J Ind Eng Int (2017) 13:417–426
DOI 10.1007/s40092-017-0197-7

ORIGINAL RESEARCH

Accelerated decomposition techniques for large discounted Markov decision processes

Abdelhadi Larach[1] · **S. Chafik**[1] · **C. Daoui**[1]

Received: 14 October 2016 / Accepted: 14 March 2017 / Published online: 23 March 2017

Abstract Many hierarchical techniques to solve large Markov decision processes (MDPs) are based on the partition of the state space into strongly connected components (SCCs) that can be classified into some levels. In each level, smaller problems named restricted MDPs are solved, and then these partial solutions are combined to obtain the global solution. In this paper, we first propose a novel algorithm, which is a variant of Tarjan's algorithm that simultaneously finds the SCCs and their belonging levels. Second, a new definition of the restricted MDPs is presented to ameliorate some hierarchical solutions in discounted MDPs using value iteration (VI) algorithm based on a list of state-action successors. Finally, a robotic motion-planning example and the experiment results are presented to illustrate the benefit of the proposed decomposition algorithms.

Keywords Markov decision process · Graph theory · Tarjan's algorithm · Strongly connected components · Decomposition

Introduction

The MDP theory is increasingly used in several problems of planning under uncertainty; it has proven tremendously useful in a wide area of disciplines (White 1993), including Communication Network, Games and various applications in Robotics. Other application areas of major importance, such as Precipitation Forecasting or Rainfall Estimations (Valipour 2016a, b), Evapotranspiration Estimations (Valipour et al. 2017; Rezaei and Valipour 2016), Water Lifting Devices (Yannopoulos et al. 2015) who need optimal decision-making are also fields of MDPs applications (Freier et al. 2011; Mishra and Singh 2011; Alighalehbabakhani et al. 2015).

Generally, these real-world problems have very large state spaces; it is impractical to solve them with the classical MDP algorithms, since their computational complexities are at least polynomials in the size of the state space (Littman et al. 1995).

Several studies have focused on tackling the curse of dimensionality: heuristic search (Bonet and Geffner 2003), action elimination techniques introduced by MacQueen (1967), decomposition techniques introduced by Ross and Varadarajan (1991) for constrained limiting average MDP and used by Abbad and Boustique (2003), Abbad and Daoui (2003), Daoui and Abbad (2007), Daoui et al. (2010) in several categories of MDPs (average, discounted and weighted MDPs. The weakness point of these decomposition methods is there polynomial run-time complexity. Dai and Goldsmith (2007) use also a decomposition technique named topological VI algorithm based on a linear Kosaraju–Sharir algorithm (Sharir 1981) but its disadvantage is to repeat, in each iteration, computing some constant terms. The parallelism is a known accelerated solution technique (Chen and Lu 2013). Chafik and Daoui (2015) combine the decomposition technique and parallelism, which leads also to a polynomial run-time decomposition algorithm.

Decomposing an MDP consists on: (1) partitioning the state space into SCCs and classifying these SCCs into some levels; (2) constructing and solving independently, in each

✉ Abdelhadi Larach
larachabdelhadi@gmail.com

[1] Faculty of Sciences and Techniques, Laboratory of Information Processing and Decision Support, Sultan Moulay Slimane University, B.P. 523, Benimellal, Morocco

level, smaller problems named restricted MDPs; (3) combining all the partial solutions to determine the global solution of the original MDP.

In this paper, discounted MDP with finite state and action space is considered. First, the authors present an accelerated version of the VI algorithm based on introducing for each state-action pair a list of successors. They also propose another variant of Tarjan's algorithm using an address table of the nodes.

The main contribution of this paper is a novel algorithm based on Tarjan's algorithm that simultaneously finds the SCCs and classifies them into some levels. This algorithm accelerates the convergence time of the decomposition and so the hierarchical solutions in discounted reward MDP. Formerly, some accelerated hierarchical VI (AHVI) algorithms are proposed. The later AHVI algorithm is based on a new definition of the restricted MDPs.

Finally, a motivating robotic motion-planning example is presented to show the advantage of the proposed decomposition algorithms.

Markov decision processes

A Markov decision process models an agent, which interacts with its environment, taking as input the states of the environment and generating actions as outputs. MDPs are defined as controlled stochastic processes satisfying the Markov property and assigning reward values to state transitions (Bellman 1957; Puterman 1994).

Formally, an MDP is defined by the five-tuple (S, A, T, P, R), where S is the state space in which the process's evolution takes place; A is the set of all possible actions which control the state dynamics; T is the set of time steps where decisions need to be made; P denotes the state transition probability function where $P(S_{t+1} = j|S_t = i, A_t = a) = P_{iaj}$ is the probability of transitioning to a state j when an action a is executed in a state i, S_t (A_t) is a random variable indicating the state (action) at time t; $-R$ provides the reward function defined on state transitions where R_{ia} denotes the reward obtained if the action a is applied in state i.

A strategy π is defined by a sequence $\pi = (\pi^1, \pi^2 \ldots)$ where $\pi^t : H_t \to \Psi$ is a decision rule, $H_t = (S \times A)^{t-1} \times S$ is the set of all histories up to time t, and $\Psi = \{(q_1, \ldots, q_A) \in \mathbb{R}^{|A|} : \sum_{i=1}^{|A|} q_i = 1, \quad q_i \geq 0, \ 1 \leq i \leq |A|\}$ is the set of probability distributions over $A = \bigcup_{i \in S} A(i)$.

A Markov strategy is a strategy π in which π^t depends only on the current state at time t, a stationary strategy is a Markov strategy with identical decision rules, and a deterministic or pure strategy is a stationary strategy whose single decision rule is nonrandomized. The core problem of MDPs is to find an optimal policy that specifies which action should be taken in each state.

Discounted reward MDPs

Let $P_\pi(S_t = j, A_t = a|S_0 = i)$ be the conditional probability that at time t the system is in state j and the action taken is a, given that the initial state is i and the decision maker is a strategy π.

If R_t denotes the variable reward at time t, then for any strategy π and initial state i, the expectation of R_t is given by:

$$E_\pi(R_t, i) = \sum_{\substack{j \in S \\ a \in A(j)}} P_\pi(S_t = j, A_t = a|S_0 = i) R_{ja} \tag{1}$$

In the discounted reward MDP, the value function, which is the expected reward when the process starts with state i and using the policy π is defined by:

$$V_\pi^\alpha(i) = E\left[\sum_{t=1}^{\infty} \alpha^t E_\pi(R_t, i)\right], \quad i \in S \tag{2}$$

where $\alpha \in [0, 1]$ is the discount factor.

The objective is to determine V^*, the maximum expected total discounted reward vector over an infinite horizon.

It is well known (Bellman 1957; Puterman 1994) that the vector V^* satisfies the optimality equation:

$$V(i) = \max_{a \in A(i)} \left\{ R_{ia} + \alpha \sum_{j \in S} P_{iaj} V(j) \right\}, \quad i \in S \tag{3}$$

The actions attaining the maximum in Eq. 3 give rise to an optimal pure policy π given by:

$$\pi^*(i) \in \arg\max_{a \in A(i)} \left\{ R_{ia} + \alpha \sum_{j \in S} P_{iaj} V^*(j) \right\}, \quad i \in S \tag{4}$$

Accelerated value iteration algorithm

Value iteration algorithm is one of the most widely standard iterative methods used for finding optimal or approximately optimal policies in discounted MDPs. In this paragraph, the authors present an accelerated version of the VI algorithm (Algorithm 1) based on introducing for each action a the list of state-action successors denoted by Γ_a^+, where $\Gamma_a^+(i) = \{j \in S : P_{iaj} > 0\}$.

Algorithm 1: Accelerated VI Algorithm

VI (**In:** MDP: S, P, A, R, Γ_a^+, α, ε; **Out**: V^*, π^*)

1: t←0; **For each** i ∈S **Do** $V^t(i)$←0;

3: **Repeat**

4: **For each** i ∈ S **Do**

$$V^{t+1}(i) \leftarrow \max_{a \in A(i)} \left\{ R_{ia} + \alpha \sum_{j \in \Gamma_a^+(i)} P_{iaj} V^t(j) \right\}$$

5: t← t +1;

6: Until (MAX$|V^{t+1}(i) - V^t(i)| < \varepsilon$)

7: **For each** i ∈ S **Do**

$$\pi^*(i) \leftarrow \operatorname*{argmax}_{a \in A(i)} \left\{ R_{ia} + \alpha \sum_{j \in \Gamma_a^+(i)} P_{iaj} V^*(j) \right\}$$

8: **Return** V^*, π^*

The Algorithm 1 permits to accelerate the iterations compared to the classical VI algorithm especially when the number of actions and successors is very less than the number of states. Indeed, the time complexity is reduced from $\mathcal{O}\left(|A||S|^2\right)$ to $\mathcal{O}\left(|\Gamma_a^+||S|\right)$ per iteration where $|\Gamma_a^+|$ denotes the average number of state-action successors.

Decomposition technique

Let $G = (S, U)$ be the associated graph to the original MDP, where the state space S represents the set of nodes and $U = \{(i,j) \in S^2 : \exists a \in A(i), P_{iaj} > 0\}$ is the set of directed arcs. There exist a unique partition $T = \{C_1, C_2, \ldots, C_p\}$of the state space S into SCCs (communicating classes in MDP theory), that can be classified into some levels (see Fig. 1). The level L_0 is formed by all closed classes C_i, that is for all $i \in C_i$, $a \in A(i) : P_{iaj} = 0$ for all $j \notin C_i$. The level L_p is formed by all classes C_i such that the end of any arc emanating from C_i is in some levels L_{p-1}, L_{p-2},…, L_0.

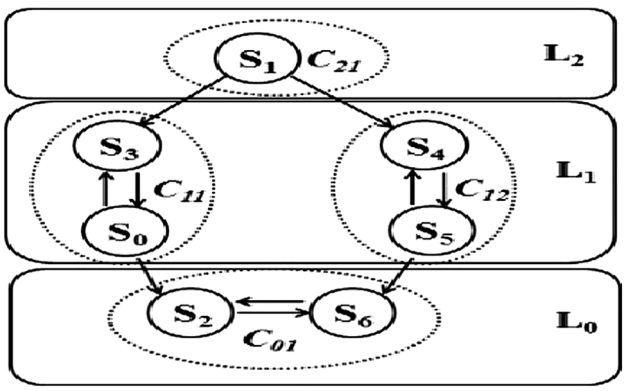

Fig. 1 Example of SCCs and there belonging levels

Restricted MDPs

In the discounted MDPs, Abbad and Daoui (2003) define and construct by induction the restricted MDP_{pk} corresponding to each SCC C_{pk} in level L_p, as follows.

- *State space* $S_{pk} = C_{pk} \cup \bar{C}_{pk}$
  $\bar{C}_{pk} = \{j \in S, j \notin C_{pk} : \exists i \in C_{pk}, P_{iaj} > 0\}$
- *Action space* $\begin{matrix} A_{pk}(i) = A(i) & \text{if } i \in C_{pk} \\ A_{pk}(i) = \theta & \text{if } i \in \bar{C}_{pk} \end{matrix}$
  θ is some fictitious action that keeps the process in the same state where it is.
- *Transition* $\begin{matrix} P_{pk}(j|i,a) = P_{iaj} & \text{if } i \in C_{pk} \\ P_{pk}(j|i,\theta) = 1 & \text{if } i \in \bar{C}_{pk} \end{matrix}$
- *Reward function* $\begin{matrix} R_{pk}(i,a) = R_{ia} & \text{if } i \in C_{pk} \\ R_{pk}(i,a) = (1-\alpha)V^*(i) & \text{if } i \in \bar{C}_{pk} \end{matrix}$
  where $V^*(i)$ is the optimal value calculated in some previous level.

Abbad and Daoui (2003) propose and show the correctness of the following decomposition algorithm (Algorithm 2) that finds an optimal strategy.

Algorithm 2: Decomposition algorithm

1. Find the SCCs in the associated graph G of MDP
2. Find the levels L_p, p=0,…,L.
3. Find the SCCs C_{pk}, k=1,…,K(p) in each level L_p.
4. **For** each level L_p, p=0,…,L **Do**
   For each class C_{pk}, k=1,…,K(p) **Do**
   Solve the restricted MDP_{pk}

In the rest of this paper, the authors consider the decomposition algorithm and propose some optimizations to speed up these steps.

A variant of Tarjan's algorithm

Tarjan's algorithm (Tarjan 1972) is one of the most known approaches used for finding SCCs; it performs a depth-first search (DFS) of the graph; it uses stack to push each visited node S_0, which is associated to the visited number S_0.idx assigned in the order in which nodes appear in the DFS. The 'Low-Return' number S_0.low is the smallest index of a node S_i in the stack reachable from the descendants of S_0, the SCC root is detected when (S_0.low = S_0.idx); the algorithm pop the stack one by one until the state popped is S_0. Tarjan's algorithm requires $\mathcal{O}(n + m)$ space and time where n is the number of nodes and m is the number of edges.

Dijkstra (1982) proposes a variant of Tarjan's algorithm that maintains a stack of possible root S_0 candidates instead of keeping track of low-return values; a SCC can be found by a second DFS starting at S_0. Couvreur (1999) designs a variant of Dijkstra's algorithm for the purpose of finding

```
struct Node{          // Element of DLLV list
   int num;           // Node identifier
   Node *Next;        // Next element
   Node *Previous;    // Previous element
};
struct  List{
  Node *Start;        // Beginning of the list
  Node *End;          // End of the list
};
Node **Addrs;         //Address table of the nodes
List *DLLV;           // Doubly Linked List
```

Fig. 2 C++ code for the definition of the address table and the DLLV list

accepting cycles that can be translated to SCC-based algorithm. Lowe (2014) proposes an iterative version for a multithreading, named concurrent algorithm.

The authors propose another variant (Algorithm 3) using an address table of the nodes (**Addrs**()) stocked in some Doubly Linked List: **DLLV** (see definition in Fig. 2). This list is initialized by an arbitrary element and constructed simultaneously with the associated graph G which is represented by Γ^+: the set of successors. Each node S_0 in **DLLV** is directly accessible using the address value **Addrs**(S_0).

In the initialisation step, the index value of each node (line 3 in Algorithm 3) is set to 0 to indicate that it is not visited; the index value is set to 2 when a node is first generated (line 5 in Algorithm 3); when a SCC is detected the index value of each node (line 22 in Algorithm 3) is set to 1 to indicate that it is in an SCC.

Algorithm 3: A variant of Tarjan's Algorithm

```
1: MTA_SCC ( In: DLLV, Γ+, Out: SCCs_List)
2:      For each node Si∈S Do
3:          Si.idx← 0; //Node not visited
4:      While ( DLLV is non-empty ) Do
5:          index ← 2;
6:          S0←DLLV.end;  //last element
7:          MTA_DFS (S0); // Depth-First Search
8: MTA_DFS (S0)
9:      S0.low← index;
10:     S0.idx← index;
11:     index ← index +1;
12:     For each Si ∈ Γ+(S0) Do
13:         If (Si is not visited ) Then // Si.idx=0
14:             Move_node(Si) //Procedure 1
15:             MTA_DFS (Si);
16:             S0.low ← Min (S0.low , Si.low)
17:         Else If (Si is not in an SCC) Then//Si.idx>1
18:             S0.low ← Min (S0.low , Si.idx)
19:     If (S0.low = S0.idx) Then
20:         SCC =Move_Class_List(S0); //Procedure 2
21:         For all node Sk in SCC Do
22:             Sk.idx ←1; //Node is in an SCC
```

The search starts from the last node of the **DLLV** list (line 6 in Algorithm 3). Each visited node is moved to the end of the list (line 14 in Algorithm 3, Procedure 1).

Procedure 1: Moving a node to the end of list

```
1: Move_node(S0)
2:   Addrs[S0].Previous.Next= Addrs[S0].Next
3:   Addrs[S0].Next.Previous= Addrs[S0].Previous
4:   Addrs[S0].Previous =DLLV.End
5:   DLLV.End = Addrs[S0]
```

When a new SCC is detected with a root node S_0, the sub-list identified by the address value **Addrs**(S_0), is moved to the SCCs list (line 20 in Algorithm 3, Procedure 2) that is a simply linked list in which each element is also a simply linked list (Fig. 3).

Procedure 2: Moving Class to the SCCs list

```
1: Move_Class_List (S0)
2:   SCC.Start=Addrs[S0]
3:   SCC.End=DLLV.End
4:   DLLV.End=Addrs[S0].Previous
5:   SCCs_List.End.Next=SCC
6:   SCCs_List.End=SCC
```

Remark 1 In Tarjan's algorithm, the step in line 4 requires $\mathcal{O}(n)$ time, but in this variant version it only requires $\mathcal{O}(k)$ time where k is generally a low constant compared to n, and depends on the structure of the graph. Eventually, the temporary complexity of Algorithm 3

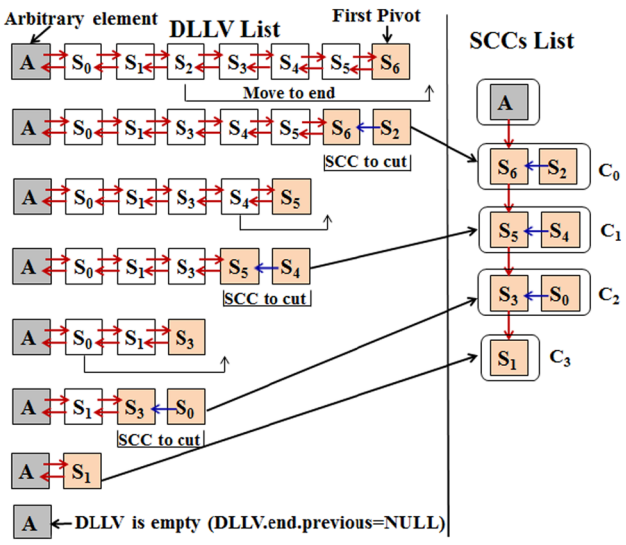

Fig. 3 Illustrative representations of the DLLV list (*left*) and SCCs list (*right*) in the example of Fig. 1

remains similar to Tarjan's algorithm ($\mathcal{O}(n + m)$), but the execution time is reduced.

Note that the first arbitrary element in **DLLV** list is useful to avoid testing if the node to be moved is in the beginning of the list or not.

Modified Tarjan's algorithm for finding SCCs and levels

After the partition of the state space into SCCs, Abbad and Boustique (2003) use the following algorithm to classify these classes into some levels.

Algorithm 4: Decomposition of the SCCs into levels

```
1: Levels (In: SCCs: C_i; Out: L_i)
2:  Ω ← S;
3:  n ← 0;
4:  L_n ← { C_i: C_i is closed in Ω };
5:  Ω ← Ω − L_n;
6:  While (Ω ≠ ∅ ) do
7:      n ← n + 1;
8:      L_n ← { C_i: C_i is closed in Ω };
9:      Ω ← Ω − L_n;
```

This algorithm requires $0(n^2)$ time, to reduce the complexity of the decomposition algorithm (Algorithm 2) the authors propose a new algorithm that finds simultaneously the SCCs and theirs belonging levels. It is based on the following proprieties:

Let C be a SCC: $\mathbb{L}(C)$ design the level of C. For all $x \in C$, $\mathbb{L}(x)$ design the level of x, and $\mathbb{L}(x) = \mathbb{L}(C)$.

Propriety 3.1 Let x and y be two nodes or states. If $y \in \Gamma^+(x)$ then $\mathbb{L}(x) \geq \mathbb{L}(y)$.

Proof Let C_x (C_y) be the SCC containing x (y). Suppose that $\mathbb{L}(x) < \mathbb{L}(y)$ then $\mathbb{L}(C_x) < \mathbb{L}(C_y)$, from the definition of the levels, there is no successors of node x in C_y, this contradicts that $y \in \Gamma^+(x)$.

Propriety 3.2 Let C be the SCC containing a node x and let y be a node such as $y \notin C$, if $y \in \Gamma^+(x)$, then $\mathbb{L}(x) > \mathbb{L}(y)$, y is called external successors.

Proof The Propriety 3.1 imply that $\mathbb{L}(x) \geq \mathbb{L}(y)$; $y \in C$ imply that $\mathbb{L}(y) = \mathbb{L}(C)$ and $x \notin C$ imply that $\mathbb{L}(x) \neq \mathbb{L}(C)$ and $\mathbb{L}(C) = \mathbb{L}(y)$ then $\mathbb{L}(x) > \mathbb{L}(y)$.

Propriety 3.3 Let C_i, $i = 1,\ldots, p$ be the successor's classes of C, $\mathbb{L}(C) = (\max_i \mathbb{L}(C_i)) + 1$; if $(p = 0)$ then $\mathbb{L}(C) = 0$.

Proof It is clear from the Propriety 3.2 and the definition of levels.

Algorithm 5: MTA for SCCs and levels.

```
1:MTA_SCC_Levels(In:DLLV,Γ+, Out: SCCs_List)
2:     For each node S_i∈S Do
3:         S_i.idx← 0; //Node not visited
4:         S_i.Level ← -1//Node not in an SCC
5:     While ( DLLV is non-empty ) Do
6:         index ← 1;
7:         S_0←DLLV.end; //last element
8:         DFS_Levels(S_0); // Depth-First Search
9: DFS_Levels(S_0)
10:    S_0.low← index;
11:    S_0.idx← index;
12:    index ← index +1;
13:    L_0←0; //Level initialisation
14:    For each S_i ∈ Γ+(S_0) Do
15:        If (S_i.idx = 0 ) Then //S_i not visited
16:            Move S_i to the end of DLLV list.
17:            L_i ← DFS_Levels(S_i);
18:            S_0.low ← Min (S_0.low , S_i.low)
19:            L_0 ← Max(L_0 , L_i); // Propriety 3.1
20:        Else If (S_i.Level = -1 ) Then
21:            S_0.low ← Min (S_0.low , S_i.idx)
22:        Else //external successors
23:            L_0 ←Max(L_0,S_i.Level+1);//Propriety 3.2
24:    If (S_0.low=S_0.idx) Then
25:        Move_Class_List( S_0);
26:        For each state S_j in class list Do
27:            S_j.Level← L_0; // Propriety 3.1, 3.2 & 3.3
28:        Return L_0+1; // Propriety 3.2
29:    Return L_0; // Propriety 3.1
```

Theorem 1 *The Algorithm 5 works correctly and runs in* $\mathcal{O}(n + m)$ *time.*

Proof The proof follows from the Proprieties 3.1, 3.2 and 3.3. Indeed, the instructions of lines 19 and 23 transmit the SCC level to its root. In fact, after each recursive call, the function **DFS_Levels**(s) returns the possibly minimum level value to its predecessor. In this case, the level value is updated using the Propriety 3.1 (line 19). If any successor is external, the level value is updated using the Propriety 3.2 (line 23). When a SCC is detected, its level is determined by the level value transmitted to its root (Propriety 3.3). In this case (line 28 in Algorithm 5), the function should return the level value incremented by one (Propriety 3.2).

The Algorithm 5 has the similar structure as Tarjan's algorithm so it runs in $\mathcal{O}(n + m)$ time.

Remark 2 These properties can be easily applied to other varieties of Tarjan's algorithm to simultaneously find the SCCs and their belonging levels, such as Dijkstra's

algorithm (Dijkstra 1982), Couvreur's algorithm (Couvreur 1999), and Sequential or Concurrent Tarjan's algorithm proposed by Lowe (2014).

Using Algorithm 5, the authors present the following first version of accelerated hierarchical VI (AHVI) algorithm (Algorithm 6).

Algorithm 6: AHVI Algorithm (version 1)

AHVI_V1 (In: MDP**; Out:** V^*, π^*)
1. Determine the SCCs and theirs belonging levels using algorithm 5
2. **For** each level Lp, p=0,…,L **Do**
  For each class C_{pk}, in level L_p **Do**
    VI(MDP_{pk}, V^*, π^*)

Remark 3

(i) The restricted MDPs in the same level L_p are independent, so they can be solved in parallel.
(ii) The DFS used in Tarjan's algorithm ensures the dependency order of the restricted MDPs. In non-parallel computing, the MDP can be solved without passing through levels (Algorithm 7).

Algorithm 7: AHVI Algorithm (version 2)

AHVI_V2 (In: MDP**; Out:** V^*, π^*)
1. Find the SCCs $\mathbf{C_i, i=0,..,N_c}$ using algorithm 3
2. **For** each class C_i, i=0,…,N_c **Do**
  VI(MDP_i, V^*, π^*)

Remark 4 The DFS avoids to finds the non-accessible SCCs from the start state S_0. For example, in Fig. 4, the initial state S_0 is in class C_{20}, Only the restricted MDPs corresponding to the SCCs: C_{00}, C_{10}, C_{11} and C_{20} are solved.

New restricted MDPs

In each restricted MDP_{pk} defined by Abbad and Daoui (2003), the Eq. 3 can be decomposed as follows:

$$V_{pk}(i) = \max_{a\in A(i)}\left\{R_{ia} + \alpha\sum_{j\in\bar{\Gamma}_a^+(i)} P_{iaj}V^*(j) + \alpha\sum_{j\in\hat{\Gamma}_a^+(i)} P_{iaj}V_{pk}(j)\right\}, \quad i\in S_{pk} \tag{5}$$

where $\bar{\Gamma}_a^+$ is the set of external successors of C_{pk} relatively to action a, $\hat{\Gamma}_a^+$ is the set of internal successors and $V^*(j)$ is the optimal value of the state j calculated in some previous level.

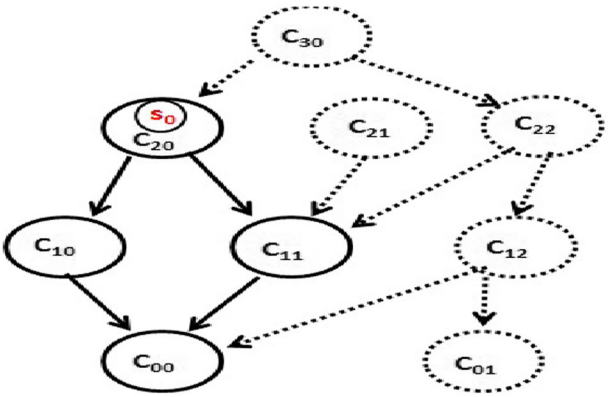

Fig. 4 Example of an aggregated acyclic graph, some SCCs are not reachable from start state S_0

The term $\alpha\sum_{j\in\bar{\Gamma}_a^+(i)} P_{iaj}V^*(j)$ is a constant value in the level L_p, it is not optimal to recalculate this term at each iteration and it can be called the resulting reward for the state-action (i, a) in the previous levels. So, the authors propose a new definition of the restricted MDP_{pk} as follows:

- *State space* $S_{pk} = C_{pk}$
- *Action space* $A_{pk}(i) = A(i),\ i \in C_{pk}$
- *Transition* $P_{pk}(j|i, a) = P_{iaj},\ i \in C_{pk}$
- *Reward function*

$$R_{pk}(i,a) = R_{ia} + \alpha\sum_{j\in\bar{\Gamma}_a^+(i)} P_{iaj}V^*(j), i\in C_{pk} \tag{6}$$

Theorem 2 *Let* $V_{pk}^*(i),\quad i \in C_{pk}$, *the optimal value in the restricted* MDP_{pk}, *then* $V_{pk}^*(i)$ *is equal to the optimal value* $V^*(i)$ *in the original MDP.*

Proof The proof is by induction. For $p = 0$ and for each C_{0k}, the optimal value V_{0k}^* is the unique solution of:

$$V_{0k}(i) = \max_{a\in A(i)}\left\{R_{0k}(i,a) + \alpha\sum_{j\in S_{0k}} P_{iaj}V_{0k}(j)\right\}, i\in S_{0k} \tag{7}$$

The fact that $R_{0k}(i,a) = R_{ia}$, $A_{0k}(i) = A(i)$, $P_{pk}(j|i,a) = P_{iaj}$ and C_{0k} is closed imply that the optimal value V_{0k}^* is the unique solution of:

$$V(i) = \max_{a\in A(i)}\left\{R_{ia} + \alpha\sum_{j\in S} P_{iaj}V(j)\right\}, \quad i\in C_{0k} \tag{8}$$

Then $V_{0k}^*(i) = V^*(i)$ for all $i \in C_{0k}$.

Let $p > 0$ and suppose that the result is true for all levels preceding p. Now, we show that the result is still true for level p.

Let $V_{pk}^*(i),\quad i \in S_{pk}$ be the optimal value in the restricted MDP_{pk}, $V_{pk}^*(i)$ is the unique solution of:

$$V_{pk}(i) = \max_{a \in A(i)} \left\{ R_{pk}(i,a) + \alpha \sum_{j \in S_{pk}} P_{iaj} V_{pk}(j) \right\} \quad (9)$$

The Eq. 6 and the fact that $C_{pk}\ U\ \{L_0,\ldots,L_{p-1}\}$ is closed imply that $V^*_{pk}(i)$ is equal to the optimal value $V^*(i)$ in the original MDP.

Remark 5 The new state space definition does not consider the external successors of each SCC compared to the old definition of restricted MDPs. This reduces the state space and so the time required.

The following procedure constructs the new restricted MDPs.

Procedure 3: New Restricted MDP

1:**NRMDP** (C_i, P, A, Γ_a^+, $\bar{\Gamma}_a^+$)
2: **For all** i ∈ C_i **Do**
3: **For all** a ∈ A(i) **Do**
4: **For all** j ∈ $\bar{\Gamma}_a^+$(i) **Do**
5: $R_{ia} \leftarrow R_{ia} + \alpha P_{iaj} V^*(j)$
6: $\Gamma_a^+(i) = \Gamma_a^+(i) - \{j\}$

For each external successor j of state-action (i, a) the resulted reward is added (line 5 in Procedure 3) and the external successor is eliminated (line 6 in Procedure 3).

Using this procedure, the following third version of AHVI algorithm for discounted MDPs is presented:

Algorithm 8: AHVI Algorithm (version 3)

1: **AHVI_V3 (In:** MDP**; Out:** V^*, π^***)**
2: Find the SCCs $\mathbf{C_i, i=0,..,N_c}$ using algorithm 3
3: **For** each class C_i, i=0,…,N_c **Do**
4: NRMDP (C_i,P, A, Γ_a^+, $\bar{\Gamma}_a^+$)
5: VI(MDP_i, V^*, π^*)

Remark 6 It is more interesting to construct the new restricted MDP during the first iteration of VI algorithm; indeed, the step 4 in Algorithm 8 can be executed in the first iteration of step 5. Also, an external successor can be detected with a different class identifier or a different level value.

Experimental results

The proposed algorithms are tested using Intel(R) Core(TM)2 Duo process (2.6 GHz), C++ implementation, Windows 7 operating system (32 bits) and random models ($\varepsilon = 10^{-5}$, $\alpha = 0.9$, $|\Gamma_a^+| = 20$).

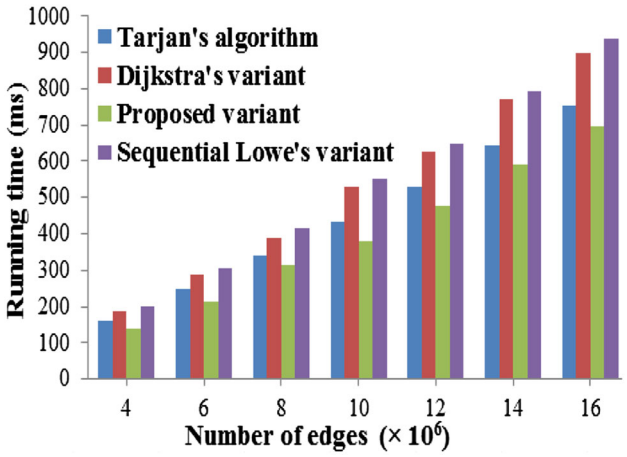

Fig. 5 Running time of several variants of Tarjan's algorithm

Figure 5 shows the running time for several variants of Tarjan's algorithm.

In the sequential Lowe's variant algorithm, each node is pushed in two stacks and popped from two stacks, thus the time needed is longer than Tarjan's algorithm.

In Dijkstra's variant for SCC, instead of keeping track of low-return values that need $\mathcal{O}(m_1)$ time, where m_1 is the number of explored edges, it maintains a stack of possible root candidates and uses a second DFS for SCC members which need $\mathcal{O}(m)$ time and generally $m_1 < m$, where m is the number of edges, so the time is longer than Tarjan's algorithm. But the memory consumption is reduced, since the low-return values are not used.

In the proposed variant, searching all SCCs (not only those reachable from one start state) need $\mathcal{O}(k)$ additional time instead of $\mathcal{O}(n)$ additional time in Tarjan's algorithm and generally $k < n$. Figure 5 shows the reduction time using the proposed variant, and it is efficient since it directly construct the SCCs list for the hierarchical solution of the original MDP, eventually, it requires more memory since it uses doubly linked list.

Figure 6 shows the linearity and the rapidity of the new decomposition algorithm into SCCS and levels compared to the classical algorithm used by Abbad and Boustique (2003). As it can be seen, the decomposition steps in Algorithm 2 becomes so fast using this novel algorithm.

To show the time gained by the new definition of the restricted MDPs, we compare the third version with the second version of AHVI algorithm using a random number of external successors. As it can be seen from Fig. 7, AHVI_V3 reduces the running time.

Note that the number of classes and the size of each class affect the performance of the hierarchical algorithms. In addition, the time gained by the new definition of the restricted MDPs depends on the number of the external successors.

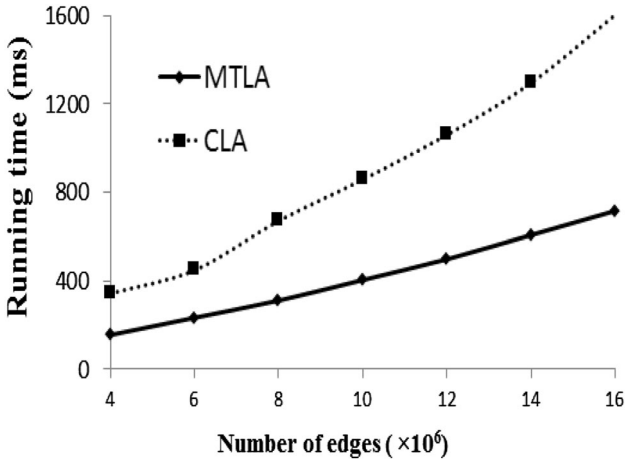

Fig. 6 Running time of the modified Tarajan's levels algorithm (MTLA) compared to the classical levels algorithm (CLA)

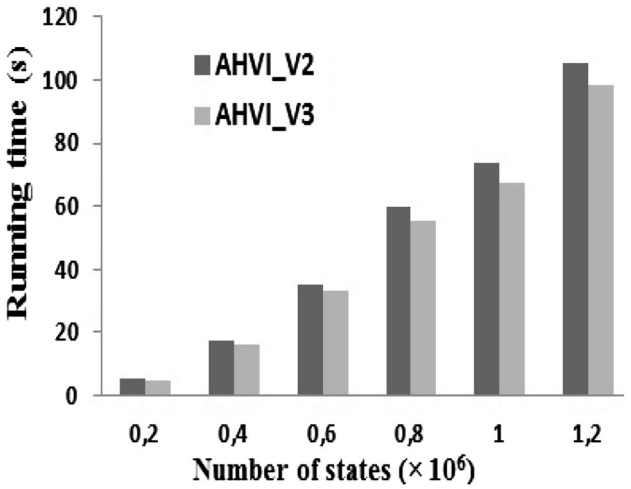

Fig. 7 Comparison between AHVI_v2 and AHVI_v3 algorithms

Problem example

In this section, the authors present an example of applying MDP decomposition in Robotics navigation where the environment structure can be decomposed into some regions. Each region corresponds to a SCC.

Model for Robotics navigation

Applying MDP theory in robotic navigation need a choice of the environment representation and a definition of the five-tuple (*S, A, T, P, R*).

Grid of the environment The grid method is used to model the environment, which is entirely discretized according to a regular grid.

States space Using the grid method, the state space is therefore a set of grids; each grid cell has an associated value stating, obstacle, free or goal state. The obstacle state can be eliminated from the state space.

Actions space The robot can be controlled through nine actions: the eight compass directions and the fictitious action θ that keep the robot on the spot. The actions that move the robot to an obstacle are eliminated. In a goal state the possible action is θ.

Note that the average number of state-action successors is very less than the number of states, so the proposed AVI algorithm (Algorithm 1) is efficient in this case.

Reward function The transition to a free state is characterized by a cost of energy equal to x when the action is diagonal, and $\frac{x}{\sqrt{2}}$ when the action is horizontal or vertical. For any transition to goal state, the reward value is equal to the constant R_b that is very higher than x.

Transition function The transition function defines the uncertainty due to the effects of the actions; it is a problem data and can be determined by reinforcement learning. In this problem, the authors use the example of transition function indicated in Fig. 8.

The transition function is similar for the other actions.

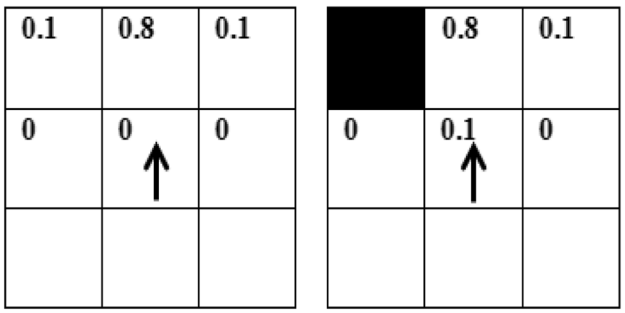

Fig. 8 Example of transition function for the action indicated by arrow (the probability of transition to the desired state is equal to 0.8)

Robot oriented to the nearest goal

Consider a robot that moves in a corridor, whenever it needs energy, it can load its battery in the nearest position in one of the desks. The battery load positions are the goal states. Figure 9 shows an example of an environment with four desks.

In each desk exists a goal state (G), the black grid is an obstacle and the blue grids represent the border of the environment.

The state space is communicant, but some states (entrances of the desks, cells with an arrow) can only perform a single deterministic action (direction of the small arrow in Fig. 9). Once the system reaches one of these states, it cannot go back. Hence, using a modified version of Tarjan's algorithm, the state space is decomposed into five

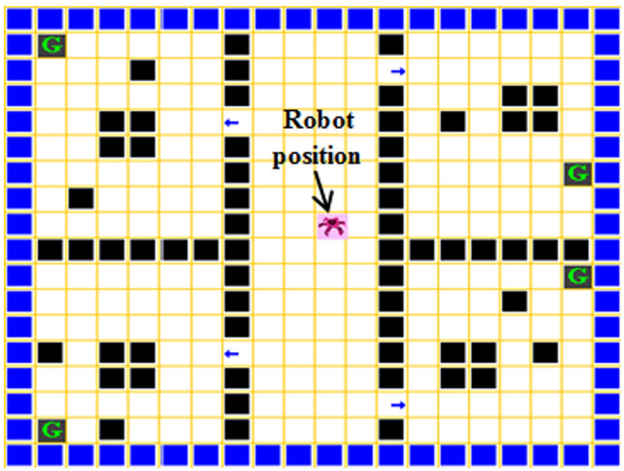

Fig. 9 Example of an environment with four desks

regions. Each region corresponds to one restricted MDPs (Fig. 10).

The fourth restricted MDPs: MDP_{pk}, $k = 0,\ldots,3$ in level $p = 0$ can be solved in parallel, thereafter the fifth restricted MDP: MDP_{p0} in level $p = 1$ is solved.

Figure 11 shows the optimal strategy generated by Algorithm 1 without using the decomposition. The arrow indicates the optimal action.

Using decomposition, the smaller restricted MDPs: MDP_{0k}, $k = 0,\ldots,3$ are solved independently; the optimal strategy generated for each restricted MDP_{0k} in level 0 is shown in Fig. 12.

After solving the fourth restricted MDPs in level 0, the fifth new restricted MDP_{10} in level 1 is constructed and solved; Fig. 13 shows the obtained solution.

The abstract goal state represents an external successor. It is eliminated during the construction of the new restricted MDP_{10}. In old definition of the restricted MDPs, it is considered as a goal state.

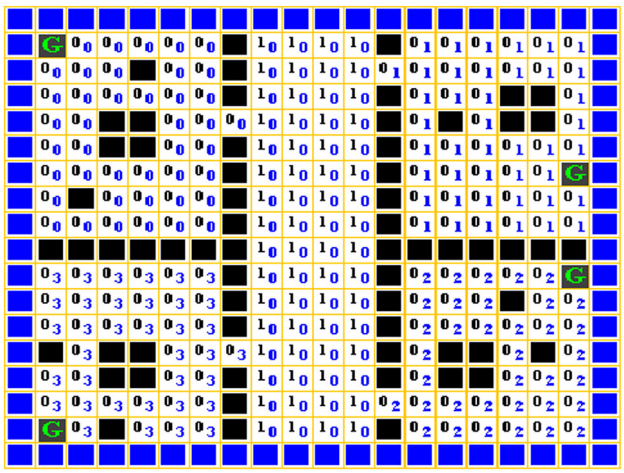

Fig. 10 SCC number (*blue*) and level value (*black*) for the state space in the example of Fig. 9

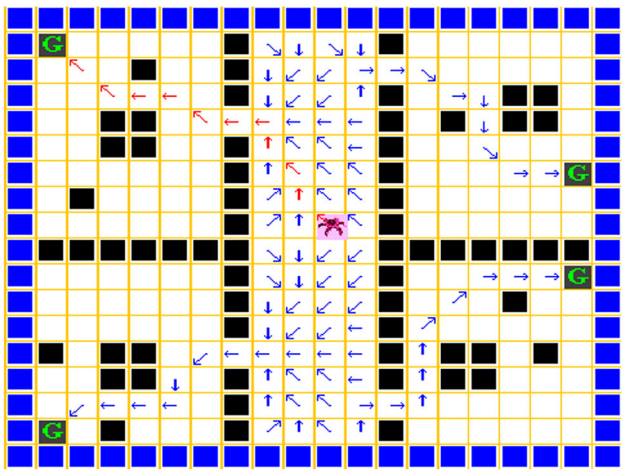

Fig. 11 An optimal strategy calculated without using the decomposition technique

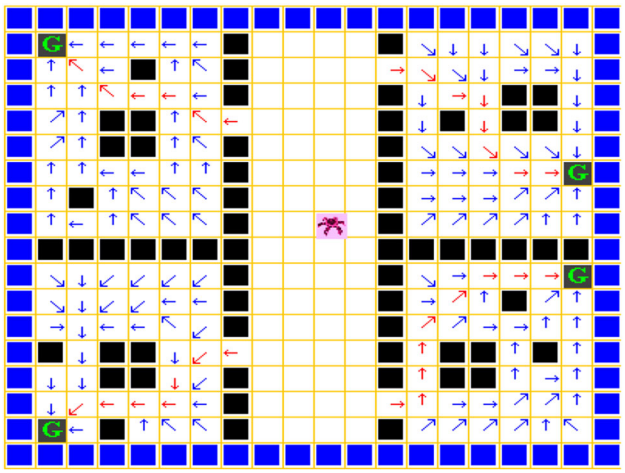

Fig. 12 An optimal strategy calculated for each restricted MDP_{0k} in level 0

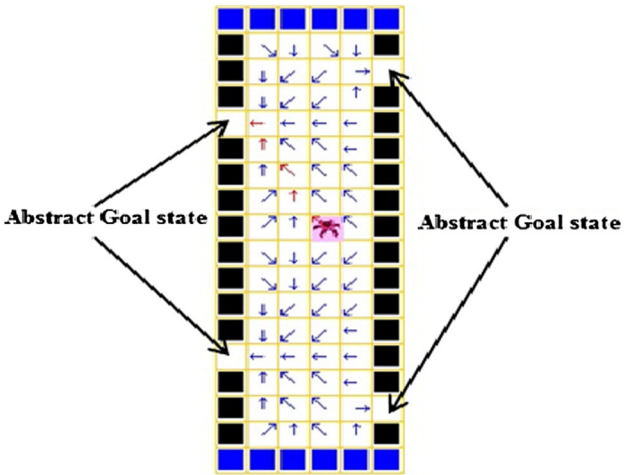

Fig. 13 An optimal strategy for the new restricted MDP_{10} in level 1, corresponding to the corridor

It can be seen from Figs. 12 and 13 that the optimal solution is similar to that obtained in Fig. 11 without using the decomposition technique. Thus, the original problem is decomposed into five smaller problems, which reduces the time complexity.

Conclusions

In this work, the authors have proposed a new algorithm that simultaneously finds the SCCs and classifies them into some levels used to accelerate the decomposition steps. The possible uses of this algorithm are certainly not limited to distributed or parallel solution for discounted MDPs. The authors have also proposed an accelerated version of HVI algorithm using a list of state–action successors and a new definition of the restricted MDPs; this allows us to reduce the time required. In future works, we try to combine the action elimination techniques with the proposed decomposition algorithms for more acceleration. The distributed solution and various areas of applications are also our perspectives.

References

Abbad M, Boustique H (2003) A decomposition algorithm for limiting average Markov decision process. Oper Res Lett 31(6):473–476

Abbad M, Daoui C (2003) Hierarchical algorithms for discounted and weighted Markov Decision Processes. Math Methods Oper Res 58(2):237–245

Alighalehbabakhani F et al (2015) Comparative evaluation of three distinct energy optimization tools applied to real water network (Monroe). Sustain Comput Inform Syst 8:29–35

Bellman RE (1957) Dynamic programming. Princeton University Press, Princeton, NJ

Bonet B, Geffner H (2003) Faster heuristic search algorithms for planning with uncertainty and full feedback. In: IJCAI, pp 1233–1238

Chafik S, Daoui C (2015) A modified value iteration algorithm for discounted Markov decision process. J Electron Commerce Organ 13(3):47–57

Chen P, Lu L (2013) Markov decision process parallel value iteration algorithm on GPU. In: International conference on information science and computer applications, ISCA, 2013, Atlantis Press

Couvreur J-M (1999) On-the-fly verification of linear temporal logic. In: Wing JM, Woodcock J, Davies J (eds) FM'99—formal methods. FM 1999, in computer science, vol 1708. Springer, Berlin

Dai P, Goldsmith J (2007) Topological value iteration algorithm for Markov decision process. In: IJCAI, pp 1860–1865

Daoui C, Abbad M (2007) One some algorithms for limiting average Markov decision processes. Oper Res Lett 35(2):261–266

Daoui C, Abbad M, Tkiouat M (2010) Exact decomposition approaches for Markov decision processes: a survey. Adv Oper Res 2010:1–19

Dijkstra E (1982) Finding the maximum strong components in a directed graph. Selected writings on computing: a personal perspective, texts and monographs in computer science. Springer, New York, pp 22–30

Freier KP, Schneider UA, Finckh M (2011) Dynamic interactions between vegetation and land use in semi-arid Morocco: using a Markov process for modelling rangelands under climate change. Agric Ecosyst Environ 140(3):462–472

Littman ML, Dean TL, Kaelbling LP (1995) On the complexity of solving Markov decision problems. In: Proceedings of the eleventh conference on uncertainty in artificial intelligence. Morgan Kaufmann Publishers Inc., pp 394–402

Lowe G (2014) Concurrent depth-first search algorithms. tools and algorithms for the construction and analysis of systems. In: Lecture notes in computer science, vol. 8413, p 202

MacQueen JA (1967) Test for suboptimal actions in Markov decision problems. Oper Res 15(3):559–561

Mishra AK, Singh VP (2011) Drought modelling. A review. J Hydrol 403(1):157–175

Puterman M (1994) Markov decision processes: discrete stochastic dynamic programming. Wiley, New York

Rezaei M, Valipour M (2016) Modeling evapotranspiration to increase the accuracy of the estimations based on the climatic parameters. Water Conserv Sci Eng 1(3):197–207

Ross KW, Varadarajan R (1991) Multi-chain Markov decision processes with a sample path constraint: a decomposition approach. Math Oper Res 16:195–207

Sharir M (1981) A strong-connectivity algorithm and its applications in data flow analysis. Comput Math Appl 7(1):67–72

Tarjan R (1972) Depth-first search and linear graph algorithms. SIAM J Comput 1(2):146–160

Valipour M (2016a) Optimization of neural networks for precipitation analysis in a humid region to detect drought and wet year alarms. Meterol Appl J 23:91–100

Valipour M (2016b) How much meteorological information is necessary to achieve reliable accuracy for rainfall estimations? Agriculture 6(4):53

Valipour M, Gholami Sefidkouhi MA, Raeini-Sarjaz M (2017) Selecting the best model to estimate potential evapotranspiration with respect to climate change and magnitudes of extreme events. Agric Water Manag 180:50–60

White DJ (1993) A survey of applications of Markov decision process. J Oper Res Soc 44(11):1069–1073

Yannopoulos SI, Lyberatos G, Theodossiou N, Li W, Valipour M, Tamburrino A, Angelakis AN (2015) Evolution of water lifting devices (pumps) over the centuries worldwide. Water 7(9):5031–5060