Artificial Intelligence 297 (2021) 103499

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

Algorithms and conditional lower bounds for planning problems

Krishnendu Chatterjee [a], Wolfgang Dvořák [b], Monika Henzinger [c], Alexander Svozil [c,*]

[a] IST Austria, Am Campus 1, A-3400 Klosterneuburg, Austria
[b] TU Wien, Institute of Logic and Computation, Favoritenstraße 9–11, A-1040 Wien, Austria
[c] University of Vienna, Faculty of Computer Science, Währinger Strasse 29, A-1090 Wien, Austria

ARTICLE INFO

Article history:
Received 3 November 2020
Received in revised form 8 March 2021
Accepted 8 March 2021
Available online 16 March 2021

Keywords:
Graph games
Conditional lower bounds
Adversarial planning
Strong exponential time hypothesis
Probabilistic planning

ABSTRACT

We consider planning problems for graphs, Markov Decision Processes (MDPs), and games on graphs in an explicit state space. While graphs represent the most basic planning model, MDPs represent interaction with nature and games on graphs represent interaction with an adversarial environment. We consider two planning problems with k different target sets: (a) the coverage problem asks whether there is a plan for each individual target set; and (b) the sequential target reachability problem asks whether the targets can be reached in a given sequence. For the coverage problem, we present a linear-time algorithm for graphs, and quadratic conditional lower bound for MDPs and games on graphs. For the sequential target problem, we present a linear-time algorithm for graphs, a sub-quadratic algorithm for MDPs, and a quadratic conditional lower bound for games on graphs. Our results with conditional lower bounds, based on the boolean matrix multiplication (BMM) conjecture and strong exponential time hypothesis (SETH), establish (i) model-separation results showing that for the coverage problem MDPs and games on graphs are harder than graphs, and for the sequential reachability problem games on graphs are harder than MDPs and graphs; and (ii) problem-separation results showing that for MDPs the coverage problem is harder than the sequential target problem.

1. Introduction

One of the fundamental algorithmic problems in artificial intelligence is the *planning problem* [1,2]. The most basic planning problem is the *Discrete Feasible Planning problem* [1]. The problem has a finite *state space* and a finite amount of *actions* for each state. Starting from an *initial state*, the planner repeatedly chooses an available action at the current state which, as a result, produces a new current state as described by a *state transition function*. The question is if the planner can produce a state which is in a certain subset of the state space called *goal* or *target*.

* Corresponding author.
E-mail addresses: krish.chat@ist.ac.at (K. Chatterjee), dvorak@dbai.tuwien.ac.at (W. Dvořák), monika.henzinger@univie.ac.at (M. Henzinger), alexander.svozil@univie.ac.at (A. Svozil).

https://doi.org/10.1016/j.artint.2021.103499
0004-3702/

Planning models. We study this problem in the following classical models:

- *Graphs.* Discrete Feasible Planning can be directly translated into a graph search problem: The vertices in the graph describe the state space and for every action in a state, there is an edge to the vertex which corresponds to the new state given by the state transition function [1,2].
- *MDPs.* In the presence of interaction with nature, the graph model is extended with probabilities or stochastic transitions, which gives rise to Markov Decision Processes (MDPs) [3–7].
- *Games on graphs.* In the presence of interaction with an adversarial environment, the graph model is extended to game graphs (or AND-OR graphs) [8,9].

Planning problems. The planner tries to solve a planning problem given one of the above-described planning models. The starting position is not restricted to the vertices controlled by the planner but can be any kind of vertex in the considered model. We consider the following basic planning problems:

- *Basic target reachability.* Given a *target set* T of vertices the goal is to determine if some vertex in T is reachable from the starting position.
- *Coverage.* In the coverage problem we are given k different target sets, namely, $T_1, \ldots, T_k$, and a starting vertex. The coverage problem asks whether we can achieve basic target reachability for all target sets T_i where $1 \leq i \leq k$. Coverage models the following scenario: Consider a robot stationed in an outpost with k different locations of interest. If an event or an attack happens in one of the locations, then that location must be reached. However, the location of the event or the attack is not known in advance and the robot must be prepared that the target set could be any of the k target sets.
- *AllCoverage*. In the AllCoverage problem there are again k different target sets $T_1, \ldots, T_k$ but in contrast to Coverage we want to determine *all starting positions* where Coverage with $T_1, \ldots, T_k$ holds. This corresponds to finding a viable outpost for robot.
- *Sequential target reachability.* In the *sequential target reachability* problem we are given k different target sets, namely, $T_1, T_2, \ldots, T_k$ and a starting position. The goal is to output whether we can first reach T_1, then T_2 and so on up to T_k from the starting position. This represents the scenario that the tasks must be completed in a sequence by the planner.

The above are natural planning problems and have been studied widely in the literature, e.g., in robot planning [10–12].

Basic planning questions. For the above problems the *basic* planning questions are as follows: (a) for graphs, the question is whether there exists a plan (or a path) such that the planning problem is solved; (b) for MDPs, the basic question is whether there exists a policy such that the planning problems are satisfied almost-surely (i.e., with probability 1); and (c) for games on graphs, the basic question is whether there exists a policy that solves the planning problem irrespective of the choices of the adversary. The almost-sure satisfaction for MDPs is also known as the strong cyclic planning in the planning literature [13], and games on graphs question represent planning in the presence of a worst-case adversary [8,9] (aka adversarial planning, strong planning [14], or conformant/contingent planning [15–17]).

Algorithmic study. In this work, we study the planning problems for graphs, MDPs, and games on graphs algorithmically. For all the above questions, polynomial-time algorithms exist. When polynomial-time algorithms exist, proving an unconditional lower bound is extremely rare. A new approach in complexity theory aims to establish a conditional lower bound (CLB) based on a well-known conjecture. Two standard conjectures for CLBs are as follows: The (a) *Boolean matrix multiplication (BMM) conjecture* states that there is no sub-cubic combinatorial algorithm for boolean matrix multiplication; and the (b) *Strong exponential-time hypothesis (SETH)* states that there is no sub-exponential time algorithm for the k-SAT problem when k grows to infinity. Many CLBs have been established based on the above conjectures, e.g., for dynamic graph algorithms and string matching [18,19].

Previous results and our contributions. We denote by n and m the number of vertices and edges of the underlying model, and k denotes the number of different target sets. The $\widetilde{O}$ notation hides poly-log factors, e.g. $O(m(\log n)^4) = \widetilde{O}(m)$. We call a running time *near-linear* if it is linear in the input but has some additional poly-logarithmic factor, e.g. $O(m(\log n)^4)$. For the basic target reachability problem, while the graphs and games on graphs problem can be solved in linear time [20,21], the current best-known bound for MDPs is $\widetilde{O}(m)$ [22, Theorem 12]. For the coverage and sequential target reachability, an $O(k \cdot m)$ upper bound follows for graphs and games on graphs, and an $\widetilde{O}(k \cdot m)$ upper bound follows for MDPs. Our contributions are as follows:

1. *Coverage problem:* First, we present an $O(m + \sum_{i=1}^{k} |T_i|)$ time algorithm for graphs; second, we present an $\Omega(k \cdot m)$ lower bound for MDPs and games on graphs, both under the BMM conjecture and the SETH. For graphs our upper bound is in linear time, however, if each $|T_i|$ is constant and $k = \theta(n)$, for MDPs and games on graphs the CLB is quadratic.
2. *Sequential target problem:* First, we present an $O(m + \sum_{i=1}^{k} |T_i|)$ time algorithm for graphs; second, we present an $\widetilde{O}(m + \sum_{i=1}^{k} |T_i|)$ time algorithm for MDPs; and third, we present an $\Omega(k \cdot m)$ lower bound for games on graphs, both under the BMM conjecture and the SETH.

Table 1
Algorithmic bounds where n and m are the numbers of vertices and edges of the underlying model, and k denotes the number of different target sets. The $\tilde{\Omega}(\cdot)$ bounds are conditional lower bounds (CLBs) under the BMM conjecture and SETH. They establish that polynomial improvements over the given bound are not possible, however, polylogarithmic improvements are not excluded. Note that CLBs are quadratic for $k = \Theta(n)$. The new results are highlighted in boldface.

Objectives	Graphs		MDPs		Games	
	Upper B.	Lower B.	Upper B.	Lower B.	Upper B.	Lower B.
Basic target	$O(m)$		$\widetilde{O}(m)$		$O(m)$	
Coverage	$O(m + \sum_{i=1}^{k} \lvert T_i \rvert)$		$\widetilde{O}(k \cdot m)$	$\tilde{\mathbf{\Omega}}(\mathbf{k \cdot m})$ (Theorem 1)	$O(k \cdot m)$	$\tilde{\mathbf{\Omega}}(\mathbf{k \cdot m})$ (Theorem 2)
AllCoverage	$O(k \cdot m)$	$\tilde{\mathbf{\Omega}}(\mathbf{k \cdot m})$ (Theorem 3)	$O(k \cdot m)$	$\tilde{\mathbf{\Omega}}(\mathbf{k \cdot m})$ (Theorem 3)	$O(k \cdot m)$	$\tilde{\mathbf{\Omega}}(\mathbf{k \cdot m})$ (Theorem 3)
Sequential target	$\mathbf{O(m + \sum_{i=1}^{k} \lvert T_i \rvert)}$ (Theorem 4)		$\widetilde{\mathbf{O}}(\mathbf{m + \sum_{i=1}^{k} \lvert T_i \rvert})$ (Theorem 5)		$O(k \cdot m)$	$\tilde{\mathbf{\Omega}}(\mathbf{k \cdot m})$ (Theorem 6)

The summary of the results is presented in Table 1. Our most interesting results are the conditional lower bounds for MDPs and game graphs for the coverage problem, the sub-quadratic algorithm for MDPs with sequential targets, and the conditional lower bound for game graphs with sequential targets.

Practical significance. The sequential reachability and coverage problems we consider are the tasks defined in [10], where the problems have been studied for games on graphs and mentioned as future work for MDPs. The applications of these problems have been demonstrated in robotics applications. We present a complete algorithmic picture for games on graphs and MDPs, settling open questions related to games and future work mentioned in [10].

Theoretical significance. Our results present a very interesting algorithmic picture for the natural planning questions in the fundamental models.

1. First, we establish results showing that some models are harder than others. More precisely,
   - for the basic target reachability problem, the MDP model seems harder than graphs and games on graphs (linear-time algorithm for graphs and games on graphs, and only near-linear time algorithms are known for MDPs);
   - for the coverage problem, MDPs and games on graphs are harder than graphs (linear-time algorithm for graphs and quadratic CLBs for MDPs and games on graphs);
   - for the sequential target problem, games on graphs are harder than MDPs and graphs (linear-time upper bound for graphs and sub-quadratic upper bound for MDPs, whereas quadratic CLB for games on graphs).

   In summary, we establish model-separation results with CLBs: For the coverage problem, MDPs and games on graphs are algorithmically harder than graphs; and for the sequential target problem, games on graphs are algorithmically harder than MDPs and graphs.
2. Second, we also establish problem-separation results. For the model of MDPs consider the different problems: Both for basic target and sequential target reachability the upper bound is sub-quadratic and in contrast to the coverage problem we establish a quadratic CLB.

Further related work In this work, our focus lies on the algorithmic complexity of fundamental planning problems and we consider *explicit state-space* graphs, MDPs, and game graphs, where the complexities are polynomial. The explicit model and algorithms for it are widely considered: For example, in LTL Synthesis [10,23–25], Probabilistic Planning [26–29], Nondeterministic Planning [30–34], Contingent Planning [35,36] and Verification [37]. In factored models such as STRIPS and SAS+ the complexities are higher (PSPACE-complete and NP-complete [38,39]), and then heuristics are the focus (e.g., [9]) rather than the exact algorithmic complexity. Notable exceptions are

1. the work on parameterized complexity of planning problems (e.g., [40]),
2. conditional lower bounds based on the ETH [41] showing that certain general propositional planning problems (e.g., propositional STRIPS with negative goals (PSN)) do not admit algorithms with running times of the form $2^{|P|^c}$ for instance size $|P|$ and concrete constants $c > 0$ [42,43],
3. conditional lower bounds based on the SETH of the form $2^{(1+\varepsilon)v} \cdot poly(|P|)$ where v is the number of variables and $\varepsilon > 0$ for very large subclasses PSN [43],
4. conditional lower bounds based on the graph colorability problem of the form $2^{v/2} \cdot poly(v)$ [43],
5. conditional lower bounds based on the ETH showing that the minimum constraint removal problem, a well-studied problem in both robotic motion planning, does not admit algorithms with running times of the form $2^{o(n)}$ [44].

2. Preliminaries

We first present formal definitions of the studied problems and then provide the necessary background on conditional lower bounds that we will use as a technique to classify the complexity of the problems throughout our paper.

2.1. Definition of the problems

Markov Decision Processes (MDPs). A *Markov decision process (MDP)* $P = ((V, E), \langle V_1, V_R \rangle, \delta)$ consists of a finite set of vertices V partitioned into the player-1 vertices V_1 and the random vertices V_R, a finite set of edges $E \subseteq (V \times V)$, and a probabilistic transition function δ. The probabilistic transition function maps every random vertex in V_R to an element of $\mathcal{D}(V)$, where $\mathcal{D}(V)$ is the set of probability distributions over the set of vertices V. A random vertex v has an edge to a vertex $w \in V$, i.e. $(v, w) \in E$ if and only if $\delta(v)[w] > 0$. For simplicity, for all random vertices v, we let $\delta(v)$ be the uniform distribution over vertices u with $(v, u) \in E$. We explain in Remark 2 why this assumption is without loss of generality. When we say that we *contract* a set of vertices X into a player-1 vertex v the *resulting MDP* P' has the vertices $V' = (V \setminus X) \cup \{v\}$, the edge set E', which contains (1) all edges of E without a vertex in X and (2) for all edges (u, x), (x, u) where $x \in X$ and $u \in V \setminus X$ in P we include the edges (u, v) and (v, u) respectively. The player-1 vertices V'_1 are defined as $(V_1 \setminus X) \cup \{v\}$ and the random vertices V'_R are $V_R \setminus X$. The new probabilistic transition function $\delta'(v)$ for $v \in V'_R$ is, again, the uniform distribution over vertices u with $(v, u) \in E$. We compare our definition of MDPs with the definition of MDPs used in most planning and AI literature in Remark 1.

Game Graphs. A game graph $\Gamma = ((V, E), \langle V_1, V_2 \rangle)$ consists of a finite set of vertices V, a finite set of edges E and a partition of the vertices V into player-1 vertices V_1 and the adversarial player-2 vertices V_2. We sometimes write player-x ($x \in \{1, 2\}$) and to describe the adversarial player we write player-$\bar{x}$, i.e., $\bar{x} = 2$ if x is 1 and $\bar{x} = 1$ if x is 2.

Graphs. A graph $G = (V, E)$ is a special case of an MDP with $V_R = \emptyset$ as well as a special case of a game graphs with $V_2 = \emptyset$. Let $Out(v) = \{u \in V \mid (v, u) \in E\}$ describe the set of successor vertices of v. The set $In(v) = \{u \in V \mid (u, v) \in E\}$ describes the set of predecessors of the vertex v. We say that two vertices u and v are strongly connected if there is a path from u to v and vice-versa.

Remark 1. A standard way to define MDPs, e.g. [7], is to consider vertices with actions and the probabilistic transition function is defined for every vertex and action. In our model, the choice of actions is represented as the choice of edges at player-1 vertices and the probabilistic transition function is represented by the random vertices. This allows us to treat MDPs and game graphs uniformly, and graphs can be described easily as a special case of MDPs.

Plays. We assume without loss of generality that every vertex has an outgoing edge.[1] A *play* is an infinite sequence $\omega = \langle v_0, v_1, v_2, \ldots \rangle$ of vertices such that each $(v_{i-1}, v_i) \in E$ for all $i \geq 1$. We denote the *set of all plays* with Ω. A play is initialized by placing a token on an initial vertex. If the token is on a vertex owned by a player (such as player 1 in MDPs, or player 1/player 2 in game graphs), then the respective player moves the token along one of the outgoing edges, whereas if the token is at a random vertex $v \in V_R$, then the next vertex is chosen according to the probability distribution $\delta(v)$. Thus an infinite sequence of vertices (or an infinite walk) is formed which is a play.

Policies. Policies are recipes for players to extend finite prefixes of plays (denoted with V^*). Formally, a player-x *policy* is a function $\sigma_x : V^* \cdot V_x \mapsto V$ which maps every finite prefix $\omega \in V^* \cdot V_x$ of a play that ends in a player-x vertex v to a successor vertex $\sigma_x(\omega) \in V$, i.e., $(v, \sigma_x(\omega)) \in E$. A player-x policy is *memoryless* if $\sigma_x(\omega) = \sigma_x(\omega')$ for all $\omega, \omega' \in V^* \cdot V_x$ that end in the same vertex $v \in V_x$, i.e., the policy does not depend on the entire prefix, but only on the last vertex.

Outcome of policies. The *outcome of a policy* is defined as follows for the models:

- In graphs, given a starting vertex, a policy for player 1 induces a unique play in the graph by applying the player-1 policy at every vertex.
- In game graphs, given a starting vertex v, and policies σ, π for player 1 and player 2 respectively, we define the unique play $\omega(v, \sigma, \pi) = \langle v_0, v_1, v_2, \ldots \rangle$, such that $v_0 = v$ and for all $i \geq 0$ if $v_i \in V_1$ then $\sigma(v_i) = v_{i+1}$ and if $v_i \in V_2$, then $\pi(v_i) = v_{i+1}$.
- In MDPs given a starting vertex v a policy for player 1 induces a distribution over the possible plays since random vertices choose their successor according to the probabilistic transition function δ.

Objectives and winning. An *objective* ϕ is a subset of Ω (the set of all plays) and describes the "winning plays". A play $\omega \in \Omega$ *achieves or is in* the objective if $\omega \in \phi$. We consider the following notion of "winning":

- *Almost-sure winning.* In MDPs, let $\Pr_v^\sigma(\phi)$ denote the probability that a play starting at vertex $v \in V$, is in ϕ when player 1 plays policy σ. A policy σ is *almost-sure winning (a.s. winning)* from a vertex $v \in V$ for an objective ϕ if and only if $\Pr_v^\sigma(\phi) = 1$. This notion is also considered strong cyclic planning [13].
- *Winning.* In game graphs a policy σ is *winning for player 1* starting at vertex v for an objective ϕ if and only if for any player-2 policy π we have $\omega(v, \sigma, \pi) \in \phi$.

[1] If a vertex v has no outgoing edge we simply add an edge (v, v). These additional edges do not affect any of the reachability notions we consider in this work.

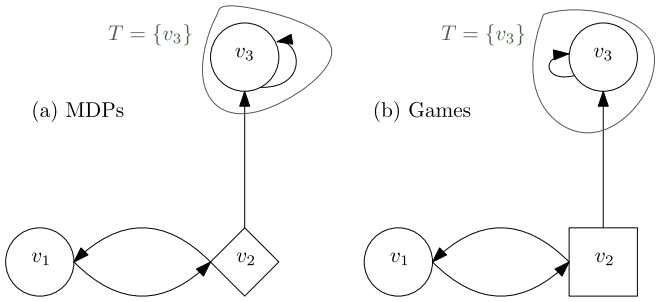

Fig. 1. Example illustrating the difference between (a) MDPs and (b) game graphs for the reachability objective *Reach*(T).

Note that in the special case of graphs both of the above winning notions requires that there exists a play from v that achieves the objective.

Remark 2. In MDPs we consider the notion of a.s. winning where the precise transition probabilities of the transition function δ do not matter because only the support of the transition function is relevant. The a.s. winning notion we use corresponds to the strong cyclic planning problem. Intuitively, if we visit a random vertex in an MDP infinitely often then all its successors are visited infinitely often. Therefore, when we consider the almost-sure winning condition in MDPs, only the underlying graph structure along with the partition of vertices into player-1 and random vertices is relevant, and the transition function δ can be treated as a uniform distribution over the support.

We have defined the notion of objectives in general above, and below we consider specific objectives and queries that are natural in planning problems. They are all variants of the most fundamental objectives in planning, namely, reachability objectives.

Basic Target Reachability. For a *target set* $T \subseteq V$, the basic target reachability objective is the set of plays that contain a vertex of T, i.e., $\mathit{Reach}(T) = \{\langle v_0, v_1, v_2, \ldots\rangle \in \Omega \mid \exists j \geq 0 : v_j \in T\}$.

Sequential Target Reachability. For a tuple of vertex sets $\mathcal{T} = (T_1, T_2, \ldots, T_k)$ the sequential target reachability objective is the set of plays that contain a vertex of T_1 followed by a vertex of T_2 and so on up to a vertex of T_k, i.e., $\mathit{Seq}(\mathcal{T}) = \{\langle v_0, v_1, v_2, \ldots\rangle \in \Omega \mid \exists j_1, j_2, \ldots j_k : v_{j_1} \in T_1, v_{j_2} \in T_2, \ldots, v_{j_k} \in T_k \text{ and } j_1 \leq j_2 \leq \cdots \leq j_k\}$.

Coverage and AllCoverage. The Coverage problems cannot be stated as a single objective but are formulated as a query involving several objectives. That is, for k vertex sets, namely $T_1, T_2, \ldots, T_k$, the coverage query $\mathit{Coverage}(T_1, \ldots, T_k)$ asks whether for each $1 \leq i \leq k$ the basic target reachability objective $\mathit{Reach}(T_i)$ can be achieved. That is, a vertex v is winning for $\mathit{Coverage}(T_1, \ldots, T_k)$ if it is winning for each objective $\mathit{Reach}(T_i)$ where $1 \leq i \leq k$.

In the *Coverage* problem we are given vertex sets $T_1, T_2, \ldots, T_k$ and a start vertex s and we decide whether there is a strategy starting at s which is winning for $\mathit{Coverage}(T_1, \ldots, T_k)$. In *AllCoverage* we are only given vertex sets $T_1, T_2, \ldots, T_k$ and have to determine the winning set, i.e., all vertices with a winning strategy for $\mathit{Coverage}(T_1, \ldots, T_k)$.

In the following example, we illustrate the differences between the notions of reachability in the different models.

Example 1 *(Reachability in MDPs and game graphs is not the same).* Consider the graph $G = (V, E)$ with vertices $V = \{v_1, v_2, v_3\}$ and the edges $E = \{(v_1, v_2), (v_2, v_1), (v_2, v_3)\}$ and let $T = \{v_3\}$ be a target set. We will now consider $\mathit{Reach}(T)$ for (a) the MDP $P = (G, \langle V_1, V_R\rangle, \delta)$ and (b) the game graph $\Gamma = (G, \langle V_1, V_2\rangle)$ with $V_1 = \{v_1, v_3\}$ and $V_2 = V_R = \{v_2\}$. The example is illustrated in Fig. 1. In the MDP the vertex v_2 is a random vertex and, thus, whenever the token is at v_2 it is moved to v_3 with non-zero probability. The player-1 policy $\sigma(v_1) = v_2$ wins almost-surely for $\mathit{Reach}(T)$ because the transition from v_2 to v_3 is taken eventually, i.e., v_3 is reached almost-surely. Note that in the game graph at vertex v_2 the adversary can force v_1 and, thus, player 1 does not have a policy which almost-surely wins for $\mathit{Reach}(T)$ starting from v_1. Thus, reachability in MDPs does not imply reachability in game graphs.

Relevant parameters. We consider the following input parameters: n denotes the number of vertices, m denotes the number of edges and k either denotes the number of target sets in the coverage problem or the size of the tuple of target sets in the sequential target reachability problem.

Algorithmic study. In this work, we study the above basic planning objectives for graphs, game graphs (i.e., winning in game graphs), and MDPs (a.s. winning in MDPs). Our goal is to clarify the algorithmic complexity of the above questions with improved algorithms and conditional lower bounds. We define the conjectured lower bounds for conditional lower bounds next.

2.2. Conjectured lower bounds

Results from classical complexity are based on standard complexity-theoretic assumptions, e.g., P $\neq$ NP. Similarly, we derive polynomial lower bounds which are based on widely believed conjectured lower bounds on well-studied algorithmic problems.

First of all, we consider conjectures on Boolean Matrix Multiplication [45, Theorem 6.1] and *triangle detection* in graphs [18, Conjecture 2]. A triangle in a graph is a triple x, y, z of vertices such that $(x, y), (y, z), (z, x) \in E$. In *triangle detection* we are given a graph and the question is if a triangle exists in the graph. We assume that no self-loops in instances of *triangle detection* exist. Note that we can easily establish this assumption by linear-time preprocessing. See Remark 3 for an explanation of the term "combinatorial algorithm".

Conjecture 1 *(Comb. Boolean Matrix Multiplication Conjecture (BMM)). There is no $O(n^{3-\epsilon})$ time combinatorial algorithm for computing the boolean product of two $n \times n$ matrices for any $\epsilon > 0$.*

Conjecture 2 *(Strong Triangle Conjecture (STC)). There is no algorithm which runs in $O(\min\{n^{\omega-\epsilon}, m^{2\omega/(\omega+1)-\epsilon}\})$ expected time and no $O(n^{3-\epsilon})$ time combinatorial algorithm that can detect whether a graph contains a triangle for any $\epsilon > 0$, where $\omega < 2.373$ is the matrix multiplication exponent.*

Williams and Williams [46, Theorem 6.1] showed that BMM is equivalent to the combinatorial part of STC. Moreover, if we do not restrict ourselves to combinatorial algorithms, STC, still gives a super-linear lower bound.

Remark 3 *(Combinatorial algorithm).* The notion of combinatorial algorithm is widely used in the field of fine-grained complexity community [47–49], despite the lack of a formal definition. The main intuition is that combinatorial algorithms do not use fast matrix multiplication [50,51], while non-combinatorial algorithms have the matrix multiplication exponent ω in the running time. To the best of our knowledge, all algorithms for deciding (almost-sure) winning conditions in game graphs and MDPs are combinatorial so far. Thus, lower bounds for combinatorial algorithms are of particular interest in our setting. For further discussion on the notion of combinatorial algorithm consider [52,53].

Secondly, we consider the Strong Exponential Time Hypothesis (SETH) used also in [18, Conjecture 1] introduced by [41, 54] for the satisfiability problem of propositional logic and the Orthogonal Vector Conjecture.

The Orthogonal Vectors Problem (OV). Given sets S_1, S_2 of d-bit vectors with $|S_1| = |S_2| = N$ and $d = \omega(\log N)$, are there $u \in S_1$ and $v \in S_2$ such that $\sum_{i=1}^{d} u_i \cdot v_i = 0$?

Conjecture 3 *(Strong Exponential Time Hypothesis (SETH)). For each $\epsilon > 0$ there is a k such that k-CNF-SAT on n variables and m clauses cannot be solved in $O(2^{(1-\epsilon)n}\,\mathrm{poly}(m))$ time.*

Conjecture 4 *(Orthogonal Vectors Conjecture (OVC)). There is no $O(N^{2-\epsilon})$ time algorithm for the Orthogonal Vectors Problem for any $\epsilon > 0$.*

SETH implies OVC [55, Theorem 5], an explicit reduction is given in the survey article [56, Theorem 3.1]. Whenever a problem is provably hard assuming OVC it is thus also hard when assuming SETH. For example, in [19, Preliminaries, A. Hardness Assumptions, OVH] the OVC is assumed to prove conditional lower bounds for the longest common subsequence problem. To the best of the author's knowledge, there is no connection between the former two and the latter two conjectures.

Remark 4. The conjectures promise that no polynomial improvements over the best-known running times are possible but do not exclude improvements by sub-polynomial factors such as poly-logarithmic factors or factors of, e.g., $2^{\sqrt{\log n}}$.

3. Basic previous results

In this section, we recall the basic algorithmic results about MDPs and game graphs known in the literature that we later use in our algorithms. They explain the results of the first row of Table 1. Note that we cannot give any quadratic conditional lower bounds for any of these problems as they all permit linear time or near-linear time algorithms.

Basic result 1: Maximal End-Component Decomposition. Given an MDP P, an *end-component* is a set of vertices $X \subseteq V$ s.t. (1) the subgraph induced by X is strongly connected (i.e., $(X, E \cap X \times X)$ is strongly connected) and (2) all random vertices have their outgoing edges in X, i.e., X is closed for random vertices, formally described as: for all $v \in X \cap V_R$ and all $(v, u) \in E$ we have $u \in X$. A *maximal end-component* (MEC) is an end-component which is maximal under set inclusion. The importance of MECs is as follows: (i) they generalize strongly connected components (SCCs) in graphs (with $V_R = \emptyset$) and closed recurrent sets of Markov chains (with $V_1 = \emptyset$); and (ii) in a MEC X from all vertices $u \in X$ every vertex $v \in X$ can be reached almost-surely.

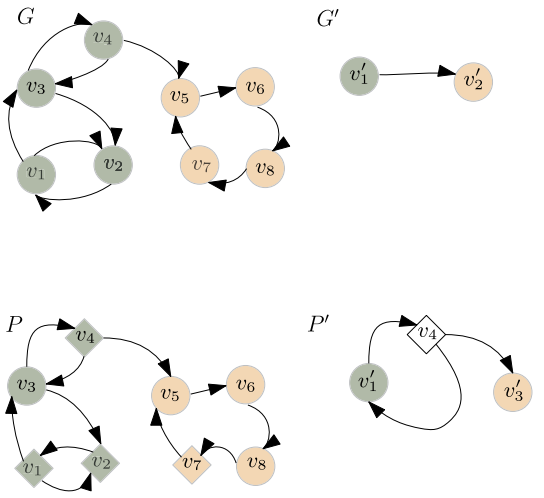

Fig. 2. A graph G and an MDP P where we contract all SCCs and MECs respectively into player-1 vertices (removing self-loops) to obtain G' and P'. The graph G' is acyclic whereas the MDP P' contains a cycle.

Example 2 *(Difference between SCC decomposition and MEC decomposition).* While in SCC decompositions we have that each vertex belongs to exactly one SCC, (which might be a trivial SCC just containing that vertex) for MEC decompositions, we might have a non-empty set of random vertices which do not belong to any MEC (still, each vertex belongs to at most one MEC). Consequently, if we contract each MECs into a player-1 vertex, the resulting MDP is not necessarily acyclic which is in contrast to the graph obtained from contracting the SCCs into player-1 vertices, which is always acyclic. In Fig. 2 we demonstrate this key difference: Contracting all SCCs of the graph G into player-1 vertices yields the DAG G'. Consider the MDP P where the vertices $\{v_1, v_2, v_4, v_7\}$ of G are random vertices and all edges remain unchanged. If we contract the MECs, i.e. $\{\{v_1, v_2, v_3\}, \{v_5, v_6, v_7, v8\}\}$ into player-1 vertices $\{v'_1, v'_3\}$ we obtain the MDP P' which has a cycle. Note that this is because v_4 does not belong to a MEC and is strongly connected with v'_1 or v_3 respectively in P' and P.

The SCC decomposition of a graph can be computed in linear time [57, Theorem 13]. MEC decomposition is computed in $O(m(\log n)^4) = \widetilde{O}(m)$ time by the fastest algorithm [22, Theorem 11].

Basic result 2: Reachability in MDPs. Given an MDP P and a target set T, the set of starting vertices from which T can be reached almost-surely can be computed in $O(m)$ time given the MEC decomposition of P [58, Theorem 4.1]. Consequently, we can solve the basic target reachability problem for MDPs in $\widetilde{O}(m)$.

Basic result 3: Reachability in game graphs. Given a game graph Γ and a target set T, the *player-x attractor* characterizes the set of vertices from which player x can reach T against all polices of the adversarial player $\bar{x}$. Formally, the player-x attractor ($x \in \{1, 2\}$) $Attr_x(S, \Gamma)$ of a given set $S \subseteq V$ is defined as the limit of the sequence $A_0 = S$; $A_{i+1} = A_i \cup \{v \in V_x \mid Out(v) \cap A_i \neq \emptyset\} \cup \{v \in V_{\bar{x}} \mid Out(v) \subseteq A_i\}$ for all $i \geq 0$. An attractor $A = Attr_x(S, \Gamma)$ can be computed in $O(m)$ time [20,21]. We will sometimes omit Γ from $Attr_x(S, \Gamma)$ if it is clear on which game graph we apply the attractor.

4. Coverage problem

In this section, we consider the coverage query problem in graphs, MDPs, and game graphs. The input is a starting vertex v and a coverage query. Our goal is to check if a set of player-1 strategies exist such that the resulting plays achieve the given coverage query when starting at v.

First, we present a linear-time algorithm for graphs and quadratic algorithms for MDPs and game graphs. Then we focus on the conditional lower bounds for MDPs and game graphs, which establish that there is no subquadratic algorithm for the coverage problem when one assumes the STC and OV conjectures.

4.1. Algorithms

The results below present the upper bound for graphs, MDPs, and game graphs of the second row of Table 1.

Coverage Problem in Graphs. For the coverage problem in graphs we are given a graph $G = (V, E)$, a coverage query $Coverage(T_1, \ldots, T_k)$ and a start vertex $s \in V$. The algorithmic problem is to find out if starting from an initial vertex v the basic target reachability, i.e., $Reach(T_i)$, can be achieved for all $1 \leq i \leq k$. The algorithmic solution is as follows: Initially, mark each $v \in T_i$ for $1 \leq i \leq k$ with i. Compute the BFS tree starting from s and check if all the targets are contained in the resulting BFS tree. This instantly gives an algorithm with a running time in $O(m + \sum_{i=0}^{k} |T_i|)$. Note that the running time is linear and thus we cannot hope to find any quadratic lower bounds.

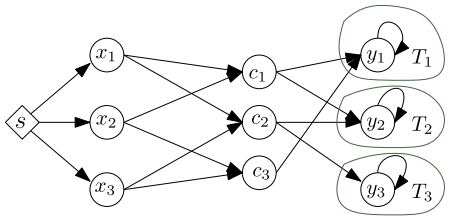

Fig. 3. An example reduction from OV to Coverage in MDPs.

Coverage Problem in MDPs and game graphs. We determine in MDPs and game graphs whether there exists a set of strategies for a given coverage query with k basic target reachability objectives and start vertex v, by applying the reachability algorithm of the respective model k times, i.e., once for each of the target sets. This yields a solution in $\widetilde{O}(km)$ time for MDPs and $O(km)$ time for game graphs respectively. Notice that for $k \in \Theta(n)$ the running time is quadratic in the input size.

4.2. Conditional lower bounds

We present conditional lower bounds for the coverage problem in MDPs and game graphs (i.e., the CLBs of the second row of Table 1). For MDPs and game graphs the conditional lower bounds complement the quadratic algorithms from the previous subsection. Note that we cannot provide a quadratic lower bound for graphs as a linear-time algorithm exists. The conditional lower bounds are due to reductions from *OV* and *triangle detection*.

4.2.1. MDPs

We present the following conditional lower bounds for MDPs:

Theorem 1. *For all $\epsilon > 0$, checking if a vertex has a set of a.s. winning policies for the coverage problem in MDPs does not admit:*

1. *an $O(m^{2-\epsilon})$ algorithm under Conjecture 4,*
2. *an $O((k \cdot m)^{1-\epsilon})$ algorithm under Conjecture 4,*
3. *a combinatorial $O(n^{3-\epsilon})$ algorithm under Conjecture 2 and*
4. *a combinatorial $O((k \cdot n^2)^{1-\epsilon})$ algorithm under Conjecture 2.*

Using the OV-Conjecture. Below, we prove the results 1–2 of Theorem 1. We reduce the OV problem to Coverage in MDPs. By applying Conjecture 4 we infer the result.

Reduction 1. Given two sets S_1, S_2 of d-dimensional vectors (both of size N), we build the MDP P as follows.

- The vertices V of P are given by a start vertex s, sets of vertices S_1 and S_2 representing the sets of vectors and vertices $\mathcal{C} = \{c_i \mid 1 \le i \le d\}$ representing the coordinates of the vectors in the OVC instance.
- The edges E of P are defined as follows: The start vertex s has an edge to every vertex of S_1. Furthermore, for each $x_i \in S_1$ there is an edge to $c_j \in C$ if and only if $x_i[j] = 1$ and for each $y_i \in S_2$ there is an edge from $c_j \in S_2$ to y_i if and only if $y_i[j] = 1$. Also, the y_i have self-loops so that every vertex has an outgoing edge.
- The set of vertices is partitioned into player-1 vertices $V_1 = S_1 \cup \mathcal{C} \cup S_2$ and random vertices $V_R = \{s\}$.

Example 3 *(Example: Reduction from OV to Coverage).* Let the OV instance be $S_1 = \{(1,1,0),(1,0,1),(0,1,1)\}$, $S_2 = \{(1,0,1),(1,1,0),(0,1,0)\}$. Notice that the second vector in S_1 and the third vector in S_2 are orthogonal. Due to the fact that s is a random vertex, there is a nonzero probability that x_2 is the successor. There is no path from x_2 to y_3. As $T_3 = \{y_3\}$, there is no a.s. winning policy from s for the given instance of coverage. We illustrate the example of the reduction in Fig. 3.

Notice that for orthogonal vectors x_i and y_j we have that for each $c_\ell \in \mathcal{C}$ either x_i is not connected to c_ℓ or y_j is not connected to c_ℓ. Thus there is no path from x_i to y_i. Starting from s there is a non-zero probability to end in x_i and, thus, also a non-zero probability to fail reaching the target set $T_j = \{y_j\}$ for all player-1 policies.

Lemma 1. *Let $P = (V, E, \langle V_1, V_R \rangle, \delta)$ be the MDP given by Reduction 1 and $T_i = \{y_i\}$ for $1 \le i \le N$. There exist orthogonal vectors $x \in S_1$, $y \in S_2$ if and only if s is not winning for Coverage$(\{T_i \mid 1 \le i \le N\})$.*

Proof. The MDP P is constructed in such a way that there is no path between vertex x_i and y_j if and only if the corresponding vectors are orthogonal in the OV instance: If x_i is orthogonal to y_j, the outgoing edges lead to no vertex which has an incoming edge to y_j as either $x_i[k] = 0$ or $y_j[k] = 0$. On the other hand, if there is no path from x_i to y_j we again

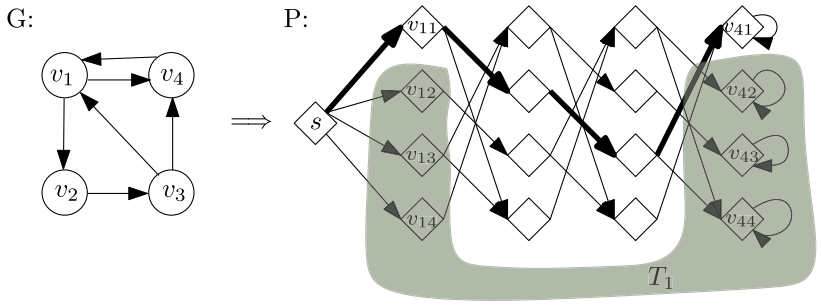

Fig. 4. Reduction from Triangle to Coverage.

have by the construction of the underlying graph that for all $1 \leq k \leq d : x_i[k] = 0$ or $y_j[k] = 0$. This is the definition of orthogonality for x_i and y_j. When starting from s the token is randomly moved to one of the vertices x_i and thus player 1 can reach each y_j almost surely from s if and only if it can reach each y_j from each x_i. Thus, we have that there is an a.s. winning player 1 policy for $Reach(T_i)$ if and only if y_i has. Hence, S_2 has no orthogonal vector in S_1 if and only if each $Reach(T_i)$ has an a.s. winning player 1 policy. □

The MDP P has only $O(N)$ many vertices and Reduction 1 can be performed in $O(N \cdot d)$ time (recall that $d = \omega(\log N)$). The number of edges m is $O(N \cdot d)$ and the number of target sets $k \in \theta(N)$. Thus the results 1–2 of Theorem 1 follow.

Using the ST-conjecture. Towards the results 3–4 in Theorem 1 we reduce the triangle detection problem to the Coverage problem in MDPs. By applying Conjecture 2 we infer the result.

Reduction 2. Given an instance of triangle detection, i.e., a graph $G = (V, E)$, we build the following MDP $P = (V', E', \langle V'_1, V'_R \rangle, \delta)$.

- The vertices V' are given as four copies V_1, V_2, V_3, V_4 of V and a start vertex s.
- The edges E' of P are defined as follows: There is an edge from s to every $v_{1i} \in V_1$ for $i = 1 \dots |V|$. In addition, for $1 \leq j \leq 4$ there is an edge from v_{ji} to $v_{(j+1)k}$ if and only if $(v_i, v_k) \in E$. Finally, v_{4i} for $i = 1 \dots |V|$ has a self-loop.
- The set of vertices V' is partitioned into player-1 vertices $V'_1 = \emptyset$ and random vertices $V'_R = \{s\} \cup V_1 \cup V_2 \cup V_3 \cup V_4$.

Notice that all the vertices of the constructed MDP are random vertices.

Example 4 *(Reducing triangle detection to Coverage.).* Let G be the graph given in Fig. 4. We construct the MDP P as in Reduction 2. Notice that G has the triangle (v_1, v_2, v_3) and the constructed MDP P has a nonzero chance to take the path marked by the fat edges that correspond to this triangle, i.e., player-1 does not have a winning policy from s for the coverage objective given in the reduction because he cannot satisfy T_1. The example is illustrated in Fig. 4.

Lemma 2. *Let P be the MDP given by Reduction 2 when applied to a graph G and let $T_i = V_1 \setminus \{v_{1i}\} \cup V_4 \setminus \{v_{4i}\}$ for $i = 1 \dots |V|$ be target sets. The graph G has a triangle if and only if s is not winning for $Coverage(\{T_i \mid 1 \leq i \leq |V|\})$ in P.*

Proof. First, s is not winning for $Coverage(T_1, \dots, T_{|V|})$ iff there is a T_i such that player-1 has no a.s. winning policy from s for $Reach(T_i)$. Second, there is a triangle in the graph G iff there is a path from some vertex v_{1i} in the first copy of G to the same vertex in the fourth copy of G, v_{4i}. Finally, notice that player 1 does not control any vertex and, thus, the policy of player 1 does not matter and each possible path is played with non-zero probability. If G has a triangle containing vertex v_i then the corresponding play from v_{1i} to v_{4i} has non-zero probability and is not in $Reach(T_i)$. That is, s is not winning for the query $Coverage(\{T_i \mid 1 \leq i \leq N\})$ for player 1. Now assume that s is not winning for $Coverage(\{T_i \mid 1 \leq i \leq N\})$ and thus not winning for $Reach(T_i)$. Then there is a path from v_{1i} to v_{4i} and thus a triangle in G. □

Moreover, the size and the construction time of the MDP P are linear in the size of the original graph G and we have $k = \theta(|V|)$ target sets. Thus 3–4 of Theorem 1 follow.

4.2.2. Game graphs

Next, we describe how the results for MDPs can be extended to game graphs. We prove the following theorem which states multiple specific lower bounds for checking if a vertex has a set of winning policies for a coverage query.

Theorem 2. *For all $\epsilon > 0$, checking if a vertex has a set of winning policies for a coverage query in game graphs does not admit:*

1. *an $O(m^{2-\epsilon})$ algorithm under Conjecture 4,*

2. *an* $O((k \cdot m)^{1-\epsilon})$ *algorithm under Conjecture 4,*
3. *a combinatorial* $O(n^{3-\epsilon})$ *algorithm under Conjecture 2 and*
4. *a combinatorial* $O((k \cdot n^2)^{1-\epsilon})$ *algorithm under Conjecture 2.*

Using the OV-Conjecture. Below we prove the results 1–2 of Theorem 2. We reduce the OV problem to Coverage in game graphs. By applying Conjecture 4 we infer the result. In Reduction 3 we change the random starting vertex of Reduction 1 to a player-2 vertex. The rest of the reduction stays the same. The proof then proceeds as before with the adversary now overtaking the role of the random choices.

Reduction 3. Given two sets S_1, S_2 of d-dimensional vectors, we build the following game graph $\Gamma = (V, E, \langle V_1, V_2 \rangle)$.

- The vertices V and edges E are defined as before in Reduction 1
- The set of vertices is now partitioned into player-1 vertices $V_1 = S_1 \cup \mathcal{C} \cup S_2$ and player-2 vertices $V_2 = \{s\}$.

Lemma 3. *Let* Γ *be the game graph given by Reduction 3 with a coverage query Coverage*$(\{T_i \mid 1 \leq i \leq n\})$ *where* $T_i = \{y_i\}$ *for* $i = 1 \ldots N$. *There exist orthogonal vectors* $x \in S_1$, $y \in S_2$ *if and only if there is no set of winning policies from start vertex s for the coverage query.*

The game graph Γ has only $O(N)$ many vertices and Reduction 3 can be performed in $O(N \cdot d)$ time (recall that $d = \omega(\log N)$). The number of edges m is $O(N \cdot d)$ and the number of target sets $k \in \theta(N)$. Thus the points 1–2 in Theorem 2 follow.

Using the STC conjecture. Below we prove the results 3–4 in Theorem 2. We reduce the triangle detection problem to Coverage in game graphs. By applying Conjecture 2 we infer the result. In Reduction 4 we change the random vertices of Reduction 2 to player-2 vertices. Notice that the resulting game graph consists of only player-2 vertices. Again, if there is a path starting from s which violates a reachability objective in the given coverage query then player 2 wins. As the reachability objectives are defined such that they rule out the triangles of the reduction, player 1 only wins if and only if there is no such path, i.e., there is no triangle in the original graph.

Reduction 4. Given an instance of triangle detection, i.e., a graph $G = (V, E)$, we build the following game graph $\Gamma = (V', E', \langle V'_1, V'_2 \rangle)$.

- The vertices V' and edges E' are the same as in Reduction 2.
- The set of vertices V' is partitioned into player-1 vertices $V'_1 = \emptyset$ and player-2 vertices $V'_2 = \{s\} \cup V_1 \cup V_2 \cup V_3 \cup V_4$.

Lemma 4. *Let* Γ *be the game graph given by Reduction 4 when applied to a graph G and let* $T_i = V_1 \setminus \{v_{1i}\} \cup V_4 \setminus \{v_{4i}\}$ *for* $i = 1 \ldots n$. *The graph G has a triangle if and only if s is winning for Coverage*$(\{T_i \mid 1 \leq i \leq n\})$ *in* Γ.

Moreover, the size and the construction time of game graph Γ are linear in the size of the original graph G and we have $k = \theta(n)$ target sets. Thus 3–4 in Theorem 2 follow.

5. AllCoverage problem

In this section, we consider the AllCoverage problem. First, we present simple algorithms for all models based on the algorithms for reachability in the respective models. Then we present a conditional lower bound for graphs which establishes that the existing algorithm cannot be polynomially improved under the STC and OV conjectures.

5.1. Algorithms

We present a quadratic algorithm for MDPs, games, and graphs. The results present the upper bounds for graphs, MDPs, and game graphs in the third row of Table 1.

Given the query *Coverage*$(\{T_i \mid 1 \leq i \leq k\})$ for graphs, MDPs, and game graphs, we propose an algorithm which first solves the k reachability objectives using the basic results. Notice that the result of the algorithms for solving the basic target reachability objective is a set of vertices that have a policy to achieve the objective. Then we take the intersection of the resulting sets. (1) For graphs using BFS which is in $O(m)$ time we obtain an $O(k \cdot m)$ time algorithm. (2) For game graphs, using the $O(m)$-time attractor computation (see basic result 3), we have an $O(k \cdot m)$ time algorithm. (3) For MDPs, the MEC decomposition followed by k many $O(m)$-time almost-sure reachability computation (see basic result 2), gives an $O(k \cdot m + \text{MEC})$ time algorithm.

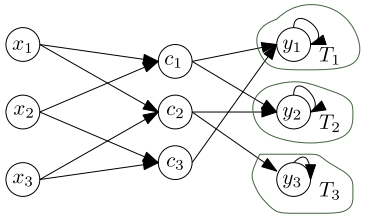

Fig. 5. Reduction from OV to AllCoverage.

5.2. Conditional lower bounds

In this section, we present conditional lower bounds for the AllCoverage problem in graphs (i.e., the CLBs of the third row of Table 1). For MDPs and game graphs the conditional lower bounds follow from Section 4 because the Coverage problem can be trivially reduced to the AllCoverage problem, i.e., once we compute all the vertices that can reach a target it is easy to check whether a specific vertex can reach that target. The conditional lower bounds are due to reductions from OV and the triangle detection problem.

Theorem 3. *For all $\epsilon > 0$, computing the solution of the AllCoverage problem in graphs does not admit*

1. *an $O(m^{2-\epsilon})$ algorithm under Conjecture 4,*
2. *an $O((k \cdot m)^{1-\epsilon})$ algorithm under Conjecture 4,*
3. *a combinatorial $O(n^{3-\epsilon})$ algorithm under Conjecture 2 and*
4. *a combinatorial $O((k \cdot n^2)^{1-\epsilon})$ algorithm under Conjecture 2.*

Using the OV-Conjecture. In this section, we prove results 1–2 in Theorem 3. We reduce the OV problem to the AllCoverage problem in graphs. By applying Conjecture 4 we infer the result.

Reduction 5. Given two sets S_1, S_2 of d-dimensional vectors (both of size N), we build the graph G as follows.

- The construction of the graph is the same as in Reduction 1 except that we do not have a vertex s.

Example 5 *(Reducing AllCoverage to OV).* Let the instance of OV be given by $S_1 = \{(1,1,0),(1,0,1),(0,1,1)\}$, $S_2 = \{(1,0,1),(1,1,0),(0,1,0)\}$. Notice that the second vector in S_1 and the third vector in S_2 are orthogonal. We construct G with Reduction 5. There is no path from x_2 to y_3. As $T_3 = \{y_3\}$, x_2 is not in the winning set of $Coverage(\{T_1, T_2, T_3\})$. Fig. 5 illustrates the reduction.

Lemma 5. *Let $G = (V, E)$ be the graph given by Reduction 5 with target sets $\mathcal{T} = \{T_i \mid T_i = \{y_i\} \text{ for } i = 1 \dots N\}$. A vector $x_i \in S_1$ is orthogonal to some vector in S_2 if and only if the vertex x_i is not in the winning set of Coverage($\mathcal{T}$).*

Proof. The graph P is constructed in such a way that there is no path between vertex x_i and y_j iff the corresponding vectors are orthogonal in the OV instance: If x_i is orthogonal to y_j, the outgoing edges lead to no vertex which has an incoming edge to y_j as either $x_i[k] = 0$ or $y_j[k] = 0$. On the other hand, if there is no path from x_i to y_j we again have by the construction of the underlying graph that for all $1 \le k \le d : x_i[k] = 0$ or $y_j[k] = 0$. This is the definition of orthogonality for x_i and y_j. Thus, x_i is in the winning set of $Coverage(\mathcal{T})$ iff x_i is orthogonal to some vector in S_2. □

Notice that we solve the given instance of *OV* with our reduction as we compute all vectors in S_1 which are orthogonal to some vector in S_2. The Graph G has only $O(N)$ many vertices and Reduction 1 can be performed in $O(N \cdot d)$ time (recall that $d = \omega(\log N)$). The number of edges m is $O(N \cdot d)$ and the number of target sets $k \in \theta(N)$. Thus the points 1–2 in Theorem 3 follow.

Using the ST-Conjecture. Below we prove 3–4 in Theorem 3. We reduce the triangle detection problem to the AllCoverage problem in graphs. By applying Conjecture 2 we infer the result.

Reduction 6. Given an instance of triangle detection, i.e., a graph $G = (V, E)$, we build the following graph $G = (V', E')$. The vertices and edges are the same as in Reduction 2 except that we have player-1 vertices instead of random vertices and there is no start vertex s.

Example 6 *(Reducing Triangle to AllCoverage).* Consider G in Fig. 6. Notice that G has the triangle (v_1, v_2, v_3) and there is a path from v_{11} to v_{41} in the constructed MDP P illustrated by the strong edges corresponding to this triangle. The winning

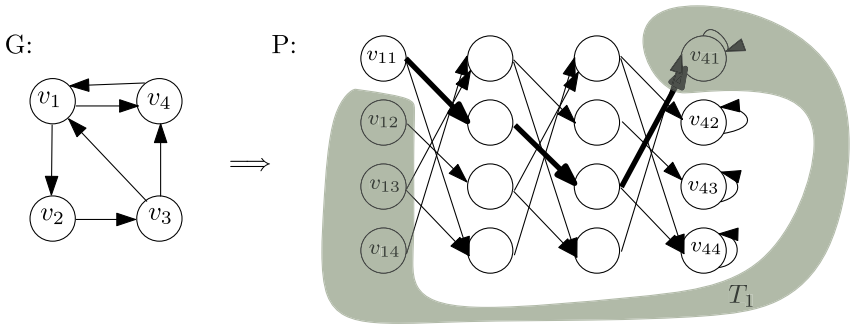

Fig. 6. Reduction from Triangle to AllCoverage.

set of *Coverage*(T_1, T_2, T_3, T_4) contains v_{11} by using the policy which uses the path from v_{11} to v_{41}: First we achieve trivially, *Reach*(T_2), *Reach*(T_3), *Reach*(T_4) with this policy by starting from v_{11}. Then *Reach*(T_1) is achieved by arriving at v_{41}.

Lemma 6. *Let G be the graph given by Reduction 6 with $|V|$ target set $T_i = V_1 \setminus \{v_{1i}\} \cup \{v_{4i}\}$ for $i = 1 \dots |V|$. A graph G has a triangle with vertex v_i if and only if the vertex v_{1i} is in the winning set of the query Coverage$(T_1, \dots, T_{|V|})$.*

Proof. Notice that there is a triangle in the graph G iff there is a path from some vertex v_{1i} in the first copy of G to the same vertex in the fourth copy of G, v_{4i}. Also, a path σ starting in v_{1i} for $1 \le i \le |V|$ is a viable policy in the Coverage query for all *Reach*(T_i) $1 \le i \le |V|$ objectives iff it is able to visit v_{4i}: By definition, T_j includes v_{1i} for $1 \le j \le |V|$ and $j \ne i$. Thus σ achieves all *Reach*(T_j) for $j \ne i$. To achieve *Reach*(T_i), there must be a path from v_{1i} to v_{4i}. Thus, there is a triangle with vertex v_i if and only if v_{1i} is in the winning set of the query *Coverage*$(\mathcal{T})$. □

Note that we solve the given instance of the *triangle detection* problem if we know all vertices which are in triangles. Moreover, the size and the construction time of the MDP P are linear in the size of the original graph G and we have $k = \theta(|V|)$ target sets. Thus 3–4 in Theorem 3 follow.

6. Sequential target problem

We consider the sequential target problem in all models. In contrast to the quadratic CLB for the coverage problem, quite surprisingly, there is a subquadratic algorithm for MDPs. We first present an algorithm for graphs and then build upon that to present the algorithm for MDPs. For games, we present a quadratic algorithm and a quadratic CLB.

6.1. Algorithms

The following results present the upper bounds of the fourth row of Table 1.

6.1.1. Algorithm for graphs

Given a graph $G = (V, E)$ and the sequential target objective *Seq*$(T_1, \dots, T_k)$, we compute the strongly connected components, contract each strongly connected component to a single vertex and remove multi-edges. This results in a *directed acyclic graph (DAG)*. Additionally, the vertex v' which represents an SCC C in the resulting DAG D, is in all target sets of its members, i.e., $v' \in T_i$ if there exists a vertex $u \in C$ such that $u \in T_i$ for all $1 \le i \le k$. Notice that this step does not change the reachability conditions of the resulting acyclic graph: Every vertex in an SCC can be reached starting from every other vertex in the same SCC. Thus, it suffices to give an algorithm for DAGs. Given a DAG $D = (V, E)$, we maintain (a) a set of unprocessed vertices S, which is initialized with V and (b) a queue Q containing the vertices which are not processed but where all successors are processed, initialized with all vertices with no outgoing edges. Notice that the queue is initially non-empty because the bottom SCCs of G are now vertices without outgoing edges in D. Additionally, for each vertex v, we maintain the values $count_v$, ℓ_v and $best_v$. The variable $count_v$ counts the number of vertices in $Out(v)$ which are not processed yet. The label ℓ_v is such that vertex v has a winning policy for the objective *Seq*$(\mathcal{T}_{\ell_v})$ where $\mathcal{T}_{\ell_v} = (T_{\ell_v}, \dots, T_k)$. In other words, there is a policy to win from v if we already visited the targets sets $\mathcal{T}_1, \dots T_{\ell_v - 1}$. The variable $best_v$ is used to store the minimum label of the already processed successors of v. The algorithm proceeds as follows. While the queue Q is not empty, we take a vertex v from the queue and call ProcessVertex$(\cdot)$. The function computes the label ℓ_v of the vertex v using $best_v$ and the target sets where v is in. Then, it removes v from S and updates the variables $best_w$ and $count_w$ of all predecessors $w \in In(v)$. In particular, we set $best_w = \min(best_w, \ell_v)$ and decrement $count_w$ by one. When the queue is empty, all vertices are processed and the algorithm terminates. We show that the described algorithm for DAGs has a linear running time, i.e., $O(m + \sum_{i=1}^{n} |T_i|)$ and the details are presented in Algorithm 1.

Algorithm 1: Sequential Targets in Graphs.

```
   Input: DAG D = (V, E), targets T = (T_1, ..., T_k)
 1 S ← V;
 2 L_v ← {i ∈ {1, ..., k} | v ∈ T_i}    for v ∈ V;
 3 count_v ← |Out(v)|    for v ∈ V;
 4 best_v ← { k+1  if Out(v) = ∅
            { null  otherwise          for v ∈ V;
 5 ℓ_v ← null    for v ∈ V;
 6 Q ← {v ∈ V | Out(v) = ∅};
 7 while S ≠ ∅ do
 8     v = Q.pop();
 9     ProcessVertex(v);
10 return {v ∈ V | ℓ_v = 1};
11 function ProcessVertex(Vertex v)
12     ℓ_v ← best_v;
13     while ℓ_v − 1 ∈ L_v do
14         ℓ_v ← ℓ_v − 1;
15     S ← S \ {v};
16     for w ∈ In(v) do
17         best_w ← min(best_w, ℓ_v);
18         count_w ← count_w − 1;
19         if count_w = 0 then
20             Q.push(w);
```

Proposition 1 *(Correctness). Given a DAG $D = (V, E)$ and a sequential reachability objective $Seq(\mathcal{T})$ with target sets $\mathcal{T} = \{T_1, \ldots, T_k\}$, Algorithm 1 returns the set of all start vertices with a path for the objective $Seq(\mathcal{T})$.*

Observation 1. *The input graph has one or more vertices v with $Out(v) = \emptyset$ and thus Q is non-empty after the initialization.*

Proof. Note that there is always a vertex $v \in V$ where $Out(v) = \emptyset$ because we assumed that D is a DAG. □

The invariants below state that (a) the variables $(best_v, count_v, \ell_v)$ have the intended meaning, (b) Q contains all the unprocessed vertices whose successors are already processed and (c) that the queue contains vertices as long as S is not empty.

Lemma 7. *The following statements are invariants of the while loop at Line 7.*

1. $count_v = |Out(v) \cap S|$
2. $v \in Q$ *if and only if* $v \in S$ *and* $Out(v) \cap S = \emptyset$.
3. *If S is not empty then the queue Q is not empty.*
4. $best_v = k + 1$ *for all* $v \in V$ *with* $Out(v) = \emptyset$.
5. *If* $v \in Q$ *then* $best_v \neq null$.
6. *If* $v \in V \setminus S$ *then* $\ell_v \neq null$.
7. $best_v = \min_{w \in Out(v) \setminus S} \ell_w$, *for all* $v \in V$ *with* $Out(v) \setminus S \neq \emptyset$.

Proof.
1. The counters $count_v$ are initialized as $|Out(v)|$ and S is initialized as V. Thus the claim holds when first entering the while loop.
   Assume the claim holds at the beginning of the iteration where vertex u is processed. The set S is only changed in Line 15. There u is removed from the set. The counters are only changed in Line 18: All counters of vertices w with $u \in Out(w)$ are decreased by one. Consequently $count_v = |Out(v) \cap S|$ holds for all $v \in V$ also after this iteration of the loop and the claim follows.
2. In the initial phase S is set to V and Q is set to $\{v \in V \mid Out(v) = \emptyset\}$. Thus the claim holds when first entering the while loop.
   Assume the claim holds at the beginning of the iteration where vertex v is processed. The set S is only changed in Line 15 where v is removed.
   First consider a vertex $w \in Q \setminus \{v\}$. As w is not removed from the set S and no vertex is added to S the claim is still true for w. Now consider a vertex w that might be added during the iteration of the loop. This can only happen in Line 20 and the if conditions ensure that $w \in S$ and $Out(v) \cap S = \emptyset$ (by the previous invariant) and thus the claim also holds for the newly added vertices.
3. Due to Observation 1 the claim holds when first entering the while loop.

Assume the claim holds at the beginning of the iteration, where vertex v is processed. The vertex v is removed from S in Line 15 and if the set S is empty now, the claim follows trivially. On the other hand, if S is non-empty and Q is also non-empty the claim follows again. In the third case S is non-empty and Q is empty. Assume for contradiction that no vertex is added at line 20. By invariant (2), every vertex $v \in S$ has a successor in S as otherwise, v would be in Q. That implies that there exists a cycle which is a contradiction with D being a DAG.

4. For $v \in V$ with $Out(v) = \emptyset$ the variables $best_v$ are initialized with $k+1$ (Line 4) and $best_v$ is only changed in Line 17 when a successor of the vertex is processed. As v has no successor, $best_v$ is not changed during the algorithm.
5. If $v \in Q$ initially, it must be due to the initialization and we have $best_v = k+1$. The claim holds when first entering the while loop. Assume the claim holds at the beginning of the iteration where v is processed. The only time we add a vertex w to Q is at Line 20. Notice that we set $best_w$ before at Line 17.
6. Initially, every vertex is in S, thus the claim holds before the first iteration of the while loop. Assume the claim holds at the beginning of the iteration where v is processed. The only time we remove a vertex from S is at Line 15, i.e., in ProcessVertex(v). Notice that we set ℓ_v in Line 12 to $best_v$ which cannot be *null* due to Lemma 7 (5).
7. Initially, V is S, and for all $v \in V$ we set $\ell_v, best_v$ to *null*. Notice that $best_v$ with $Out(v) \neq \emptyset$ are not changed at Line 4. Thus the claim holds when the algorithm enters the loop.
   Now consider the iteration of vertex v and assume the claim is true at the beginning. The set S is only changed in Line 15 where v is removed. Let S_{old} be the set at the beginning of the iteration and $S_{new} = S_{old} \setminus \{v\}$ the updated set. Due to Lemma 7 (6) $\ell_v \neq null$. For a vertex $w \in In(v)$, the value $best_w$ is updated to $\min(best_w, \ell_v)$ (Line 17) which by assumption is equal to $\min_{x \in (Out(w) \setminus S_{old}) \cup \{v\}} \ell_x = \min_{x \in (Out(w) \setminus S_{new})} \ell_x$, i.e., the equation holds. For vertices $w \notin In(v)$ both $best_w$ as well as the right hand side of the equation are unchanged. Hence, the claim holds also after the iteration. □

From the following invariants, we obtain the correctness of our algorithm.

Lemma 8. *The following statements are invariants of the while loop at Line 7 for all $v \in V \setminus S$:*

1. *there exists a path $p_v \in Seq(\mathcal{T}_{\ell_v})$,*
2. *there exists no path $p_v \in Seq(\mathcal{T}_{\ell_v - 1})$,*

where $\mathcal{T}_{\ell_v} = \{T_{\ell_v}, \ldots, T_k\}$ or $\ell_v > k$.

Proof. As S is initialized with the set of vertices V the two statements trivially hold after the initialization.

Now consider the iteration where vertex v is processed and assume the invariants hold at the beginning of the iteration. Let $be(v) = \min_{w \in Out(v)} \ell_w$. By Lemma 7 (2) we have $Out(v) \cap S = \emptyset$ and by Lemma 7 (6) also $\ell_w \neq null$ for all $w \in Out(v)$. Thus by Lemma 7 (7) we have $be(v) = best_v$ and ℓ_v can be computed. The while loop in Line 13 decrements ℓ_v which is initialized to $best_v - 1$ as long has $best_v - 1 \in L_v$. L_v contains $\ell_v, \ldots, best_v - 1$ but does not contain $\ell_v - 1$.

1. We next show that there is a path p_v in $Seq(\mathcal{T}_{\ell_v})$: Let $w = be(v)$. A path for vertex w where $p_w \in Seq(\mathcal{T}_{\ell_w})$, exists by induction hypothesis. The targets $\{T_{\ell_v}, \ldots, T_{\ell_{w-1}}\}$ are visited by starting from v. The path is obtained as follows: $p_v = v, p_w$, which proves the claim.
2. We next show that there is no path p_v in $Seq(\mathcal{T}_{\ell_v - 1})$. The current vertex v is not in the set $T_{\ell_v - 1}$ and no successor w has a path p_w with $p_w \in Seq(\mathcal{T}_{\ell_v - 1})$ because $\ell_v - 1 < \ell_v \leq \ell_w$ (Lines 12–14). Thus there is also no path $p_v \in Seq(\mathcal{T}_{\ell_v - 1})$ which concludes the proof. □

Proposition 2. *Algorithm 1 has running time $O(m + \sum_{i=1}^{k} |T_i|)$.*

Proof. We first argue that the initialization takes $O(m + \sum_{i=1}^{k} |T_i|)$ time. Initializing the sets L_v can be done by first initializing the set L_v as $\emptyset$ (in $O(n)$) and then iterate over all set T_i and for each $v \in T_i$ add i to L_v (in $O(\sum_{i=1}^{k} |T_i|)$). The other variables can be initialized by iterating over all vertices and for each vertex consider all outgoing edges. That is in $O(n+m) = O(m)$ time. Now consider the main part of the algorithm. In the while loop we process each vertex $v \in V$ once (recall that D is a DAG) and call the function ProcessVertex(v) at Line 9 In the function call ProcessVertex(v), we iterate over the set L_v (Lines 13–14) and all incoming edges of v (Lines 16–20). If we sum over all the vertices we obtain a running time of $O(m + \sum_{v \in V} |L_v|) = O(m + \sum_{i=1}^{n} |T_i|)$. □

Theorem 4. *Given a graph $G = (V, E)$, a sequential target objective and a vertex $s \in V$ we can decide whether s is winning for $Seq(\mathcal{T})$ in $O(m + \sum_{i=1}^{k} |T_i|)$ time.*

6.1.2. Algorithm for MDPs

The algorithm for MDPs builds on the algorithm for graphs. The key difference is that instead of computing an SCC decomposition and contracting SCCs, for MDPs, we compute a MEC decomposition and contract MECs into player-1 vertices:

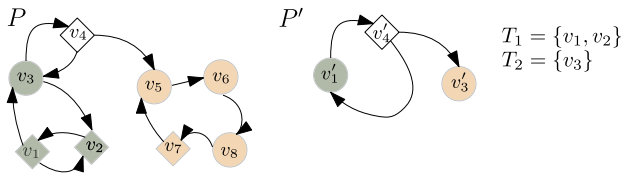

Fig. 7. Key difficulty when computing Sequential Reachability in MDPs.

Given an MDP $P = (V, E, \langle V_1, V_R \rangle, \delta)$ with the sequential target objective $Seq(T_1, \ldots, T_k)$, we compute the MEC decomposition of the MDP. Then, each MEC M is contracted into a player-1 vertex v' without self-loops. The resulting MDP is P'. The target sets of P' are as follows: The vertex v' is in all the target sets of the corresponding vertices in M, i.e., $v' \in T_i$ if there exists a vertex $u \in M$ such that $u \in T_i$ for all $1 \leq i \leq k$. Notice that this step does not change the reachability conditions of the resulting MDP: Every vertex in the MEC can be reached almost-surely starting from every other vertex in the same MEC, regardless of their type (player-1, random). Thus, it suffices to give an algorithm for MEC-free MDPs.

Key challenge. When computing a MEC decomposition and contracting the MECs we get an MDP that may still contain cycles. Thus, our MDP algorithm has to deal with cycles unlike the graph setting where we only had to deal with DAGs, i.e., in Algorithm 1 we maintained a Queue Q which contained all unprocessed vertices where all successors are processed. In each iteration we process one such vertex. Notice that when the queue is the only mechanism to process the vertices, we need the fact (which we also show in Lemma 7 (3)) that there either exists a vertex where all successors have been processed or all vertices have been processed and the algorithm can terminate. When running the algorithm on MEC-free MDPs there might be a situation where the Queue Q is empty and some vertices have not been processed yet. We illustrate such a situation in Example 7. Thus, we need an additional mechanism to process vertices in this situation.

Example 7 *(Queue empty but graph not processed).* Consider the MDP P given in Fig. 7: Contracting the MECs of P into player-1 vertices, we obtain P'. Notice that P' still contains a cycle. Using only the queue to obtain the next vertex to process we have the following problem: After v'_3 is processed, the queue Q is empty because v'_2 has still has an unprocessed successor, namely v'_1. Notice that v'_2 and v'_1 have not been processed yet.

Algorithm Description. The algorithm for MEC-free MDPs maintains the set of unprocessed vertices S and a queue Q, the values $count_v$, ℓ_v, and $best_v$ for each vertex v. The value $count_v$, as in Algorithm 1, stores the number of vertices in $Out(v)$ which are not processed yet. The label ℓ_v for v is now such that v has an *almost-sure winning policy* for the objective $Seq(\mathcal{T}_{\ell_v})$ where $\mathcal{T}_{\ell_v} = (T_{\ell_v}, \ldots, T_k)$. Random vertices might choose the worst possible successor with nonzero probability, i.e., the vertex with highest ℓ_v, whereas player-1 vertices always choose the vertex with the lowest ℓ_v. We reflect this fact as follows in the variable $best_v$: The variable $best_v$ stores the maximum (for $v \in V_R$) / minimum (for $v \in V_1$) label of the already processed successors of v. The set S is initialized with V, and, initially, all vertices with no outgoing edges are added to the queue Q. Notice that the bottom MECs of P are now vertices without outgoing edges in P' and thus Q is initially non-empty. If the queue Q is non-empty, a vertex from the queue is processed as in Algorithm 1. When Q is empty, the algorithm has to process a vertex where some successors are not processed yet. In that case, we consider all the random vertices for which at least one successor is processed and choose the random vertex with the maximum $best_v$ to process next. We show that, as the graph has no MECs, whenever Q is empty (and S is not) there exists such a random vertex. Moreover, whenever Q is empty, all vertices in the set of unprocessed vertices S have a policy that satisfies $Seq(\mathcal{T}_m)$ for $m = \max_{v \in V_R \cap S} best_v$: Intuitively, this is due to the fact that all vertices in S can reach a vertex v' (which is possibly different to $v = \text{argmax}_{v \in V_R \cap S} best_v$) in the set of already processed vertices and in the worst case $\ell_{v'} = m$, i.e., they can satisfy $Seq(\mathcal{T}_m)$. For the vertex $v = \arg\max_{v \in V_R \cap S} best_v$ all successors w without a label are in S and are going to obtain a label ℓ_w of at most m. The current value of $best_v$ is m, i.e., v has a successor w with $\ell_w = m$. As v is in V_R the final value of $best_v$ must be m. Hence, one can process v without knowing the exact label of all the successors. We present the details in Algorithm 2 and prove a running time which is in $O(m \log n + \sum_{i=0}^{k} |T_i|)$. Notice that the running is near-linear time, linear up to a $\log n$ factor.

Proposition 3 *(Correctness). Given an MDP P and a sequential target objective $Seq(\mathcal{T})$ with targets $\mathcal{T} = (T_1, \ldots, T_k)$, Algorithm 2 returns the set of all start vertices with a player-1 policy for the objective $Seq(\mathcal{T})$.*

We next state the invariants of the while loop (see Line 7) that will enable us to show the correctness of the algorithm. The invariants state that (a) the variables $best_v$ and $count_v$ have the meaning as described in the algorithm description for all $v \in V$, (b) Q contains all the unprocessed vertices whose successors are already processed, and (c) that the function argmax is well-defined whenever called, i.e., there is a random vertex where $best_v$ is not *null*.

Algorithm 2: Sequential target Reachability for MEC-free MDPs.

Input: MEC-free MDP $P = (V, E, \langle V_1, V_R \rangle, \delta)$, targets $\mathcal{T} = (T_1, \ldots, T_k)$
Output: All vertices with a policy for $Seq(\mathcal{T})$.

1 $S \leftarrow V$;
2 $L_v \leftarrow \{i \in \{1, \ldots, k\} \mid v \in T_i\}$ for $v \in V$;
3 $count_v \leftarrow |Out(v)|$ for $v \in V$;
4 $best_v \leftarrow \begin{cases} k+1 & \text{if } Out(v) = \emptyset \\ null & \text{otherwise} \end{cases}$ for $v \in V$;
5 $\ell_v \leftarrow null$ for $v \in V$;
6 $Q \leftarrow \{v \in V \mid Out(v) = \emptyset\}$;
7 **while** $S \neq \emptyset$ **do**
8 **if** $Q \neq \emptyset$ **then**
9 v = $Q.pop()$;
10 ProcessVertex(v);
11 **else**
12 $v \leftarrow \text{argmax}_{v \in V_R \cap S}\ best_v$;
13 ProcessVertex(v);
14 **return** $\{v \in V \mid \ell_v = 1\}$;
15 **function** ProcessVertex(*Vertex* v)
16 $\ell_v \leftarrow best_v$;
17 **while** $\ell_v - 1 \in L_v$ **do**
18 $\ell_v \leftarrow \ell_v - 1$;
19 $S \leftarrow S \setminus \{v\}$;
20 **for** $w \in \{w : (w, v) \in E\}$ **do**
21 **if** $w \in V_1$ **then**
22 $best_w \leftarrow \min(best_w, \ell_v)$
23 **else**
24 $best_w \leftarrow \max(best_w, \ell_v)$
25 $count_w \leftarrow count_w - 1$;
26 **if** $count_w = 0 \wedge w \in S$ **then**
27 $Q.push(w)$

Lemma 9. *The following statements are invariants of the while loop in Line 7.*

1. $count_v = |Out(v) \cap S|$;
2. $v \in Q$ *if and only if* $v \in S$ *and* $Out(v) \cap S = \emptyset$;
3. $best_v = k + 1$, *for all* $v \in V$ *with* $Out(v) = \emptyset$.
4. *If* $v \in Q$ *we have* $best_v \neq null$.
5. *If* $v \in V \setminus S$ *we have* $\ell_v \neq null$.
6. *For all* $v \in V$ *with* $Out(v) \setminus S \neq \emptyset$: $best_v = \begin{cases} \min_{w \in Out(v) \setminus S} \ell_w & v \in V_1 \\ \max_{w \in Out(v) \setminus S} \ell_w & v \in V_R \end{cases}$
7. *If* $S \neq \emptyset$ *and* $Q = \emptyset$ *there is a* $v \in S \cap V_R$ *such that* $best_v \neq null$.

Proof. The proofs of (1) − (5) proceed as the proofs of the corresponding statements in the proof of Lemma 7.

6. Initially $S = V$ and for all $v \in V$ we set $\ell_v, best_v$ to *null*. Also, $best_v$ with $Out(v) \neq \emptyset$ are not changed at Line 4 and the claim holds when the algorithm enters the loop.
   Now consider the iteration of vertex v and assume the claim is true at the beginning. The set S is only changed in Line 19 where v is removed. Let S_{old} be the set at the beginning of the iteration and $S_{new} = S_{old} \setminus \{v\}$ the updated set. First notice that $best_v \neq null$ as v is either chosen by (a) as element of Q or (b) by argmax. In the former case we apply Lemma 9 (4) and in the latter case $best_v \neq null$ by the definition of argmax. For a vertex $w \in In(v) \cap V_1$ the value $best_w$ is updated to $\min(best_w, \ell_v)$ (Line 22) which by assumption is equal to $\min_{x \in (Out(w) \setminus S_{old}) \cup \{v\}} \ell_x = \min_{x \in (Out(w) \setminus S_{new})} \ell_x$, i.e., the equation holds. For a vertex $w \in In(v) \cap V_R$ the value $best_w$ is updated to $\max(best_w, \ell_v)$ (Line 24) which by assumption is equal to $\max_{x \in (Out(w) \setminus S_{old}) \cup \{v\}} \ell_x = \max_{x \in (Out(w) \setminus S_{new})} \ell_x$, i.e., the equation holds. For vertices $w \notin In(v)$ $best_w$ remains unchanged. Hence, the claim holds for w by the assumption that the invariant is true before the iteration.
7. Initially the statement is true as each MEC-free MDP has a vertex v with $Out(v) = \emptyset$ and thus Q is non-empty (otherwise there would be an SCC with no outgoing edge which thus would be a MEC).
   Now consider the iteration processing vertex v and assume the claim is true at the beginning and $Q = \emptyset$. Notice that $best_w$ is set for vertices as soon as one vertex in $Out(w)$ was processed. Towards a contradiction assume that all vertices $w \in S \cap V_R$ have $best_w = null$, i.e., no vertex $w \in S \cap V_R$ has a successor in $V \setminus S$. Note that S contains only the vertices

which are not processed yet. Each $w \in S$ has at least one successor in S as otherwise, w would be in Q. Thus S is either empty which would make the statement trivially true or has again a bottom SCC (on the induced subgraph G_P) with more than one vertex that has no random outgoing edges. Again such an SCC would be a MEC and we obtain our desired contradiction. □

From the following invariant, we obtain the correctness of our algorithm.

Lemma 10. *The following statements are invariants of the while loop in Line 7 for all $v \in V \setminus S$:*

1. *there exists a player 1 policy σ s.t. $\Pr_v^\sigma(Seq(\mathcal{T}_{\ell_v})) = 1$; and*
2. *there is no player 1 policy σ s.t. $\Pr_v^\sigma(Seq(\mathcal{T}_{\ell_v - 1})) = 1$.*

where $\mathcal{T}_{\ell_v} = \{T_{\ell_v}, \ldots, T_k\}$ or $\ell_v > k$.

Proof. As S is initialized as set V the two statements hold after the initialization.

Now consider the iteration where vertex v is processed and assume the invariants hold at the beginning of the iteration. Notice that we do not change ℓ'_v for any other vertex $v' \neq v$ and the invariant holds trivially for v'. We first introduce the following notation

$$be(v) = \begin{cases} \min_{w \in Out(v)} \ell_w & v \in V_1 \\ \max_{w \in Out(v)} \ell_w & v \in V_R \end{cases}$$

We distinguish the case where Q is non-empty and the case where Q is empty.

- *Case $Q \neq \emptyset$*: By Lemma 9 (2) we have $Out(v) \cap S = \emptyset$. Because we only remove vertices from S if we process them, all $w \in Out(v)$ are processed and thus $\ell_w \neq null$. Thus by Lemma 9 (6) we have $be(v) = best_v$. By the while-loop in Line 17 we have $L_v \supseteq \{\ell_v, \ldots, best_v - 1\}$ but does not contain $\ell_v - 1$, i.e., $\ell_v - 1 \notin L_v$.
  (1) Thus we can easily obtain a policy σ with $\Pr_v^\sigma(Seq(\mathcal{T}_{\ell_v})) = 1$ as follows.
  If $v \in V_1$ pick the vertex w that corresponds to $be(v)$ and then player 1 can follow the existing policy σ' for vertex w. Because the invariant holds for w, there exists a policy σ' such that $\Pr_w^{\sigma'}(Seq(\mathcal{T}_{be(v)})) = 1$.
  If $v \in V_R$ let vertex $w \in Out(v)$ be the randomly chosen vertex. By the invariant which holds during the iteration, w has a policy σ such that $\Pr_w^\sigma(Seq(\mathcal{T}_{be(v)})) = 1$. Combined with L_v, this is the desired policy, i.e., $\Pr_v^\sigma(Seq(\mathcal{T}_{\ell_v})) = 1$
  (2) We next show that there is no policy for $Seq(\mathcal{T}_{\ell_v - 1})$. By Line 17 we have $v \notin T_{\ell_v - 1}$. If $v \in V_1$ no successor $w \in Out(v)$ has a policy σ with $\Pr_w^\sigma(Seq(\mathcal{T}_{\ell_v - 1})) = 1$ as the invariant holds also for w. Thus there is also no policy σ for v such that $\Pr_v^\sigma(Seq(\mathcal{T}_{\ell_v - 1})) = 1$. If $v \in V_R$ there is at least one successor w (because the invariant holds also for w) which has no policy σ such that $\Pr_w^\sigma(Seq(\mathcal{T}_{\ell_v - 1})) = 1$. Consequently there is no policy σ for v with $\Pr_v^\sigma(Seq(\mathcal{T}_{\ell_v - 1})) = 1$ as there is a non-zero chance that a vertex w is picked that, by the fact that the invariant holds at the current iteration, cannot reach a node in $T_{l_v - 1}$.
- *Case $Q = \emptyset$*: Due to Lemma 9 (7) there is at least one vertex in $V_R \cap S$ such that $best_v \neq null$. Let $best_{\max} = \max_{v \in V_R \cap S} best_v$.
  (1) As we have no MEC (in S), there is a policy σ, so that the play almost surely leaves S by using one of the outgoing edges of a random node: Note that for all random nodes between S and $V \setminus S$ we have a policy which achieves at least $Seq(\mathcal{T}_{best_{\max}})$. The policy σ can be arbitrary, except that for a player-1 vertex $x \in S$ with an edge (x, y) where $y \in V \setminus S$ we choose $\sigma(x) \in S$ (which must exist as x would be in Q otherwise). As there are no MECs (in S) the policy σ_1 will eventually lead to a vertex in $V \setminus S$ using a random node. This implies that from each vertex in S player 1 has a policy to reach a vertex in $V \setminus S$ coming from a random vertex. Because the invariant holds at the current iteration each successor of a random vertex v' where $best_{v'} \neq null$ has a policy to satisfy $Seq(\mathcal{T}_{best_{\max}})$. Thus it follows that from each vertex in S player 1 has a policy to satisfy $Seq(\mathcal{T}_{best_{\max}})$. Now consider the random vertex v that was chosen by the algorithm as $\operatorname{argmax}_{v \in V_R \cap S} best_v$. Because v' is a random vertex, all successors have a policy to satisfy $Seq(\mathcal{T}_{best_{\max}})$ almost-surely. As L_v contains $\ell_v, \ldots, best_v - 1$ but does not contain $\ell_v - 1$ we obtain a policy σ with $\Pr_v^\sigma(Seq(\mathcal{T}_{\ell_v})) = 1$.
  (2) By the choice of v there is also a successor (that is chosen with non-zero probability) that, by assumption, has no policy for $Seq(\mathcal{T}_{best_{\max} - 1})$ and, moreover, L_v does not contain $\ell_v - 1$. Thus, when starting in v each policy will fail to satisfy $Seq(\mathcal{T}_{best_{\max} - 1})$ with non-zero probability, i.e., there is no policy σ for $\Pr_v^\sigma(Seq(\mathcal{T}_{\ell_v - 1})) = 1$. □

Proposition 4 *(Running Time). Algorithm 2 runs in $O(m \log n + \sum_{i=0}^{k} |T_i|)$ time.*

Proof. Initializing the algorithm takes $O(m + \sum_{i=0}^{k} |T_i|)$ time and calling the function ProcessVertex(v) takes time $O(|In(v)| + |L_v|)$ (cf. proof of Proposition 2). Consider the main while-loop at Line 7 where every vertex is processed once (recall that either Q is nonempty or there exists a $v \in S \cap V_R$ such that $best_v \neq null$ due to Lemma 9). The costly operations are the calls to the ProcessVertex($\cdot$) function and the evaluation of the argmax function. Summing up over all vertices we

Algorithm 3: Sequential Target Reachability for Games.

Input: Game graph $\Gamma = ((V, E), \langle V_1, V_2 \rangle)$ and target sets $T = (T_1, \ldots, T_k)$

```
S_{k+1} ← V;
ℓ ← k;
while ℓ > 0 do
    S_ℓ ← Attr_1(T_ℓ ∩ S_{ℓ+1});
    ℓ ← ℓ − 1;
return S_1;
```

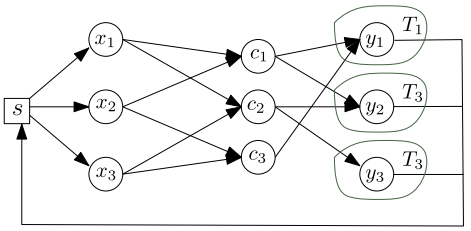

Fig. 8. Reduction from OV to Sequential Targets.

obtain a $O(m + \sum_{i=0}^{k} |T_i|)$ bound for the calls to ProcessVertex$(\cdot)$. To compute argmax efficiently we have to maintain a priority queue containing all not yet processed random vertices. As we have $O(m)$ updates this costs only $O(m \log n)$ for one of the standard implementations of priority queues. Summing up this yields a $O(m \log n + \sum_{i=0}^{k} |T_i|)$ running time for Algorithm 2. □

By considering also the time MEC for the MEC decomposition we obtain the desired bound and the following theorem.

Theorem 5. *Given an MDP P, a start vertex s and a sequential target objective $Seq(\mathcal{T})$, we can compute whether there is a player-1 policy σ_1 at s for $Seq(\mathcal{T})$ in $O(\text{MEC} + m \log n + \sum_{i=0}^{k} |T_i|)$ time.*

6.1.3. Algorithm for games

Given a game graphs with sequential target objectives $Seq(\mathcal{T})$ where $\mathcal{T} = (T_1, \ldots, T_k)$, the basic algorithm (stated as Algorithm 3) performs k player-1 attractor computations. It starts with computing the attractor $S_k = Attr_1(T_k)$ of T_k, and then iteratively computes the sets $S_\ell = Attr_1(S_{\ell+1} \cap T_\ell)$ for $1 \leq \ell < k$, and finally returns the set S_1 as the start vertices from which player 1 can reach all the target sets in the given order. This gives an $O(k \cdot m)$-time algorithm. Note that for $k = \Theta(n)$ the running time is quadratic in the input size.

6.2. Conditional lower bounds

We present CLBs for game graphs based on the conjectures STC and OVC which establish the CLBs for the fourth row of Table 1. Notice that we cannot provide conditional lower bounds for graphs and MDPs as linear time algorithms for these two models exist.

Theorem 6. *For all $\epsilon > 0$, checking if a vertex has a winning policy for the sequential target problem in game graphs does not admit*

1. *an $O(m^{2-\epsilon})$ algorithm under Conjecture 4,*
2. *an $O((k \cdot m)^{1-\epsilon})$ algorithm under Conjecture 4,*
3. *a combinatorial $O(n^{3-\epsilon})$ algorithm under Conjecture 2 and*
4. *a combinatorial $O((k \cdot n^2)^{1-\epsilon})$ algorithm under Conjecture 2.*

Using the OV-Conjecture. Below we prove results 1–2 of Theorem 6 by reducing the OV problem to the sequential target problem in game graphs. The reduction is an extension of Reduction 1, where we (a) produce a player-2 vertex instead of a random vertex and (b) also every vertex of S_2 has an edge back to s.

Example 8. Let the OV instance be given by $S_1 = \{(1,1,0), (1,0,1), (0,1,1)\}$, $S_2 = \{(1,0,1), (1,1,0), (0,1,0)\}$. Notice that the second vector in S_1 and the third vector in S_2 are orthogonal. Due to the fact that s is a player-2 vertex, it can choose x_2 as the successor. There is no path from x_2 to y_3. As $T_3 = \{y_3\}$, there is no winning policy for player 1 from s for the given sequential target objective. We illustrate the reduction in Fig. 8.

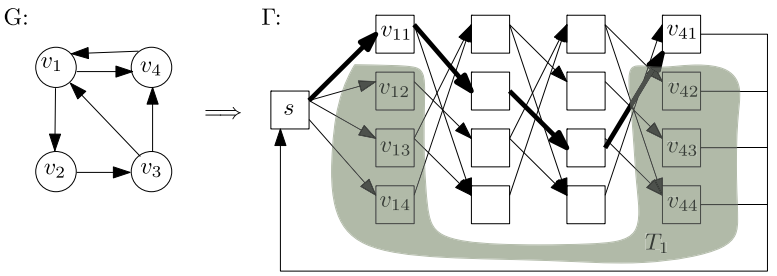

Fig. 9. Reduction from Triangle to Sequential Targets.

Reduction 7. Given two sets S_1, S_2 of d-dimensional vectors (both of size N), we build the following game graph Γ.

- The vertices V of the game graph are given by a start vertex s, sets of vertices S_1 and S_2 representing the sets of vectors and vertices $\mathcal{C} = \{c_i \mid 1 \leq i \leq d\}$ representing the coordinates of the vectors in the OVC instance.
- The edges E of Γ are defined as follows: the start vertex s has an edge to every vertex of S_1 and every vertex of S_2 has an edge back to s; furthermore, for each $x_i \in S_1$ there is an edge to $c_j \in C$ if and only if $x_i[j] = 1$ and for each $y_i \in S_2$ there is an edge from $c_j \in S_2$ to y if and only if $y_i[j] = 1$.
- The set of vertices is partitioned into player-1 vertices $V_1 = S_1 \cup \mathcal{C} \cup S_2$ and player-2 vertices $V_2 = \{s\}$.

Lemma 11. *Let Γ be the game graph given by Reduction 7 with a sequential objective Seq($\mathcal{T}$) where $\mathcal{T} = (T_1, \ldots, T_k)$ and $T_i = \{y_i\}$ for $i = 1 \ldots N$. There exist orthogonal vectors $x_i \in S_1$, $y_j \in S_2$ if and only if s has no player-1 policy σ_1 to ensure winning for the objective Seq($\mathcal{T}$).*

Proof. Notice that the game graph Γ is constructed in such a way that there is no path between x_i and y_j iff they are orthogonal in the OV instance. Notice that each play starting at s revisits s every four steps and if there is no path between x_i and y_j then player 2 can disrupt player 1 from visiting a target T_j by moving the token to x_i whenever the token is in s. However, if there is no such x_i and y_j, player 2 cannot disrupt player 1 from s because no matter which vertex x_i player 2 chooses, player 1 has a policy to reach the next target set. If s has no player-1 policy σ_1 to ensure winning for the objective Seq($\mathcal{T}$) there must be a target player 1 cannot reach. This must be due to the fact that there is no path between some x_i and y_j and player 2 always chooses x_i. □

The number of vertices in Γ, constructed by Reduction 1 is $O(N)$ and the construction can be performed in $O(N \log N)$ time (recall that $(d = \omega(\log N))$). The number of edges m is $O(N \log N)$ and the number of target sets $k \in \theta(N) = \theta(m/\log N)$. Thus (1–2) in Theorem 6 follow.

Using the ST-Conjecture. In this section, we prove the results 3–4 in Theorem 6. We reduce the triangle detection problem to the sequential target problem in game graphs. The reduction extends Reduction 2, where we (a) produce player-2 vertices instead of random vertices and (b) every vertex in the fourth copy has an edge back to s.

Reduction 8. Given an instance of triangle detection, i.e., a graph $G = (V, E)$, we build the following game graph $\Gamma = (V', E', \langle V'_1, V'_2 \rangle)$.

- The vertices V' are given as four copies V_1, V_2, V_3, V_4 of V and a start vertex s.
- The edges E' are defined as follows: There is an edge from s to every $v_{1i} \in V_1$ where $i = 1 \ldots |V|$. In addition for $1 \leq j \leq 3$ there is an edge from v_{ji} to $v_{(j+1)k}$ if and only if $(v_i, v_k) \in E$. Furthermore there are edges from every $v_{4i} \in V_4$ to the start vertex s.
- The set of vertices V' is partitioned into player-1 vertices $V'_1 = \emptyset$ and player-2 vertices $V'_2 = \{s\} \cup V_1 \cup V_2 \cup V_3 \cup V_4$.

Example 9 *(Reduction Triangle Detection to Sequential Targets in Games.).* Consider the graph G given in Fig. 9. The vertices of G are player-2 vertices in Γ and the graph is copied four times. The edges of Γ go to the same target but to next copy of the graph. Notice that G has the triangle (v_1, v_2, v_3) and the constructed game graph Γ enables player-2 to take the path marked by the fat edges, i.e., player-1 does not have a winning policy from s for the sequential target objective given in the reduction because he cannot satisfy T_1. We illustrate the example of the reduction in Fig. 9.

Lemma 12. *Let Γ' be the game graphs given by Reduction 8 with Seq($\mathcal{T}$) as follows: $\mathcal{T} = (T_1, T_2, \ldots, T_k)$ where $T_i = V_1 \setminus \{v_{1i}\} \cup V_4 \setminus \{v_{4i}\}$ for $i = 1 \ldots k$. The graph G has a triangle if and only if there is no policy σ_1 to ensure winning for the objective Seq($\mathcal{T}$) from start vertex s.*

Proof. For the correctness of the reduction notice that there is a triangle in the graph G iff there is a path from some vertex v_{1i} in the first copy of G to the same vertex in the fourth copy of G, v_{4i} in P. Player 2 then has a policy to always visit only v_{1i} from the first copy and only v_{4i} from the fourth copy which prevents player 1 from visiting target T_i. □

The size and the construction time of graph Γ, given by Reduction 2, are linear in the size of the original graph G and we have $k = \Theta(|V|)$ target sets. Thus (3–4) in Theorem 6 follow.

7. Discussion and conclusion

In this work, we presented lower bound results for planning objectives in explicit state space. We next discuss implications from these results for the same planning objectives in factored models and then end this paper with concluding remarks.

7.1. Implications for factored models

Here we relate our results for the explicit state space to factored models, like STRIPS [2]. For the basic reachability problem different conditional lower bounds were established in [42,43]. In the following, we discuss to which extent our conditional lower bounds provide lower bounds for the corresponding problems in the factored models. We use the AllCoverage problem on graphs as an example but similar arguments apply to the other planning problems as well. A planning instance of the AllCoverage problem for graphs in a factored model is given by variables V, the domain of the variables D, the actions A and a set of conditions defining the target sets $s_{G_1}, \ldots, s_{G_k}$. A state is a function that specifies a value in D to every variable in V. The planner can go from one state to another by applying the actions defined in the function A. A is a mapping from an input state to an output state, i.e., the possible transition between states is defined by the actions. The goal of the planner is to output all states which can reach all sets s_{G_i} $(1 \leq i \leq k)$ using the actions described in A. We next investigate how our graph-based lower bounds can be interpreted in the factored model. To obtain lower bounds similar to Theorem 3 we aim encoding the graphs computed by our reductions (see e.g. Reduction 5 and Reduction 6) in the factored model. A naive encoding that simply numbers the vertices and then uses a binary encoding of these numbers can represent n states with $v = \log_2 n$ variables. This encoding would give lower bounds w.r.t. the number of variables v (and thus the state space) that exclude $O(k \cdot 2^{(1-\epsilon)v} \cdot poly(v))$ algorithms for the AllCoverage problem in the factored planning model. However, these lower bounds are only w.r.t. the number of variables and not w.r.t. the total size of the problem instance which, in particular, also includes the number of actions. The naive encoding requires as many actions as there are edges in the graph, i.e., the size of the problem instance is dominated by the number of actions. Thus, using this naive encoding we do not get interesting lower bounds w.r.t. the instance size. To obtain lower bounds w.r.t. the instance size we have to encode the vertices of the graph in a way that also allows to encode the edges of the graphs compactly. For our reductions from triangle detection, i.e., Reduction 6, the edge relation can be rather arbitrary and, thus, the reduction is unlikely to allow for a compact representation in the factored model (without making additional assumptions). In contrast, our reductions from the OV-problem, i.e., Reduction 5, are well-suited for such a compact encoding as the resulting graphs are sparse and the edge relation is defined systematically based on the content of the vertices. Thus, if one uses the binary vectors as a basis for the encoding of the vertices in the factored model then the transitions can be represented with a relatively low number of actions. However, the details of such an encoding and the corresponding lower bounds depend on the actual factored model. Investigating such encodings for concrete factored models is beyond the scope of this paper and we thus leave it open as an interesting direction for future research.

7.2. Concluding remarks

In this work, we study several natural planning problems in graphs, MDPs, and game graphs, which are basic algorithmic problems in artificial intelligence. Our main contributions are a sub-quadratic algorithm for sequential target in MDPs, and quadratic conditional lower bounds. Note that graphs are a special case of both MDPs and game graphs, and the algorithmic problems are simplest for graphs, and in all cases except for AllCoverage, we have linear-time upper bounds. The key highlight of our results is an interesting separation of MDPs and game graphs: for basic target reachability, MDPs are harder than game graphs; for the coverage problem, both MDPs and game graphs are hard (quadratic CLBs); for sequential target reachability, game graphs are harder than MDPs.

In this work, we clarified the algorithmic landscape of basic planning problems with CLBs and better algorithms. An interesting direction of future work would be to consider CLBs for other polynomial-time problems in planning and AI in general. For MDPs with sequential targets, we establish sub-quadratic upper bounds, and hence the techniques of the paper that establish quadratic CLBs are not applicable. Other CLB techniques for this problem are an interesting topic to investigate as future work.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

A preliminary version of this work appeared in the *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling (ICAPS '18)* [59]. The authors are very grateful to the anonymous referees from ICAPS'18 and the journal of Artificial Intelligence for their valuable comments that helped to greatly improve the presentation of this work. A.S. is fully supported by the Vienna Science and Technology Fund (WWTF) through project ICT15–003. K.C. is supported by the Austrian Science Fund (FWF) NFN Grant No. S11407-N23 (RiSE/SHiNE) and by the ERC CoG 863818 (ForM-SMArt). For M.H. the research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007–2013) / ERC Grant Agreement No. 340506.

References

[1] S.M. LaValle, Planning Algorithms, Cambridge University Press, 2006.
[2] S.J. Russell, P. Norvig, Artificial Intelligence - A Modern Approach, Third International Edition, Pearson Education, 2010.
[3] H. Howard, Dynamic Programming and Markov Processes, MIT Press, 1960.
[4] M.L. Puterman, Markov Decision Processes: Discrete Stochastic Dynamic Programming, Wiley, 1994.
[5] J.A. Filar, K. Vrieze, Competitive Markov Decision Processes, Springer, 1997.
[6] C.H. Papadimitriou, J.N. Tsitsiklis, The complexity of Markov decision processes, Math. Oper. Res. 12 (1987) 441–450.
[7] C. Guestrin, D. Koller, R. Parr, S. Venkataraman, Efficient solution algorithms for factored MDPs, J. Artif. Intell. Res. 19 (2003) 399–468.
[8] A. Mahanti, A. Bagchi, AND/OR graph heuristic search methods, J. ACM 32 (1985) 28–51.
[9] E.A. Hansen, S. Zilberstein, Heuristic search in cyclic AND/OR graphs, in: AAAI, 1998, pp. 412–418.
[10] H. Kress-Gazit, G.E. Fainekos, G.J. Pappas, Temporal-logic-based reactive mission and motion planning, IEEE Trans. Robot. 25 (2009) 1370–1381.
[11] L.P. Kaelbling, M.L. Littman, A.R. Cassandra, Planning and acting in partially observable stochastic domains, Artif. Intell. 101 (1998) 99–134.
[12] H.M. Choset, Principles of Robot Motion: Theory, Algorithms, and Implementation, MIT Press, 2005.
[13] A. Cimatti, M. Pistore, M. Roveri, P. Traverso, Weak, strong, and strong cyclic planning via symbolic model checking, Artif. Intell. 147 (2003) 35–84.
[14] S. Maliah, R. Brafman, E. Karpas, G. Shani, Partially observable online contingent planning using landmark heuristics, in: ICAPS, 2014, pp. 163–171.
[15] B. Bonet, H. Geffner, Planning with incomplete information as heuristic search in belief space, in: AIPS, 2000, pp. 52–61.
[16] J. Hoffmann, R. Brafman, Contingent planning via heuristic forward search with implicit belief states, in: ICAPS, 2005, pp. 71–88.
[17] H. Palacios, H. Geffner, From conformant into classical planning: efficient translations that may be complete too, in: ICAPS, 2007, pp. 264–271.
[18] A. Abboud, V.V. Williams, Popular conjectures imply strong lower bounds for dynamic problems, in: FOCS, 2014, pp. 434–443.
[19] K. Bringmann, M. Künnemann, Quadratic conditional lower bounds for string problems and dynamic time warping, in: FOCS, 2015, pp. 79–97.
[20] C. Beeri, On the membership problem for functional and multivalued dependencies in relational databases, ACM Trans. Database Syst. 5 (1980) 241–259.
[21] N. Immerman, Number of quantifiers is better than number of tape cells, J. Comput. Syst. Sci. 22 (1981) 384–406.
[22] K. Chatterjee, W. Dvořák, M. Henzinger, A. Svozil, Near-linear time algorithms for Streett objectives in graphs and MDPs, in: CONCUR, 2019, pp. 7:1–7:16.
[23] A. Camacho, J.A. Baier, C. Muise, S.A. McIlraith, Finite LTL synthesis as planning, in: ICAPS, 2018, pp. 29–38.
[24] A. Camacho, C.J. Muise, J.A. Baier, S.A. McIlraith, LTL realizability via safety and reachability games, in: IJCAI, 2018, pp. 4683–4691.
[25] A. Camacho, M. Bienvenu, S.A. McIlraith, Finite LTL synthesis with environment assumptions and quality measures, in: KR, 2018, pp. 454–463.
[26] A. Kolobov Mausam, D.S. Weld, H. Geffner, Heuristic search for generalized stochastic shortest path MDPs, in: ICAPS, 2011, pp. 130–137.
[27] F. Teichteil-Königsbuch, Stochastic safest and shortest path problems, in: J. Hoffmann, B. Selman (Eds.), AAAI, 2012, pp. 1826–1831.
[28] T. Keller, P. Eyerich, PROST: probabilistic planning based on UCT, in: L. McCluskey, B.C. Williams, J.R. Silva, B. Bonet (Eds.), ICAPS, 2012, pp. 119–127.
[29] A. Camacho, C.J. Muise, S.A. McIlraith, From FOND to robust probabilistic planning: computing compact policies that bypass avoidable deadends, in: ICAPS, 2016, pp. 65–69.
[30] R. Mattmüller, M. Ortlieb, M. Helmert, P. Bercher, Pattern database heuristics for fully observable nondeterministic planning, in: R.I. Brafman, H. Geffner, J. Hoffmann, H.A. Kautz (Eds.), ICAPS, AAAI, 2010, pp. 105–112.
[31] J. Fu, V. Ng, F.B. Bastani, I. Yen, Simple and fast strong cyclic planning for fully-observable nondeterministic planning problems, in: T. Walsh (Ed.), IJCAI, IJCAI/AAAI, 2011, pp. 1949–1954.
[32] C.J. Muise, S.A. McIlraith, J.C. Beck, Improved non-deterministic planning by exploiting state relevance, in: L. McCluskey, B.C. Williams, J.R. Silva, B. Bonet (Eds.), ICAPS, AAAI, 2012, pp. 172–180.
[33] R. Alford, U. Kuter, D.S. Nau, R.P. Goldman, Plan aggregation for strong cyclic planning in nondeterministic domains, Artif. Intell. 216 (2014) 206–232.
[34] A. Camacho, E. Triantafillou, C.J. Muise, J.A. Baier, S.A. McIlraith, Non-deterministic planning with temporally extended goals: LTL over finite and infinite traces, in: AAAI, 2017, pp. 3716–3724.
[35] C.J. Muise, V. Belle, S.A. McIlraith, Computing contingent plans via fully observable non-deterministic planning, in: AAAI, 2014, pp. 2322–2329.
[36] B. Bonet, H. Geffner, Planning under partial observability by classical replanning: theory and experiments, in: T. Walsh (Ed.), IJCAI, 2011, pp. 1936–1941.
[37] K. Chatterjee, M. Henzinger, Efficient and dynamic algorithms for alternating Büchi games and maximal end-component decomposition, J. ACM 61 (2014) 15:1–15:40.
[38] T. Bylander, The computational complexity of propositional STRIPS planning, Artif. Intell. 69 (1994) 165–204.
[39] C. Bäckström, B. Nebel, Complexity results for SAS+ planning, Comput. Intell. 11 (1995) 625–656.
[40] M. Kronegger, A. Pfandler, R. Pichler, Parameterized complexity of optimal planning: a detailed map, in: IJCAI, 2013, pp. 954–961.
[41] R. Impagliazzo, R. Paturi, Complexity of k-SAT, in: CCC, 1999, pp. 237–240.
[42] M. Aghighi, C. Bäckström, P. Jonsson, S. Ståhlberg, Refining complexity analyses in planning by exploiting the exponential time hypothesis, Ann. Math. Artif. Intell. 78 (2016) 157–175.
[43] C. Bäckström, P. Jonsson, Time and space bounds for planning, J. Artif. Intell. Res. 60 (2017) 595–638.
[44] E. Eiben, J. Gemmell, I.A. Kanj, A. Youngdahl, Improved results for minimum constraint removal, in: AAAI, 2018, pp. 6477–6484.
[45] V.V. Williams, R. Williams, Subcubic equivalences between path, matrix, and triangle problems, J. ACM 65 (2018) 27:1–27:38.
[46] V.V. Williams, R. Williams, Subcubic equivalences between path, matrix and triangle problems, in: FOCS, 2010, pp. 645–654.
[47] A. Abboud, A. Backurs, V.V. Williams, If the current clique algorithms are optimal, so is Valiant's parser, SIAM J. Comput. 47 (2018) 2527–2555.
[48] A. Lincoln, V.V. Williams, R.R. Williams, Tight hardness for shortest cycles and paths in sparse graphs, in: SODA, 2018, pp. 1236–1252.
[49] K. Bringmann, N. Fischer, M. Künnemann, A fine-grained analogue of Schaefer's theorem in P: dichotomy of existsk-forall-quantified first-order graph properties, in: CCC, 2019, pp. 31:1–31:27.
[50] V.V. Williams, Multiplying matrices faster than Coppersmith-Winograd, in: STOC, 2012, pp. 887–898.

[51] F. Le Gall, Powers of tensors and fast matrix multiplication, in: ISSAC, 2014, pp. 296–303.
[52] G. Ballard, J. Demmel, O. Holtz, O. Schwartz, Graph expansion and communication costs of fast matrix multiplication, J. ACM 59 (2012) 32:1–32:23.
[53] M. Henzinger, S. Krinninger, D. Nanongkai, T. Saranurak, Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture, in: STOC, 2015, pp. 21–30.
[54] R. Impagliazzo, R. Paturi, F. Zane, Which problems have strongly exponential complexity?, in: FOCS, 1998, pp. 653–662.
[55] R. Williams, A new algorithm for optimal 2-constraint satisfaction and its implications, Theor. Comput. Sci. 348 (2005) 357–365.
[56] V. Vassilevska-Williams, On some fine-grained questions in algorithms and complexity, in: ICM, 2018, pp. 3447–3487.
[57] R. Tarjan, Depth-first search and linear graph algorithms, SIAM J. Comput. 1 (1972) 146–160.
[58] K. Chatterjee, W. Dvořák, M. Henzinger, V. Loitzenbauer, Model and objective separation with conditional lower bounds: disjunction is harder than conjunction, in: LICS, 2016, pp. 197–206.
[59] K. Chatterjee, W. Dvořák, M. Henzinger, A. Svozil, Algorithms and conditional lower bounds for planning problems, in: ICAPS, AAAI Press, 2018, pp. 56–64.