Exact computation of the number of accepting paths of an NTM

Subrahmanyam Kalyanasundaram[1] Kenneth W. Regan[2]

[1]Department of Computer Science and Engineering
Indian Institute of Technology Hyderabad

[2]Department of Computer Science and Engineering
University at Buffalo

Feburary 16, 2018
CALDAM 2018, IIT Guwahati

Exact computation of the number of accepting paths of an NTM

Subrahmanyam Kalyanasundaram[1] Kenneth W. Regan[2]

[1]Department of Computer Science and Engineering
Indian Institute of Technology Hyderabad

[2]Department of Computer Science and Engineering
University at Buffalo

Feburary 16, 2018
CALDAM 2018, IIT Guwahati

Outline

Outline

Trying to Understand Nondeterminism

- One of the fundamental goals is to understand the power of nondeterminism.
- Is nondeterministic computation really more powerful than deterministic computation?
- A concrete answer would resolve the P vs. NP question.

- In this paper, we study how fast we can count the number of accepting paths of an NTM.

Trying to Understand Nondeterminism

- One of the fundamental goals is to understand the power of nondeterminism.
- Is nondeterministic computation really more powerful than deterministic computation?
- A concrete answer would resolve the P vs. NP question.

- In this paper, we study how fast we can count the number of accepting paths of an NTM.

The question

Question

If an NTM N runs in time $t = t(n)$, how fast can we deterministically count the number of accepting computations?

- We can count using the configuration graph.
- For a graph of size S, this results in an $O(S)$ algorithm.
- Typically $S \sim a^{kt}$.

Our answer

We show that this can be done in time roughly *square root* of the size of the configuration graph.

The question

Question

If an NTM N runs in time $t = t(n)$, how fast can we deterministically count the number of accepting computations?

- We can count using the configuration graph.
- For a graph of size S, this results in an $O(S)$ algorithm.
- Typically $S \sim a^{kt}$.

Our answer

We show that this can be done in time roughly *square root* of the size of the configuration graph.

The question

Question

If an NTM N runs in time $t = t(n)$, how fast can we deterministically count the number of accepting computations?

- We can count using the configuration graph.
- For a graph of size S, this results in an $O(S)$ algorithm.
- Typically $S \sim a^{kt}$.

Our answer

We show that this can be done in time roughly *square root* of the size of the configuration graph.

The question

Question

If an NTM N runs in time $t = t(n)$, how fast can we deterministically count the number of accepting computations?

- We can count using the configuration graph.
- For a graph of size S, this results in an $O(S)$ algorithm.
- Typically $S \sim a^{kt}$.

Our answer

We show that this can be done in time roughly *square root* of the size of the configuration graph.

The question

Question

If an NTM N runs in time $t = t(n)$, how fast can we deterministically count the number of accepting computations?

- We can count using the configuration graph.
- For a graph of size S, this results in an $O(S)$ algorithm.
- Typically $S \sim a^{kt}$.

Our answer

We show that this can be done in time roughly *square root* of the size of the configuration graph.

Main Result

Theorem

Given an NTM N, which runs in time t, we can count the number of accepting paths of N on a given input in time

$$a^{kt/2}\ H_a^{k\sqrt{t}\log t} q^2 \mathrm{poly}(\log q, k, t, a).$$

Parameters of NTM N	Denoted by
Number of tapes	k
Alphabet Size	a
Number of States	q
Running time	$t = t(n)$

Main Result

Theorem

Given an NTM N, which runs in time t, we can count the number of accepting paths of N on a given input in time

$$a^{kt/2} H_a^{k\sqrt{t}\log t} q^2 \mathrm{poly}(\log q, k, t, a).$$

Parameters of NTM N	Denoted by
Number of tapes	k
Alphabet Size	a
Number of States	q
Running time	$t = t(n)$

Related Results: What is known already

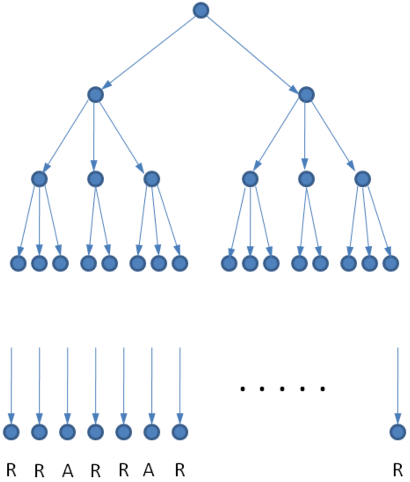
- Counting variants of different problems behave differently.
  - Polynomial time: Kirchhoff's matrix-tree theorem and Kasteleyn's theorem.
  - #P-complete: Perfect matchings in an arbitrary graph and satisfying assignments of a CNF formula.
  - FPRAS: Satisfying assignments of a DNF formula and perfect matchings in a bipartite graph.
- But no result for general nondeterministic machines.
- [vMS 05]: Faster simulation of probabilistic polytime machines in time $o(2^t)$.
  - Model of [vMS 05] restrict the amount of nondeterministic choices.

Our approach

- [KLRS 2011] showed that NTM simulation can be performed in $a^{kt/2}$ time.
- Combined two approaches: BFS and Block Trace.
- We extend the above to the problem of counting the number of accepting paths.

Outline

Configuration Tree



The Naive Approach

Parameters of NTM N	Denoted by
Number of tapes	k
Alphabet Size	a
Number of States	q
Running time	$t = t(n)$

- The straightforward approach; check each computation path.
- This approach takes c^t time, where c is the maximum degree of the computation tree.

The Naive Approach

Parameters of NTM N	Denoted by
Number of tapes	k
Alphabet Size	a
Number of States	q
Running time	$t = t(n)$

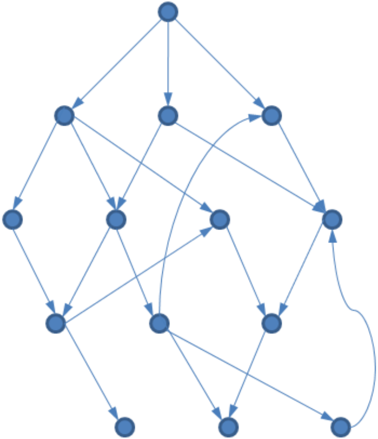
- The straightforward approach; check each computation path.
- This approach takes c^t time, where c is the maximum degree of the computation tree.

The Naive Approach

Parameters of NTM N	Denoted by
Number of tapes	k
Alphabet Size	a
Number of States	q
Running time	$t = t(n)$

- The straightforward approach; check each computation path.
- This approach takes c^t time, where c is the maximum degree of the computation tree.

BFS on Configuration Graph



BFS on Configuration Graph

- BFS can be used to count the number of shortest paths.
  - But each accepting path need not be a shortest path.
- We modify the configuration graph as follows:
  - In place of each configuration ρ, we have (ρ, i).
  - For a directed edge $\rho \longrightarrow \rho'$, we have $(\rho, i) \longrightarrow (\rho', i+1)$.
  - All paths are shortest paths.
- Total no. of vertices is $S \cdot (t+1) = a^{kt} t^k q \cdot (t+1)$.

BFS on Configuration Graph

- BFS can be used to count the number of shortest paths.
  - But each accepting path need not be a shortest path.
- We modify the configuration graph as follows:
  - In place of each configuration ρ, we have (ρ, i).
  - For a directed edge $\rho \longrightarrow \rho'$, we have $(\rho, i) \longrightarrow (\rho', i+1)$.
  - All paths are shortest paths.
- Total no. of vertices is $S \cdot (t+1) = a^{kt} t^k q \cdot (t+1)$.

BFS on Configuration Graph

- BFS can be used to count the number of shortest paths.
  - But each accepting path need not be a shortest path.
- We modify the configuration graph as follows:
  - In place of each configuration ρ, we have (ρ, i).
  - For a directed edge $\rho \longrightarrow \rho'$, we have $(\rho, i) \longrightarrow (\rho', i+1)$.
  - All paths are shortest paths.
- Total no. of vertices is $S \cdot (t+1) = a^{kt} t^k q \cdot (t+1)$.

BFS on Configuration Graph

- Total no. of vertices is $a^{kt}\ t^k q \cdot (t+1)$.
  - For each vertex (ρ, i), we compute the number of (shortest) paths from $(\rho_x, 0)$.
  - Then sum up the number of accepting computation paths.

Theorem

This approach takes $a^{kt}\ q^2(3at)^k \mathrm{poly}(\log q, k, t, a)$ time.

- The dominant factor above comes from the number of configurations.

BFS on Configuration Graph

- Total no. of vertices is $a^{kt}\ t^k q \cdot (t+1)$.
  - For each vertex (ρ, i), we compute the number of (shortest) paths from $(\rho_x, 0)$.
  - Then sum up the number of accepting computation paths.

Theorem

This approach takes $a^{kt}\ q^2(3at)^k \mathrm{poly}(\log q, k, t, a)$ time.

- The dominant factor above comes from the number of configurations.

BFS on Configuration Graph

- Total no. of vertices is $a^{kt}\ t^k q \cdot (t+1)$.
  - For each vertex (ρ, i), we compute the number of (shortest) paths from $(\rho_x, 0)$.
  - Then sum up the number of accepting computation paths.

Theorem

This approach takes $a^{kt}\ q^2(3at)^k\text{poly}(\log q, k, t, a)$ time.

- The dominant factor above comes from the number of configurations.

BFS on Configuration Graph

- Total no. of vertices is $a^{kt}\ t^k q \cdot (t+1)$.
  - For each vertex (ρ, i), we compute the number of (shortest) paths from $(\rho_x, 0)$.
  - Then sum up the number of accepting computation paths.

Theorem

This approach takes $a^{kt}\ q^2 (3at)^k \text{poly}(\log q, k, t, a)$ *time.*

- The dominant factor above comes from the number of configurations.

BFS on Configuration Graph

- Total no. of vertices is a^{kt} $t^k q \cdot (t+1)$.
  - For each vertex (ρ, i), we compute the number of (shortest) paths from $(\rho_x, 0)$.
  - Then sum up the number of accepting computation paths.

Theorem

This approach takes a^{kt} $q^2(3at)^k \text{poly}(\log q, k, t, a)$ time.

- The dominant factor above comes from the number of configurations.

Outline

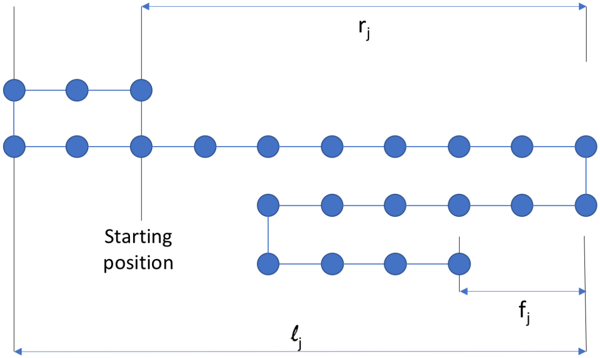
Block Traces

- A segment of block size *d* consists of the following over the next *d* steps:
  - How far to the right do the tape heads go?
  - How far to the left do the tape heads go?
  - Where do the tape heads end up?
  - What are contents of the cells traversed?
- A block trace is a sequence of such segments.
- Each computation path correspond to a distinct block trace witness.

Block Traces

- A segment of block size d consists of the following over the next d steps:
  - How far to the right do the tape heads go?
  - How far to the left do the tape heads go?
  - Where do the tape heads end up?
  - What are contents of the cells traversed?
- A block trace is a sequence of such segments.
- Each computation path correspond to a distinct block trace witness.

Block Traces



Block Trace Approach

Lemma

The number of accepting computations on a given input that are compatible with a given block trace witness can be calculated in time $q^2 a^{3kd} \mathrm{poly}(\log q, k, t, a, d)$.

- We try all possible block traces and compute the number of accepting paths.
- Number of block traces = $a^{kt} 32^{kt/d}$.
- Optimizing for the block size d, we get the following:

Block Trace Approach

Lemma

The number of accepting computations on a given input that are compatible with a given block trace witness can be calculated in time $q^2 a^{3kd} \mathrm{poly}(\log q, k, t, a, d)$.

- We try all possible block traces and compute the number of accepting paths.
- Number of block traces = $a^{kt} 32^{kt/d}$.
- Optimizing for the block size d, we get the following:

Block Trace Approach

Lemma

The number of accepting computations on a given input that are compatible with a given block trace witness can be calculated in time $q^2 a^{3kd} \text{poly}(\log q, k, t, a, d)$.

- We try all possible block traces and compute the number of accepting paths.
- Number of block traces = $a^{kt} 32^{kt/d}$.
- Optimizing for the block size d, we get the following:

Block Trace Approach

Lemma

The number of accepting computations on a given input that are compatible with a given block trace witness can be calculated in time $q^2 a^{3kd} \text{poly}(\log q, k, t, a, d)$.

- We try all possible block traces and compute the number of accepting paths.
- Number of block traces = $a^{kt} 32^{kt/d}$.
- Optimizing for the block size d, we get the following:

Block Trace Approach

Theorem

The number of accepting computation paths on a given input can be computed in time

$$a^{kt} C_a^{k\sqrt{t}} \cdot q^2 \mathrm{poly}(\log q, k, t, a),$$

where C_a is a constant that depends only on a.

Block Trace Approach

Theorem

The number of accepting computation paths on a given input can be computed in time

$$a^{kt} C_a^{k\sqrt{t}} \cdot q^2 \text{poly}(\log q, k, t, a),$$

where C_a is a constant that depends only on a.

Outline

Idea: Combine the approaches

- Two approaches: BFS and Block Traces.
- Both have comparable running time with a^{kt} being the dominant factor.
- The idea is to mix the two cleverly.

Tape Usage is Less than Half

- In the BFS approach, a^{kt} factor was due to number of tape configurations.
- Maximum possible tape usage is kt.
- If the tape usage is less, then we could save time on the BFS approach.

First Observation

If the total tape use is $\leq kt/2$, then the BFS approach runs in time roughly $a^{kt/2}$.

- But what if tape usage is more?

Tape Usage is Less than Half

- In the BFS approach, a^{kt} factor was due to number of tape configurations.
- Maximum possible tape usage is kt.
- If the tape usage is less, then we could save time on the BFS approach.

First Observation

If the total tape use is $\leq kt/2$, then the BFS approach runs in time roughly $a^{kt/2}$.

- But what if tape usage is more?

Tape Usage is Less than Half

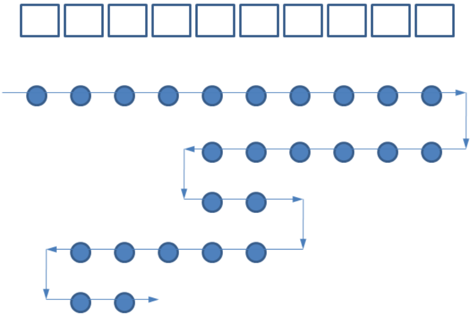
- In the BFS approach, a^{kt} factor was due to number of tape configurations.
- Maximum possible tape usage is kt.
- If the tape usage is less, then we could save time on the BFS approach.

First Observation

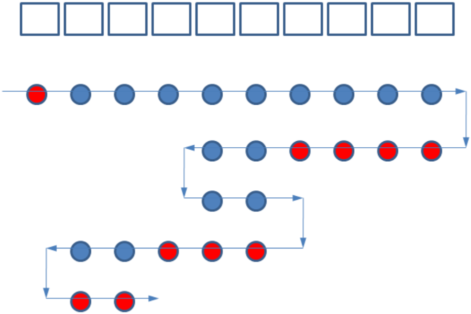
If the total tape use is $\leq kt/2$, then the BFS approach runs in time roughly $a^{kt/2}$.

- But what if tape usage is more?

In Figures



In Figures



Tape Usage is More than Half

- For every location visited, there is a last visit.
- If the total tape use is $\geq kt/2$, over half the visits are last visits.
- There is no need to write anything during the last visit.
- This saves a factor of $a^{kt/2}$ in the block trace approach.

Second Observation

Thus the block trace approach would yield a running time roughly $a^{kt/2}$.

Tape Usage is More than Half

- For every location visited, there is a last visit.
- If the total tape use is $\geq kt/2$, over half the visits are last visits.
- There is no need to write anything during the last visit.
- This saves a factor of $a^{kt/2}$ in the block trace approach.

Second Observation

Thus the block trace approach would yield a running time roughly $a^{kt/2}$.

Tape Usage is More than Half

- For every location visited, there is a last visit.
- If the total tape use is $\geq kt/2$, over half the visits are last visits.
- There is no need to write anything during the last visit.
- This saves a factor of $a^{kt/2}$ in the block trace approach.

Second Observation

Thus the block trace approach would yield a running time roughly $a^{kt/2}$.

Tape Usage is More than Half

- For every location visited, there is a last visit.
- If the total tape use is $\geq kt/2$, over half the visits are last visits.
- There is no need to write anything during the last visit.
- This saves a factor of $a^{kt/2}$ in the block trace approach.

Second Observation

Thus the block trace approach would yield a running time roughly $a^{kt/2}$.

The Whole Algorithm

- List down all possible directional paths.
- Compare the total tape usage to $kt/2$.
- Depending on the comparison, choose the approach.

Theorem (Main Theorem)

The number of accepting computations of an NTM on a given input can be computed in time

$$a^{kt/2}\ H_a^{k\sqrt{t}\log t} q^2 \mathrm{poly}(\log q, k, t, a).$$

The Whole Algorithm

- List down all possible directional paths.
- Compare the total tape usage to $kt/2$.
- Depending on the comparison, choose the approach.

Theorem (Main Theorem)

The number of accepting computations of an NTM on a given input can be computed in time

$$a^{kt/2}\ H_a^{k\sqrt{t}\log t} q^2 \mathrm{poly}(\log q, k, t, a).$$

The Whole Algorithm

- List down all possible directional paths.
- Compare the total tape usage to $kt/2$.
- Depending on the comparison, choose the approach.

Theorem (Main Theorem)

The number of accepting computations of an NTM on a given input can be computed in time

$$a^{kt/2}\, H_a^{k\sqrt{t}\log t} q^2 \mathrm{poly}(\log q, k, t, a).$$

The Whole Algorithm

- List down all possible directional paths.
- Compare the total tape usage to $kt/2$.
- Depending on the comparison, choose the approach.

Theorem (Main Theorem)

The number of accepting computations of an NTM on a given input can be computed in time

$$a^{kt/2}\ H_a^{k\sqrt{t}\log t} q^2 \text{poly}(\log q, k, t, a).$$

The Whole Algorithm

- List down all possible directional paths.
- Compare the total tape usage to $kt/2$.
- Depending on the comparison, choose the approach.

Theorem (Main Theorem)

The number of accepting computations of an NTM on a given input can be computed in time

$$a^{kt/2} H_a^{k\sqrt{t}\log t} q^2 \text{poly}(\log q, k, t, a).$$

Outline

Concluding Remarks

- This implies a faster deterministic simulation of the following counting classes:
  - Parity classes $\oplus$P and Mod_kP.
  - Probabilistic classes PP, BPP, ZPP and BQP (an improvement over [vMS 05]).
- Can we improve the exponent of the running time, to say $kt/3$?
- Could we extend this framework to simulate classes higher up in the polynomial hierarchy, like Σ_2P?

Concluding Remarks

- This implies a faster deterministic simulation of the following counting classes:
  - Parity classes $\oplus$P and $\mathrm{Mod_k P}$.
  - Probabilistic classes PP, BPP, ZPP and BQP (an improvement over [vMS 05]).
- Can we improve the exponent of the running time, to say $kt/3$?
- Could we extend this framework to simulate classes higher up in the polynomial hierarchy, like $\Sigma_2 P$?

Concluding Remarks

- This implies a faster deterministic simulation of the following counting classes:
  - Parity classes $\oplus$P and $\mathrm{Mod_k P}$.
  - Probabilistic classes PP, BPP, ZPP and BQP (an improvement over [vMS 05]).
- Can we improve the exponent of the running time, to say $kt/3$?
- Could we extend this framework to simulate classes higher up in the polynomial hierarchy, like Σ_2P?

Concluding Remarks

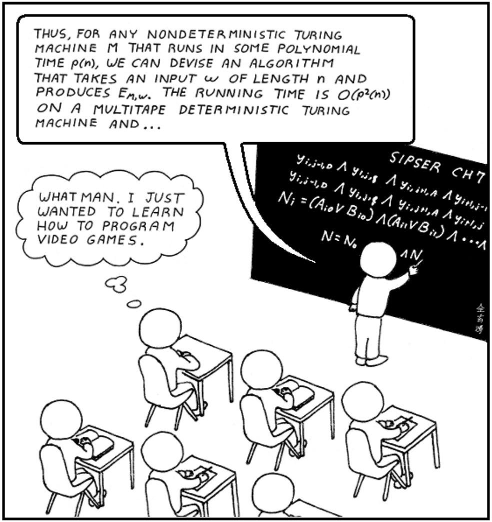
- This implies a faster deterministic simulation of the following counting classes:
  - Parity classes $\oplus$P and Mod_kP.
  - Probabilistic classes PP, BPP, ZPP and BQP (an improvement over [vMS 05]).
- Can we improve the exponent of the running time, to say $kt/3$?
- Could we extend this framework to simulate classes higher up in the polynomial hierarchy, like Σ_2P?

Concluding Remarks

- This implies a faster deterministic simulation of the following counting classes:
  - Parity classes $\oplus$P and Mod_kP.
  - Probabilistic classes PP, BPP, ZPP and BQP (an improvement over [vMS 05]).

- Can we improve the exponent of the running time, to say $kt/3$?
- Could we extend this framework to simulate classes higher up in the polynomial hierarchy, like Σ_2P?

Thank You



“Rite of Passage”: Abstruse Goose Comic, available at `http://abstrusegoose.com/206` with minor changes.