types of MDP, including Finite-Horizon MDP, Infinite-Horizon MDP and Stochastic Shortest Path MDP (SSP-MDP) [2]. The first two of them can be viewed as special cases of SSP-MDP [8]. In this work, we focus on SSP-MDPs, which we describe formally in Definition 1 below.

Definition 1 A *Stochastic Shortest Path MDP* (SSP-MDP) is given by a tuple (S, A, T, C, G), where:

- S is a finite set of states;
- A is a finite set of actions;
- $T\colon S \times A \times S \to [0,1]$ is a transition function, where $T(s,a,s')$ is the probability of reaching state s' when applying action a while in state s;
- $C\colon S \times A \to \mathbb{R}^+$ is a cost function, where $C(s,a)$ gives the cost of applying the action a while in state s;
- $G \subseteq S$ is the set of goal states (which can be assumed to be sink states).

We generally search for a policy $\pi\colon S \to A$ that tells us which action should be executed at each state, such that an execution following the actions given by π until a goal is reached has a minimal expected cost. This expected cost is given by a value function $V^\pi\colon S \to \mathbb{R}$. The Bellman Optimality Equations are a system of equations satisfied by any optimal policy.

Definition 2 The Bellman Optimality Equations are the following:

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

The expression between square brackets is called the *Q-value* of a state-action pair:

$$Q(s,a) = C(s,a) + \sum_{s' \in S} T(s,a,s') V(s).$$

When an optimal value function $V^\star$ has been computed, an optimal policy $\pi^\star$ can be found greedily:

$$\pi^\star(s) = \text{argmin}_{a \in A} Q^\star(s,a).$$

Most MDP solvers are based on dynamic programming algorithms like Value Iteration (VI), which update iteratively an arbitrarily initialized value function until convergence with a given precision ϵ. In the worst case, VI needs to do $|S|$ sweeps of the state space, where one sweep consists in updating the value estimate of every state using the Bellman Optimality Equations. Hence, the number of state updates (called a *backup*) is $\mathcal{O}(|S|^2)$. When the MDP is acyclic, most of these backups are wasteful, since the MDP can in this situation be solved using only $|S|$ backups (ordered in reverse topological order), thus allowing one to find an optimal policy in $\mathcal{O}(|S|)$ [8].

4 Parallel-Chained TVI

In this section, we describe an improvement to the TVI algorithm, named pcTVI (Parallel-Chained Topological Value Iteration), which is able to solve an MDP in parallel (as P3VI). pcTVI uses the decomposition proposed by TVI, known to give good performance on many planning domains. We start by summarizing how the original TVI algorithm works.

First, TVI uses Kosaraju's graph algorithm on a given MDP to find the strongly connected components (SCCs) of its graphical structure (the graph corresponding to its all-outcomes determinization).The SCCs are found by Kosaraju's algorithm in reverse topological order, which means that for every $i < j$, there is no path from a state in the i^{th} SCC to a state in the j^{th} SCC. This property ensures that every SCC can be solved separately by VI sweeps if previous SCCs (according to the reverse topological order) have already been solved. The second step of TVI is thus to solve every SCC one by one in that order. Since TVI divides the MDP in multiple subparts, it maximizes the usefulness of every state backup by ensuring that only useful information (i.e., converged state values) is propagated through the state-space.

Unfortunately, TVI can only solve one SCC at a time. Since modern computers have many computing units (cores) which can work in parallel, we could theoretically solve many SCCs in parallel to greatly reduce computation time. Instead of choosing SCCs to solve in parallel arbitrarily or using a priority metric (as in P3VI), which incur a computational overhead to propagate the values between the threads, we want to consider their topological order (as in TVI) to minimize redundant or useless computations. One way to share the work between the processes is to find independent chains of SCCs which can be solved in parallel. The advantage of independent chains is that no coordination and communication is needed between the SCCs, which both removes some running-time overhead and simplifies the implementation.

The Parallel-Chained TVI algorithm we propose (Algorithm 1) works as follows. First, we find the graph G corresponding to the graphical structure of the MDP, decompose it into SCCs, and find the reverse topological order of the SCCs (as in TVI, but we use Tarjan's algorithm instead of Kosaraju's algorithm since it is about twice as fast). We then build the condensation of the graph G, i.e., the graph G_c whose vertices are SCCs of G, where an edge is present between two vertices scc_1 and scc_2 if there exists an edge in G between a state $s_1 \in scc_1$ and a state $s_2 \in scc_2$. We also store the reversed edges in G_c and a counter c_{scc} on every vertex scc which indicates how many incoming neighbors have not yet been computed. We use this (usually small) graph G_c to detect which SCCs are ready to be considered (the SCCs whose incoming neighbors have all been determined with precision ϵ, i.e., the SCCs whose associated counter c_{scc} is 0). When a new SCC is ready, it is inserted into a work queue from which the waiting threads acquire their next task.

Algorithm 1 Parallel-Chained Topological Value Iteration

```
1: procedure pcTVI($M$: MDP, $t$: Number of threads)
2:     ▷ Find the SCCs of $M$
3:     $G \leftarrow$ Graph($M$)        ▷ $G$ implicitly shares the same data structures as $M$
4:     $SCCs \leftarrow$ Tarjan($G$)          ▷ SCCs are found in reverse topological order
5:
6:     ▷ Build the graph of SCCs of $G$
7:     $G_c \leftarrow$ GraphCondensation($G, SCCs$)
8:
9:     ▷ Solve in parallel independent SCCs
10:    $Pool \leftarrow$ CreateThreadPool($t$)                          ▷ Create $t$ threads
11:    $V \leftarrow$ NewValueFunction()   ▷ Arbitrarily initialized; Shared by all threads
12:    $Q \leftarrow$ CreateQueue()                                 ▷ Shared by all threads
13:    Insert($Q$, Head($SCCs$))                  ▷ The goal SCC is inserted in the queue
14:    while NotEmpty($Q$) do                       ▷ Only one thread runs this loop
15:        $scc \leftarrow$ ExtractNextItem($Q$)
16:        for all $neighbor \in$ Neighbors($scc$) do
17:            Decrement NumIncomingNeighbors($neighbor$)
18:            if NumIncomingNeighbors($neighbor$) $= 0$ then
19:                AssignTaskToAvailableThread($Pool$, PartialVI($M, V, scc$))
20:                Push($Q, scc$)    ▷ Neighbors of $scc$ are ready to be considered next
21:
22:    ▷ Compute and return an optimal policy using the computed value function
23:    $\Pi \leftarrow$ GreedyPolicy($V$)
24:    return $\Pi$
```

5 Empirical Evaluation

In this section, we evaluate empirically the performance of pcTVI, comparing it to the three following algorithms: (1) VI – the standard dynamic programming algorithm (here we use its asynchronous round-robin variant), (2) LRTDP – a well-known heuristic search algorithm, and (3) TVI – the Topological Value Iteration algorithm described in Section 4. In the case of LRTDP, we carried out the admissible and domain-independent $h_{\min}$ heuristic, first described in the original paper introducing LRTDP [5]:

$$h_{\min}(s) = \begin{cases} 0, & \text{if } s \in G. \\ \min\limits_{a \in A_s} \left[C(s,a) + \min\limits_{s' \in succ_a(s)} V(s') \right], & \text{otherwise,} \end{cases}$$

where A_s denotes the set of applicable actions in state s and $succ_a(s)$ is the set of successors when applying action a at state s. The four competing algorithms (VI, TVI, LRTDP and pcTVI) were implemented in C++ by the authors of this paper and compiled using the GNU g++ compiler (version 11.2). All tests were performed on a computer equipped with four Intel Xeon E5-2620V4 processors (each of them having 8 cores at 2.1 GHz, for a total of 32 cores). For every test domain, we measured the running time of the four compared algorithms carried out until convergence to an ϵ-optimal value function (we used $\epsilon = 10^{-6}$). Every domain was tested 15 times with randomly generated MDP instances. To minimize random factors, we report the median values obtained over these 15 MDP instances.

Since there is no standard MDP domain in the scientific literature suitable to benchmark a parallel MDP solver, we propose a new general parametric MDP domain that we use to evaluate the algorithms. This domain, which we call chained-MDP, uses 5 parameters: (1) k, the number of independent chains $\{c_1, c_2, \ldots, c_k\}$ in the MDP; (2) n_{scc}, the number of SCCs $\{scc_{i,1}, scc_{i,2}, \ldots, scc_{i,n_{scc}}\}$ in every chain c_i; (3) n_{sps}, the number of states per SCC; (4) n_a the number of applicable actions per state, and (5) n_e the number of probabilistic effects per action. The possible successors $succ(s)$ of a state s in $scc_{i,j}$ are states in $scc_{i,j}$ and either the states in $scc_{i,j+1}$ if it exists, or the goal state otherwise. When generating the transition function of a state-action pair (s, a), we sampled n_e states uniformly from $succ(s)$ with random probabilities. In each of our tests, we used $n_{scc} = 2$, $n_a = 5$ and $n_e = 5$. A representation of a Chained-MDP instance is shown in Figure 1.

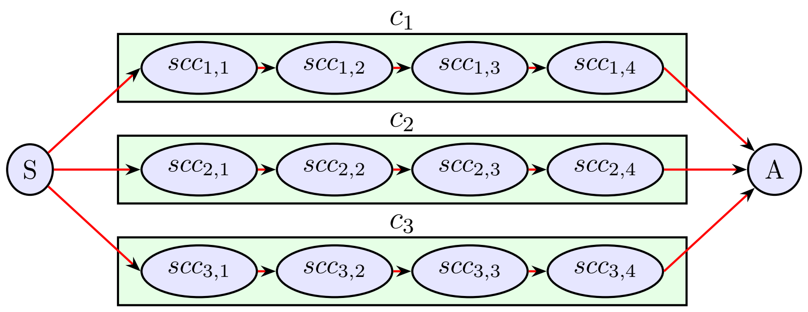

Fig. 1 A chained-MDP instance where $n_c = 3$ and $n_{scc} = 4$. Each ellipse represents a strongly connected component.

Figure 2 presents the obtained results for the Chained-MDP domain when varying the number of states and fixing the number of chains (32). We can observe that when the number of states is small, pcTVI does not provide an important advantage over the existing algorithms since the overhead of creating and managing the threads is taking most of the possible gains. However, as the number of states increases, the gap in the running time between

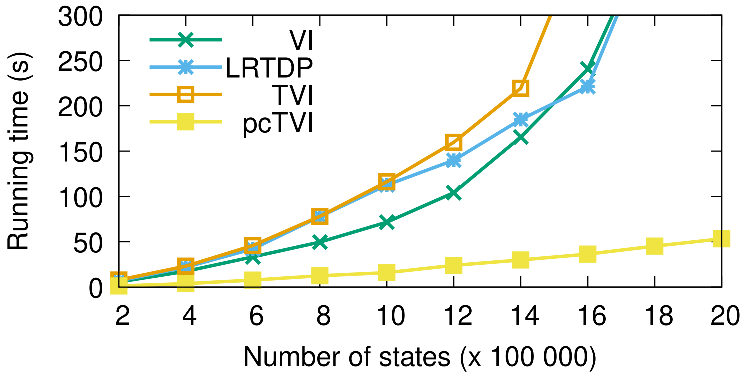

Fig. 2 Average running times (in s) for the Chained-MDP domain with varying number of states and fixed number of chains (32).

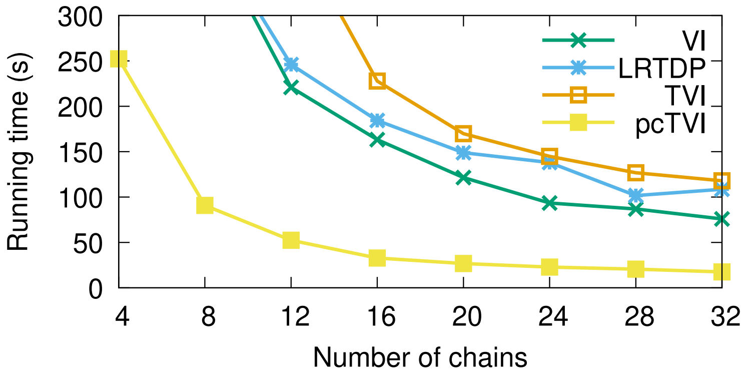

Fig. 3 Average running times (in s) for the Chained-MDP domain with varying number of chains and fixed number of states (1M).

pcTVI and the three other algorithms increases. This indicates that pcTVI is particularly useful on very large MDPs, which are usually needed when considering real-world domains.

Figure 3 presents the obtained results for the same Chained-MDP domain when varying the number of chains and fixing the number of states (1M). When the number of chains increases, the total number of SCCs implicitly increases (which also implies the number of states per SCC decreases). This explains why each tested algorithms becomes faster (TVI becomes faster by design, since it solves SCCs one-by-one without doing useless state backups, and VI and LRTDP become faster due to an increased locality of the considered states in memory, which improves cache performance). The performance of pcTVI increases as the number of chains increases (for the same reason as the others algorithms, but also due to increased parallelization opportunities). We can also observe that for domains with 4 chains only, pcTVI still clearly outperforms the other methods. This means that pcTVI does not need a highly parallel server CPU and can be used on standard 4-core computer.

6 Conclusion

The main contributions of this paper are two-fold. First, we presented a new algorithm, pcTVI, which is, to the best of our knowledge, the first MDP solver that takes into account both the topological structure of the MDP (as in TVI) and the parallel capacities of modern computers (as in P3VI). Second, we introduced a new parametric planning domain, Chained-MDP, which models any situation where different strategies (corresponding to a chain) can reach a goal, but where, once committed to a strategy, it is not possible to switch to a different one. This domain is ideal to evaluate the parallel performance of an MDP solver. Our experiments indicate that pcTVI outperforms the other competing methods (VI, LRTDP, and TVI) on every tested instance of the Chained-MDP domain. Moreover, pcTVI is particularly effective when the considered MDP has many SCC chains (for increased parallelization opportunities) of large size (for decreased overhead of assigning small tasks to the threads). As future work, we plan to investigate ways of pruning provably suboptimal actions, which would allow more SCCs to be found. While this paper focuses on the automated planning side of MDPs, the proposed optimization and parallel computing approaches could also be applied when using MDPs with Reinforcement Learning and other ML algorithms.

Acknowledgements This research has been supported by the *Natural Sciences and Engineering Research Council of Canada* (NSERC) and the *Fonds de Recherche du Québec — Nature et Technologies* (FRQNT).

References

1. Champagne Gareau, J., Beaudry E., Makarenkov, V.: A Fast Electric Vehicle Planner Using Clustering. In: Stud. in Classif., Data Anal., and Knowl. Organ., vol. 5, pp. 17–25. Springer (2021)
2. Mausam, Kolobov, A.: Planning with Markov Decision Processes: An AI Perspective. Morgan & Claypool (2012)
3. Bellman, R.: Dynamic Programming. Prentice Hall (1957)
4. Dai, P., Mausam, Weld, D., Goldsmith, J.: Topological value iteration algorithms. J. Artif. Intell. Res., vol. 42, pp. 181–209 (2011)
5. Bonet, B., Geffner, H.: Labeled RTDP: Improving the Convergence of Real-Time Dynamic Programming, In: Proc. of ICAPS, pp. 12–21 (2013)
6. Hansen, E., Zilberstein, S.: LAO*: A heuristic search algorithm that finds solutions with loops, Artif. Intell., vol. 129, no. 1–2, pp. 35–62 (2001)
7. Wingate, D., Seppi, K.: P3VI: A partitioned, prioritized, parallel value iterator. In: Proc. of the Int. Conf. on Mach. Learn. (ICML), pp. 863–870 (2004)
8. Bertsekas, D.: Dynamic programming and optimal control, vol. 2. Athena scientific Belmont, MA (2001)