Beyond Value Iteration for Parity Games: Strategy Iteration with Universal Trees

Zhuan Khye Koh
Department of Mathematics, London School of Economics and Political Science, United Kingdom

Georg Loho
Discrete Mathematics and Mathematical Programming, University of Twente, The Netherlands

Abstract

Parity games have witnessed several new quasi-polynomial algorithms since the breakthrough result of Calude et al. (STOC 2017). The combinatorial object underlying these approaches is a *universal tree*, as identified by Czerwiński et al. (SODA 2019). By proving a quasi-polynomial lower bound on the size of a universal tree, they have highlighted a barrier that must be overcome by all existing approaches to attain polynomial running time. This is due to the existence of worst case instances which force these algorithms to explore a large portion of the tree.

As an attempt to overcome this barrier, we propose a strategy iteration framework which can be applied on any universal tree. It is at least as fast as its value iteration counterparts, while allowing one to take bigger leaps in the universal tree. Our main technical contribution is an efficient method for computing the least fixed point of 1-player games. This is achieved via a careful adaptation of shortest path algorithms to the setting of ordered trees. By plugging in the universal tree of Jurdziński and Lazić (LICS 2017), or the Strahler universal tree of Daviaud et al. (ICALP 2020), we obtain instantiations of the general framework that take time $O(mn^2 \log n \log d)$ and $O(mn^2 \log^3 n \log d)$ respectively per iteration.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Theory of computation → Logic and verification; Mathematics of computing → Discrete mathematics

Keywords and phrases parity games, strategy iteration, value iteration, progress measure, universal trees

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.63

Related Version *Full Version*: `https://arxiv.org/abs/2108.13338`

Funding This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement no. 757481–ScaleOpt).

Acknowledgements We are grateful for the helpful comments and support by László A. Végh. We would also like to thank Xavier Allamigeon, Nathanaël Fijalkow and Marcin Jurdziński for inspiring discussions.

1 Introduction

A *parity game* is an infinite duration game between two players Even and Odd. It takes place on a sinkless directed graph $G = (V, E)$ equipped with a *priority* function $\pi : V \to \{1, 2, \dots, d\}$. Let $n = |V|$ and $m = |E|$. The node set V is partitioned into $V_0 \sqcup V_1$ such that nodes in V_0 and V_1 are owned by Even and Odd respectively. The game starts when a token is placed on a node. In each turn, the owner of the current node moves the token along an outgoing arc to the next node, resulting in an infinite walk. If the highest priority occurring infinitely often in this walk is even, then Even wins. Otherwise, Odd wins.

By the positional determinacy of parity games [8], there exists a partition of V into two subsets from which Even and Odd can force a win respectively. The main algorithmic problem of parity games is to determine this partition, or equivalently, to decide the winner given a starting node. This is a notorious problem that lies in NP $\cap$ co-NP [9], and also in UP $\cap$ co-UP [14], with no known polynomial algorithm to date.

Due to its intriguing complexity status, as well as its fundamental role in automata theory and logic [9, 18], parity games have been intensely studied over the past three decades. Prior to 2017, algorithms for solving parity games, e.g. [27, 15, 26, 4, 24, 17, 25, 20, 2], are either exponential or subexponential. In a breakthrough result, Calude et al. [5] gave the first quasi-polynomial algorithm. Since then, many other quasi-polynomial algorithms [10, 16, 19, 22, 3] have been developed. Most of them have been unified by Czerwiński et al. [6] via the concept of a *universal tree*. A universal tree is an ordered tree into which every ordered tree of a certain size can be isomorphically embedded. They proved a quasi-polynomial lower bound on the size of a universal tree.

Value iteration. The starting point of this paper is the classic *progress measure* algorithm [15, 16] for solving parity games. It belongs to a broad class of algorithms called *value iteration* – a well-known method for solving more general games on graphs such as mean payoff games and stochastic games. In value iteration, every node v in G is assigned a value $\mu(v) \in \mathcal{V}$ from some totally ordered set $\mathcal{V}$, and the values are locally improved until we reach the *least fixed point* of a set of operators associated with the game. The set $\mathcal{V}$ is called the *value domain*, which is usually a bounded set of real numbers or integers. For the progress measure algorithm, its value domain is the set of leaves $L(T)$ in a universal tree T. As the values are monotonically improved, the running time is proportional to $|L(T)|$. The first progress measure algorithm of Jurdziński [15] uses a perfect n-ary tree, which runs in exponential time. Its subsequent improvement by Jurdziński and Lazić [16] uses a quasi-polynomial-sized tree, which runs in $n^{\log(d/\log n)+O(1)}$ time.

Despite having good theoretical efficiency, the progress measure algorithm is not robust against its worst-case behaviour. In fact, it is known to realize its worst-case running time on very simple instances. As an example, let (G, π) be an arbitrary instance with maximum priority d, with d being even. For a small odd constant k, if we add two nodes of priority k as shown in Figure 1, then the progress measure algorithm realizes its worst-case running time. This is because the values of those nodes are updated superpolynomially many times.

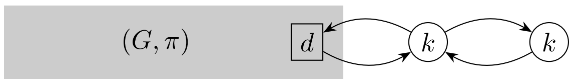

Figure 1 A worst-case construction for the progress measure algorithm. Nodes in V_0 and V_1 are drawn as squares and circles, respectively.

Strategy iteration. A different but related method for solving games on graphs is *strategy iteration*. For a parity game (G, π), a *(positional) strategy* τ for a player (say Odd) is a choice of an outgoing arc from every node in V_1. Removing the unchosen outgoing arcs from every node in V_1 results in a *strategy subgraph* $G_\tau \subseteq G$. A general framework for strategy iteration is given, e.g., in [12]. Following that exposition, to rank the strategies for Odd, one fixes a suitable value domain $\mathcal{V}$ and associates a valuation $\mu : V \to \mathcal{V}$ to each strategy. This induces a partial order over the set of strategies for Odd. Note that most valuations used in the literature can be thought of as fixed points of a set of operators associated with the

1-player game (G_τ, π) for Even. In every iteration, the algorithm maintains a strategy τ for Odd and its corresponding valuation $\mu : V \to \mathcal{V}$. Based on a *pivot rule*, it modifies τ to a better strategy τ', and updates μ to the valuation μ' of τ'. Note that $\mu' \geq \mu$. This process is repeated until we reach the optimal strategy for Odd.

Originally introduced by Hoffman and Karp for stochastic games [13], variants of strategy iteration for parity games have been developed [23, 26, 4, 24]. They usually perform well in practice, but tedious constructions of their worst case (sub)exponential complexity are known [11]. Motivated by the construction of small universal trees [16, 7], a natural question is whether there exists a strategy iteration algorithm with value domain $L(T)$ for a universal tree T. It is not hard to see that with value domain $L(T)$, unfortunately, the fixed point of a 1-player game (G_τ, π) may not be unique. Moreover, in a recent thesis [21], Ohlmann showed that a valuation that is fit for strategy iteration cannot be defined using $L(T)$.

Our contribution. We show that an adaptation of strategy iteration with value domain $L(T)$ is still possible. To circumvent the impossibility result of Ohlmann [21], we slightly alter the strategy iteration framework as follows. After pivoting to a strategy τ' in an iteration, we update the current node labeling μ to the least fixed point of $(G_{\tau'}, \pi)$ that is *pointwise at least* μ. In other words, we force μ to increase (whereas this happens automatically in the previous framework). Since the fixed point of a 1-player game may not be unique, this means that we may encounter a strategy more than once during the course of the algorithm. The motivation of our approach comes from tropical geometry, as discussed in the full version.

To carry out each iteration efficiently, we give a combinatorial method for computing the least fixed point of 1-player games with value domain $L(T)$. It relies on adapting the classic techniques of label-correcting and label-setting from the shortest path problem to the setting of ordered trees. When T is instantiated as a specific universal tree constructed in the literature, we obtain the following running times:

- The universal tree of Jurdziński and Lazić [16] takes $O(mn^2 \log n \log d)$.
- The Strahler universal tree of Daviaud et al. [7] takes $O(mn^2 \log^3 n \log d)$.
- The perfect n-ary tree of height $d/2$ takes $O(d(m + n \log n))$.

The total number of strategy iterations is trivially bounded by $n|L(T)|$, the same bound for the progress measure algorithm. Whereas we do not obtain a strict improvement over previous running time bounds, it is conceivable that our algorithm would terminate in fewer iterations than the progress measure algorithm on most examples. Moreover, our framework provides large flexibility in the choice of pivot rules. Identifying a pivot rule that may provide strictly improved (and possibly even polynomial) running time is left for future research.

Computing the least fixed point of 1-player games. Let (G_τ, π) be a 1-player game for Even, and μ^* be its least fixed point with value domain $L(T)$ for some universal tree T. Starting from $\mu(v) = \min L(T)$ for all $v \in V$, the progress measure algorithm successively lifts the label of a node based on the labels of its out-neighbours until μ^* is reached. However, this is not polynomial in general, even on 1-player games. So, instead of approaching μ^* from below, we approach it from above. This is reminiscent of shortest path algorithms, where node labels form upper bounds on the shortest path distances throughout the algorithm. In a label-correcting method like the Bellman–Ford algorithm, to compute shortest paths to a target node t, the label at t is initialized to 0, while the label at all other nodes is initialized to $+\infty$. By iteratively checking if an arc violates feasibility, the node labels are monotonically decreased. We refer to Ahuja et al. [1] for an overview on label-correcting and label-setting techniques for computing shortest paths.

In our setting, the role of the target node t is replaced by a (potentially empty) set of *even* cycles in G_τ. A cycle is said to be *even* if its maximum priority is even. However, this set is not known to us a priori. To overcome this issue, we define *base nodes* as candidate target nodes. A node $w \in V$ is a *base node* if it *dominates* an even cycle in G_τ, that is, it is a node with the highest priority in the cycle. Note that $\pi(w)$ is even.

To run a label-correcting method, we need to assign initial labels ν to the nodes in G_τ. The presumably obvious choice is to set $\nu(w) \leftarrow \min L(T)$ if w is a base node, and $\nu(w) \leftarrow \top$ otherwise, where $\top$ is bigger than every element in $L(T)$ ($\top$ is analogous to $+\infty$ for real numbers). However, this only works when T is a perfect n-ary tree. For a more complicated universal tree, the number of children at each internal vertex of T is not the same. Hence, it is possible to have $\nu(w) < \mu^*(w)$ for a base node w. We also cannot make $\nu(w)$ too large, as otherwise we may converge to a fixed point that is not pointwise minimal.

To correctly initialize $\nu(w)$ for a base node w, let us consider the cycles dominated by w in G_τ. Every such cycle C induces a subgame (C, π) on which Even wins because C is even. The least fixed point of (C, π) consists of leaves of an ordered tree T_C of height $j := \pi(w)/2$. Initializing $\nu(w)$ essentially boils down to finding such a cycle C with the "narrowest" T_C. To this end, let $\mathcal{T}_j$ be the set of *distinct* subtrees of height j of our universal tree T. We will exploit the fact that $\mathcal{T}_j$ is a poset with respect to the partial order of embeddability. In particular, let $\mathcal{C}_j$ be a set of chains covering $\mathcal{T}_j$, and fix a chain $\mathcal{C}_j^k$ in $\mathcal{C}_j$. We define the *width* of a cycle C as the "width" of the smallest tree in $\mathcal{C}_j^k$ into which T_C is embeddable. Then, we show that our problem reduces to finding a minimum width cycle dominated by w in G_τ.

To solve the latter problem, we construct an arc-weighted *auxiliary digraph* D on the set of base nodes. Every arc uv in D represents a path from base node u to base node v in G_τ, in such a way that minimum bottleneck cycles in D correspond to minimum width cycles in G_τ. It follows that the desired cycle C can be obtained by computing a minimum bottleneck cycle in D containing w. After getting C, we locate the corresponding subtree T' of T into which T_C is embeddable. Then, the label at w is initialized as $\nu(w) \leftarrow \min L(T')$.

With these initial labels, we show that a generic label-correcting procedure returns the desired least fixed point μ^* in $O(mn)$ time. The overall running time of this label-correcting method is dominated by the initialization phase, whose running time is proportional to the size of the chain cover $\mathcal{C}_j$. We prove that the quasi-polynomial universal trees constructed in the literature [16, 7] admit small chain covers. Using this result, we then give efficient implementations of our method for these trees.

In the full version of the paper, we also develop a label-setting method for computing μ^*, which is faster but only applicable when T is a perfect n-ary tree. Unlike the label-correcting approach, in a label-setting method such as Dijkstra's algorithm, the label of a node is fixed in each iteration. In the shortest path problem, Dijkstra's algorithm selects a node with the smallest label to be fixed in every iteration. When working with labels given by the leaves of a universal tree, this criterion does not work anymore. Let H be the subgraph of G_τ obtained by deleting all the base nodes. For $p \in \mathbb{N}$, let H_p be the subgraph of H induced by the nodes with priority at most p. We construct a suitable potential function by interlacing each node label with a tuple that encodes the topological orders in $H_2, H_4, \dots$. In every iteration, a node with the smallest potential is selected, and its label is fixed.

Paper organization. In Section 2, we introduce notation and provide the necessary preliminaries on parity games and universal trees. Section 3 contains our strategy iteration framework based on universal trees. In Section 4, we give a label-correcting method for computing the least fixed point of 1-player games. The label-setting method, on the other hand, is given in the full version. Missing proofs can also be found in the full version.

2 Preliminaries on Parity Games and Universal Trees

For $d \in \mathbb{N}$, let $[d] = \{1, 2, \ldots, d\}$. For a graph G, we use $V(G)$ as its vertex set, and $E(G)$ as its edge set. A parity game instance is given by (G, π), where $G = (V, E)$ is a sinkless directed graph with $V = V_0 \sqcup V_1$, and $\pi : V \to [d]$ is a priority function. Without loss of generality, we may assume that d is even. In this paper, we are only concerned with positional strategies. A *strategy* for Odd is a function $\tau : V_1 \to V$ such that $v\tau(v) \in E$ for all $v \in V_1$. Its *strategy subgraph* is $G_\tau = (V, E_\tau)$, where $E_\tau := \{vw \in E : v \in V_0\} \cup \{v\tau(v) : v \in V_1\}$. A strategy for Even and its strategy subgraph are defined analogously. We always denote a strategy for Even as σ, and a strategy for Odd as τ. If we fix a strategy τ for Odd, the resulting instance (G_τ, π) is a *1-player game* for Even.

For the sake of brevity, we overload the priority function π as follows. Given a subgraph $H \subseteq G$, let $\pi(H)$ be the highest priority in H. The subgraph H is said to be *even* if $\pi(H)$ is even, and *odd* otherwise. For a fixed π, we denote by $\Pi(H)$ the set of nodes with the highest priority in H. If $v \in \Pi(H)$, we say that v *dominates* H. For $p \in [d]$, H_p refers to the subgraph of H induced by nodes with priority at most p. For a node v, let $\delta^-_H(v)$ and $\delta^+_H(v)$ be the incoming and outgoing arcs of v in H respectively. Similarly, let $N^-_H(v)$ and $N^+_H(v)$ be the in-neighbors and out-neighbors of v in H respectively. When H is clear from context, we will omit it from the subscripts.

The win of a player can be certified by *node labels* from a *universal tree*, as stated in Theorem 3. We give the necessary background for this now.

2.1 Ordered Trees and Universal Trees

An *ordered tree* T is a prefix-closed set of tuples, whose elements are drawn from a linearly ordered set M. The linear order of M lexicographically extends to T. Equivalently, T can be thought of as a rooted tree, whose root we denote by r. Under this interpretation, elements in M correspond to the branching directions at each vertex of T (see Figures 2 and 3 for examples). Every tuple then corresponds to a vertex $v \in V(T)$. This is because the tuple can be read by traversing the unique r-v path in T. Observe that v is an h-tuple if and only if v is at depth h in T. In particular, r is the empty tuple.

In this paper, we always use the terms "vertex" and "edge" when referring to an ordered tree T. The terms "node" and "arc" are reserved for the game graph G.

Given an ordered tree T of height h, let $L(T)$ be the set of leaves in T. For convenience, we assume that every leaf in T is at depth h throughout. The tuple representing a leaf $\xi \in L(T)$ is denoted as $\xi = (\xi_{2h-1}, \xi_{2h-3}, \ldots, \xi_1)$, where $\xi_i \in M$ for all i. We refer to ξ_{2h-1} as the *first* component of ξ, even though it has index $2h - 1$. For a fixed $p \in [2h]$, the *p-truncation* of ξ is $\xi|_p := \begin{cases} (\xi_{2h-1}, \xi_{2h-3}, \ldots, \xi_{p+1}), & \text{if } p \text{ is even} \\ (\xi_{2h-1}, \xi_{2h-3}, \ldots, \xi_p), & \text{if } p \text{ is odd.} \end{cases}$
In other words, the p-truncation of a tuple is obtained by deleting the components with index less than p. Note that a truncated tuple is an ancestor of the untruncated tuple in T.

▶ **Definition 1.** *Given ordered trees T and T', we say that T* embeds into *T' (denoted $T \sqsubseteq T'$) if there exists an* injective *and* order-preserving *homomorphism from T to T' such that leaves in T are mapped to leaves in T'. Formally, this is an injective function $f : V(T) \to V(T')$ which satisfies the following properties:*
1. *For all $u, v \in V(T)$, $uv \in E(T)$ implies $f(u)f(v) \in E(T')$;*
2. *For all $u, v \in V(T)$, $u \leq v$ implies $f(u) \leq f(v)$.*
3. *$f(u) \in L(T')$ for all $u \in L(T)$.*

We write $T \equiv T'$ if $T \sqsubseteq T'$ and $T' \sqsubseteq T$. Also, $T \sqsubset T'$ if $T \sqsubseteq T'$ and $T \not\equiv T'$.

In the definition above, since f is order-preserving, the children of every vertex in T are mapped to the children of its image injectively such that their order is preserved. As an example, the tree in Figure 3 embeds into the tree in Figure 2. It is easy to verify that $\sqsubseteq$ is a partial order on the set of all ordered trees.

▶ **Definition 2.** *An* (ℓ, h)-universal tree *is an ordered tree T' of height h such that $T \sqsubseteq T'$ for every ordered tree T of height h and with at most ℓ leaves, all at depth exactly h.*

The simplest example of an (ℓ, h)-universal tree is the perfect ℓ-ary tree of height h, which we call a *perfect universal tree*. The linearly ordered set M for this tree can be chosen as $\{0, 1, \dots, \ell - 1\}$ (see Figure 2 for an example). It has ℓ^h leaves, which grows exponentially with h. Jurdziński and Lazić [16] constructed an (ℓ, h)-universal tree with at most $\ell^{\log h + O(1)}$ leaves, which we call a *succinct universal tree*. In this tree, every leaf ξ corresponds to an h-tuple of binary strings with at most $\lfloor \log(\ell) \rfloor$ bits in total[1]. We use $|\xi|$ and $|\xi_i|$ to denote the total number of bits in ξ and ξ_i respectively. The linearly ordered set M for this tree consists of finite binary strings, where $\varepsilon \in M$ is the empty string (see Figure 3 for an example). For any pair of binary strings $s, s' \in M$ and a bit b, the linear order on M is defined as $0s < \varepsilon < 1s'$ and $bs < bs' \iff s < s'$.

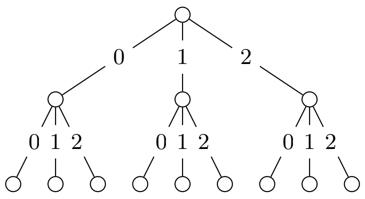

Figure 2 The perfect (3,2)-universal tree.

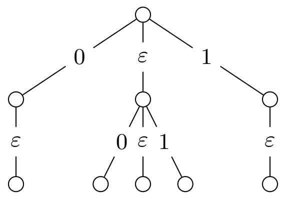

Figure 3 The succinct (3,2)-universal tree.

2.2 Node Labelings from Universal Trees

Let (G, π) be a parity game instance and T be an ordered tree of height $d/2$. We augment the set of leaves with an extra *top* element $\top$, denoted $\bar{L}(T) := L(T) \cup \{\top\}$, such that $\top > v$ for all $v \in V(T)$. We also set $\top|_p := \top$ for all $p \in [d]$. A function $\mu : V \to \bar{L}(T)$ which maps the nodes in G to $\bar{L}(T)$ is called a *node labeling*. For a subgraph H of G, we say that μ is *feasible in* H if there exists a strategy $\sigma : V_0 \to V$ for Even with $v\sigma(v) \in E(H)$ whenever $\delta^+_H(v) \neq \emptyset$, such that the following condition holds for every arc vw in $H \cap G_\sigma$:

- If $\pi(v)$ is even, then $\mu(v)|_{\pi(v)} \geq \mu(w)|_{\pi(v)}$.
- If $\pi(v)$ is odd, then $\mu(v)|_{\pi(v)} > \mu(w)|_{\pi(v)}$ or $\mu(v) = \mu(w) = \top$.

An arc vw which does not satisfy the condition above is called *violated* (with respect to μ). On the other hand, if $\mu(v)$ is the smallest element in $\bar{L}(T)$ such that vw is non-violated, then vw is said to be *tight*. Any arc which is neither tight nor violated is called *loose*. We say that a subgraph is tight if it consists of tight arcs.

In the literature, a node labeling which is feasible in G is also called a *progress measure*. The node labeling given by $\mu(v) = \top$ for all $v \in V$ is trivially feasible in G. However, we are primarily interested in progress measures with minimal top support, i.e. such that the set of nodes having label $\top$ is inclusion-wise minimal.

[1] A slightly looser bound of $\lceil \log \ell \rceil$ was derived in [16, Lemma 1]. It can be strengthened to $\lfloor \log \ell \rfloor$ with virtually no change in the proof.

▶ **Theorem 3** ([15, Corollaries 7–8]). *Given an $(n, d/2)$-universal tree T, let $\mu^* : V \to \bar{L}(T)$ be a node labeling which is feasible in G and has minimal top support. Then, Even wins from $v \in V$ if and only if $\mu^*(v) \neq \top$.*

The above theorem formalizes the following intuition: nodes with smaller labels are more advantageous for Even to play on. Note that if μ is a minimal node labeling which is feasible in G, i.e. μ' is infeasible in G for all $\mu' < \mu$, then there exists a strategy σ for Even such that $v\sigma(v)$ is tight for all $v \in V_0$. The next observation is well-known (see, e.g., [16, Lemma 2]) and follows directly from the definition of feasibility.

▶ **Lemma 4** (Cycle Lemma). *Let μ be a node labeling and C be a cycle such that $\mu(v) \neq \top$ for all $v \in V(C)$. If μ is feasible in C, then C is even. If C is also tight, then $\mu(v) = \mu(w)$ for all $v, w \in \Pi(C)$.*

We assume to have access to the following algorithmic primitive, whose running time we denote by $\gamma(T)$. Its implementation depends on the ordered tree T. For instance, $\gamma(T) = O(d)$ if T is a perfect $(n, d/2)$-universal tree. If T is a succinct $(n, d/2)$-universal tree, Jurdziński and Lazić [16, Theorem 7] showed that $\gamma(T) = O(\log n \log d)$.

Tighten(μ, vw)

Given a node labeling $\mu : V \to \bar{L}(T)$ and an arc $vw \in E$, return the unique element $\xi \in \bar{L}(T)$ such that vw is tight after setting $\mu(v)$ to ξ.

Given a node labeling $\mu : V \to \bar{L}(T)$ and an arc $vw \in E$, let $\mathrm{lift}(\mu, vw)$ be the smallest element $\xi \in \bar{L}(T)$ such that $\xi \geq \mu(v)$ and vw is not violated after setting $\mu(v)$ to ξ. Observe that if vw is violated, $\mathrm{lift}(\mu, vw)$ is given by Tighten(μ, vw). Otherwise, it is equal to $\mu(v)$. Hence, it can be computed in $\gamma(T)$ time.

Let $\mathcal{L}$ be the finite lattice of node labelings mapping V to $\bar{L}(T)$. For a sinkless subgraph $H \subseteq G$, consider the following operators. For every node $v \in V_0$, define $\mathrm{Lift}_v : \mathcal{L} \times V \to \bar{L}(T)$ as $\mathrm{Lift}_v(\mu, u) := \min_{vw \in E(H)} \mathrm{lift}(\mu, vw)$ if $u = v$, and $\mu(u)$ otherwise. For every arc $vw \in E(H)$ where $v \in V_1$, define $\mathrm{Lift}_{vw} : \mathcal{L} \times V \to \bar{L}(T)$ as $\mathrm{Lift}_{vw}(\mu, u) := \mathrm{lift}(\mu, vw)$ if $u = v$, and $\mu(u)$ otherwise. We denote $\mathcal{H}^\uparrow = \{\mathrm{Lift}_v : v \in V_0\} \cup \{\mathrm{Lift}_{vw} : v \in V_1\}$ as the operators in H. Since they are inflationary and monotone, for any $\mu \in \mathcal{L}$, the *least* simultaneous fixed point of $\mathcal{H}^\uparrow$ that is *pointwise at least* μ exists. It is denoted as $\mu^{\mathcal{H}^\uparrow}$. Note that a node labeling is a simultaneous fixed point of $\mathcal{H}^\uparrow$ if and only if it is feasible in H. The *progress measure algorithm* [15, 16] is an iterative application of the operators in $\mathcal{G}^\uparrow$ to μ to obtain $\mu^{\mathcal{G}^\uparrow}$.

3 Strategy Iteration with Tree Labels

In this section, we present a strategy iteration algorithm (Algorithm 1) whose pivots are guided by a universal tree. It takes as input an instance (G, π), a universal tree T, and an initial strategy τ_1 for Odd. Throughout, it maintains a node labeling $\mu : V \to \bar{L}(T)$, initialized as the least simultaneous fixed point of $\mathcal{G}^\uparrow_{\tau_1}$. At the start of every iteration, the algorithm maintains a strategy τ for Odd, and a node labeling $\mu : V \to \bar{L}(T)$ which is feasible in G_τ. Furthermore, there are no loose arcs in G_τ with respect to μ. So, every arc in G_τ is either tight (usable by Even in her counterstrategy σ) or violated (not used by Even). Note that our initial node labeling satisfies these conditions with respect to τ_1.

For $v \in V_1$, we call a violated arc $vw \in E$ with respect to μ *admissible* (as it admits Odd to perform an improvement). If there are no admissible arcs in G, then the algorithm terminates. In this case, μ is feasible in G. Otherwise, Odd pivots to a new strategy τ' by

Algorithm 1 Strategy iteration with tree labels: (G, π) instance, T universal tree, τ_1 initial strategy for Odd.

```
1: procedure StrategyIteration($(G, \pi), T, \tau_1$)
2:     $\mu(v) \leftarrow \min L(T)\ \forall v \in V$
3:     $\tau \leftarrow \tau_1$, $\mu \leftarrow \mu^{\mathcal{G}_\tau^\uparrow}$
4:     while $\exists$ an admissible arc in $G$ with respect to $\mu$ do
5:         Pivot to a strategy $\tau'$ by selecting admissible arc(s)      ▷ requires a pivot rule
6:         $\tau \leftarrow \tau'$, $\mu \leftarrow \mu^{\mathcal{G}_\tau^\uparrow}$
7:     return $\tau$, $\mu$
```

switching to admissible arc(s). The choice of which admissible arc(s) to pick is governed by a *pivot rule*. Then, μ is updated to $\mu^{\mathcal{G}_{\tau'}^\uparrow}$. Due to the minimality of $\mu^{\mathcal{G}_{\tau'}^\uparrow}$, there are no loose arcs in $G_{\tau'}$ with respect to $\mu^{\mathcal{G}_{\tau'}^\uparrow}$, so this invariant continues to hold in the next iteration.

The correctness of Algorithm 1 follows from the Knaster–Tarski Theorem. We remark that a strategy τ may occur more than once during the course of the algorithm, as mentioned in the description of strategy iteration in Section 1. This is because the fixed points of $\mathcal{G}_\tau^\uparrow$ are not necessarily unique. See Figure 4 for an example run with a succinct universal tree.

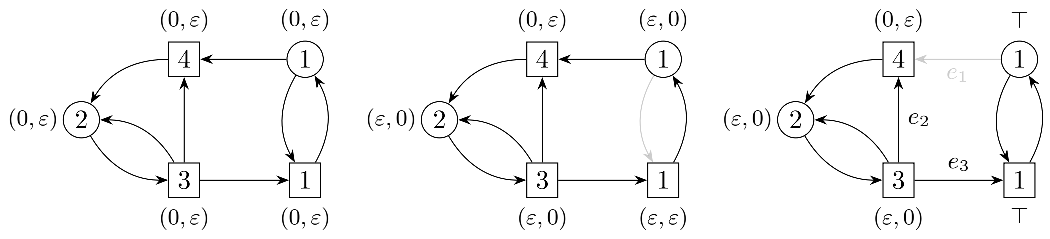

Figure 4 An example run of Algorithm 1 with the succinct (3,2)-universal tree. The left figure depicts a game instance (nodes in V_0 and V_1 are drawn as squares and circles respectively). The next two figures show Odd's strategy and the node labeling at the start of Iteration 1 and 2. Arcs not selected by Odd are greyed out. In the right figure, e_1 is loose, e_2 is tight, and e_3 is violated.

4 Computing the Least Fixed Point of 1-Player Games

Let (G_τ, π) be a 1-player game for Even, and let $\mu \in \mathcal{L}$ be a node labeling such that there are no loose arcs in G_τ. In this section, we develop an efficient method for computing $\mu^{\mathcal{G}_\tau^\uparrow}$. We know that applying the operators in $\mathcal{G}_\tau^\uparrow$ to μ is not polynomial in general. So, we will approach $\mu^{\mathcal{G}_\tau^\uparrow}$ from above instead.

Given a node labeling $\nu : V \to \bar{L}(T)$ and an arc $vw \in E$, let $\mathrm{drop}(\nu, vw)$ be the largest element $\xi \in \bar{L}(T)$ such that $\xi \leq \nu(v)$ and vw is not loose after setting $\nu(v)$ to ξ. Observe that if vw is loose, then $\mathrm{drop}(\nu, vw)$ is given by $\textsc{Tighten}(\nu, vw)$. Otherwise, it is equal to $\nu(v)$. Hence, it can be computed in $\gamma(T)$ time.

We are ready to define the deflationary counterpart of Lift_{vw}. For every arc $vw \in E_\tau$, define the operator $\mathrm{Drop}_{vw} : \mathcal{L} \times V \to \bar{L}(T)$ as $\mathrm{Drop}_{vw}(\nu, u) := \mathrm{drop}(\nu, vw)$ if $u = v$, and $\nu(v)$ otherwise. For a subgraph $H \subseteq G_\tau$, we denote $\mathcal{H}^\downarrow = \{\mathrm{Drop}_e : e \in E(H)\}$ as the operators in H. Since they are deflationary and monotone, for any $\nu \in \mathcal{L}$, the *greatest* simultaneous fixed point of $\mathcal{H}^\downarrow$ that is *pointwise at most* ν exists. It is denoted as $\nu^{\mathcal{H}^\downarrow}$. Note that a node labeling is a simultaneous fixed point of $\mathcal{H}^\downarrow$ if and only if there are no loose arcs in H with respect to it.

Our techniques are inspired by the methods of *label-correcting* and *label-setting* for the shortest path problem. In the shortest path problem, we have a designated target node t whose label is initialized to 0. For us, the role of t is replaced by a (potentially empty) set of even cycles in G_τ, which we do not know a priori. So, we define a set of candidates nodes called *base nodes*, whose labels need to be initialized properly.

▶ **Definition 5.** *Given a 1-player game (G_τ, π) for Even, we call $v \in V$ a* base node *if $v \in \Pi(C)$ for some even cycle C in G_τ. Denote $B(G_\tau)$ as the set of base nodes in G_τ.*

The base nodes can be found by recursively decomposing G_τ into strongly connected components. Initially, for each strongly connected component K of G_τ, we delete $\Pi(K)$. If $\pi(K)$ is even and $|V(K)| > 1$, then $\Pi(K)$ are base nodes and we collect them. Otherwise, we ignore them. Then, we are left with a smaller subgraph of G, so we repeat the process. Using Tarjan's strongly connected components algorithm, this procedure takes $O(dm)$ time.

In the next subsection, we develop a label-correcting method for computing $\mu^{\mathcal{G}_\tau^\uparrow}$, and apply it to the quasi-polynomial universal trees constructed in the literature [16, 7]. The label-setting method, which is faster but only applicable to perfect universal trees, is deferred to the full version.

4.1 Label-Correcting Method

The Bellman–Ford algorithm for the shortest path problem is a well-known implementation of the generic label-correcting method [1]. We start by giving its analogue for ordered trees. Algorithm 2 takes as input a 1-player game (G_τ, π) for Even and a node labeling $\nu : V \to \bar{L}(T)$ from some ordered tree T. Like its classical version for shortest paths, the algorithm runs for $n-1$ iterations. In each iteration, it replaces the tail label of every arc $e \in E_\tau$ by $\mathrm{drop}(\nu, e)$. Clearly, the running time is $O(mn\gamma(T))$. Moreover, if ν' is the returned node labeling, then $\nu' \geq \nu^{\mathcal{G}_\tau^\downarrow}$.

Algorithm 2 Bellman–Ford: (G_τ, π) 1-player game for Even, $\nu : V \to \bar{L}(T)$ node labeling from an ordered tree T.

```
1: procedure BellmanFord($(G, \pi), \nu$)
2:     for $i = 1$ to $n - 1$ do
3:         for all $vw \in E$ do                              ▷ In any order
4:             $\nu(v) \leftarrow \mathrm{drop}(\nu, vw)$
5:     return $\nu$
```

Recall that we have a node labeling $\mu \in \mathcal{L}$ such that G_τ does not have loose arcs, and our goal is to compute $\mu^{\mathcal{G}_\tau^\uparrow}$. We first state a sufficient condition on the input node labeling ν such that Algorithm 2 returns $\mu^{\mathcal{G}_\tau^\uparrow}$. In the shortest path problem, we set $\nu(t) = 0$ at the target node t, and $\nu(v) = \infty$ for all $v \in V \setminus \{t\}$. When working with node labels given by an ordered tree, one has to ensure that the algorithm does not terminate with a fixed point larger than $\mu^{\mathcal{G}_\tau^\uparrow}$, motivating the following definition.

▶ **Definition 6.** *Given a node labeling $\mu \in \mathcal{L}$, the* threshold label *of a base node v is*

$$\hat{\mu}(v) := \min_{\tilde{\mu} \in \mathcal{L}} \{\tilde{\mu}(v) : \tilde{\mu}(v) \geq \mu(v) \text{ and } \tilde{\mu} \text{ is feasible in a cycle dominated by } v\} \quad .$$

The next lemma follows directly from the pointwise minimality of $\mu^{\mathcal{G}_\tau^\uparrow}(v)$.

▶ **Lemma 7.** *Let $\mu \in \mathcal{L}$ be a node labeling such that G_τ does not have loose arcs. For every base node $v \in B(G_\tau)$, we have $\widehat{\mu}(v) \geq \mu^{\mathcal{G}_\tau^\uparrow}(v)$.*

The next theorem shows that if we initialize the base nodes with their corresponding threshold labels, then Algorithm 2 returns $\mu^{\mathcal{G}_\tau^\uparrow}$. Even more, it suffices to have an initial node labeling $\nu \in \mathcal{L}$ such that $\mu^{\mathcal{G}_\tau^\uparrow}(v) \leq \nu(v) \leq \widehat{\mu}(v)$ for all $v \in B(G_\tau)$. For the other nodes $v \notin B(G_\tau)$, we can simply set $\nu(v) \leftarrow \top$.

▶ **Theorem 8.** *Let $\mu \in \mathcal{L}$ be a node labeling such that G_τ does not have loose arcs. Given $\nu \in \mathcal{L}$ where $\nu \geq \mu^{\mathcal{G}_\tau^\uparrow}$ and $\nu(v) \leq \widehat{\mu}(v)$ for all $v \in B(G_\tau)$, Algorithm 2 returns $\mu^{\mathcal{G}_\tau^\uparrow}$.*

Our strategy for computing such a ν is to find the cycles in Definition 6. In particular, for every base node $v \in B(G_\tau)$, we aim to find a cycle C dominated by v such that $\widehat{\mu}(v)$ can be extended to a node labeling that is feasible in C. To accomplish this goal, we first introduce the notion of *width* in Section 4.1.1, which allows us to evaluate how "good" a cycle is. It is defined using chains in the poset of subtrees of T, where the partial order is given by $\sqsubseteq$. Then, in Section 4.1.2, we show how to obtain the desired cycles by computing minimum bottleneck cycles on a suitably defined auxiliary digraph.

4.1.1 Width from a Chain of Subtrees in T

Two ordered trees T' and T'' are said to be *distinct* if $T' \not\equiv T''$ (not isomorphic in the sense of Definition 1). Let h be the height of our universal tree T. For $0 \leq j \leq h$, denote $\mathcal{T}_j$ as the set of distinct (whole) subtrees rooted at the vertices of depth $h - j$ in T. For example, $\mathcal{T}_h = \{T\}$, while $\mathcal{T}_0$ contains the trivial tree with a single vertex. Since we assumed that all the leaves in T are at the same depth, every tree in $\mathcal{T}_j$ has height j. We denote $\mathcal{T} = \cup_{j=0}^{h} \mathcal{T}_j$ as the union of all these subtrees. The sets $\mathcal{T}$ and $\mathcal{T}_j$ form posets with respect to the partial order $\sqsubseteq$. The next definition is the usual chain cover of a poset, where we additionally require that the chains form an indexed tuple instead of a set.

▶ **Definition 9.** *For $0 \leq j \leq h$, let $\mathcal{C}_j = (\mathcal{C}_j^0, \mathcal{C}_j^1, \ldots, \mathcal{C}_j^\ell)$ be a tuple of chains in the poset $(\mathcal{T}_j, \sqsubseteq)$. We call $\mathcal{C}_j$ a* cover of $\mathcal{T}_j$ *if $\cup_{k=0}^{\ell} \mathcal{C}_j^k = \mathcal{T}_j$. A* cover of $\mathcal{T}$ *is a tuple $\mathcal{C} = (\mathcal{C}_0, \mathcal{C}_1, \ldots, \mathcal{C}_h)$ where $\mathcal{C}_j$ is a cover of $\mathcal{T}_j$ for all $0 \leq j \leq h$. We refer to $\mathcal{C}_j$ as the* jth-subcover *of $\mathcal{C}$. Given a cover $\mathcal{C}$ of $\mathcal{T}$, we denote the trees in the chain $\mathcal{C}_j^k$ as $T_{0,j}^k \sqsubset T_{1,j}^k \sqsubset \cdots \sqsubset T_{|\mathcal{C}_j^k|-1,j}^k$.*

An example of an ordered tree with its cover is given in Figure 5. We are ready to introduce the key concept of this subsection.

▶ **Definition 10.** *Let $\mathcal{C}$ be a cover of $\mathcal{T}$. Let H be a subgraph of G_τ and $j = \lceil \pi(H)/2 \rceil$. For a fixed chain $\mathcal{C}_j^k$ in $\mathcal{C}_j$, the* kth-width *of H, denoted $\alpha_{\mathcal{C}}^k(H)$, is the smallest integer $i \geq 0$ such that there exists a node labeling $\nu : V(H) \to L(T_{i,j}^k)$ which is feasible in H. If i does not exist, then $\alpha_{\mathcal{C}}^k(H) = \infty$.*

Note that $T_{i,j}^k$ is the $(i+1)$-th smallest tree in the chain $\mathcal{C}_j^k$. We are mainly interested in the case when H is a cycle, and write $\alpha^k(H)$ whenever the cover $\mathcal{C}$ is clear from context. Observe that the definition above requires $\nu(v) \neq \top$ for all $v \in V(H)$. Hence, an odd cycle has infinite kth-width by the Cycle Lemma. As $(\mathcal{C}_j^k, \sqsubseteq)$ is a chain, for all finite $i \geq \alpha^k(H)$, there exists a node labeling $\nu : V(H) \to L(T_{i,j}^k)$ which is feasible in H. The next lemma illustrates the connection between the kth-width of an even cycle and its path decomposition.

▶ **Lemma 11.** *Let $\mathcal{C}$ be a cover of $\mathcal{T}$. For an even cycle C, let $\Pi(C) = \{v_1, v_2, \ldots, v_\ell\}$ and $j = \pi(C)/2$. Decompose C into arc-disjoint paths $P_1, P_2, \ldots, P_\ell$ such that each P_i ends at v_i. Then, $\alpha^k(C) = \max_{i \in [\ell]} \alpha^k(P_i)$ for all $0 \leq k < |\mathcal{C}_j|$.*

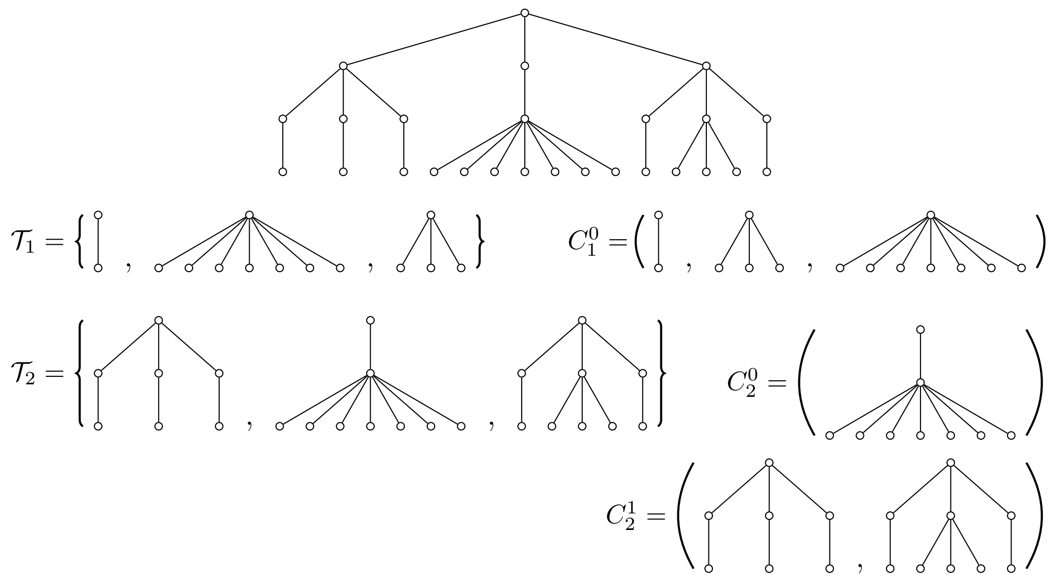

Figure 5 An ordered tree T of height 3, and a cover $\mathcal{C}_j$ of $\mathcal{T}_j$ for all $0 < j < 3$. Recall that $\mathcal{T}_j$ is the set of distinct subtrees of T rooted at depth $3 - j$, while $\mathcal{C}_j^k$ is the kth chain in $\mathcal{C}_j$.

For a base node $v \in B(G_\tau)$, let us consider the cycles in G_τ which are dominated by v. Among them, we are interested in finding one with the smallest kth-width. So, we extend the notion of kth-width to base nodes in the following way.

▶ **Definition 12.** *Let $\mathcal{C}$ be a cover of $\mathcal{T}$. Let $v \in B(G_\tau)$ be a base node and $j = \pi(v)/2$. For $0 \leq k < |\mathcal{C}_j|$, define the* kth-width *of v as $\alpha^k_{\mathcal{C}}(v) := \min\left\{\alpha^k_{\mathcal{C}}(C) : C \text{ is a cycle dominated by } v\right\}$.*

Again, we write $\alpha^k(v)$ whenever $\mathcal{C}$ is clear from context. Observe that $T^k_{\alpha^k(v),\pi(v)/2}$ is the smallest tree in the chain $\mathcal{C}^k_{\pi(v)/2}$ which can encode a node labeling that is feasible on some cycle dominated by v.

Given a leaf $\xi \in L(T)$ and integers $i, j, k \in \mathbb{Z}_{\geq 0}$, the following subroutine locates a subtree of T that is a member of the chain $\mathcal{C}^k_j$ and into which $T^k_{i,j}$ is embeddable.

Raise(ξ, i, j, k)

Given a leaf $\xi \in L(T)$ and integers $i, j, k \in \mathbb{Z}_{\geq 0}$, return the smallest leaf $\xi' \in L(T)$ such that (1) $\xi' \geq \xi$; and (2) ξ' is the smallest leaf in the subtree $T^k_{i',j}$ for some $i' \geq i$. If ξ' does not exist, then return $\top$.

Now, fix a base node $v \in B(G_\tau)$. For any $0 \leq k < |\mathcal{C}_{\pi(v)/2}|$, notice that the label returned by Raise$(\mu(v), \alpha^k(v), \frac{\pi(v)}{2}, k)$ is an upper bound on the threshold label $\widehat{\mu}(v)$. The smallest such label over all k is precisely the threshold label $\widehat{\mu}(v)$, as the next lemma shows.

▶ **Lemma 13.** *Fix $v \in B(G_\tau)$. Let ξ^k be the label returned by* Raise$(\mu(v), \alpha^k(v), \frac{\pi(v)}{2}, k)$ *for all $0 \leq k < |\mathcal{C}_{\pi(v)/2}|$. If there are no loose arcs in G_τ with respect to μ, then $\widehat{\mu}(v) = \min_k \xi^k$.*

The necessary number of chains in the subcover $\mathcal{C}_{\pi(v)/2}$ can be large if T is an arbitrary ordered tree. Fortunately, the universal trees constructed in the literature admit covers with small subcovers. In the full version, we prove that a succinct (n, h)-universal tree has a cover with only 1 chain per subcover, whereas a succinct Strahler (n, h)-universal tree (introduced by Daviaud et al. [7]) has a cover with at most $\log n$ chains per subcover.

Let $\rho(T, \mathcal{C})$ denote the running time of RAISE. We provide efficient implementations of RAISE for succinct universal trees and succinct Strahler universal trees in the full version. They have the same running time as TIGHTEN, i.e., $\rho(T, \mathcal{C}) = O(\log n \log h)$.

4.1.2 Estimating the Width of Base Nodes

In light of the previous discussion, we can now focus on computing the kth-width of a base node $w \in B(G_\tau)$. Fix a $0 \leq k < |\mathcal{C}_{\pi(w)/2}|$. Since we ultimately need a label that lies between $\mu^{\mathcal{G}_\tau^\uparrow}(w)$ and $\widehat{\mu}(w)$ in order to initialize Algorithm 2, it suffices to compute a "good" under-estimation of $\alpha^k(w)$. In this subsection, we reduce this problem to computing a minimum bottleneck cycle in an auxiliary digraph D with nonnegative arc costs $c^k \geq 0$.

For a base node $w \in B(G_\tau)$, let K_w denote the strongly connected component containing w in $(G_\tau)_{\pi(w)}$, the subgraph of G_τ induced by nodes with priority at most $\pi(w)$. Let $K'_w \subseteq K_w$ be the subgraph obtained by deleting the incoming arcs $\delta^-(v)$ for all $v \in \Pi(K_w) \setminus \{w\}$. Then, we define J_w as the subgraph of K'_w induced by those nodes which can reach w in K'_w. These are the nodes which can reach w in K_w without encountering an intermediate node of priority $\pi(w)$.

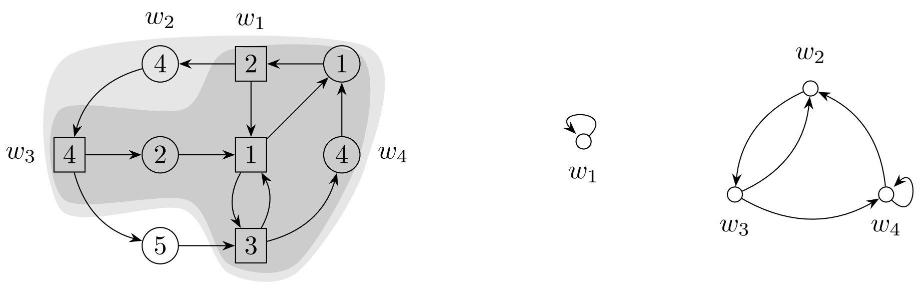

Figure 6 An example of a 1-player game (G_τ, π) for Even is given on the left, with its auxiliary digraph D on the right. Nodes in V_0 and V_1 are drawn as squares and circles respectively. Base nodes are labeled as w_1, w_2, w_3, w_4. The light gray region is K_{w_4}, while the dark gray region is J_{w_4}.

The auxiliary digraph D is constructed as follows. Its node set is $B(G_\tau)$. For every ordered pair (v, w) of base nodes where $\pi(v) = \pi(w)$, add the arc vw if v has an outgoing arc in J_w. Note that if $(v, w) \in D$, then v can reach w by only seeing smaller priorities on the intermediate nodes. As ordered pairs of the form (v, v) are also considered, D may contain self-loops. Observe that D is a disjoint union of strongly connected components, each of which consists of base nodes with the same priority (see Figure 6 for an example). For $w \in B(G_\tau)$, we denote D_w as the component in D which contains w.

To finish the description of D, it is left to assign the arc costs c^k. Note that the graph structure of D is independent of k. We give a range in which the cost of each arc should lie. Fix a base node $w \in B(G_\tau)$ and let $j = \pi(w)/2$. Recall that $\mathcal{J}_w^\downarrow = \{\text{Drop}_e : e \in E(J_w)\}$ is the set of Drop operators in the subgraph $J_w \subseteq G_\tau$. For each $0 \leq i < |\mathcal{C}_j^k|$, let $\lambda_{i,w}^k : V(J_w) \to \bar{L}(T_{i,j}^k)$ be the greatest simultaneous fixed point of $\mathcal{J}_w$ subject to $\lambda_{i,w}^k(w) = \min L(T_{i,j}^k)$. Then, for each arc $vw \in E(D)$, the lower and upper bounds of $c^k(vw)$ are given by

$$\begin{aligned} \underline{c}^k(vw) &:= \min \left\{ i : \lambda_{i,w}^k(u) \neq \top \text{ for some } u \in N_{J_w}^+(v) \right\} \\ \bar{c}^k(vw) &:= \min \left\{ \alpha^k(P) : P \text{ is a } u\text{-}w \text{ path in } J_w \text{ where } u \in N_{J_w}^+(v) \right\} \end{aligned} \tag{1}$$

respectively. The lower bound $\underline{c}^k(vw)$ is the smallest integer $i \geq 0$ such that the greatest simultaneous fixed point $\lambda_{i,w}^k$ assigns a non-top label to an out-neighbor of v in J_w. On the other hand, the upper bound $\bar{c}^k(vw)$ is the minimum kth-width of a path from an out-neighbor of v to w in J_w. Note that these quantities could be equal to $+\infty$.

▶ **Lemma 14.** *For every arc $vw \in E(D)$, we have $\underline{c}^k(vw) \leq \overline{c}^k(vw)$.*

For a cycle C in D, its (bottleneck) c^k-*cost* is defined as $c^k(C) := \max_{e \in E(C)} c^k(e)$. Note that self-loops in D are considered cycles. The next theorem allows us to obtain the desired initial node labeling ν for Algorithm 2 by computing minimum bottleneck cycles in D.

▶ **Theorem 15.** *Let $\mathcal{C}$ be a cover of $\mathcal{T}$. Let $\mu \in \mathcal{L}$ be a node labeling such that G_τ does not have loose arcs. For a base node w, let c^k be arc costs in D_w such that $\underline{c}^k \leq c^k \leq \overline{c}^k$ for all $0 \leq k < |\mathcal{C}_{\pi(w)/2}|$. For each k, let i^k be the minimum c^k-cost of a cycle containing w in D_w, and ξ^k be the label returned by* Raise$(\mu(w), i^k, \frac{\pi(w)}{2}, k)$*. Then, $\mu^{\mathcal{G}_\tau^\uparrow}(w) \leq \min_k \xi^k \leq \widehat{\mu}(w)$.*

4.1.3 The Label-Correcting Algorithm

The overall algorithm for computing $\mu^{\mathcal{G}_\tau^\uparrow}$ is given in Algorithm 3. The main idea is to initialize the labels on base nodes via the recipe given in Theorem 15, before running Algorithm 2. The labels on $V \setminus B(G_\tau)$ are initialized to $\top$. The auxiliary graph D serves as a condensed representation of the "best" paths between base nodes. The arc costs are chosen such that minimum bottleneck cycles in D give a good estimate on the width of base nodes.

Algorithm 3 Label-Correcting: (G_τ, π) 1-player game for Even, $\mathcal{C}$ cover of $\mathcal{T}$ for some universal tree T, $\mu : V \to \bar{L}(T)$ node labeling such that G_τ does not contain loose arcs.

```
1:  procedure LabelCorrecting($(G_\tau, \pi), \mathcal{C}, \mu$)
2:      $\nu(v) \leftarrow \top$ for all $v \in V$
3:      Construct auxiliary digraph $D$
4:      for all components $H$ in $D$ do
5:          for $k = 0$ to $|\mathcal{C}_{\pi(H)/2}| - 1$ do
6:              Assign arc costs $c^k$ to $H$ where $\underline{c}^k \leq c^k \leq \overline{c}^k$
7:              for all $v \in V(H)$ do
8:                  $i^k \leftarrow$ minimum $c^k$-cost of a cycle containing $v$ in $H$
9:                  $\nu(v) \leftarrow \min(\nu(v), \text{Raise}(\mu(v), i^k, \frac{\pi(v)}{2}, k))$
10:     $\nu \leftarrow$ BellmanFord($(G_\tau, \pi), \nu$)
11:     return $\nu$
```

In Algorithm 3, the arc costs c^k can be obtained using Algorithm 2. For each base node $w \in B(G_\tau)$, in order to compute $c^k(e)$ for all incoming arcs $e \in \delta_D^-(w)$, we run Algorithm 2 on the subgraph J_w for $|\mathcal{C}^k_{\pi(w)/2}|$ times. If the chain $\mathcal{C}^k_{\pi(w)/2}$ is too long, then this can be sped up using binary search. For specific families of trees such as succinct universal trees, one can compute c^k even faster. More details are given in the full version. Overall, this yields the following generic running time bound.

▶ **Theorem 16.** *In $O(mn^2\gamma(T) \cdot \max_{j,k} |\mathcal{C}_j| \min\{|\mathcal{C}_j^k|, n \log |\mathcal{C}_j^k|\} + n\rho(T, \mathcal{C}) \cdot \max_j |\mathcal{C}_j|)$ time, Algorithm 3 returns $\mu^{\mathcal{G}_\tau^\uparrow}$.*

As mentioned in Section 4.1.1, a succinct $(n, d/2)$-universal tree has a cover with 1 chain per subcover, while a succinct Strahler $(n, d/2)$-universal tree has a cover with at most $\log n$ chains per subcover. So, applying Algorithm 3 to these trees yields the following runtimes.

▶ **Corollary 17.** *For a succinct universal tree, $\mu^{\mathcal{G}_\tau^\uparrow}$ is returned in $O(mn^2 \log n \log d)$.*

▶ **Corollary 18.** *For a succinct Strahler universal tree, $\mu^{\mathcal{G}_\tau^\uparrow}$ is returned in $O(mn^2 \log^3 n \log d)$.*

References

1 Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network flows. Theory, algorithms, and applications.* Englewood Cliffs, NJ: Prentice Hall, 1993.

2 Massimo Benerecetti, Daniele Dell'Erba, and Fabio Mogavero. Solving parity games via priority promotion. *Formal Methods Syst. Des.*, 52(2):193–226, 2018.

3 Massimo Benerecetti, Daniele Dell'Erba, Fabio Mogavero, Sven Schewe, and Dominik Wojtczak. Priority promotion with parysian flair. *arXiv*, abs/2105.01738, 2021. `arXiv:2105.01738`.

4 Henrik Björklund, Sven Sandberg, and Sergei G. Vorobyov. A discrete subexponential algorithm for parity games. In *20th Annual Symposium on Theoretical Aspects of Computer Science, STACS*, volume 2607 of *Lecture Notes in Computer Science*, pages 663–674, 2003.

5 Cristian S. Calude, Sanjay Jain, Bakhadyr Khoussainov, Wei Li, and Frank Stephan. Deciding parity games in quasipolynomial time. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC*, pages 252–263, 2017.

6 Wojciech Czerwinski, Laure Daviaud, Nathanaël Fijalkow, Marcin Jurdzinski, Ranko Lazic, and Pawel Parys. Universal trees grow inside separating automata: Quasi-polynomial lower bounds for parity games. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 2333–2349, 2019.

7 Laure Daviaud, Marcin Jurdzinski, and K. S. Thejaswini. The strahler number of a parity game. In *47th International Colloquium on Automata, Languages, and Programming, ICALP*, volume 168 of *LIPIcs*, pages 123:1–123:19, 2020.

8 E. Allen Emerson and Charanjit S. Jutla. Tree automata, mu-calculus and determinacy. In *32nd Annual Symposium on Foundations of Computer Science, FOCS*, pages 368–377, 1991.

9 E. Allen Emerson, Charanjit S. Jutla, and A. Prasad Sistla. On model-checking for fragments of μ-calculus. In *5th International Conference on Computer-Aided Verification, CAV*, volume 697 of *Lecture Notes in Computer Science*, pages 385–396, 1993.

10 John Fearnley, Sanjay Jain, Bart de Keijzer, Sven Schewe, Frank Stephan, and Dominik Wojtczak. An ordered approach to solving parity games in quasi-polynomial time and quasi-linear space. *Int. J. Softw. Tools Technol. Transf.*, 21(3):325–349, 2019.

11 Oliver Friedmann. An exponential lower bound for the parity game strategy improvement algorithm as we know it. In *Proceedings of the 24th Annual IEEE Symposium on Logic in Computer Science, LICS*, pages 145–156, 2009.

12 Oliver Friedmann. *Exponential Lower Bounds for Solving Infinitary Payoff Games and Linear Programs.* PhD thesis, University of Munich, 2011. URL: `http://files.oliverfriedmann.de/theses/phd.pdf`.

13 A. J. Hoffman and R. M. Karp. On nonterminating stochastic games. *Manage. Sci.*, 12(5):359–370, 1966.

14 Marcin Jurdzinski. Deciding the winner in parity games is in UP $\cap$ co-UP. *Inf. Process. Lett.*, 68(3):119–124, 1998.

15 Marcin Jurdzinski. Small progress measures for solving parity games. In *17th Annual Symposium on Theoretical Aspects of Computer Science, STACS*, volume 1770 of *Lecture Notes in Computer Science*, pages 290–301, 2000.

16 Marcin Jurdzinski and Ranko Lazic. Succinct progress measures for solving parity games. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS*, pages 1–9, 2017.

17 Marcin Jurdzinski, Mike Paterson, and Uri Zwick. A deterministic subexponential algorithm for solving parity games. *SIAM J. Comput.*, 38(4):1519–1532, 2008.

18 Orna Kupferman and Moshe Y. Vardi. Weak alternating automata and tree automata emptiness. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pages 224–233, New York, NY, USA, 1998. Association for Computing Machinery. `doi:10.1145/276698.276748`.

19 Karoliina Lehtinen. A modal μ perspective on solving parity games in quasi-polynomial time. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS*, pages 639–648, 2018.

20 Matthias Mnich, Heiko Röglin, and Clemens Rösner. New deterministic algorithms for solving parity games. *Discret. Optim.*, 30:73–95, 2018.
21 Pierre Ohlmann. *Monotonic Graphs for Parity and Mean-Payoff Games*. PhD thesis, University of Paris, 2021. URL: `https://www.irif.fr/~ohlmann/contents/these.pdf`.
22 Pawel Parys. Parity games: Zielonka's algorithm in quasi-polynomial time. In *44th International Symposium on Mathematical Foundations of Computer Science, MFCS*, volume 138 of *LIPIcs*, pages 10:1–10:13, 2019.
23 Anuj Puri. *Theory of hybrid systems and discrete event systems*. PhD thesis, EECS Department, University of California, Berkeley, 1995.
24 Sven Schewe. An optimal strategy improvement algorithm for solving parity and payoff games. In *22nd International Workshop on Computer Science Logic, CSL*, volume 5213 of *Lecture Notes in Computer Science*, pages 369–384, 2008.
25 Sven Schewe. Solving parity games in big steps. *J. Comput. Syst. Sci.*, 84:243–262, 2017.
26 Jens Vöge and Marcin Jurdzinski. A discrete strategy improvement algorithm for solving parity games. In *12th International Conference on Computer-Aided Verification, CAV*, volume 1855 of *Lecture Notes in Computer Science*, pages 202–215, 2000.
27 Wieslaw Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theor. Comput. Sci.*, 200(1-2):135–183, 1998.