Plan for Today

- Basic Games
- Algorithms & Data Structures
- Advanced Games
- Temporal Logic Synthesis
Plan for Today

- Basic Games
- Algorithms & Data Structures
- Advanced Games
  - Parity Games
- Temporal Logic Synthesis
Motivation

Four foundational winning conditions:

**Reachability** reaching a goal (at least once)

**Safety** staying safe (at all times)

**Recurrence** reaching a goal infinitely often

**Persistence** staying safe from some point onwards
Motivation

Four foundational winning conditions:

**Reachability** reaching a goal (at least once)

**Safety** staying safe (at all times)

**Recurrence** reaching a goal infinitely often

**Persistence** staying safe from some point onwards

There are still specifications not expressible with these conditions, e.g., Boolean combinations of basic conditions.
Never visit red vertex and
if infinitely many blue vertices are visited, then infinitely many green vertices are visited.
Never visit red vertex and

if infinitely many blue vertices are visited, then infinitely many green vertices are visited.

\[ \text{Safety}(\text{not red}) \cap (\text{coB\"uchi}(\text{not blue}) \cup \text{B\"uchi}(\text{green})) \]
Player 0 wins if the maximal color visited infinitely often is even.
Parity Games

Player 0 wins if the maximal color visited infinitely often is even.
Parity Games

Recall
\( \text{Inf}(\rho) := \{ v \in V \mid v_n = v \text{ for infinitely many } n \} \): the set of vertices occurring infinitely often in \( \rho \).

Definition
Let \( \mathcal{A} = (V, V_0, V_1, E) \) be an arena and let \( \Omega : V \to \mathbb{N} \) be a coloring of \( \mathcal{A} \)'s vertices. Then, the parity condition \( \text{Parity}(\Omega) \) is defined as

\[
\text{Parity}(\Omega) := \{ \rho \in V^\omega \mid \max \text{Inf}(\Omega(\rho_0)\Omega(\rho_1)\Omega(\rho_2) \cdots) \text{ is even} \}.
\]

We call a game \( \mathcal{G} = (\mathcal{A}, \text{Parity}(\Omega)) \) a parity game.

Intuition: color \( c \) is desirable for Player \( c \mod 2 \).
Never visit red vertex and
if infinitely many blue vertices are visited, then infinitely many
green vertices are visited.

This can be expressed as a parity condition

- **gray** vertices $\mapsto 0$
- **red** vertices $\mapsto 3$
- **blue** vertices $\mapsto 1$
- **green** vertices $\mapsto 2$
Parity games play a central role in automata and logics:

- Normal form for $\omega$-regular languages: deterministic parity automata (next week).
- Model-checking games of the modal $\mu$-calculus.
- Emptiness of parity tree automata is equivalent to parity games.
- Semantics of alternating automata on infinite objects.
- Intriguing complexity-theoretic status (later today).
Properties

Lemma
Every parity condition is a Boolean combination of Büchi conditions.
Properties

Lemma
Every parity condition is a Boolean combination of Büchi conditions.

Lemma
1. Büchi conditions are exactly those parity conditions with an odd color $c_1$ and an even color $c_2 > c_1$.
2. Co-Büchi conditions are exactly those parity conditions with an even color $c_0$ and an odd color $c_1 > c_0$. 
Properties

Lemma

Every parity condition is a Boolean combination of Büchi conditions.

Lemma

1. Büchi conditions are exactly those parity conditions with an odd color $c_1$ and an even color $c_2 > c_1$.
2. Co-Büchi conditions are exactly those parity conditions with an even color $c_0$ and an odd color $c_1 > c_0$.

Lemma

The parity condition is self-dual, i.e., $\text{Par}(\Omega) = V^\omega \setminus \text{Par}(\Omega')$ where $\Omega'(v) = \Omega(v) + 1$. 
More Properties: Prefix-independence

Definition
A winning condition $\text{Win} \subseteq V^\omega$ is prefix-independent, if adding or removing a prefix of a play does not change the winner, e.g., if $\rho \in \text{Win} \iff w\rho \in \text{Win}$ for all $\rho \in V^\omega$ and all $w \in V^*$. 

Remark
Parity conditions are prefix-independent.
Reachability and safety conditions are not prefix-independent.

Lemma
Let $\sigma$ be a winning strategy for Player $i$ from a set $W$ of vertices in a game with prefix-independent winning condition. If a play $\rho$ has a suffix starting in $W$ that is consistent with $\sigma$, then $\rho$ is winning for Player $i$. 

Reactive Synthesis (2017/18)
More Properties: Prefix-independence

Definition
A winning condition $\text{Win} \subseteq V^\omega$ is prefix-independent, if adding or removing a prefix of a play does not change the winner, e.g., if $\rho \in \text{Win} \iff w\rho \in \text{Win}$ for all $\rho \in V^\omega$ and all $w \in V^*$.  

Remark

- Parity conditions are prefix-independent.
- Reachability and safety conditions are not prefix-independent.
More Properties: Prefix-independence

**Definition**
A winning condition $\text{Win} \subseteq V^\omega$ is prefix-independent, if adding or removing a prefix of a play does not change the winner, e.g., if $\rho \in \text{Win} \iff \omega\rho \in \text{Win}$ for all $\rho \in V^\omega$ and all $\omega \in V^\ast$.

**Remark**
- Parity conditions are prefix-independent.
- Reachability and safety conditions are not prefix-independent.

**Lemma**
Let $\sigma$ be a winning strategy for Player $i$ from a set $W$ of vertices in a game with prefix-independent winning condition. If a play $\rho$ has a suffix starting in $W$ that is consistent with $\sigma$, then $\rho$ is winning for Player $i$. 
More Properties: Traps

Definition
Let $\mathcal{A} = (V, V_0, V_1, E)$ be an arena. A set $T \subseteq V$ is a trap for Player $i$, if

- every vertex $v \in T \cap V_i$ has only successors in $T$, i.e., $(v, v') \in E$ implies $v' \in T$, and
- every vertex $v \in T \cap V_{1-i}$ there is a successor in $T$, i.e., there is some $v' \in T$ with $(v, v') \in E$.

Remark
The complement of a Player $i$ attractor is a trap for Player $i$.

Lemma
Let $G = (\mathcal{A}, \text{Win})$ be a game with prefix-independent winning condition $\text{Win}$. Then, $W_i(G)$ is a trap for Player $1-i$. 
**Definition**

Let $\mathcal{A} = (V, V_0, V_1, E)$ be an arena. A set $T \subseteq V$ is a *trap* for Player $i$, if

- every vertex $v \in T \cap V_i$ has only successors in $T$, i.e., $(v, v') \in E$ implies $v' \in T$, and
- every vertex $v \in T \cap V_{1-i}$ there is a successor in $T$, i.e., there is some $v' \in T$ with $(v, v') \in E$.

**Remark**

*The complement of a Player $i$ attractor is a trap for Player $i$.***
More Properties: Traps

Definition
Let $\mathcal{A} = (V, V_0, V_1, E)$ be an arena. A set $T \subseteq V$ is a trap for Player $i$, if

- every vertex $v \in T \cap V_i$ has only successors in $T$, i.e., $(v, v') \in E$ implies $v' \in T$, and
- every vertex $v \in T \cap V_{1-i}$ there is a successor in $T$, i.e., there is some $v' \in T$ with $(v, v') \in E$.

Remark
The complement of a Player $i$ attractor is a trap for Player $i$.

Lemma
Let $\mathcal{G} = (\mathcal{A}, \text{Win})$ be a game with prefix-independent winning condition $\text{Win}$. Then, $W_i(\mathcal{G})$ is a trap for Player $1 - i$. 

Reactive Synthesis (2017/18)
Main Theorem

Theorem

Parity games are determined with positional winning strategies.
Theorem
Parity games are determined with positional winning strategies.

Proof.
By induction over the number \( n \) of vertices.
Theorem

*Parity games are determined with positional winning strategies.*

Proof.
By induction over the number \( n \) of vertices.

- The induction start \( n = 1 \) is trivial:

  - Player \((c \mod 2)\) wins with a positional strategy.

- Player \((c \mod 2)\) wins with a positional strategy.
Let $d$ be the maximal color in $G$ and define $i = d \mod 2$, i.e., $d$ is desirable for Player $i$. 

"G"
Let $d$ be the maximal color in $G$ and define $i = d \mod 2$, i.e., $d$ is desirable for Player $i$. 

$D = \Omega^{-1}(d)$
Let $d$ be the maximal color in $G$ and define $i = d \mod 2$, i.e., $d$ is desirable for Player $i$. 

Proof

\[ D = \Omega^{-1}(d) \]

\[ A = \text{Attr}_i(D) \]

\[ V \setminus A \]
First assume $A = V$. 

$D = \Omega^{-1}(d)$

$A = \text{Attr}_i(D)$
Claim $W_i(G) = V$: 

$D = \Omega^{-1}(d)$ 

$A = Attr_i(D)$
Claim \( W_i(G) = V \):

- In \( A \setminus D \) use attractor strategy.
- In \( D \) move arbitrarily.
Claim $W_i(G) = V$:

- In $A \setminus D$ use attractor strategy.
- In $D$ move arbitrarily.

Every play consistent with this strategy visits $D$ infinitely often and is therefore winning for Player $i$. 
Now assume $A \neq V$. 

$D = \Omega^{-1}(d)$ 

$A = \text{Attr}_i(D)$ 

$V \setminus A$ 

$G' \setminus W_i(G')$ 

$W_i(G')$ 

$W_{i-1}(G')$ 

$B = \text{Attr}_{1-i}(W_{i-1}(G'))$ 

$V \setminus B$ 

$W_{i-1}(G'')$ 

$W_{i-1}(G''(W_i(G'))$
Proof

\[ D = \Omega^{-1}(d) \]

\[ A = \text{Attr}_i(D) \]

\[ G' \]

\[ W_i(G') \]
The induction hypothesis is applicable to $G'$. 
Proof

\[ D = \Omega^{-1}(d) \]

\[ A = \text{Attr}_i(D) \]

\[ W_i(G') \]

\[ W_{1-i}(G') \]
First, assume $W_{1-i}(G)$ is empty.
Claim $W_i(G) = V$: 
Proof

\[ D = \Omega^{-1}(d) \]
\[ A = \text{Attr}_{i}(D) \]
\[ W_{i}(G') \]

**Claim** \( W_{i}(G) = V \):
- In \( W_{i}(G') \) use winning strategy \( \sigma' \) from induction hypothesis.
- In \( A \setminus D \) use attractor strategy.
- In \( D \) do anything.
Proof

\[ D = \Omega^{-1}(d) \]

\[ A = \text{Attr}_i(D) \]

\[ W_i(G') \]

**Claim** \( W_i(G) = V \):

- In \( W_i(G') \) use winning strategy \( \sigma' \) from induction hypothesis.
- In \( A \setminus D \) use attractor strategy.
- In \( D \) do anything.

Every consistent play either

- visits \( D \) infinitely often, or
- has a suffix that is consistent with \( \sigma' \).

In both cases, Player \( i \) wins.
Proof

Now, assume \( W_{1-i}(G) \) is non-empty.
Proof

\[ W_{1-i}(G') \]
Proof

\[ V \setminus B \]

\[ B = \text{Attr}_{1-i}(W_{1-i}(G')) \]

\[ W_{1-i}(G') \]
Proof

$$G''$$

$$B = \text{Attr}_{1-i}(W_{1-i}(G'))$$

$$W_{1-i}(G')$$
Proof

\[ G'' = \Omega - 1(d) \]

\[ A = \text{Attr}(D) \]

\[ D = \Omega - 1(d) \]

\[ A = \text{Attr}(D) \]

\[ W_i(G') \]

\[ W_1 - i(G') \]

\[ B = \text{Attr}_{1-i}(W_{1-i}(G')) \]

The induction hypothesis is applicable to \( G'' \).
Proof

\[ \mathcal{G} = \Omega \setminus \text{attr}(D) \]

\[ W_1 - i(\mathcal{G}') = B = \text{attr}_{1 - i}(W_{1 - i}(\mathcal{G}')) \]

\[ W_i(\mathcal{G}'') \]
Claim $W_i(G) = W_i(G'')$: 
Claim \( W_i(G) = W_i(G'') \):

In \( W_i(G'') \) use winning strategy from induction hypothesis.
Proof

Claim \( W_i(\mathcal{G}) = W_i(\mathcal{G}'') \):
In \( W_i(\mathcal{G}'') \) use winning strategy from induction hypothesis.

- \( W_i(\mathcal{G}'') \) is a trap for Player \( i - 1 \) in \( \mathcal{A} \).
- Hence, \( \sigma' \) is also winning from \( W_i(\mathcal{G}'') \) in \( \mathcal{G} \).
Claim \( W_{i-1}(G) = W_{1-i}(G'') \cup B \):
Claim \( W_{i-1}(G) = W_{1-i}(G'') \cup B \):  
In \( W_i(G') \) and \( W_i(G'') \) use winning strategies \( \sigma' \) and \( \sigma'' \) from induction hypothesis, on attractor use attractor strategy.
Claim $W_{i-1}(\mathcal{G}) = W_{1-i}(\mathcal{G}'') \cup B$:

In $W_i(\mathcal{G}')$ and $W_i(\mathcal{G}'')$ use winning strategies $\sigma'$ and $\sigma''$ from induction hypothesis, on attractor use attractor strategy.

- Each such play starting in $W_{1-i}(\mathcal{G}'') \cup B$ has a suffix that is consistent with $\sigma'$ or $\sigma''$. Hence, it is winning for Player $1-i$. 
Determinacy proof yields recursive algorithm with exponential running time.
- Determinacy proof yields recursive algorithm with exponential running time.
- Intriguing complexity-theoretic status: in \( \text{NP} \cap \text{Co-NP} \) and thus unlikely to be complete for \( \text{NP} \) or \( \text{Co-NP} \).
- Even in \( \text{UP} \cap \text{Co-UP} \) and other smaller complexity classes ("the easiest problem not in \( \text{P} \" ").
Determinacy proof yields recursive algorithm with exponential running time.

- Intriguing complexity-theoretic status: in $\text{NP} \cap \text{Co-NP}$ and thus unlikely to be complete for $\text{NP}$ or $\text{Co-NP}$.

- Even in $\text{UP} \cap \text{Co-UP}$ and other smaller complexity classes (”the easiest problem not in P”).

- Until recently, the best deterministic algorithms had running times $O(m \cdot n^{d/3})$ or $n^{O(\sqrt{n})}$.

- But: in practice, the recursive algorithm is typically the fastest.
Determinacy proof yields recursive algorithm with exponential running time.

Intriguing complexity-theoretic status: in $\text{NP} \cap \text{Co-NP}$ and thus unlikely to be complete for $\text{NP}$ or $\text{Co-NP}$.

Even in $\text{UP} \cap \text{Co-UP}$ and other smaller complexity classes (“the easiest problem not in $\text{P}$”).

Until recently, the best deterministic algorithms had running times $O(m \cdot n^{\frac{d}{3}})$ or $n^{O(\sqrt{n})}$.

But: in practice, the recursive algorithm is typically the fastest.

Open problem: is solving parity games in polynomial time?
A Breakthrough

Theorem
Parity games can be solved in quasi-polynomial time, i.e., in time $O(n^{\log d+6})$.

The following is based on slides by John Fearnley (University of Liverpool).
Intuition

Consider the following “finite-duration” variant of parity games:

- The players move a token through the arena until a cycle is closed.
- Player 0 wins if and only if the maximal color on the cycle is even.
Consider the following “finite-duration” variant of parity games:

- The players move a token through the arena until a cycle is closed.
- Player 0 wins if and only if the maximal color on the cycle is even.
Intuition

Consider the following “finite-duration” variant of parity games:

- The players move a token through the arena until a cycle is closed.
- Player 0 wins if and only if the maximal color on the cycle is even.

Lemma

Player 0 wins a parity game from $v$ if and only if she wins the finite-duration variant from $v$.

Proof.

By positional determinacy.
The finite-duration game is essentially a reachability game:

- Vertices are cycle-free play prefixes.
- Once a cycle is closed, the play ends in an accepting sink if the maximal color on the cycle is even. Otherwise, it ends in a rejecting sink.
- Player 0 wins if the accepting sink is reached.
Intuition

The finite-duration game is essentially a reachability game:

- Vertices are cycle-free play prefixes.
- Once a cycle is closed, the play ends in an accepting sink if the maximal color on the cycle is even. Otherwise, it ends in a rejecting sink.
- Player 0 wins if the accepting sink is reached.

Problem
The reachability game has $n^n$ vertices in the worst case.
The finite-duration game is essentially a reachability game:

- Vertices are cycle-free play prefixes.
- Once a cycle is closed, the play ends in an accepting sink if the maximal color on the cycle is even. Otherwise, it ends in a rejecting sink.
- Player 0 wins if the accepting sink is reached.

**Problem**
The reachability game has $n^n$ vertices in the worst case.

The quasi-polynomial time algorithm is based on the following idea:

- Save resources by not aiming for the first cycle.
- Instead find a witness that is “easier” to check while still being complete.
An $i$-sequence is a set of $2^i$ positions such that

- **Evenness**: every position has an even color except (possibly) the last,
- **Inner domination**: every subsequence is dominated by one of its end points, and
- **Outer domination**: the final position dominates the rest of the sequence
Intuition

- Player 0 wins $\Rightarrow$ we should see a $\log(n + 1)$-sequence.
  - Take each position to be the largest color seen infinitely often.

- Player 1 wins $\Rightarrow$ we should not see a $\log(n + 1)$-sequence.
  - The largest color seen infinitely often is odd.
  - Player 1 can force it to be seen before move $n$. 
Intuition

- Player 0 wins $\Rightarrow$ we should see a $\log(n + 1)$-sequence.
  - Take each position to be the largest color seen infinitely often.

- Player 1 wins $\Rightarrow$ we should not see a $\log(n + 1)$-sequence.
  - The largest color seen infinitely often is odd.
  - This color cannot be inner dominated.
  - Player 1 can force it to be seen before move $n$. 

---

$i$-sequences
The idea behind the algorithm:

- Watch the players play the game.
- Uses a compact data structure to track the longest $i$-sequence.
- If we see a $(\log n)$-sequence, declare Player 0 to be the winner.
Each component of the data structure is either \(-\) or a color.

- **Sequences**: If component \(i\) is not \(-\) then
  - the play contains an \(i\)-sequence and
  - the component records the outer domination color

- **Order**: The sequences are ordered, i.e,
  - the 1-sequence comes after the 2-sequence, etc.
The data structure has $\log n$ components.

- Each component needs $\log(d + 1)$ bits.

- So the total size is $(\log n)(\log(d + 1)) \in O(\log^2 n)$. 
We will feed a play into the data structure.

- We start with $-$ $-$ $-$ $-$.
- Each time we see a new color, we use an update rule...
Type 1 Update: New color $c$

- Find the largest $i$ such that
  - position $i$ is blank or $c < \text{color at position } i$ and
  - all colors from position $i$ to the right are even.
- Then:
  - Put the new color at position $i$.
  - Blank all lower positions.
Type 2 Update: New color $c$

- Find the largest $i$ such that $c > \text{color at position } i$.
- Then:
  - Put the new color at position $i$.
  - Blank all lower positions.
**Type 2 Update:** New color $c$

- Find the largest $i$ such that $c >$ color at position $i$.
- Then:
  - Put the new color at position $i$.
  - Blank all lower positions.

First apply Type 1, then apply Type 2
An Example Play

2

- - - 2

Reactive Synthesis (2017/18)
An Example Play

2 \rightarrow 1

- - - 2
\rightarrow - - 1 -
An Example Play

2 → 1 → 4

→ - - 2
→ - - 1 -
→ - - 4 -
An Example Play

\[\begin{array}{c}
\circ \quad 2 \quad \circ \quad 1 \quad \circ \quad 4 \quad \circ \quad 2
\end{array}\]

\[\begin{array}{c}
\rightarrow \quad - - - \quad 2
\rightarrow \quad - - \quad 1 -
\rightarrow \quad - - \quad 4 -
\rightarrow \quad - - \quad 4 2
\end{array}\]
An Example Play

2 → 1 → 4 → 2 → 8

- - - 2
→ - - 1 -
→ - - 4 -
→ - - 4 2
→ - 8 - -
An Example Play

2 1 4 2 8 7

→ - - 2
→ - - 1 -
→ - - 4 -
→ - - 4 2
→ - 8 - -
→ - 8 - 7
An Example Play

\[ \begin{array}{cccccc}
2 & 1 & 4 & 2 & 8 & 7 & 2 \\
\rightarrow & - & - & 2 & & & \\
\rightarrow & - & - & 1 & - & & \\
\rightarrow & - & - & 4 & - & & \\
\rightarrow & - & - & 4 & 2 & & \\
\rightarrow & - & 8 & - & - & & \\
\rightarrow & - & 8 & - & 7 & & \\
\rightarrow & - & 8 & - & 2 & & \\
\end{array} \]
An Example Play

2 1 4 2 8 7 2 1

- - - 2
→ - - 1 -
→ - - 4 -
→ - - 4 2
→ - 8 - -
→ - 8 - 7
→ - 8 - 2
→ - 8 1 -
An Example Play

2 1 4 2 8 7 2 1 4

- - - 2
→ - - 1 -
→ - - 4 -
→ - - 4 2
→ - 8 - -
→ - 8 - 7
→ - 8 - 2
→ - 8 1 -
→ - 8 4 -
An Example Play

2 1 4 2 8 7 2 1 4 1

- - - 2
→ - - 1 -
→ - - 4 -
→ - - 4 2
→ - 8 - -
→ - 8 - 7
→ - 8 - 2
→ - 8 1 -
→ - 8 4 -
→ - 8 4 1
An Example Play

- - - 2
→ - - 1 -
→ - - 4 -
→ - - 4 2
→ - 8 - -
→ - 8 - 7
→ - 8 - 2
→ - 8 1 -
→ - 8 4 -
→ - 8 4 1
→ - 8 4 2
An Example Play

2 1 4 2 8 7 2 1 4 1 2 9

- - - 2
→ - - 1 -
→ - - 4 -
→ - - 4 2
→ - 8 - -
→ - 8 - 7
→ - 8 - 2
→ - 8 1 -
→ - 8 4 -
→ - 8 4 1
→ - 8 4 2
→ 9 - - -
An Example Play

2 → 1 → 4 → 2 → 8 → 7 → 2 → 1 → 4 → 1 → 2 → 9

- - - 2
→ - - 1 -
→ - - 4 -
→ - - 4 2
→ - 8 - -
→ - 8 - 7
→ - 8 - 2
→ - 8 1 -
→ - 8 4 -
→ - 8 4 1
→ - 8 4 2
→ 9 - - -
→ ...
The Algorithm

Use the data structure as memory structure and play a reachability game in the product arena.

- Initialize memory with $\_\_\_\_\_\_\_\_$ and use update rules to implement update function.
- Goal vertices: those whose first position is set.
- The resulting game has $n^{O(\log n)}$ vertices and can be constructed and solved in time $n^{O(\log n)}$.

Lemma
Player 0 wins the parity game from $v$ if and only if she wins the reachability game from $(v, \_\_\_\_\_\_\_\_)$. Note this is not a game reduction as introduced in the previous lecture!
The Algorithm

Use the data structure as memory structure and play a reachability game in the product arena.

- Initialize memory with $-$ $-$ $-$ $-$ and use update rules to implement update function.
- Goal vertices: those whose first position is set.
- The resulting game has $n^{O(\log n)}$ vertices and can be constructed and solved in time $n^{O(\log n)}$.

**Lemma**

*Player 0 wins the parity game from $v$ if and only if she wins the reachability game from $(v, - - - -)$.*
The Algorithm

Use the data structure as memory structure and play a reachability game in the product arena.

- Initialize memory with $- - - -$ and use update rules to implement update function.
- Goal vertices: those whose first position is set.
- The resulting game has $n^{O(\log n)}$ vertices and can be constructed and solved in time $n^{O(\log n)}$.

Lemma

Player 0 wins the parity game from $v$ if and only if she wins the reachability game from $(v, - - - -)$.

Note

This is not a game reduction as introduced in the previous lecture!
[CJKLS 17] Quasi-polynomial time and space.
[CJKLS 17] Quasi-polynomial time and space.

[GI-J 17] A simpler proof, slightly better running time, and finer running time analysis.

[JL 17] Quasi-polynomial time and linear space with a completely different approach and a fine running time analysis.

[FJSW 17] Quasi-polynomial time and linear space by a refinement of the original approach.

But still no polynomial-time algorithm.
The Latest

[CJKLS 17] Quasi-polynomial time and space.

[GI-J 17] A simpler proof, slightly better running time, and finer running time analysis.

[JL 17] Quasi-polynomial time and linear space with a completely different approach and a fine running time analysis.
The Latest

[CJKLS 17] Quasi-polynomial time and space.

[GI-J 17] A simpler proof, slightly better running time, and finer running time analysis.

[JL 17] Quasi-polynomial time and linear space with a completely different approach and a fine running time analysis.

[FJSW 17] Quasi-polynomial time and linear space by a refinement of the original approach.
The Latest

[CJKLS 17] Quasi-polynomial time and space.

[GI-J 17] A simpler proof, slightly better running time, and finer running time analysis.

[JL 17] Quasi-polynomial time and linear space with a completely different approach and a fine running time analysis.

[FJSW 17] Quasi-polynomial time and linear space by a refinement of the original approach.

But still no polynomial-time algorithm.