Faster Algorithms for Bounded Liveness in Graphs and Game Graphs

Krishnendu Chatterjee
IST Austria, Klosterneuburg, Austria

Monika Henzinger
University of Vienna, Faculty of Computer Science, Vienna, Austria

Sagar Sudhir Kale
University of Vienna, Faculty of Computer Science, Vienna, Austria

Alexander Svozil
University of Vienna, Faculty of Computer Science, Vienna, Austria

Abstract

Graphs and games on graphs are fundamental models for the analysis of reactive systems, in particular, for model-checking and the synthesis of reactive systems. The class of ω-regular languages provides a robust specification formalism for the desired properties of reactive systems. In the classical infinitary formulation of the liveness part of an ω-regular specification, a "good" event must happen eventually without any bound between the good events. A stronger notion of liveness is bounded liveness, which requires that good events happen within d transitions. Given a graph or a game graph with n vertices, m edges, and a bounded liveness objective, the previous best-known algorithmic bounds are as follows: (i) $O(dm)$ for graphs, which in the worst-case is $O(n^3)$; and (ii) $O(n^2d^2)$ for games on graphs. Our main contributions improve these long-standing algorithmic bounds. For graphs we present: (i) a randomized algorithm with one-sided error with running time $O(n^{2.5}\log n)$ for the bounded liveness objectives; and (ii) a deterministic linear-time algorithm for the complement of bounded liveness objectives. For games on graphs, we present an $O(n^2d)$ time algorithm for the bounded liveness objectives.

2012 ACM Subject Classification Theory of computation → Modal and temporal logics

Keywords and phrases Graphs, Game Graphs, Büchi

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.124

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Funding *Krishnendu Chatterjee*: Supported by the ERC CoG 863818 (ForM-SMArt).
Monika Henzinger: Supported by the Austrian Science Fund (FWF) and netIDEE SCIENCE project P 33775-N.
Sagar Sudhir Kale: Partially supported by the Vienna Science and Technology Fund (WWTF) through project ICT15-003.
Alexander Svozil: Fully supported by the Vienna Science and Technology Fund (WWTF) through project ICT15-003.

1 Introduction

Graphs and games on graphs. Graphs and two-player games played on graphs provide a general mathematical framework for a wide range of problems in computer science: in particular, for the analysis of reactive systems, where the vertices of the graph represent the states of a reactive system and the edges represent the transitions between the states. The classical synthesis problem (the problem of Church) asks for the construction of a winning strategy in a game played on the graph [13, 21, 20] and the fundamental model-checking problem is an algorithmic graph problem [14].

Omega-regular specifications: strength and weakness. In the analysis of reactive systems, the desired temporal properties that the system should satisfy constitute the specification. The class of ω-regular languages provides a robust specification formalism [18, 20]. Every ω-regular objective can be decomposed into a safety part and a liveness part [3]. The safety part ensures that the system will not do anything "bad" (such as violating an invariant) within any finite number of transitions. The liveness part ensures that the system will do something "good" (such as proceed or respond) in the long-run. Liveness can be violated only in the limit, by infinite sequences of transitions, as no bound is specified on when a "good" event must happen. This infinitary formulation has several strengths, such as robustness and simplicity [18, 23]. However, there is also a weakness of the classical definition of liveness: it can be satisfied by systems that are unsatisfactory because no bound can be put between the occurrence of desired events.

Stronger notion of liveness. For the weakness of the infinitary formulation of liveness, alternative and stronger formulations of liveness have been proposed. The first formulation is *bounded liveness* which ensures, given a bound d, that eventually, good events happen within d transitions. The second formulation is *finitary liveness* which requires the existence of a bound such that eventually good events happen within the bound. Finitary liveness was proposed in [4] and has been widely studied; e.g., games on graphs with finitary ω-regular objectives [12], and logics such as PromptLTL based on finitary liveness [17]. The notion of bounded liveness has also been investigated in many contexts, such as MSO with bounding quantifiers [7], bounded model-checking [6], and "bounded until" in logics such as RTCTL [15].

Algorithmic questions for bounded liveness. In this work, we consider graphs and games on graphs with bounded liveness objectives. Consider a graph with n vertices, m edges, and a bounded liveness objective with bound d. A basic algorithmic approach is to reduce the bounded liveness objective to a liveness objective on a larger graph (that we call the auxiliary graph) that explicitly keeps track of the number of transitions since the last good event. This basic approach yields the following bounds: (a) an $O(dm)$-time algorithm for graphs (applying the linear-time algorithm for liveness objectives on graphs), and (b) an $O(n^2d^2)$-time algorithm for games on graphs (applying the current best-known $O(n^2)$-time algorithm for games on graphs with liveness objectives [11]). A fundamental algorithmic question is whether the above bounds can be improved.

Our contributions. In this work, our main contributions are improved algorithmic bounds for bounded liveness on graphs and games on graphs.

- In graphs, there are two relevant semantics: (a) an existential semantic that asks whether there exists a path to satisfy the objective, and (b) a universal semantic that asks whether all paths satisfy the objective. The answer to the universal semantics with bounded liveness is "Yes" if and only if the answer is "No" for existential semantics with the complementary bounded coliveness objective. We consider graphs with the existential semantics and bounded liveness and bounded coliveness objectives. For bounded liveness objectives, all previous algorithmic approaches yield an $O(n^3)$ worst-case time-bound (where $d = O(n)$) and we present a randomized algorithm with one-sided error whose worst-case time-bound is $O(n^{2.5} \log n)$. For bounded coliveness objectives, we present a deterministic linear-time algorithm.
- For games on graphs with bounded liveness objectives, we present an $O(n^2d)$-time algorithm that improves the previous $O(n^2d^2)$-time algorithm.

Significance of the contributions. On the technical front, it is threefold.

1. To break the $O(n^3)$-time barrier for graphs, we exploit randomization to estimate for all pairs of good events how far they are from each other. Using this information along with a suitably modified auxiliary graph results in the faster $O(n^{2.5} \log n)$-time algorithm.

To get the improved time bound of $O(n^2 d)$ for game graphs:

2. we construct an auxiliary game graph (similar to the graph case) and make a crucial observation that this game graph after each iteration has a lot of structure, a property we call *induced symmetry*;
3. we strategically introduce as many "layover" vertices as there are good events; in combination with induced symmetry, this enables us to prove that a significant chunk of the auxiliary game graph is deleted after each iteration.

Furthermore, there are several important implications of our contributions. First, for graphs with bounded liveness objectives, the previous worst-case time-bound is $O(n^3)$. In recent years, many such algorithmic problems with $O(n^3)$ bound have been shown to be conditionally optimal with a reduction from classical problems such as BMM (boolean matrix multiplication) [1, 2, 8, 9, 10, 24]. Our new algorithm breaks the $O(n^3)$ barrier and shows that such conditional lower bound approaches do not apply for bounded liveness in graphs. Second, for graphs with bounded coliveness objectives our linear-time bound shows that there is a very efficient algorithm for the complement of the bounded liveness objectives. Finally, we show that the basic algorithmic approach for games on graphs can also be improved. Given our results improve the bounds for graphs and games on graphs with bounded liveness objectives, there are several interesting questions for future work. Whether the bounds can be further improved or a deterministic sub-cubic time algorithm can be obtained for graphs with bounded liveness objectives are the most interesting algorithmic open questions.

2 Preliminaries

Since the notation and definitions are standard, we base this section on the definitions section by Chatterjee and Henzinger [11].

Game graphs and graphs. A game graph $\Gamma = ((V, E), \langle V_1, V_2 \rangle)$ is a directed graph, where V is a finite set of vertices, E is a finite set of edges, and $\langle V_1, V_2 \rangle$ is a partition of V into player-1 vertices V_1 and the adversarial player-2 vertices V_2. Graphs are a special case of game graphs with $V_2 = \emptyset$. Define $Out(v) = \{u \in V \mid (v, u) \in E\}$ to be the set of vertices to which v has an outgoing edge and $In(v) = \{u \in V \mid (u, v) \in E\}$ to be the set of vertices from which v has an incoming edge. As is standard, we assume that there are no self-loops and that every vertex has an outgoing edge. Let $n = |V|$ be the number of vertices and $m = |E|$ be the number of edges.

Plays. A *play* $\langle v_0, v_1, v_2, \ldots \rangle$ is an infinite sequence of vertices in Γ such that each $(v_{i-1}, v_i) \in E$ for all $i \geqslant 1$. We denote by Ω the set of all plays. A *finite play* V^* is a prefix of a play.

Strategies. A player-ρ strategy tells which edge to follow next given a finite play that ends in a player-ρ vertex. More formally, a player-1 *strategy* is a function $\sigma : V^* \cdot V_1 \mapsto V$ such that for $\omega \in V^* \cdot V_1$ and v being the last vertex, $(v, \sigma(\omega)) \in E$. A player-2 strategy is defined in the same way. We denote by Σ the set of all player-1 strategies and by Π the set of all player-2 strategies.

Outcome of strategies. Given a starting vertex v and the strategies $\sigma \in \Sigma$ and $\pi \in \Pi$, there is a unique play $\omega(v, \sigma, \pi) = \langle v_0, v_1, v_2, \ldots \rangle$, which is defined as follows: $v_0 = v$; for all $i > 0$ if $v_i \in V_1$ then $\sigma(\langle v_0, \ldots, v_i \rangle) = v_{i+1}$, and if $v_i \in V_2$, then $\pi(\langle v_0, \ldots, v_i \rangle) = v_{i+1}$.

Objectives. An objective $\Phi \subseteq \Omega$ is a set of "winning" plays. The main objectives of this paper are the bounded Büchi objective for player 1 and the complementary bounded coBüchi

objective for player 2. For a play ω, we define by $Inf(\omega)$ the set of vertices that occur infinitely often in ω. More formally, if $\omega = \langle v_0, v_1, v_2, \ldots\rangle \in \Omega$, then $Inf(\omega) = \{v \in V \mid \forall i \geqslant 0 \exists j > i : v_j = v\}$. We also need the reachability, safety, Büchi and the coBüchi objectives for the analyses. In the following definitions, assume that we are given a game graph Γ.

1. *Reachability and Safety objectives.* For $T \subseteq V$, the reachability objective states that *at least one* vertex in T be visited, and dually, the safety objective states that *only* vertices in C be visited. Formally, $\text{Reach}(T, \Gamma) = \{\langle v_0, v_1, v_2, \ldots\rangle \in \Omega \mid \exists k \geqslant 0 : v_k \in T\}$ and $\text{Safety}(C, \Gamma) = \{\langle v_0, v_1, v_2, \ldots\rangle \in \Omega \mid \forall k \geqslant 0 : v_k \in C\}$. The two objectives are dual, i.e., $\text{Reach}(T, \Gamma) = \Omega \setminus \text{Safety}(V \setminus T, \Gamma)$.
2. *Büchi and coBüchi objectives.* Given a set of *Büchi* vertices, the Büchi objective states that some Büchi vertex be visited infinitely often, and dually, the coBüchi objective states that only vertices in a given set C be visited infinitely often. Formally, given $B \subseteq V$, define $\text{Büchi}(B, \Gamma) = \{\omega \in \Omega \mid Inf(\omega) \cap B \neq \emptyset\}$ and given $C \subseteq V$, define $\text{coBüchi}(C, \Gamma) = \{\omega \in \Omega \mid Inf(\omega) \subseteq C\}$. The two objectives are dual, i.e., $\text{Büchi}(B, \Gamma) = \Omega \setminus \text{coBüchi}(V \setminus B, \Gamma)$.
3. *Bounded Büchi and bounded coBüchi objectives.* Given a set of Büchi vertices and an integer $d \geqslant 0$, the bounded Büchi objective states that from some point on, the distance between any two consecutive Büchi vertices is at most d. Dually, given $C \subseteq V$, the bounded coBüchi objective requires that there are at least d consecutive vertices in C infinitely often. Formally, the sets of winning plays are $\text{boundedBüchi}(B, d, \Gamma) = \{\omega \in \Omega \mid \exists i \geqslant 0 \forall j \geqslant i : \{v_j, v_{j+1}, \ldots, v_{j+d-1}\} \cap B \neq \emptyset\}$ and $\text{boundedcoBüchi}(C, d, \Gamma) = \{\omega \in \Omega \mid \forall i \geqslant 0\ \exists j \geqslant i \text{ s.t. } \{v_j, v_{j+1}, \ldots, v_{j+d-1}\} \subseteq C\}$. These are also dual, i.e., $\text{boundedBüchi}(B, d, \Gamma) = \Omega \setminus \text{boundedcoBüchi}(V \setminus B, d, \Gamma)$.

When studying bounded Büchi (and bounded coBüchi) objectives, one can assume without loss of generality that $d \leqslant n$, because otherwise they are equivalent to Büchi objectives. We omit Γ from the definition of the objectives if it is obvious on which game graph the objectives are defined.

For an objective Φ, a strategy $\sigma \in \Sigma$ is a *winning strategy* for player 1 from vertex v if for all player-2 strategies $\pi \in \Pi$ the resulting play $\omega(v, \sigma, \pi) \in \Phi$, and the *set of winning vertices for player 1* is $W_1(\Phi) = \{v \in V \mid \exists \sigma \in \Sigma \text{ s.t. } \forall \pi \in \Pi : \omega(v, \sigma, \pi) \in \Phi\}$. Player-2 winning strategies and winning vertices are defined in the same way.

Remark about determinacy. The following theorem shows that every vertex in V either belongs to the winning set of bounded Büchi objectives of player 1 or to the winning set of bounded coBüchi objectives for player 2. The same holds for Büchi and coBüchi objectives. We say that a vertex is either *winning for player 1* or *winning for player 2.*

▶ **Theorem 1** (Determinancy [19]). *For all game graphs Γ, all (bounded) Büchi objectives Φ for player 1 and the complementary (bounded) coBüchi objectives $\Psi = \Omega \setminus \Phi$ for player 2 we have $W_1(\Phi) = V \setminus W_2(\Psi)$.*

Observe that for (bounded) Büchi objectives Φ for player 1 and the (bounded) coBüchi objectives $\Psi = \Omega \setminus \Phi$, by definition, we have $V \setminus W_2(\Psi) = \{v \in V \mid \forall \pi \in \Pi\ \exists \sigma \in \Sigma \text{ s.t. } \omega(v, \sigma, \pi) \in \Phi\}$. Theorem 1 allows to change existential and universal quantifiers, i.e., $V \setminus W_2(\Psi) = \{v \in V \mid \exists \sigma \in \Sigma \text{ s.t. } \forall \pi \in \Pi\ \omega(v, \sigma, \pi) \in \Phi\} = W_1(\Phi)$. If for every strategy π of player 2, there exists a strategy σ for player 1 that wins from vertex v, then there exists a (unique) strategy σ for player 1 that wins against every strategy π of player 2.

The computational problem. Given a game graph with bounded Büchi objective Φ the goal is to compute the set $W_1(\Phi)$. The focus of this paper is on bounded Büchi and bounded

coBüchi objectives, and when we mention winning vertices or winning strategies, we mean winning for bounded Büchi objectives, unless stated otherwise.

Closed Sets. A set $U \subseteq V$ of vertices is a closed set for player 1 if $\forall u \in (U \cap V_1) : Out(u) \subseteq U$ and $\forall u \in (U \cap V_2) : Out(u) \cap V_2 \neq \emptyset$. We define player-2 closed sets analogously. Observe that every closed set U induces a subgame graph denoted $G \restriction U$.

A connection between closed sets, winning for safety, reachability and coBüchi objectives in the following proposition.

▶ **Proposition 2** ([11] Proposition 2.2). *Consider a game graph Γ, and a closed set U for player 1. Then, the following assertions hold:*

1. *Player 2 has a winning strategy for the objective Safety(U) for all vertices in U, that is, player 2 can ensure that if the play starts in U, then the play never leaves the set U.*
2. *If $U \cap B = \emptyset$ (i.e., there is no Büchi vertex in U), then every vertex in U is winning for player 2 for the coBüchi objective.*

Attractors. For a set of "target" vertices $T \subseteq V$, the set of vertices from which player ρ can reach T against all strategies of the other player, is called the *player-ρ attractor* of T; formally [25, 23], $attr_\rho(T, \Gamma) = W_\rho(\text{Reach}(T, \Gamma))$. An attractor $A = attr_\rho(T, \Gamma)$ can be computed in $O(m)$ time [5, 16].

The following observation stipulates the connection between closed sets and attractors.

▶ **Observation 3** ([11]). *For all game graphs Γ, all players $\rho \in \{1, 2\}$, and all sets $U \subseteq V$ we have the following: The set $V \setminus attr_\rho(U, \Gamma)$ is a closed set for player ρ, i.e., no player-ρ vertex in $V \setminus attr_\rho(U, \Gamma)$ has an edge to $attr_\rho(U, \Gamma)$ and every vertex of the other player in $V \setminus attr_\rho(U, \Gamma)$ has an edge in $V \setminus attr_\rho(U, \Gamma)$.*

3 Algorithms for Graphs

Graphs are a special case of game graphs with $V_2 = \emptyset$. Hereon, we will call this "the graph case" as opposed to "the game graph case" (where $V_1 \neq \emptyset$ and $V_2 \neq \emptyset$). The objectives we consider are *prefix independent*, i.e., if $\omega \in \Omega$, then any play obtained by adding or removing a finite prefix to or from ω is also in Ω. Hence, with respect to computing winning vertices, it is enough to focus on strongly connected graphs. The reasoning is as follows.

In the input graph, we call a strongly connected component (SCC) S *good* if the graph restricted to S has a winning vertex. Due to prefix independence, all vertices in a good SCC and those from which you can reach a good SCC are winning. We will prove that such vertices are exactly the winning vertices, and that this set can be computed by the following procedure:

- Compute the SCCs of the input graph (can be done in linear time [22]).
- Determine for each SCC if it is good (this step depends on the objective).
- Consider the set of all vertices belonging to a good SCC. Perform reachability to this set. (This can also be done in linear time.)

▶ **Lemma 4.** *A vertex v is a winning vertex if and only if it there is path from v to some vertex in a good SCC.*

Proof. As mentioned before, due to prefix independence, if v has a path to some vertex in a good SCC, then it is winning. Next, we show the converse.

If v is winning, then there is a winning play ω starting at v. Since SCCs themselves form a directed acyclic graph (DAG), ω must eventually enter an SCC S and stay there. Again,

due to prefix independence, the vertices visited by ω in S are also winning, i.e., S is a good SCC. ◀

By Lemma 4 and the procedure described above it, the problem of computing the winning vertices is reduced to determining, given a strongly-connected graph, whether there is a winning vertex or not. More formally, we get the following lemma.

▶ **Lemma 5.** *Let $S_1, S_2, \ldots$ be SCCs of the graph $G = (V, E)$. When $V_2 = \emptyset$, i.e., in the graph case, for a prefix independent objective, the set of winning vertices can be computed in time $O(m + \sum_i t(S_i))$ time, where $m = |E|$ and $t(S_i)$ is the time required to compute whether S_i is a good SCC or not.*

In this paper, we consider bounded Büchi and bounded coBüchi objectives.

3.1 The Bounded Büchi Objective

We are given a graph $G = (V, E)$, a set B of Büchi vertices, and a positive integer d. A cyclic-walk in G is a walk $(v_1, v_2, \ldots, v_\ell)$ such that $v_1 = v_\ell$. We say that a cyclic-walk C is *feasible* if it has at least one Büchi vertex and the number of edges in C between any two consecutive Büchi vertices is at most d. We assume that G is strongly connected, and our goal is to determine if there is a winning vertex in G. Then, using Lemma 5, we generalize the result to a graph that might not be strongly connected. The following lemma reduces this problem to finding a feasible cyclic-walk in G.

▶ **Lemma 6.** *The strongly-connected input graph G has a winning vertex with respect to the bounded Büchi objective if and only if it has a feasible cyclic-walk.*

Proof. If G has a winning vertex, say v, then there is a winning play ω that starts at v. Let $\omega = \langle v_0 = v, v_1, v_2, \ldots \rangle$; so by the definition of winning play, $\exists i \geqslant 1$ such that $\forall j \geqslant i : \{v_j, v_{j+1}, \ldots, v_{j+d-1}\} \cap B \neq \emptyset$. Consider the set $Inf(\omega)$ of vertices that appear infinitely often in ω. Since ω is winning, $Inf(\omega) \cap B \neq \emptyset$. Thus, we can choose a $j' \geqslant i$ such that $v_{j'} \in Inf(\omega) \cap B$. Since $v_{j'}$ appears infinitely often, for some $j'' > j'$, we have that $v_{j''} = v_{j'}$. Thus $(v_{j'}, v_{j'+1}, \ldots, v_{j''} = v_{j'})$ is a feasible cyclic-walk because $j' \geqslant i$ and, as mentioned earlier, $\forall j \geqslant i : \{v_j, v_{j+1}, \ldots, v_{j+d-1}\} \cap B \neq \emptyset$.

In the other direction, if G has a feasible cyclic-walk, then we can keep traversing it to construct a winning play, which means G has a winning vertex. ◀

An $O(dm)$-time algorithm for bounded Büchi

Next, we recall the basic $O(dm)$-time algorithm to determine if there is a feasible cyclic-walk. This algorithm tries to trace a feasible cycle by maintaining a counter with each possible non-Büchi vertex denoting how far away we are from the last visit to a Büchi vertex. We construct a $(d+1)$-layered auxiliary graph $G^* = (V^*, E^*)$, where $V^* = (B \times \{0\}) \cup ((V \setminus B) \times \{1, \ldots, d\})$. We define a more general graph here that we also use in Section 4. We illustrate an example in Figure 1. So, for $(v, \ell) \in V^*$, the integer ℓ corresponds to the aforementioned counter. We call the vertices in $B \times \{0\}$ Büchi vertices and the vertices in $(V \setminus B) \times \{1, \ldots, d\}$ non-Büchi vertices. The edge set E^* is constructed by Algorithm 1. The last layer of the auxiliary graph is actually not needed for the graph case but is needed for the game graph case later. Observe that the auxiliary graph is also a game graph. (The ownership of the vertices will be defined later in a natural way.)

Algorithm 1 Construction of the auxiliary graph G^* from G, B, and d. It is easy to see that the running time of this algorithm is $O(dm)$ and G^* has at most dm edges.

procedure ConstructAuxiliaryGraph($G = (V, E), B \subseteq V$, d)
  $V^* \leftarrow (B \times \{0\}) \cup ((V \setminus B) \times \{1, \ldots, d\})$ and $E^* \leftarrow \emptyset$.
  for $(u, v) \in E$ such that $v \notin B$ (add counter-incrementing edges) **do**
    if $u \notin B$ **then**
      for $i \in \{1, \ldots, d-1\}$ **do**
        Add $((u, i), (v, i+1))$ to E^*.
      Add $((u, d), (v, d))$ to E^* (edges in the last layer to $V \setminus B$ stay in the last layer).
    else Add $((u, 0), (v, 1))$ to E^*.
  for $(u, v) \in E$ such that $v \in B$ (add counter-resetting edges) **do**
    if $u \notin B$ **then**
      for $i \in \{1, \ldots, d\}$ **do**
        Add $((u, i), (v, 0))$ to E^*.
    else Add $((u, 0), (v, 0))$ to E^*.
  return $G^* = (V^*, E^*)$
procedure AuxiliaryGraph-d-Layers($G = (V, E), B \subseteq V$, d)
  $G^* \leftarrow$ ConstructAuxiliaryGraph($G = (V, E), B \subseteq V, d$)
  Return the graph resulted by removing layer-d from G^*, called $G' = (V', E')$.

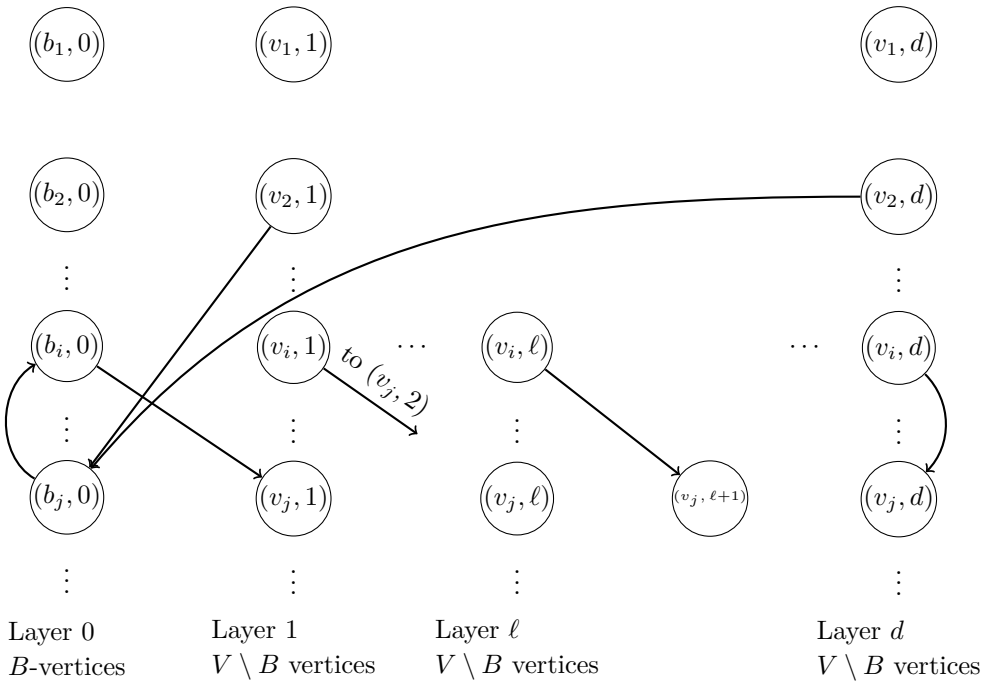

Figure 1 An illustration of how the auxiliary layered graph is constructed. If G contains the edges (b_j, b_i), (b_i, v_j), (v_2, b_j), and (v_i, v_j), then the auxiliary layered graph G^* will have shown edges.

▶ **Lemma 7.** *The running time of the procedures* ConstructAuxiliaryGraph *and* AuxiliaryGraph-d-Layers *in Algorithm 1 is* $O(dm)$.

Proof. In ConstructAuxiliaryGraph, each of the outer for loop runs for at most m iterations, and each of the inner for loops runs for at most d iterations. AuxiliaryGraph-

D-Layers just calls ConstructAuxiliaryGraph and removes the last layer, which takes $O(dm)$ time. ◀

For the graph case, we are interested in G^* induced on layers-$\{0, 1, \ldots, d-1\}$. Let G' denote this graph.

▶ **Lemma 8.** *The strongly-connected input graph G has a feasible cyclic-walk if and only if G' has a cycle.*

Proof. Let $C = (b_1, v_{1,1}, \ldots, v_{1,\ell_1}, b_2, v_{2,1}, \ldots, v_{2,\ell_2}, b_3, \ldots, b_1)$, where each $b_i \in B$, each $v_{i,j} \in V \setminus B$, and each $\ell_i \leqslant d-1$, be a feasible cyclic-walk in G. There is a corresponding cyclic-walk C' in G':

- for each $(b_i, v_{i,1}) \in C$, the edge $((b_i, 0), (v_{i,1}, 1)) \in E'$,
- for each $(v_{i,j}, v_{i,j+1}) \in C$, the edge $((v_{i,j}, j), (v_{i,j+1}, j{+}1)) \in E'$,
- for each $(v_{i,\ell_j}, b_{i+1}) \in C$, the edge $((v_{i,\ell_j}, \ell_j), (b_{i+1}, 0)) \in E'$, and
- for the final edge $(v_{i,\ell_j}, b_1) \in C$, the edge $((v_{i,\ell_j}, \ell_j), (b_1, 0)) \in E'$.

If C' consists of union of cycles can be short-cut to get a cycle in G'.

In the other direction, consider a cycle in G'. A projection of this cycle on the first coordinate of the vertices, by construction, gives a feasible cyclic-walk in G, because the number of edges between consecutive Büchi vertices is at most d. ◀

Thus, by Lemmas 6 and 8, we get Algorithm 2.

Algorithm 2 This algorithm determines if the strongly-connected input graph has a winning vertex with respect to the bounded Büchi objective.

```
procedure BoundedBüchi(G = (V, E), B ⊆ V, d)
    G' ← AuxiliaryGraph-d-Layers(G, B, d)
    Run depth-first search on G' to determine if it has a cycle.
    if G' has a cycle then
      return "G has a winning vertex."
    else
      return "G does not have a winning vertex."
```

▶ **Lemma 9.** *Algorithm 2 determines if the strongly-connected input graph G has a winning vertex with respect to the bounded Büchi objective in $O(dm)$ time.*

Proof. By Lemmas 6 and 8, G has a winning vertex if and only if G' has a cycle. Since a depth-first search finds if there is a cycle in G', the correctness of the algorithm is established. By Lemma 7, AuxiliaryGraph-d-Layers takes $O(dm)$ time, and a depth-first search on G' takes time $O(dm)$, because the number of edges in G' is $O(dm)$. ◀

Thus, by Lemma 5, we get the following theorem.

▶ **Theorem 10.** *The set of winning vertices for the bounded Büchi objective in the graph case can be computed in time $O(dm)$.*

Proof. Let $S_1, S_2, \ldots$ be SCCs of the input graph $G = (V, E)$. Let $m_1, m_2, \ldots$ be the number of edges in the SCCs $S_1, S_2, \ldots$. Then, by Lemma 9, for $i = 1, 2, \ldots$, we can determine in time $O(dm_i)$ whether S_i is good. Since $m \geqslant \sum_i m_i$, the proof is complete by Lemma 5. ◀

An $O(|B|m)$-time algorithm for bounded Büchi

Now, we briefly discuss an $O(|B|m)$-time algorithm for bounded Büchi. Given $G = (V, E)$ and B, consider the graph $G' = (B, E')$ such that $(b, b') \in E'$ if the distance from b to b' in G is at most d. We allow self loops in G'. It is easy to see that G has a feasible-cyclic walk if and only if G' has a cycle. To construct G', we perform $|B|$ breadth-first searches, one starting from each vertex in B. This takes time $O(|B|m)$. Then, by a similar argument as in the proof of Theorem 10, we get the following theorem.

▶ **Theorem 11.** *The set of winning vertices for the bounded Büchi objective in the graph case can be computed in time $O(|B|m)$.*

▶ Remark 12. Note that both algorithms that we have seen so far can take $\Theta(n^3)$ time if $m = \Theta(n^2)$ and B and d are $\Theta(n)$. The next algorithm we see is combinatorial and has running time $O(n^{2.5} \log n)$ for the worst setting of the parameters and breaks the cubic barrier. This also rules out any conditional lower bound approaches to get an $\Omega(n^3)$ lower bound for combinatorial algorithms.

An $O((m + |B|^2)\sqrt{n} \log n)$-time algorithm for bounded Büchi

In this section, we present an $O((m+|B|^2)\sqrt{n} \log n)$-time algorithm for bounded Büchi in the graph case. This is one of our main contributions. Here, we give a procedure that computes distances between all pairs of Büchi vertices if the distance is at least $\sqrt{N}$, where $N \geqslant |V|$ is a parameter that we will fix later. This information can be used to reduce the number of layers in the auxiliary graph to $\sqrt{N}$. By dist, we denote the distance function with respect to G. For any $u, v \in V$, if $u \neq v$, then $\mathrm{dist}(u, v)$ denotes the length of a shortest path from u to v, and for any $u \in V$, $\mathrm{dist}(u, u)$ denotes the length of a shortest cycle through u.

Algorithm 3 This algorithm determines if the strongly-connected input graph has a winning vertex with respect to the bounded Büchi objective.

procedure RandBoundedBüchi($G = (V, E), B \subseteq V$, d, N)
  if $d < \sqrt{N}$ **then**
    return BoundedBüchi($G = (V, E), B \subseteq V$, d)
  Sample $4\sqrt{N} \ln N$ vertices uniformly at random, independently, and with replacement.
  $S \leftarrow$ the set of sampled vertices.
  for $s \in S$ **do**
    Perform incoming and outgoing breadth-first search (BFS) to and from s.
    Compute distances $\mathrm{dist}(b, s)$ and $\mathrm{dist}(s, b)$ for each $b \in B$ during the BFSs.
  $G' \leftarrow$ AuxiliaryGraph-d-Layers($G, B, \sqrt{N} - 1$)
  for $b \in B$ **do**
    for $b' \in B$ **do**
      $\mathrm{dist}^S(b, b') \leftarrow \infty$
      for $s \in S$ **do**
        $\mathrm{dist}^S(b, b') \leftarrow \min\{\mathrm{dist}^S(b, b'), \mathrm{dist}(b, s) + \mathrm{dist}(s, b')\}$
      if $\mathrm{dist}^S(b, b') \leqslant d$ **then**
        Add $((b, 0), (b', 0))$ to E' (this would be a self-loop if $b = b'$).
  Run depth-first search on G' to determine if it has a cycle.
  if G' has a cycle **then**
    return "G has a winning vertex."
  else
    return "G does not have a winning vertex."

▶ **Lemma 13.** *Let $N \geqslant |V|$. Algorithm 3 determines with probability at least $1 - 1/N^2$ if the strongly-connected input graph G has a winning vertex with respect to the bounded Büchi objective in $O((m + |B|^2)\sqrt{N} \log N)$ time. It never returns a* false positive*, i.e., if it outputs that G has a winning vertex, then it is correct with probability* 1*. Its running time is $O((m + |B|^2)\sqrt{N} \log N)$.*

Proof. If $d < \sqrt{N}$, then we are done by Lemma 9. Thus, we assume for the rest of the proof that $d \geqslant \sqrt{N}$.

For any $b, b' \in B$, by $T(b, b')$, we denote a fixed shortest cycle through b if $b = b'$ or a fixed shortest path from b to b' otherwise. Let the event that a vertex $v(b, b') \in T(b, b')$ is sampled into S be denoted by $\mathcal{E}(b, b')$. Since $v(b, b') \in T(b, b')$, we have that $\text{dist}(b, b') = \text{dist}(b, v(b, b')) + \text{dist}(v(b, b'), b')$. This implies that if $\mathcal{E}(b, b')$ occurs, then $\text{dist}(b, v(b, b'))$ and $\text{dist}(v(b, b'), b')$ are computed by the algorithm using the incoming and outgoing BFS at $v(b, b')$, and hence $\text{dist}^S(b, b') = \text{dist}(b, b')$. Let $\mathcal{E}^c(b, b')$ be the complement of $\mathcal{E}(b, b')$. Now, $\Pr[\mathcal{E}^c(b, b')] = (1 - \text{dist}(b, b')/|V|)^{4\sqrt{N} \ln N}$, because $1 - \text{dist}(b, b')/|V|$ is the probability that a fixed sample does not contain a vertex of $T(b, b')$ and we draw $4\sqrt{N} \ln N$ independent samples.

For any $b, b' \in B$, where $\text{dist}(b, b') \geqslant \sqrt{N}$, we denote the event that $\text{dist}^S(b, b') = \text{dist}(b, b')$ by $\mathcal{E}'(b, b')$. As noted earlier, $\text{dist}^S(b, b') = \text{dist}(b, b')$ if $\mathcal{E}(b, b')$ occurs, hence:

$$
\begin{aligned}
\Pr[\mathcal{E}'(b, b')] &\geqslant \Pr[\mathcal{E}(b, b')] = 1 - \Pr[\mathcal{E}^c(b, b')] && \mathcal{E}(b, b') \text{ is a subevent of } \mathcal{E}'(b, b'),\\
&= 1 - \left(1 - \frac{\text{dist}(b, b')}{|V|}\right)^{4\sqrt{N} \ln N} && \text{by the argument earlier,}\\
&\geqslant 1 - \left(1 - \frac{1}{\sqrt{N}}\right)^{4\sqrt{N} \ln N} && \text{because } \text{dist}(b, b')/|V| \geqslant 1/\sqrt{N},\\
&\geqslant 1 - \frac{1}{N^4} && \text{by well-known fact } (1 - 1/x)^x \leqslant 1/e.
\end{aligned}
$$

Since $N \geqslant |B|$, by the union bound and because the $\mathcal{E}'(b, b')$ are independent, we have $\Pr[\forall (b, b') \in B \times B : \mathcal{E}'(b, b')] \geqslant 1 - 1/N^2$. Let us condition on the event that for all $(b, b') \in B \times B : \mathcal{E}'(b, b')$, and let G' be the auxiliary graph constructed by the algorithm.

Suppose G has a winning vertex. By Lemma 6, there is a feasible cyclic-walk C in G. Then for any consecutive Büchi vertices b and b' in C, either $\text{dist}(b, b') \geqslant \sqrt{N}$, in which case there is an edge $((b, 0), (b', 0))$ or $\text{dist}(b, b') < \sqrt{N}$, in which case there exists a cycle $((b, 0), (u_1, 1), (u_2, 2), \ldots, (u_\ell, \ell), (b', 0))$ in G', where $\ell < \sqrt{N} - 1$. Thus, C induces a cycle in G'.

On the other hand, if there is a cycle C' in G', then a projection of C' on the first coordinate of the vertices, by construction of G', gives a feasible cyclic-walk in G after replacing all edges in C' of the form $((b, 0), (b', 0))$ by corresponding paths of length at most d that certify $\text{dist}^S(b, b')$. By Lemma 6, G has a winning vertex.

Also, if the algorithm does return that G has a winning vertex, then G' has a cycle, and existence of a feasible cyclic-walk in G can be shown in the same way as above. This shows that the algorithm never returns a false positive.

Running time

Incoming and outgoing BFSs from the vertices in S take time $O(m\sqrt{N} \log N)$. Auxiliary-Graph-d-Layers takes $O(m\sqrt{N})$ time. Computing dist^S takes time $O(|B|^2\sqrt{N} \log N)$. DFS on G' takes time $O(|B|^2 + m\sqrt{N})$. In total, Algorithm 3 has running time $O((m + |B|^2)\sqrt{N} \log N)$. ◀

Finally, we use Lemma 5 to generalize the above to a graph that may not be strongly connected. Fix N to be n in Algorithm 3 when running it for each SCC. Then, by a similar argument as in the proof of Theorem 10, we get the following theorem.

▶ **Theorem 14.** *The set of winning vertices for the bounded Büchi objective can be computed with probability at least* $1-1/n$ *in time* $O((m+|B|^2)\sqrt{n}\log n)$ *which is* $O(n^{2.5}\log n)$*. Moreover, the algorithm never returns a* false positive*, i.e., each vertex in the set it outputs is a winning vertex with probability* 1*.*

Proof. Let $S_1, S_2, \ldots$ be SCCs of the input graph $G = (V, E)$. Let $m_1, m_2, \ldots$ be the number of edges and by $\beta_1, \beta_2, \ldots,$ be the number of Büchi vertices in the SCCs $S_1, S_2, \ldots$, respectively. Then, by Lemma 13, for $i = 1, 2, \ldots$, the algorithm outputs in time $O((m_i + \beta_i^2)\sqrt{n}\log n)$ whether S_i is good. Since $m \geqslant \sum_i m_i$ and $|B|^2 = (\sum_i \beta_i)^2 \geqslant \sum_i \beta_i^2$, the running time bound is proved.

The probability bound is obtained by a union bound over at most n SCCs. Moreover, the algorithm never returns a *false positive* by Lemma 13. ◀

3.2 The Bounded coBüchi Objective

Given a graph $G = (V, E)$, a set C of vertices, and a positive integer d, a walk W is called a *feasible* walk if $W \subseteq C$ and the number of vertices in W is at least d. Let $G[C]$ be the graph induced by C. The bounded coBüchi problem reduces to finding a feasible walk, which further reduces to finding whether there is a cycle in $G[C]$ (can be done in linear time), and if not $G[C]$ is a directed acyclic graph (DAG), so it reduces to determining whether the length of a longest path in the DAG $G[C]$ is at least d (also can be done in linear time). This gives us the following theorem.

▶ **Theorem 15.** *The set of winning vertices for the bounded coBüchi objective in the graph case can be computed in time* $O(m)$*.*

4 Algorithms for Game Graphs

In this section, we present algorithms for the bounded Büchi objective in game graphs. We first introduce the auxiliary *game graph* similar to the auxiliary graph defined earlier. We then show that we can compute in $O(n^2d^2)$ time the winning set of a given bounded Büchi objective on game graphs by computing the winning set of a coBüchi objective on the auxiliary game graph. Finally, we show how to improve the running time to $O(n^2d)$ by using structural properties of the auxiliary game graph and adapting a known technique for solving Büchi Games [11].

The Auxiliary Game Graph. Given a game graph $\Gamma = (V, E, \langle V_1, V_2\rangle)$ with n vertices, m edges and a bounded Büchi objective boundedBüchi(B, d), we first construct the auxiliary graph by calling ConstructAuxiliaryGraph$((V, E), B, d)$ in Algorithm 1 and additionally partition the vertices of the auxiliary graph V^* into player-1 vertices V_1^* and player-2 vertices V_2^*, i.e., for each $(v, \ell) \in V^*$ we get $(v, \ell) \in V_1^*$ if $v \in V_1$ and $(v, \ell) \in V_2^*$ if $v \in V_2$. The auxiliary game graph has $O(nd) = O(n^2)$ vertices and $O(md) = O(mn)$ edges. We say that a vertex $(v, \ell) \in V^*$ is a *layer-ℓ vertex* and v is its *first component*.

For any play λ, we denote by λ_k the kth vertex of the play. If a play has a superscript, it denotes the starting vertex of the play, e.g λ^v means that the play λ starts at v. By λ_k^v we refer to the kth vertex of the play λ^v which starts at v. Given a finite feasible play $\lambda^{(w,\ell)}$ in Γ^* starting at (w, ℓ), we define Proj$(\lambda^{(w,\ell)})$ to be the projection of $\lambda^{(w,\ell)}$ on the

first component of the vertices in it; by definition, this finite play starts at w and is feasible in Γ. Analogously, given a finite feasible play λ^w in Γ, we define $\text{Lift}(\lambda^w, \ell)$ to be the unique finite feasible play in Γ^* starting at (w, ℓ) such that the first component of $\text{Lift}(\lambda^w, \ell)_k$ is the same as λ^w_k. For (u, v) in E such that $(u, j) \in V^*$ define (the appropriate next layer number if you followed the copy of (u, v) starting in layer j)

$$\text{NxtLyr}(u, v, j) = \begin{cases} j + 1 \text{ if } j < d \text{ and } v \notin B \\ d \text{ if } j = d \text{ and } v \notin B \\ 0 \text{ if } v \in B\,. \end{cases}$$

Now, define $\text{Lift}(\lambda^w, \ell)_1 = (w, \ell)$, and for $k > 1$, given $\text{Lift}(\lambda^w, \ell)_{k-1} = (\lambda^w_{k-1}, j)$ define $\text{Lift}(\lambda^w, \ell)_k = (\lambda^w_k, \text{NxtLyr}(\lambda^w_{k-1}, \lambda^w_k, j))$. Similarly, given the finite feasible play $\lambda^{(w,\ell)}$ in Γ^*, we define $\text{Shift}(\lambda^{(w,\ell)}, \ell')$ to be the finite play that starts at (w, ℓ') in Γ^* such that, for any k, the first components of $\lambda^{(w,\ell)}_k$ and $\text{Shift}(\lambda^{(w,\ell)}, \ell')_k$ are the same. By construction of Γ^* the finite play $\text{Shift}(\lambda^{(w,\ell)}, \ell')$ is well-defined because (1) edges going from layer-i vertices to layer-$(i + 1)$ vertices ($1 \leqslant i \leqslant d - 1$) exist in all layers with the same respective first components except in layer-d where these edges go again to layer-d, (2) edges going to layer-0 vertices exist in all layers ($1 \leqslant i \leqslant d$) and (3) because edges originating from layer-0 vertices implies that both plays are currently visiting the same layer-0 vertex.

In comparison, the goal of the two operations $\text{Proj}(\cdot)$ and $\text{Lift}(\cdot)$ is to map finite plays between Γ^* and Γ such that the finite play in Γ^* has, for all vertices, the same first component as the corresponding finite play in Γ and vice versa. In contrast, $\text{Shift}(\lambda^{(w,\ell)}, \ell')$ maps a finite play in Γ^* to a finite play also in Γ^* which has the same first component but a "shifted" starting vertex.

4.1 An $O(n^2d^2)$-time Algorithm for Bounded Büchi in Games

In this section, we show that we can compute the winning set of a given *bounded Büchi objective* on game graphs by computing the winning set of a *coBüchi objective* on the auxiliary game graph. Then we apply the best-known algorithm for computing the winning set of a Büchi objective on the auxiliary game graph to get the desired result.

In the following lemma, we prove that computing $W_1(\text{boundedBüchi}(B, d, \Gamma))$ is the same as computing $W_1(\text{coBüchi}(C^*, \Gamma^*))$ where C^* are the vertices in layers-$\{0, 1, \ldots, d-1\}$. Intuitively, when a play ϕ in $\text{coBüchi}(C^*, \Gamma^*)$ stays in layers-$\{0, 1, \ldots, d-1\}$, it reaches a vertex in layer 0 every at most d steps by construction of Γ^*. The layer-0 vertices correspond to the vertices in B which means that a play ϕ' in Γ defined as the projection on the first component of the vertices in ϕ visits a vertex in B every at most d steps which implies that $\phi' \in \text{boundedBüchi}(B, d, \Gamma)$. On the other hand, when player 1 has a strategy in Γ to visit a vertex in B every at most d steps, a similar strategy which visits the same vertices in the first component in Γ^* allows player 1 to stay in the first d layers of the auxiliary graph.

▶ **Lemma 16.** *Let $\Gamma = (V, E, \langle V_1, V_2\rangle)$ be a game graph with bounded Büchi objective boundedBüchi(B, d), let $\Gamma^* = (V^*, E^*, \langle V_1^*, V_2^*\rangle)$ be the corresponding auxiliary game graph, and let C^* be the vertices in the first d layers of the auxiliary graph, i.e., $C^* = \{(v, i) \in V^* \mid 0 \leqslant i \leqslant d - 1\}$. Then $\{w \mid (w, i) \in W_1(\text{coBüchi}(C^*, \Gamma^*))$, for some $0 \leqslant i \leqslant d\} = W_1(\text{boundedBüchi}(B, d, \Gamma))$.*

Proof. We first prove that $\{w \mid (w, i) \in W_1(\text{coBüchi}(C^*, \Gamma^*))$, for some $0 \leqslant i \leqslant d\} \subseteq W_1(\text{boundedBüchi}(B, d, \Gamma))$. Let $(w, i) \in W_1(\text{coBüchi}(C^*, \Gamma^*))$. Then player 1 has a winning

strategy σ^* in Γ^* such that for all player-2 strategies π^*, we have that $\omega((w,i),\sigma^*,\pi^*) \in \text{coBüchi}(C^*,\Gamma^*)$.

Whenever player 1 makes a move in Γ^*, we define the corresponding move in Γ as follows: For any finite play λ^w in Γ that ends in a player-1 vertex, define $\sigma(\lambda^w)$ to be the first component of $\sigma^*(\text{Lift}(\lambda^w,i))$. (It does not matter how we define σ for plays that do not start at w.)

Next, we argue why σ is a winning player-1 strategy for boundedBüchi(B,d,Γ) starting at w. Let π be an arbitrary player-2 strategy in Γ. We define a corresponding player-2 strategy π^* in Γ^*: for $\lambda^{(w,i)}$ that ends in a player-2 vertex (u,j), let $v = \pi(\text{Proj}(\lambda^{(w,i)}))$ and define $\pi^*(\lambda^{(w,i)}) = (v, \text{NxtLyr}(u,v,j))$.

Now, it is straightforward to show that the first component of $\omega((w,i),\sigma^*,\pi^*)_k$ is equal to $\omega(w,\sigma,\pi)_k$ by induction on k.

Since the play $\omega((w,i),\sigma^*,\pi^*) \in \text{coBüchi}(C^*,\Gamma^*)$, it stays in C^* after a finite number of steps. Note that to stay in C^* means to visit a layer-0 vertex after every at most d steps because there are only d layers in C^* and each step that does not go to a layer-0 vertex increases the layer counter. Since the first component of each layer-0 vertex is in B, the play $\omega(w,\sigma,\pi)$ visits a vertex in B every at most d steps after a finite number of steps and is in boundedBüchi(B,d,Γ).

The other direction, $W_1(\text{boundedBüchi}(B,d,\Gamma)) \subseteq \{w \mid (w,i) \in W_1(\text{coBüchi}(C^*,\Gamma^*)), \text{for some } 0 \leqslant i \leqslant d\}$ can be shown with a similar argument. ◀

To compute $W_1(\text{coBüchi}(C^*))$ in Γ^*, we observe that, by Theorem 1, $W_1(\text{coBüchi}(C^*)) = V^* \setminus W_2(\text{Büchi}(V^* \setminus C^*)) = V^* \setminus W_2(\text{Büchi}(\{(v,d) \in V^*\}))$. Since, traditionally, we always compute the player-1 winning set of a given objective, we swap player-1 and player-2 vertices in Γ^*. Then we compute $W = W_1(\text{Büchi}(\{(v,d) \in V^*\}))$ using the algorithm of Chatterjee and Henzinger [11], which is the fastest algorithm for Büchi games known, and project $V^* \setminus W$ on the first coordinate. We illustrate the details in Algorithm 4.

Algorithm 4 Determine $W_1(\text{boundedBüchi}(B,d))$, given a game graph Γ

```
1: procedure BoundedBüchiGames(Γ = (V, E, ⟨V1, V2⟩), B, d)
2:     (V*, E*) ← ConstructAuxiliaryGraph((V, E))
3:     V1* ← {(v,i) ∈ V* | v ∈ V1}, V2* ← {(v,i) ∈ V* | v ∈ V2}
4:     Γ* ← (V*, E*, V1*, V2*); B* ← {(v,d) ∈ V* | v ∈ V \ B}
5:     W ← BüchiGamesFast(Γ* = (V*, E*, ⟨V2*, V1*⟩), B*) ([11], Algorithm 5)
6:     return {x | (x,i) ∈ V* \ W for some 0 ⩽ i ⩽ d}
```

The correctness of Algorithm 4 is due to the correctness of the fast Büchi games algorithm [11, Theorem 2.14], the argument above, and Lemma 16. The argument for the running time of Algorithm 4 is as follows. We first construct Γ^* in $O(md)$ time and then compute the winning set of coBüchi(C^*) in time $O(|V^*|^2)$ [11, Theorem 2.14]. As $|V^*| = O(nd)$ and $d = O(n)$, we get the following theorem.

▶ **Theorem 17.** *The set of winning vertices for the bounded Büchi objectives in games can be computed in time $O(n^2d^2) = O(n^4)$.*

4.2 An $O(n^2d)$-time Algorithm for Bounded Büchi in Games

In this section, we give a refined running time analysis of Algorithm 4 giving us an $O(n^2d)$-time algorithm for bounded Büchi games. We first describe the fastest algorithm for Büchi Games [11] for completeness. Then, we identify key ideas of the refined running time analysis when the input is an auxiliary game graph and prove the improved running time formally.

4.2.1 The Büchi Games Algorithm of [11]

Given a game graph $\Gamma = (V, E, \langle V_1, V_2\rangle)$ and a set B of Büchi vertices[1], we fix an order on the edges. In this fixed order, the edges (u, v) where u is a non-Büchi player-2 vertex, i.e., $u \in (V_2 \setminus B)$, come before all other edges. We call them priority-1 edges. All the other edges are priority-0 edges.

▶ **Definition 18.** *Given a game graph $\Gamma = (V, E, \langle V_1, V_2\rangle)$, let $\Gamma_i = (V, E_i, \langle V_1, V_2\rangle)$ for $1 \leqslant i \leqslant \log n$ be a subgraph of Γ which we define as follows: For all $u \in V$, the set E_i contains the following edges:*

1. *If the outdegree of u in E is at most 2^i, E_i contains all edges of the form (u, v), i.e., if $|Out(u)| \leqslant 2^i$ then the set $\{(u, v) \mid v \in Out(u)\} \subseteq E_i$.*
2. *If the edge (v, u) belongs to the first 2^i inedges of vertex u in E, we have $(v, u) \in E_i$ ("first" means with respect to the fixed order we specified above).*

Note that $E_{i-1} \subseteq E_i$ since the order of the edges is fixed. We form a partition of V in Γ_i by giving each vertex a color:

- Blue*: A player-1 vertex v in Γ_i is blue if the outdegree of v is greater than 2^i.*
- Red*: A player-2 vertex u in Γ_i is red if it has no outedge in E_i.*[2]
- *All other vertices are* white*.*

Thus, if a player-1 vertex is white then all its outedges are in E_i, and if a player-2 vertex is white then it has at least one outgoing edge in E_i.

Algorithm description. The input of Algorithm 5 is a game graph Γ and a set of Büchi vertices B. Recall that every vertex in a player-1 closed set S without Büchi vertices cannot be in the player-1 winning set of the given Büchi objective $W_1(\text{Büchi}(B))$ (Proposition 2 (2)). We repeatedly find such a set S by removing from V the player-1 attractor of the set B (Proposition 3) and forming S from all the remaining vertices. Then we remove the player-2 attractor of S. In the algorithm, we identify such a set S_j at Line 11 and remove the attractor at Line 15. Note that a naive algorithm would take $O(nm)$ time, as the attractor of S could always be of size 1 and computing the attractor is in $O(m)$ time. To obtain a quadratic-time (in the number of vertices) algorithm, the improved algorithm of Chatterjee and Henzinger constructs, for $i = 1, \ldots, \log n$, the graph Γ_i which has at most 2^i edges. Due to the properties of Γ_i, it can be shown that the set S_j has size of at least 2^{i-1}. In this way, the attractor computation take time proportional to the removed vertices. Since player-1 vertices with missing outgoing edges or player-2 vertices with no outgoing edge in Γ^i, i.e., non-white vertices might still be able to reach a vertex in B, we compute the player-1 attractor of the non-white vertices combined with the vertices in B. We illustrate the details in Algorithm 5.

The definition of a *separating cut* further refines the definition of the winning regions for player 2 in this regard.

Separating cut. A set S of vertices induces a separating cut in a game graph Γ_i or Γ_i^j in Algorithm 5 if

1. the only edges from S to $V \setminus S$ come from player-2 vertices in S
2. every player-2 vertex in S has an edge to another vertex in S

[1] not to be confused with the input for the bounded Büchi problem in the previous and later sections

[2] In the algorithm of Chatterjee and Henzinger [11] red vertices are player-2 vertices where an edge of E is missing. We change this definition slightly, i.e., without changing their algorithm or correctness argument, by saying that player-2 vertices are red if they do not have any outedges in E_i.

Algorithm 5 Determine $W_1(\text{Büchi}(B))$, given a game graph Γ [11]

```
1: procedure BüchiGamesFast(Γ = (V, E, ⟨V_1, V_2⟩), B)
2:     Let j ← 0; U ← ∅; Y_0 ← attr_1(B, Γ); S_0 ← V \ Y_0; D_0 ← attr_2(S_0, Γ); Γ^j ← Γ;
3:     j ← j + 1;
4:     while D_{j−1} ≠ ∅ do
5:         Remove the vertices in D_{j−1} from Γ^{j−1} to obtain Γ^j; and U ← U ∪ D_{j−1};
6:         i ← 1;
7:         repeat
8:             Construct Γ_i^j from Γ^j as described in Definition 18.
9:             Let Z_i^j be the vertices of V^j that are either red or blue;
10:            Y_i^j ← attr_1(B^j ∪ Z_i^j, Γ_i^j);
11:            S_j ← V^j \ Y_i^j;
12:            i ← i + 1
13:        until S_j is nonempty or i ⩾ 1 + log n
14:        if S_j ≠ ∅ then
15:            D_j ← attr_2(S_j, Γ^j)
16:        else
17:            return V \ U
18:        j ← j + 1
```

3. every player-1 vertex in S is white and
4. $B \cap S = \emptyset$.

Thus, a separating cut S is a player-1 closed set where (i) player-1 vertices are white and which (ii) does not contain a vertex in B.

The following lemmas are needed to establish the improved running time guarantees in the next section. Detailed proofs can be found in the paper by Chatterjee and Henzinger [11].

Lemma 19 below says that the set S_j is indeed a separating cut in Γ^j (not only in Γ_i^j) and that due to the careful construction of Γ_i^j from the game graph Γ^j in iteration j, S_j does not include a vertex of the player-1 attractor of the Büchi vertices in Γ^j.

▶ **Lemma 19** ([11], Lemma 2.9). *Let S_j be the non-empty set computed by Algorithm 5 in iteration j. Then, (1) S_j is a separating cut in Γ^j; and (2) $S_j \cap attr_1(B^j, \Gamma^j) = \emptyset$.*

Lemma 20 establishes that the separating cut found in Γ_i^j is indeed the maximum separating cut in Γ_i^j. Also, if Γ_i^j contains a separating cut, Algorithm 5 finds it.

▶ **Lemma 20** ([11], Lemma 2.11). *Let Γ_i^j be the game graph in iteration j of the outer loop and iteration i of the inner loop. If S induces a separating cut in Γ_i^j, then $S \subseteq S_j$.*

Lemma 21 says that the set S_j is a separating cut in Γ_i^j. This does not follow from Lemma 19(1) because Γ_i^j might have less edges than Γ^j and separating cuts are not preserved if we only consider a subset of edges in Γ^j (property 2 might be violated).

▶ **Lemma 21** ([11], Lemma 2.12). *Consider an iteration j of the outer loop of Algorithm 5 such that the algorithm stops the inner loop at value i and identifies a non-empty set S_j. Then, S_j is a separating cut in Γ_i^j.*

4.2.2 Faster Algorithm for Bounded Büchi Games

In this section, we give the refined running time analysis of Algorithm 4. We note that Γ^* gets redefined to be $(V^*, E^*, \langle V_2^*, V_1^* \rangle)$ in Algorithm 4 on Line 4. Therefore, from hereon,

when we say player 1 (respectively player 2), we mean the player controlling the vertices in V_2^* (respectively, those in V_1^*).

Distinct vertices. We call a set of vertices S in Γ^* *distinct* if, for each pair of vertices $(v,\ell),(v',\ell') \in S$, we have $v \neq v'$.

Copies of a vertex. Let $\mathit{Copies}(v)$ denote the set of "copies" of a vertex $v \in V^*$, i.e., for a layer-0 vertex $(v,0)$ we have that $\mathit{Copies}((v,0)) = \{(v,0)\}$ and for a vertex (v,ℓ), where $\ell > 0$, we have $\mathit{Copies}((v,\ell)) = \{(v,1),\ldots,(v,d)\}$.

The improved running time guarantee is due to two key ideas.

Key idea 1. When *there is a* vertex (v,ℓ) in D_j then $\mathit{Copies}((v,\ell)) \subseteq D_j$, i.e., *all* its copies are in D_j.

On a very high level, the argument is that if there is a player-2 strategy to go from a vertex to S_j, then there exists a player-2 strategy from all copies of that vertex to S_j. While the idea is simple to state, a complicated machinery is needed to prove it formally. We prove the key idea in Claim 27 building on Definition 25 and Claim 26.

Now, if we follow the original running-time argument [11], then we can only claim that we remove 2^{i-1} vertices in *total* if the inner loop at Line 7 stops at iteration i, but the second key idea states something stronger.

Key idea 2. If the inner loop at Line 7 stops at iteration i^*, we remove 2^{i^*-1} *distinct* vertices.

Combining the key ideas, we remove from the game graph in iteration j all copies of those distinct vertices. The ith iteration of the loop at Lines 7–13 takes time $O(2^i nd)$ for constructing the auxiliary version of $(\Gamma^*)_i$ and performing the attractor computations. The iterations of the loop in Lines 7–13 before $i' < i$ amount to a total running time of $O(2^i nd)$. Thus, we charge the 2^{i-1} removed distinct vertices the cost of the iteration and the iterations before, i.e., each such removed orginal vertex is charged $O(nd)$. As we can remove only n distinct vertices since they correspond to the vertices in the game graph Γ, we have a total cost of $O(n^2 d)$.

For the second key idea to work, we must modify the original bounded Büchi instance (Γ, B, d) carefully. For every vertex in $v \in B$ we add a player-2 vertex v' which is not in B and an edge (v', v). Then we redirect all edges which go to v in the original instance and make them go to v' instead, i.e., for all $v \in B$ we have $V_2 \leftarrow V_2 \cup \{v'\}$ and $E \leftarrow (E \cup \{(v',v)\} \cup \{(u,v') \mid (u,v) \in E\}) \setminus \{(u,v) \in E\}$. Also, we increase d by one, as we increase the distance to all vertices in B by one. Note that this simple modification allows us to assume, without loss of generality, that all vertices in B have incoming edges from player-2 vertices only. Since we swap the player-1 vertices with player-2 vertices in Algorithm 4 we can assume that all incoming edges to a layer-0 vertex are from player-1 vertices. This adds at most n vertices and edges to Γ.

▶ **Observation 22.** *We can assume, without loss of generality, that all layer-0 vertices $v \in V^*$ of the auxiliary game graph Γ^* created at Line 4 in Algorithm 4 have no incoming edges from player-2 vertices, i.e., if $(v,0) \in V^*$ then $\mathit{In}((v,0)) \cap V_2^* = \emptyset$.*

With the above observation, we can prove the following proposition which is the crux of this section.

▶ **Proposition 23.** *Algorithm 4 runs in time $O(n^2 d) = O(n^3)$.*

Proof. In this proof we denote by (Γ^*, B^*) the input of Algorithm 5 at Line 4 of Algorithm 4. The input to Algorithm 4 is (Γ, B, d). If we can show that the running time of the call to Algorithm 5 at Line 4 is in $O(n^2 d) = O(n^3)$ we are done, as the rest of the operations of

Algorithm 4 are in $O(md)$. This entails constructing (Γ^*, B^*) and going through W. We therefore prove the following lemma.

▶ **Lemma 24.** *The total time Algorithm 4 spends in Algorithm 5 is $O(n^2 d) = O(n^3)$.*

Every vertex v in Γ^* has only $O(n)$ out-edges by the definition of the auxiliary game graph. Thus, when we consider the graphs $(\Gamma^*)_i$ of Definition 18 for $1 \leqslant i \leqslant \log n$, we have $(\Gamma^*)_{\log n} = \Gamma^*$. The construction of $(\Gamma^*)_i$ $(1 \leqslant i \leqslant \log n)$ takes time $O(nd \cdot 2^i)$.

We split the running time argument into two parts. In the first part, we bound the running time of all except the last iteration of the while loop at Line 4. In the second part of the analysis, we bound the running time of the last iteration of the same loop.

Running time bound for all iterations of the while loop except the last. Consider iteration j, and assume that Algorithm 5 stops the repeat-until loop at Line 13 with value i^* and it is not the last iteration of the while loop at Line 4. Thus, S_j is not empty. By Lemma 21, the set S_j is a separating cut in $(\Gamma^*)^j_{i^*}$. We make a detour to set up some claims.

We need the following definition because it helps us translate plays and strategies from a vertex to its copies.

▶ **Definition 25.** *If Γ^*_s is an induced subgraph of Γ^* such that for all (u, ℓ_s) in Γ^*_s we have that Copies$((u, \ell_s))$ are also in Γ^*_s, then we say that Γ^*_s has the* induced-symmetry *property or that it is* symmetrically induced*.*

The following claim is about the translation of a strategy from a vertex to its copy.

▷ Claim 26. Suppose Γ^*_s is symmetrically induced. Then, in Γ^*_s, if a player has a strategy to reach a copy of w from a copy of u, then from all copies of u, she has a strategy to reach some copy of w. More formally, in Γ^*_s, if player ρ has a strategy π to reach (w, ℓ_d) from (u, ℓ_s), then for all copies (u, ℓ'_s), she also has a strategy π' to reach (w, ℓ'_d) for some ℓ'_d.

Proof. We define π'. Consider a finite feasible play $\lambda^{(u,\ell'_s)}$ that ends in a player-ρ vertex (v, j). Let $\pi(\text{Shift}(\lambda^{(u,\ell'_s)}, \ell_s)) = (y, p)$. Define $\pi'(\lambda^{(v,\ell'_s)}) = (y, \text{NxtLyr}(v, y, j))$. Now, the play $\text{Shift}(\lambda^{(u,\ell'_s)}, \ell_s)$ is feasible and the strategy π' is well defined because Γ^*_s is symmetrically induced.

We argue why player ρ can reach a copy of w using π'. Let σ' be an arbitrary strategy for the other player, i.e., player $(3 - \rho)$. For any finite feasible play $\lambda^{(u,\ell_s)}$ that ends in a player-$(3 - \rho)$ vertex (v, j), let $\sigma'(\text{Shift}(\lambda^{(u,\ell_s)}, \ell'_s)) = (y, p)$. Define $\sigma(\lambda^{(u,\ell_s)}) = (y, \text{NxtLyr}(v, y, j))$. Again, $\text{Shift}(\lambda^{(u,\ell_s)}, \ell'_s)$ is feasible and σ is well defined because Γ^*_s is symmetrically induced.

Now, it is straightforward to show by induction on k that the first components of $\omega((u, \ell_s), \sigma, \pi)_k$ and $\omega((u, \ell'_s), \sigma', \pi')_k$ are the same. This means that if $\omega((u, \ell_s), \sigma, \pi)_k$ reaches (w, ℓ_d), then $\omega((u, \ell'_s), \sigma', \pi')_k$ reaches (w, ℓ'_d) for some ℓ'_d. ◀

The following claim is a formal version of the first key idea.

▷ Claim 27. If a vertex (v, ℓ) is in D_j, then $\mathit{Copies}((v, \ell)) \subseteq D_j$; and, $(\Gamma^*)^j$ has induced symmetry.

Proof. We prove the claim by induction on j.

Base case, $j = 0$. If $(v, \ell) \in D_0$, then there is a player-2 strategy π_1 to reach $(w, p) \in S_0$. The set $S_0 = V \setminus \mathit{attr}_1(B^*, \Gamma^*)$ is a player-1 closed set by Observation 3: This means that there is a player-2 strategy π_2 to stay inside S_0. By construction of Γ^*, any edge from a non-layer-d vertex goes to the next layer or to layer-0. Then, since $S_0 \cap B^* = \emptyset$, that is, since S_0 does not contain any layer-d vertices, any (infinite) play that stays inside S_0 must

eventually return to layer-0. Thus, player 2 can first use π_1 to reach $(w,p) \in S_0$ from (v,ℓ), then use π_2 to reach $(x,0) \in S_0$ from (w,p); effectively, this gives a player-2 strategy to go to $(x,0) \in S_0$ from (v,ℓ). Then, by Claim 26, player 2 has a strategy to reach a copy of $(x,0)$ from (v,ℓ') for any ℓ' because Γ^* itself has induced symmetry. Now, $(x,0)$ does not have any other copy, this means player 2 has a strategy to reach $(x,0) \in S_0$ from (v,ℓ'). By induced symmetry of Γ^* again, we have that all copies of (v,ℓ), i.e., $\mathit{Copies}((v,\ell))$ are in Γ^*; moreover, by the above argument, for each of these copies, there is a player-2 strategy to reach S_0, which implies that $\mathit{Copies}((v,\ell)) \subseteq D_0$. Noting that $(\Gamma^*)^0 = \Gamma^*$ has induced symmetry finishes the base case.

Induction step, $j \geqslant 1$. By induction hypothesis, $(\Gamma^*)^{j-1}$ has induced symmetry, and if a vertex (v,ℓ) is in D_{j-1}, then $\mathit{Copies}((v,\ell)) \subseteq D_{j-1}$. This implies that deleting D_{j-1} from $(\Gamma^*)^{j-1}$ to get $(\Gamma^*)^j$ means deleting all copies of a vertex being deleted. Therefore, since $(\Gamma^*)^{j-1}$ has induced symmetry, $(\Gamma^*)^j$ also has induced symmetry.

Since S_j is a separating cut (by Lemma 19), it is a player-1 closed set. Thus, by the same argument as in the base case that uses the induced symmetry of $(\Gamma^*)^j$, if (v,ℓ) is in D_j, then $\mathit{Copies}((v,\ell)) \subseteq D_j$. This completes the induction step and the proof. ◀

The following claim is the formal proof of the second key idea.

▷ Claim 28. The set S_j contains at least 2^{i^*-1} *distinct* vertices.

Proof. The proof is similar to the proof of [11, Lemma 2.13] except that we must now argue that all of the 2^{i^*-1} vertices are distinct. Consider the set S_j in the game graph of the iteration before, i.e., we argue about S_j in $(\Gamma^*)^j_{i^*-1}$. Note that we have the following two cases.

- In the first case, S_j contains a player-1 vertex (x,ℓ) for $1 \leqslant \ell \leqslant d$ that is blue in $(\Gamma^*)^j_{i^*-1}$. Thus, (x,ℓ) has outdegree at least 2^{i^*-1} in $(\Gamma^*)^j_{i^*}$ and none of these edges go to vertices in $V^j \setminus S_j$ in $(\Gamma^*)^j_{i^*}$. Thus, S_j contains at least 2^{i^*-1} vertices. Note that vertex (x,ℓ) can only have edges to vertices which are distinct to (x,ℓ), i.e., for all $((x,\ell),(y,\ell')) \in E^*$ we have $x \neq y$ because the game graph Γ does not have self loops.
- In the second case, all player-1 vertices in S_j are white in $(\Gamma^*)^j_{i^*-1}$. Thus, their outedges in $(\Gamma^*)^j_{i^*}$ and $(\Gamma^*)^j_{i^*-1}$ are identical. We now argue, why a player-2 vertex in S_j exists: Assume for contradiction that no player-2 vertex in S_j exists. Hence, S_j is a separating cut only consisting of player-1 vertices. As S_j is a separating cut in $(\Gamma^*)^j_{i^*}$ we have $S_j \cap B = \emptyset$. Thus, S_j is also a separating cut in $(\Gamma^*)^j_{i^*-1}$. But then, by Lemma 20, the algorithm would have terminated in iteration $i^* - 1$ which is a contradiction because it terminated in iteration i^*.

  Note that repeat-until loop at Lines 7–13 would have stopped in iteration $i^* - 1$ in $(\Gamma^*)^j_{i^*-1}$ as all player-1 vertices in S_j are white.

  Consider a player-2 vertex u in S_j. Note that u must have an edge $(u,v) \in (E^*)^j_i$ with $v \in S_j$ because S_j is a separating cut in $(\Gamma^*)^j_{i^*}$ (Lemma 21). Again, there are two possibilities:
  - For all player-2 vertices $u \in S_j$ there exists a vertex $v \in S_j$ with $(u,v) \in (E^*)^j_{i^*-1}$. But then S_j would be a separating cut in $(\Gamma^*)^j_{i^*-1}$ as the outedges of player 1 are identical in $(\Gamma^*)^j_{i^*}$ and $(\Gamma^*)^j_{i^*-1}$. By Lemma 20, the separating cut would have been found in iteration $i^* - 1$ of the repeat-until loop at Line 7, which is a contradiction.
  - Therefore, there exists a player-2 vertex $u \in S_j$ that has an edge $(u,v) \in (E^*)^j_{i^*}$ to a vertex $v \in S_j$ but this edge is not contained in $(E^*)^j_{i^*-1}$. This can only happen if v has at least 2^{i^*-1} other inedges in $(E^*)^j_{i^*-1}$. Note that u is a player-2 vertex not in $(B^*)^j$

(because all vertices of $(B^*)^j$ belong to Y^j), and hence the edge (u, v) has priority 1 and recall that by the fixed inorder of edges priority-1 edges come before all priority-0 edges. Thus, it follows that since the edge (u, v) is not in $(\Gamma^*)^j_{i^*-1}$, all inedges of v that are in $(\Gamma^*)^j_{i^*-1}$ must have priority 1 by the fixed order of inedges, that is, all the inedges of v in $(\Gamma^*)^j_{i^*-1}$ are from non-Büchi player-2 vertices. Note that $v \in S_j$ and since S_j is a separating cut and, thus, a closed set, all player-2 vertices which are not in B^* with an edge to v are also in S_j. Since v has at least 2^{i^*-1} inedges from player-2 vertices which are not in B^*, the set S_j must contain at least 2^{i^*-1} vertices.
Furthermore, all incoming edges are from distinct vertices: Note that v cannot be a layer 0 vertex of Γ^*, because by Observation 22 all vertices in B of the given bounded Büchi objective have no incoming edges from a player-2 vertex. Also, layer-d vertices cannot be in S_j as they are in B^* and would be in the player-1 attractor $Y^j_{i^*}$ computed at Line 10. All other vertices in Γ^* have incoming edges only from distinct vertices. Thus, all 2^{i^*-1} such vertices are distinct. ◀

Due to Claim 28, S_j contains at least 2^{i^*-1} distinct vertices, and since $S_j \subseteq D_j$, the set D_j also contains all copies of all vertices in S_j due to Claim 27. All of D_j is deleted. We resume from the detour. The time spent in all graphs $(\Gamma^*)^j_1, \ldots, (\Gamma^*)^j_{i^*}$, i.e., the time spent in the repeat-until loop at Line 7 for the graph construction and the attractor computations, sums up to $O(2^{i^*} \cdot nd)$. We charge $O(nd)$ work to each distinct vertex. This accounts for all the running time except for the last iteration of the outer loop. Since we always remove all copies of a vertex $v \in S_j$, the algorithm deletes at most n distinct vertices throughout a run of the algorithm. Thus, the total time spent over the whole algorithm other than the last iteration is $O(n^2 d)$.

The last iteration of the outer loop. In the last iteration j^* of the outer loop, when no vertex is deleted, the algorithm works on all $\log n$ game graphs, spending time $O(n \cdot 2^i)$ on game graph $(\Gamma^*)^{j^*}_i$. Since each graph $(\Gamma^*)^{j^*}_i$ has at most $nd \cdot 2^{i+1}$ edges and there are $\log n$ graphs, the total number of edges worked in the last iteration is $\sum_{i=1}^{\log n} nd \cdot 2^{i+1} = 4nd \sum_{i=1}^{\log n} 2^{i-1} = 4nd(2^{\log n} - 1) = 4nd(n-1) = O(n^2 d)$. ◀

▶ **Theorem 29.** *The set of winning vertices for the bounded Büchi objective and bounded coBüchi objectives in game graphs can be computed in time $O(n^2 d) = O(n^3)$.*

References

1. Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *FOCS*, pages 434–443. IEEE Computer Society, 2014. URL: https://doi.org/10.1109/FOCS.2014.53, doi:10.1109/FOCS.2014.53.
2. Amir Abboud, Virginia Vassilevska Williams, and Huacheng Yu. Matching triangles and basing hardness on an extremely popular conjecture. *SIAM J. Comput.*, 47(3):1098–1122, 2018. URL: https://doi.org/10.1137/15M1050987, doi:10.1137/15M1050987.
3. Bowen Alpern and Fred B. Schneider. Defining Liveness. *Information Processing Letters*, 21(4):181–185, 1985.
4. Rajeev Alur and Thomas A. Henzinger. Finitary Fairness. *ACM Transactions on Programming Languages and Systems*, 20(6):1171–1194, 1998.
5. C. Beeri. On the membership problem for functional and multivalued dependencies in relational databases. *ACM Trans. Datab. Sys.*, 5(3):241–259, 1980.
6. A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, and Y. Zhu. Bounded model checking. *Advances in Computers*, 58:117–148, 2003.

7 Mikolaj Bojanczyk and Thomas Colcombet. Bounds in ω-Regularity. In *Proceedings of the 21st Annual IEEE Symposium on Logic in Computer Science, LICS'06*, pages 285–296. IEEE Computer Society, 2006.

8 Krishnendu Chatterjee, Wolfgang Dvorák, Monika Henzinger, and Veronika Loitzenbauer. Conditionally optimal algorithms for generalized büchi games. In *MFCS*, volume 58 of *LIPIcs*, pages 25:1–25:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. URL: https://doi.org/10.4230/LIPIcs.MFCS.2016.25, doi:10.4230/LIPIcs.MFCS.2016.25.

9 Krishnendu Chatterjee, Wolfgang Dvorák, Monika Henzinger, and Veronika Loitzenbauer. Model and objective separation with conditional lower bounds: Disjunction is harder than conjunction. In *LICS*, pages 197–206. ACM, 2016. doi:10.1145/2933575.2935304.

10 Krishnendu Chatterjee, Wolfgang Dvorák, Monika Henzinger, and Alexander Svozil. Algorithms and conditional lower bounds for planning problems. In Mathijs de Weerdt, Sven Koenig, Gabriele Röger, and Matthijs T. J. Spaan, editors, *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling, ICAPS 2018, Delft, The Netherlands, June 24-29, 2018*, pages 56–64. AAAI Press, 2018. URL: https://aaai.org/ocs/index.php/ICAPS/ICAPS18/paper/view/17639.

11 Krishnendu Chatterjee and Monika Henzinger. Efficient and dynamic algorithms for alternating büchi games and maximal end-component decomposition. *J. ACM*, 2014.

12 Krishnendu Chatterjee, Thomas A. Henzinger, and Florian Horn. Finitary Winning in ω-regular Games. *ACM Transactions on Computational Logic*, 11(1), 2009.

13 Alonzo Church. Logic, arithmetic, and automata. In *Proceedings of the International Congress of Mathematicians*, pages 23–35, 1962.

14 Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors. *Handbook of Model Checking*. Springer, 2018.

15 E. A. Emerson, A. K. Mok, A. P. Sistla, and J. Srinivasan. Quantitative temporal reasoning. *Real-Time Systems*, 4(4):331–352, 1992.

16 N. Immerman. Number of quantifiers is better than number of tape cells. *Journal of Computer and System Sciences*, pages 384–406, 1981. doi:10.1016/0022-0000(81)90039-8.

17 Orna Kupferman, Nir Piterman, and Moshe Y. Vardi. From liveness to promptness. *Formal Methods Syst. Des.*, 34(2):83–103, 2009.

18 Zohar Manna and Amir Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, 1992.

19 Donald A. Martin. Borel determinacy. *Annals of Mathematics*, 102(2):363–371, 1975.

20 Amir Pnueli and Roni Rosner. On the Synthesis of a Reactive Module. In *Proceedings of the 16th Annual ACM Symposium on Principles of Programming Languages, POPL'89*, pages 179–190, 1989.

21 Michael Oser Rabin. Automata on Infinite Objects and Church's Problem. *Transactions of the American Mathematical Society*, 141:1–35, 1969.

22 Robert E. Tarjan. Depth first search and linear graph algorithms. *SIAM J. Comput.*, 1(2):146–160, 1972.

23 Wolfgang Thomas. Languages, Automata, and Logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, Beyond Words. Springer, 1997.

24 Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proceedings of the ICM*, volume 3, pages 3431–3472, 2018.

25 W. Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200(1–2):135–183, 1998. doi:10.1016/S0304-3975(98)00009-7.