Complexity Models for Incremental Computation

Shahbaz Khan, PhD CSE

Paper by
Peter Bro Miltersen, Sairam Subramanian,
Jeffery Scott Vitter and Roberto Tamassia

Outline

Computation Model

Motivation

Computation Model

Motivation

- The efficiency of an algorithm is judged by application
  Worst Case Time, Expected Time, Space, Updation Time etc.

Computation Model

Motivation

- The efficiency of an algorithm is judged by application
  Worst Case Time, Expected Time, Space, Updation Time etc.
- Given an instance of the problem that changes over time.

Computation Model

Motivation

- The efficiency of an algorithm is judged by application
  Worst Case Time, Expected Time, Space, Updation Time etc.
- Given an instance of the problem that changes over time.
- Aim is to preprocess such that it can be updated easily.

Computation Model

Motivation

- The efficiency of an algorithm is judged by application
  Worst Case Time, Expected Time, Space, Updation Time etc.
- Given an instance of the problem that changes over time.
- Aim is to preprocess such that it can be updated easily.
- Trivial: Recompute from scratch after every update.

Computation Model

Motivation

- The efficiency of an algorithm is judged by application
  Worst Case Time, Expected Time, Space, Updation Time etc.
- Given an instance of the problem that changes over time.
- Aim is to preprocess such that it can be updated easily.
- Trivial: Recompute from scratch after every update.
- Idea: Data structure that is queried and updated.

Computation Model

Motivation

- The efficiency of an algorithm is judged by application
  Worst Case Time, Expected Time, Space, Updation Time etc.
- Given an instance of the problem that changes over time.
- Aim is to preprocess such that it can be updated easily.
- Trivial: Recompute from scratch after every update.
- Idea: Data structure that is queried and updated.

Replacement Model of Computation

Computation Model

Motivation

- The efficiency of an algorithm is judged by application
  Worst Case Time, Expected Time, Space, Updation Time etc.
- Given an instance of the problem that changes over time.
- Aim is to preprocess such that it can be updated easily.
- Trivial: Recompute from scratch after every update.
- Idea: Data structure that is queried and updated.

Replacement Model of Computation

- Given an instance I of a decision problem π.

Computation Model

Motivation

- The efficiency of an algorithm is judged by application
  Worst Case Time, Expected Time, Space, Updation Time etc.
- Given an instance of the problem that changes over time.
- Aim is to preprocess such that it can be updated easily.
- Trivial: Recompute from scratch after every update.
- Idea: Data structure that is queried and updated.

Replacement Model of Computation

- Given an instance I of a decision problem π.
- We allow an algorithm to preprocess I to build D.

Computation Model

Motivation

- The efficiency of an algorithm is judged by application
  Worst Case Time, Expected Time, Space, Updation Time etc.
- Given an instance of the problem that changes over time.
- Aim is to preprocess such that it can be updated easily.
- Trivial: Recompute from scratch after every update.
- Idea: Data structure that is queried and updated.

Replacement Model of Computation

- Given an instance I of a decision problem π.
- We allow an algorithm to preprocess I to build D.
- An update is in form of Δ bit flips of I.

Contents

Topics covered

Contents

Topics covered

- Define incremental complexity classes and reductions.

Contents

Topics covered

- Define incremental complexity classes and reductions.
- Problems hard to parallelize are hard to dynamize.

Contents

Topics covered

- Define incremental complexity classes and reductions.
- Problems hard to parallelize are hard to dynamize.
- Problems hard to solve in small space are hard to dynamize.

Contents

Topics covered

- Define incremental complexity classes and reductions.
- Problems hard to parallelize are hard to dynamize.
- Problems hard to solve in small space are hard to dynamize.
- Describe the complete problems for class P.

Contents

Topics covered

- Define incremental complexity classes and reductions.
- Problems hard to parallelize are hard to dynamize.
- Problems hard to solve in small space are hard to dynamize.
- Describe the complete problems for class P.
- Problems solvable is small space have better dynamic solutions.

Outline

General Definitions

Basic Notation

General Definitions

Basic Notation

- Given decision problem π with initial instance I^o.

General Definitions

Basic Notation

- Given decision problem π with initial instance I^o.
- Positive Instance $(\pi(I) = 1)$ and Negative otherwise.

General Definitions

Basic Notation

- Given decision problem π with initial instance I^o.
- Positive Instance $(\pi(I) = 1)$ and Negative otherwise.
- Update Δ changes current instance I to I'.

General Definitions

Basic Notation

- Given decision problem π with initial instance I^o.
- Positive Instance $(\pi(I) = 1)$ and Negative otherwise.
- Update Δ changes current instance I to I'.
- Size of instance $|I| = |I'| = |I^o| = n$.

General Definitions

Basic Notation

- Given decision problem π with initial instance I^o.
- Positive Instance ($\pi(I) = 1$) and Negative otherwise.
- Update Δ changes current instance I to I'.
- Size of instance $|I| = |I'| = |I^o| = n$.
- Any algorithm $\mathcal{A}$ has two stages: *preprocess and update*.

General Definitions

Basic Notation

- Given decision problem π with initial instance I^o.
- Positive Instance $(\pi(I) = 1)$ and Negative otherwise.
- Update Δ changes current instance I to I'.
- Size of instance $|I| = |I'| = |I^o| = n$.
- Any algorithm $\mathcal{A}$ has two stages: *preprocess and update*.
- We allow an algorithm to preprocess I to build D.

General Definitions

Basic Notation

- Given decision problem π with initial instance I^o.
- Positive Instance $(\pi(I) = 1)$ and Negative otherwise.
- Update Δ changes current instance I to I'.
- Size of instance $|I| = |I'| = |I^o| = n$.
- Any algorithm $\mathcal{A}$ has two stages: *preprocess and update*.
- We allow an algorithm to preprocess I to build D.
- $\mathcal{A}$ preprocesses I^o to form data structure D_{I^o}.

General Definitions

Basic Notation

- Given decision problem π with initial instance I^o.
- Positive Instance ($\pi(I) = 1$) and Negative otherwise.
- Update Δ changes current instance I to I'.
- Size of instance $|I| = |I'| = |I^o| = n$.
- Any algorithm $\mathcal{A}$ has two stages: *preprocess and update*.
- We allow an algorithm to preprocess I to build D.
- $\mathcal{A}$ preprocesses I^o to form data structure D_{I^o}.
- $\mathcal{A}$ processes Δ by reporting $\pi(I')$ and updating D_I to $D_{I'}$.

General Definitions

Definition

incr-**TIME[$f(n)$]:** (*analogous to* DTIME[$f(n)$])

Decision problem π belongs to *incr*-TIME[$f(n)$] if there exists RAM programs P_1 and P_2 such that $\forall n \in \mathbb{N}$

General Definitions

Definition

incr-**TIME[$f(n)$]:** (*analogous to* DTIME[$f(n)$])

Decision problem π belongs to *incr*-TIME[$f(n)$] if there exists RAM programs P_1 and P_2 such that $\forall n \in \mathbb{N}$

- P_1 efficiently processes I^o, where $|I^o| = n$ to compute D_{I^o}.

General Definitions

Definition

incr-**TIME[$f(n)$]:** (*analogous to* DTIME[$f(n)$])
Decision problem π belongs to *incr*-TIME[$f(n)$] if there exists RAM programs P_1 and P_2 such that $\forall n \in \mathbb{N}$

- P_1 efficiently processes I^o, where $|I^o| = n$ to compute D_{I^o}.
- Given update Δ on I and current data structure D_I in RAM, P_2 computes $\pi(I')$ and updates data structure D_I to $D_{I'}$ in $O(|\Delta| f(n))$ time.

General Definitions

Definition

incr-**TIME[$f(n)$]:** (*analogous to* DTIME[$f(n)$])
Decision problem π belongs to *incr*-TIME[$f(n)$] if there exists RAM programs P_1 and P_2 such that $\forall n \in \mathbb{N}$

- P_1 efficiently processes I^o, where $|I^o| = n$ to compute D_{I^o}.
- Given update Δ on I and current data structure D_I in RAM, P_2 computes $\pi(I')$ and updates data structure D_I to $D_{I'}$ in $O(|\Delta| f(n))$ time.

Basic Classes

incr-CONSTANT-TIME, *incr*-LOG-TIME

$$incr\text{-POLYLOGTIME} = \bigcup_{k \geq 0} incr\text{-TIME}[\log^k n]$$

General Definitions

Definition

incr-**SPACE[$f(n)$]:** (*analogous to* DSPACE[$f(n)$])
Decision problem π belongs to *incr*-SPACE[$f(n)$] if there exists RAM programs P_1 and P_2 such that $\forall n \in \mathbb{N}$

General Definitions

Definition

incr-**SPACE[$f(n)$]:** (*analogous to* DSPACE[$f(n)$])
Decision problem π belongs to *incr*-SPACE[$f(n)$] if there exists RAM programs P_1 and P_2 such that $\forall n \in \mathbb{N}$

- P_1 efficiently processes I^o, where $|I^o| = n$ to compute D_{I^o}.

General Definitions

Definition

incr-**SPACE[$f(n)$]:** (*analogous to* DSPACE[$f(n)$])
Decision problem π belongs to *incr*-SPACE[$f(n)$] if there exists RAM programs P_1 and P_2 such that $\forall n \in \mathbb{N}$

- P_1 efficiently processes I^o, where $|I^o| = n$ to compute D_{I^o}.
- Given update Δ on I and current data structure D_I in read only RAM, P_2 computes $\pi(I')$ and constructs data structure $D_{I'}$ on write-only-memory using $O(|\Delta| f(n))$ work space.

General Definitions

Definition

incr-**SPACE[$f(n)$]:** (*analogous to* DSPACE[$f(n)$])
Decision problem π belongs to *incr*-SPACE[$f(n)$] if there exists RAM programs P_1 and P_2 such that $\forall n \in \mathbb{N}$

- P_1 efficiently processes I^o, where $|I^o| = n$ to compute D_{I^o}.
- Given update Δ on I and current data structure D_I in read only RAM, P_2 computes $\pi(I')$ and constructs data structure $D_{I'}$ on write-only-memory using $O(|\Delta| f(n))$ work space.

Basic Classes

incr-LOGSPACE

$$incr\text{-POLYLOGSPACE} = \bigcup_{k \geq 0} incr\text{-SPACE}[\log^k n]$$

Incremental Reductions

Motivation

Incremental Reductions

Motivation

- To compare hardness of solving two problems dynamically.

Incremental Reductions

Motivation

- To compare hardness of solving two problems dynamically.
- The complexity of P_1 is not significant, main focus on P_2. (f)

Incremental Reductions

Motivation

- To compare hardness of solving two problems dynamically.
- The complexity of P_1 is not significant, main focus on P_2. (f)
- Relative size of updates i.e. Δ_1 and Δ_2. (g)

Incremental Reductions

Motivation

- To compare hardness of solving two problems dynamically.
- The complexity of P_1 is not significant, main focus on P_2. (f)
- Relative size of updates i.e. Δ_1 and Δ_2. (g)
- Relative size of mapping i.e. π_1 and π_2. (p)

Incremental Reductions

Motivation

- To compare hardness of solving two problems dynamically.
- The complexity of P_1 is not significant, main focus on P_2. (f)
- Relative size of updates i.e. Δ_1 and Δ_2. (g)
- Relative size of mapping i.e. π_1 and π_2. (p)
- Represented as $\pi_1 \leq_{incr[f(n),g(n),p(n)]} \pi_2$

Incremental Reducibility

Definition $\pi_1 \leq_{incr[f(n),g(n),p(n)]} \pi_2$

Decision problem π_1 is *incrementally reducible* to π_2 if there exist transformation T and RAM programs P and Q such that

Incremental Reducibility

Definition $\pi_1 \leq_{incr[f(n),g(n),p(n)]} \pi_2$

Decision problem π_1 is *incrementally reducible* to π_2 if there exist transformation T and RAM programs P and Q such that

- $T : \pi_1 \rightarrow \pi_2$, where $|\pi_2| = p(n)$ and $\pi_2(T(I)) = \pi_1(I)$.

Incremental Reducibility

Definition $\pi_1 \leq_{incr[f(n),g(n),p(n)]} \pi_2$

Decision problem π_1 is *incrementally reducible* to π_2 if there exist transformation T and RAM programs P and Q such that

- $T : \pi_1 \rightarrow \pi_2$, where $|\pi_2| = p(n)$ and $\pi_2(T(I)) = \pi_1(I)$.
- Given $I^o \in \pi_1$, P efficiently computes $T(I^o)$ and S_{I^o}.

Incremental Reducibility

Definition $\pi_1 \leq_{incr[f(n),g(n),p(n)]} \pi_2$

Decision problem π_1 is *incrementally reducible* to π_2 if there exist transformation T and RAM programs P and Q such that

- $T : \pi_1 \rightarrow \pi_2$, where $|\pi_2| = p(n)$ and $\pi_2(T(I)) = \pi_1(I)$.
- Given $I^o \in \pi_1$, P efficiently computes $T(I^o)$ and S_{I^o}.
- Given update Δ_1 on I with S_I in RAM, Q computes Δ_2 on $T(I)$ such that $|\Delta_2| \leq g(n)|\Delta_1|$ and modifies data structure S_I to $S_{I'}$ using $O(|\Delta|f(n))$ time.

Incremental Reducibility

Definition $\pi_1 \leq_{incr[f(n),g(n),p(n)]} \pi_2$

Decision problem π_1 is *incrementally reducible* to π_2 if there exist transformation T and RAM programs P and Q such that

- $T : \pi_1 \rightarrow \pi_2$, where $|\pi_2| = p(n)$ and $\pi_2(T(I)) = \pi_1(I)$.
- Given $I^o \in \pi_1$, P efficiently computes $T(I^o)$ and S_{I^o}.
- Given update Δ_1 on I with S_I in RAM, Q computes Δ_2 on $T(I)$ such that $|\Delta_2| \leq g(n)|\Delta_1|$ and modifies data structure S_I to $S_{I'}$ using $O(|\Delta| f(n))$ time.

Theorem

If $\pi_1 \leq_{incr[f(n),g(n),p(n)]} \pi_2$ and $\pi_2 \in incr\text{-TIME}[h(n)]$ then $\pi_1 \in incr\text{-TIME}[f(n) + g(n).h(p(n))]$.

Outline

Some Definitions and Theorems

Definition

Decision problem π is $incr[f(n), g(n), p(n)]$-Complete for class C if

Some Definitions and Theorems

Definition

Decision problem π is $incr[f(n), g(n), p(n)]$-Complete for class C if

1. $\pi \in C$.

Some Definitions and Theorems

Definition

Decision problem π is *incr*$[f(n), g(n), p(n)]$-Complete for class C if

1. $\pi \in C$.
2. $\forall \pi_1 \in C$, $\pi_1 \leq_{incr[f(n),g(n),p(n)]} \pi$.

Some Definitions and Theorems

Definition

Decision problem π is *incr*$[f(n), g(n), p(n)]$-Complete for class C if

1. $\pi \in C$.
2. $\forall \pi_1 \in C$, $\pi_1 \leq_{incr[f(n),g(n),p(n)]} \pi$.

Definition (*incr*-PLTC)

incr-POLYLOGTIME-Complete $=$*incr*$[\log^{k_1} n, \log^{k_2}, n^{k_3}]$-Complete

Some Definitions and Theorems

Definition

Decision problem π is $incr[f(n), g(n), p(n)]$-Complete for class C if

1. $\pi \in C$.
2. $\forall \pi_1 \in C$, $\pi_1 \leq_{incr[f(n),g(n),p(n)]} \pi$.

Definition (*incr*-PLTC)

incr-POLYLOGTIME-Complete $= incr[\log^{k_1} n, \log^{k_2}, n^{k_3}]$-Complete

Theorem

Some Definitions and Theorems

Definition

Decision problem π is $incr[f(n), g(n), p(n)]$-Complete for class C if

1. $\pi \in C$.
2. $\forall \pi_1 \in C$, $\pi_1 \leq_{incr[f(n),g(n),p(n)]} \pi$.

Definition (*incr*-PLTC)

incr-POLYLOGTIME-Complete $= incr[\log^{k_1} n, \log^{k_2}, n^{k_3}]$-Complete

Theorem

- General P-Complete problems are *incr*-PLTC for P.

Some Definitions and Theorems

Definition

Decision problem π is *incr*$[f(n), g(n), p(n)]$-Complete for class C if

1. $\pi \in C$.
2. $\forall \pi_1 \in C$, $\pi_1 \leq_{incr[f(n),g(n),p(n)]} \pi$.

Definition (*incr*-PLTC)

incr-POLYLOGTIME-Complete $=$*incr*$[\log^{k_1} n, \log^{k_2}, n^{k_3}]$-Complete

Theorem

- General P-Complete problems are *incr*-PLTC for P.
- There exist P-Complete problems in *incr*-POLYLOGTIME.

P-Completeness

Definition

P-Complete is a class complete for the class P, under two reductions

P-Completeness

Definition

P-Complete is a class complete for the class P, under two reductions

- Problems difficult to parallelize.
  P-Hard problems in P under NC reductions. P=NC?

P-Completeness

Definition

P-Complete is a class complete for the class P, under two reductions

- Problems difficult to parallelize.
  P-Hard problems in P under NC reductions. P=NC?
- Problems difficult to solve in small space.
  P-Hard problems in P under L reductions. P=L?

P-Completeness

Definition

P-Complete is a class complete for the class P, under two reductions

- Problems difficult to parallelize.
  P-Hard problems in P under NC reductions. P=NC?
- Problems difficult to solve in small space.
  P-Hard problems in P under L reductions. P=L?

Comments

P-Completeness

Definition

P-Complete is a class complete for the class P, under two reductions

- Problems difficult to parallelize.
  P-Hard problems in P under NC reductions. P=NC?
- Problems difficult to solve in small space.
  P-Hard problems in P under L reductions. P=L?

Comments

- L reduction are weaker than NC reductions.

P-Completeness

Definition

P-Complete is a class complete for the class P, under two reductions

- Problems difficult to parallelize.
  P-Hard problems in P under NC reductions. P=NC?
- Problems difficult to solve in small space.
  P-Hard problems in P under L reductions. P=L?

Comments

- L reduction are weaker than NC reductions.
- P-Complete under L $\subseteq$ P-Complete under NC.

P-Completeness

Definition

P-Complete is a class complete for the class P, under two reductions

- Problems difficult to parallelize.
  P-Hard problems in P under NC reductions. P=NC?
- Problems difficult to solve in small space.
  P-Hard problems in P under L reductions. P=L?

Comments

- L reduction are weaker than NC reductions.
- P-Complete under L $\subseteq$ P-Complete under NC.
- We consider L reduction variant.

P-Completeness and *incr*-POLYLOGTIME-Completeness

Circuit Value Problem

Given a circuit in form of a DAG, where each node is either input, output or gate(AND,OR,NOT). Given an assignment of 0 and 1 for each input node, aim is to find value of an output node.

P-Completeness and *incr*-POLYLOGTIME-Completeness

Circuit Value Problem

Given a circuit in form of a DAG, where each node is either input, output or gate(AND,OR,NOT). Given an assignment of 0 and 1 for each input node, aim is to find value of an output node.

Theorem

CVP is P-Complete under logspace reduction for P. *[Lardner 1975]*
For any problem $\pi \in$ P, a circuit whose inputs are the bits of input instance of π and simulates turing machine use to solve problem π.

P-Completeness and *incr*-POLYLOGTIME-Completeness

Circuit Value Problem

Given a circuit in form of a DAG, where each node is either input, output or gate(AND,OR,NOT). Given an assignment of 0 and 1 for each input node, aim is to find value of an output node.

Theorem

CVP is P-Complete under logspace reduction for P. *[Lardner 1975]*
For any problem $\pi \in$ P, a circuit whose inputs are the bits of input instance of π and simulates turing machine use to solve problem π.

Reduction

One bit change in instance of π refers to exactly one bit change in instance of CV, i.e. the corresponding input bit. Done in constant time, so CV is incr-POLYLOGTIME-Compelete for P.

P-Completeness and *incr*-POLYLOGTIME-Completeness

Corollary

If the P-Complete problems as CVP are in *incr*-POLYLOGTIME then all of P are in *incr*-POLYLOGTIME.

P-Completeness and *incr*-POLYLOGTIME-Completeness

Corollary

If the P-Complete problems as CVP are in *incr*-POLYLOGTIME then all of P are in *incr*-POLYLOGTIME.

Comments

P-Completeness and *incr*-POLYLOGTIME-Completeness

Corollary

If the P-Complete problems as CVP are in *incr*-POLYLOGTIME then all of P are in *incr*-POLYLOGTIME.

Comments

- All mentions here are for some given P-Complete problems.

P-Completeness and *incr*-POLYLOGTIME-Completeness

Corollary

If the P-Complete problems as CVP are in *incr*-POLYLOGTIME then all of P are in *incr*-POLYLOGTIME.

Comments

- All mentions here are for some given P-Complete problems.
- Some P-Complete problems are in *incr*-POLYLOGTIME.

P-Completeness and *incr*-POLYLOGTIME-Completeness

Corollary

If the P-Complete problems as CVP are in *incr*-POLYLOGTIME then all of P are in *incr*-POLYLOGTIME.

Comments

- All mentions here are for some given P-Complete problems.
- Some P-Complete problems are in *incr*-POLYLOGTIME.
- However those are not *incr*-POLYLOGTIME-Complete.

P-Completeness and *incr*-POLYLOGTIME-Completeness

Corollary

If the P-Complete problems as CVP are in *incr*-POLYLOGTIME then all of P are in *incr*-POLYLOGTIME.

Comments

- All mentions here are for some given P-Complete problems.
- Some P-Complete problems are in *incr*-POLYLOGTIME.
- However those are not *incr*-POLYLOGTIME-Complete.
- Infact any P-Complete problem can be converted such that

P-Completeness and *incr*-POLYLOGTIME-Completeness

Corollary

If the P-Complete problems as CVP are in *incr*-POLYLOGTIME then all of P are in *incr*-POLYLOGTIME.

Comments

- All mentions here are for some given P-Complete problems.
- Some P-Complete problems are in *incr*-POLYLOGTIME.
- However those are not *incr*-POLYLOGTIME-Complete.
- Infact any P-Complete problem can be converted such that
    - It remains P-Complete.

P-Completeness and *incr*-POLYLOGTIME-Completeness

Corollary

If the P-Complete problems as CVP are in *incr*-POLYLOGTIME then all of P are in *incr*-POLYLOGTIME.

Comments

- All mentions here are for some given P-Complete problems.
- Some P-Complete problems are in *incr*-POLYLOGTIME.
- However those are not *incr*-POLYLOGTIME-Complete.
- Infact any P-Complete problem can be converted such that
  - It remains P-Complete.
  - It becomes *incr*-POLYLOGTIME.

P-Completeness and *incr*-POLYLOGTIME-Completeness

Corollary

If the P-Complete problems as CVP are in *incr*-POLYLOGTIME then all of P are in *incr*-POLYLOGTIME.

Comments

- All mentions here are for some given P-Complete problems.
- Some P-Complete problems are in *incr*-POLYLOGTIME.
- However those are not *incr*-POLYLOGTIME-Complete.
- Infact any P-Complete problem can be converted such that
  - It remains P-Complete.
  - It becomes *incr*-POLYLOGTIME.
  - It is no longer *incr*–POLYLOGTIME-Complete.

P-Complete problems in *incr*-POLYLOGTIME

Construction

P-Complete problems in *incr*-POLYLOGTIME

Construction

- Consider a given P-Complete language L over $\sigma = \{0, 1\}$.

P-Complete problems in *incr*-POLYLOGTIME

Construction

- Consider a given P-Complete language L over $\sigma = \{0, 1\}$.
- Let it be in DTIME$\{n^c\}$ and hence *incr*-TIME$\{n^c\}$.

P-Complete problems in *incr*-POLYLOGTIME

Construction

- Consider a given P-Complete language L over $\sigma = \{0, 1\}$.
- Let it be in DTIME$\{n^c\}$ and hence *incr*-TIME$\{n^c\}$.
- We construct a language $L' = \{w^{|w|} | w \in L\}$.

P-Complete problems in *incr*-POLYLOGTIME

Construction

- Consider a given P-Complete language L over $\sigma = \{0, 1\}$.
- Let it be in DTIME$\{n^c\}$ and hence *incr*-TIME$\{n^c\}$.
- We construct a language $L' = \{w^{|w|} | w \in L\}$.

Comments

P-Complete problems in *incr*-POLYLOGTIME

Construction

- Consider a given P-Complete language L over $\sigma = \{0, 1\}$.
- Let it be in DTIME$\{n^c\}$ and hence *incr*-TIME$\{n^c\}$.
- We construct a language $L' = \{w^{|w|} | w \in L\}$.

Comments

- By construction if $|w| = n$, $w^{|w|} = n^2$.

P-Complete problems in *incr*-POLYLOGTIME

Construction

- Consider a given P-Complete language L over $\sigma = \{0, 1\}$.
- Let it be in DTIME$\{n^c\}$ and hence *incr*-TIME$\{n^c\}$.
- We construct a language $L' = \{w^{|w|} | w \in L\}$.

Comments

- By construction if $|w| = n$, $w^{|w|} = n^2$.
- L is reducible to L' under L and NC reduction.

P-Complete problems in *incr*-POLYLOGTIME

Construction

- Consider a given P-Complete language L over $\sigma = \{0, 1\}$.
- Let it be in DTIME$\{n^c\}$ and hence *incr*-TIME$\{n^c\}$.
- We construct a language $L' = \{w^{|w|} | w \in L\}$.

Comments

- By construction if $|w| = n$, $w^{|w|} = n^2$.
- L is reducible to L' under L and NC reduction.
- L' is solvable in $O(n^c) = O(n'^{c/2})$ and hence is in P.

P-Complete problems in *incr*-POLYLOGTIME

Construction

- Consider a given P-Complete language L over $\sigma = \{0, 1\}$.
- Let it be in DTIME$\{n^c\}$ and hence *incr*-TIME$\{n^c\}$.
- We construct a language $L' = \{w^{|w|} | w \in L\}$.

Comments

- By construction if $|w| = n$, $w^{|w|} = n^2$.
- L is reducible to L' under L and NC reduction.
- L' is solvable in $O(n^c) = O(n'^{c/2})$ and hence is in P.
- Divide Update work in $O(n)$ parts to get *incr*-TIME$\{n^{c-1}\}$.

P-Complete problems in *incr*-POLYLOGTIME

Construction

- Consider a given P-Complete language L over $\sigma = \{0, 1\}$.
- Let it be in DTIME$\{n^c\}$ and hence *incr*-TIME$\{n^c\}$.
- We construct a language $L' = \{w^{|w|} | w \in L\}$.

Comments

- By construction if $|w| = n$, $w^{|w|} = n^2$.
- L is reducible to L' under L and NC reduction.
- L' is solvable in $O(n^c) = O(n'^{c/2})$ and hence is in P.
- Divide Update work in $O(n)$ parts to get *incr*-TIME$\{n^{c-1}\}$.
- Repeated to get P-Complete Problem in *incr*-POLYLOGTIME.

P-Complete problems in *incr*-POLYLOGTIME

Summary of Algorithm

P-Complete problems in *incr*-POLYLOGTIME

Summary of Algorithm

- Let S_L be a subroutine that checks for membership in L.

P-Complete problems in *incr*-POLYLOGTIME

Summary of Algorithm

- Let S_L be a subroutine that checks for membership in L.
- Let the string be divided into equal sized $a_1, a_2, a_3, \ldots, a_n$.

P-Complete problems in *incr*-POLYLOGTIME

Summary of Algorithm

- Let S_L be a subroutine that checks for membership in L.
- Let the string be divided into equal sized $a_1, a_2, a_3, \dots, a_n$.
- Return 0 untill $n/2$ of a_i's are same.

P-Complete problems in *incr*-POLYLOGTIME

Summary of Algorithm

- Let S_L be a subroutine that checks for membership in L.
- Let the string be divided into equal sized $a_1, a_2, a_3, ..., a_n$.
- Return 0 untill $n/2$ of a_i's are same.
- Then start process S_L part wise in each update.

P-Complete problems in *incr*-POLYLOGTIME

Summary of Algorithm

- Let S_L be a subroutine that checks for membership in L.
- Let the string be divided into equal sized $a_1, a_2, a_3, ..., a_n$.
- Return 0 untill $n/2$ of a_i's are same.
- Then start process S_L part wise in each update.
- Which will take atleast $n/2$ steps to form $w^{|w|}$.

P-Complete problems in *incr*-POLYLOGTIME

Summary of Algorithm

- Let S_L be a subroutine that checks for membership in L.
- Let the string be divided into equal sized $a_1, a_2, a_3, ..., a_n$.
- Return 0 untill $n/2$ of a_i's are same.
- Then start process S_L part wise in each update.
- Which will take atlaest $n/2$ steps to form $w^{|w|}$.
- Each update takes $O(n^c/n)$ times.

P-Complete problems in *incr*-POLYLOGTIME

Summary of Algorithm

- Let S_L be a subroutine that checks for membership in L.
- Let the string be divided into equal sized $a_1, a_2, a_3, ..., a_n$.
- Return 0 untill $n/2$ of a_i's are same.
- Then start process S_L part wise in each update.
- Which will take atlaest $n/2$ steps to form $w^{|w|}$.
- Each update takes $O(n^c/n)$ times.
- Only problem is figuring out that $n/2$ of a_i's are same.

P-Complete problems in *incr*-POLYLOGTIME

Data Structure

P-Complete problems in *incr*-POLYLOGTIME

Data Structure

- Divide each a_i into k words of size $\log n$, $a_i^1, \ldots, a_i^k$.

P-Complete problems in *incr*-POLYLOGTIME

Data Structure

- Divide each a_i into k words of size $\log n$, $a_i^1, \ldots, a_i^k$.
- Construct k sets, where S_j has jth word of each a_i.

P-Complete problems in *incr*-POLYLOGTIME

Data Structure

- Divide each a_i into k words of size $\log n$, $a_i^1, \ldots, a_i^k$.
- Construct k sets, where S_j has jth word of each a_i.
- Each set S_j stores two flags f_j and g_j denoting half and full.

P-Complete problems in *incr*-POLYLOGTIME

Data Structure

- Divide each a_i into k words of size $\log n$, $a_i^1, \ldots, a_i^k$.
- Construct k sets, where S_j has jth word of each a_i.
- Each set S_j stores two flags f_j and g_j denoting half and full.
- Each set stores majority word w_j for set that is alteast half.

P-Complete problems in *incr*-POLYLOGTIME

Data Structure

- Divide each a_i into k words of size $\log n$, $a_i^1, \ldots, a_i^k$.
- Construct k sets, where S_j has jth word of each a_i.
- Each set S_j stores two flags f_j and g_j denoting half and full.
- Each set stores majority word w_j for set that is alteast half.

Algorithm

P-Complete problems in *incr*-POLYLOGTIME

Data Structure

- Divide each a_i into k words of size $\log n$, $a_i^1, \ldots, a_i^k$.
- Construct k sets, where S_j has jth word of each a_i.
- Each set S_j stores two flags f_j and g_j denoting half and full.
- Each set stores majority word w_j for set that is atleast half.

Algorithm

- If atleast one g_j is 0 answer 0.

P-Complete problems in *incr*-POLYLOGTIME

Data Structure

- Divide each a_i into k words of size $\log n$, $a_i^1, \ldots, a_i^k$.
- Construct k sets, where S_j has jth word of each a_i.
- Each set S_j stores two flags f_j and g_j denoting half and full.
- Each set stores majority word w_j for set that is atleast half.

Algorithm

- If atleast one g_j is 0 answer 0.
- If and all f_j are 1 and S_L not started, start S_L on $w_1 \ldots w_k$.

P-Complete problems in *incr*-POLYLOGTIME

Data Structure

- Divide each a_i into k words of size $\log n$, $a_i^1, \ldots, a_i^k$.
- Construct k sets, where S_j has jth word of each a_i.
- Each set S_j stores two flags f_j and g_j denoting half and full.
- Each set stores majority word w_j for set that is atleast half.

Algorithm

- If atleast one g_j is 0 answer 0.
- If and all f_j are 1 and S_L not started, start S_L on $w_1 \ldots w_k$.
- If atleast one f_j is 0 stop S_L.

P-Complete problems in *incr*-POLYLOGTIME

Data Structure

- Divide each a_i into k words of size $\log n$, $a_i^1, \ldots, a_i^k$.
- Construct k sets, where S_j has jth word of each a_i.
- Each set S_j stores two flags f_j and g_j denoting half and full.
- Each set stores majority word w_j for set that is alteast half.

Algorithm

- If atleast one g_j is 0 answer 0.
- If and all f_j are 1 and S_L not started, start S_L on $w_1 \ldots w_k$.
- If atleast one f_j is 0 stop S_L.
- If all g_j equal to 1 return answer of S_L.

P-Complete problems in *incr*-POLYLOGTIME

Correctness

P-Complete problems in *incr*-POLYLOGTIME

Correctness

- To prove that S_L is completed on $w_1 w_2 ... w_k$ when all $g_j = 1$.

P-Complete problems in *incr*-POLYLOGTIME

Correctness

- To prove that S_L is completed on $w_1 w_2 ... w_k$ when all $g_j = 1$.
- Starts with all $f_j = 1$, take atleast $n/2$ steps till all $g_j = 1$.

P-Complete problems in *incr*-POLYLOGTIME

Correctness

- To prove that S_L is completed on $w_1 w_2 \ldots w_k$ when all $g_j = 1$.
- Starts with all $f_j = 1$, take atleast $n/2$ steps till all $g_j = 1$.
- Each S_j maintained dynamically using augmented balanced BST T_j.

P-Complete problems in *incr*-POLYLOGTIME

Correctness

- To prove that S_L is completed on $w_1 w_2 \ldots w_k$ when all $g_j = 1$.
- Starts with all $f_j = 1$, take atleast $n/2$ steps till all $g_j = 1$.
- Each S_j maintained dynamically using augmented balanced BST T_j.

Maintaining S_j

P-Complete problems in *incr*-POLYLOGTIME

Correctness

- To prove that S_L is completed on $w_1 w_2 ... w_k$ when all $g_j = 1$.
- Starts with all $f_j = 1$, take atleast $n/2$ steps till all $g_j = 1$.
- Each S_j maintained dynamically using augmented balanced BST T_j.

Maintaining S_j

- Each update acts on only one S_j.

P-Complete problems in *incr*-POLYLOGTIME

Correctness

- To prove that S_L is completed on $w_1 w_2 \ldots w_k$ when all $g_j = 1$.
- Starts with all $f_j = 1$, take atleast $n/2$ steps till all $g_j = 1$.
- Each S_j maintained dynamically using augmented balanced BST T_j.

Maintaining S_j

- Each update acts on only one S_j.
- Each word in S_j ie. $a_j^1, \ldots, a_j^n$ are stored at leaves.

P-Complete problems in *incr*-POLYLOGTIME

Correctness

- To prove that S_L is completed on $w_1 w_2 \ldots w_k$ when all $g_j = 1$.
- Starts with all $f_j = 1$, take atleast $n/2$ steps till all $g_j = 1$.
- Each S_j maintained dynamically using augmented balanced BST T_j.

Maintaining S_j

- Each update acts on only one S_j.
- Each word in S_j ie. $a_j^1, \ldots, a_j^n$ are stored at leaves.
- They are lexicographically sorted.

P-Complete problems in *incr*-POLYLOGTIME

Correctness

- To prove that S_L is completed on $w_1 w_2 ... w_k$ when all $g_j = 1$.
- Starts with all $f_j = 1$, take atleast $n/2$ steps till all $g_j = 1$.
- Each S_j maintained dynamically using augmented balanced BST T_j.

Maintaining S_j

- Each update acts on only one S_j.
- Each word in S_j ie. $a_j^1, ..., a_j^n$ are stored at leaves.
- They are lexicographically sorted.
- An update is performed as deletion followed by insertion of a_j^i.

P-Complete problems in *incr*-POLYLOGTIME

Correctness

- To prove that S_L is completed on $w_1 w_2 ... w_k$ when all $g_j = 1$.
- Starts with all $f_j = 1$, take atleast $n/2$ steps till all $g_j = 1$.
- Each S_j maintained dynamically using augmented balanced BST T_j.

Maintaining S_j

- Each update acts on only one S_j.
- Each word in S_j ie. $a_j^1, ..., a_j^n$ are stored at leaves.
- They are lexicographically sorted.
- An update is performed as deletion followed by insertion of a_j^i.
- Internal node store *max*, *left*,*right* and corresponding words.

P-Complete problems in *incr*-POLYLOGTIME

Correctness

- To prove that S_L is completed on $w_1 w_2 ... w_k$ when all $g_j = 1$.
- Starts with all $f_j = 1$, take atleast $n/2$ steps till all $g_j = 1$.
- Each S_j maintained dynamically using augmented balanced BST T_j.

Maintaining S_j

- Each update acts on only one S_j.
- Each word in S_j ie. $a_j^1, ..., a_j^n$ are stored at leaves.
- They are lexicographically sorted.
- An update is performed as deletion followed by insertion of a_j^i.
- Internal node store *max*, *left*,*right* and corresponding words.
- Root visited to check for f_i or g_i after an update.

P-Complete problems in *incr*-POLYLOGTIME

Comments

P-Complete problems in *incr*-POLYLOGTIME

Comments

- Some P-Complete problems are not *incr*-PLTC.

P-Complete problems in *incr*-POLYLOGTIME

Comments

- Some P-Complete problems are not *incr*-PLTC.
- NC and L reductions do not capture this extensions.

P-Complete problems in *incr*-POLYLOGTIME

Comments

- Some P-Complete problems are not *incr*-PLTC.
- NC and L reductions do not capture this extensions.
- Hence some new more restrictive reduction required.

P-Complete problems in *incr*-POLYLOGTIME

Comments

- Some P-Complete problems are not *incr*-PLTC.
- NC and L reductions do not capture this extensions.
- Hence some new more restrictive reduction required.
- Important to address the redundancy issue.

P-Complete problems in *incr*-POLYLOGTIME

Comments

- Some P-Complete problems are not *incr*-PLTC.
- NC and L reductions do not capture this extensions.
- Hence some new more restrictive reduction required.
- Important to address the redundancy issue.
- Stricter definition of P-Completeness in terms of projections.

Outline

Projection of a function

Definition

A function $f(x_1, x_2, ..., x_n)$ is called a projection of a function $g(y_1, y_2, ..., y_m)$ if

Projection of a function

Definition

A function $f(x_1, x_2, ..., x_n)$ is called a projection of a function $g(y_1, y_2, ..., y_m)$ if

- There is a mapping $\sigma : \{y_1, ..., y_m\} \rightarrow \{x_1, \overline{x_1}, ..., x_n, \overline{x_n}, 0, 1\}$

Projection of a function

Definition

A function $f(x_1, x_2, ..., x_n)$ is called a projection of a function $g(y_1, y_2, ..., y_m)$ if

- There is a mapping $\sigma : \{y_1, ..., y_m\} \rightarrow \{x_1, \overline{x_1}, ..., x_n, \overline{x_n}, 0, 1\}$
- Where $f(x_1, x_2, ..., x_n) = g(\sigma(y_1), \sigma(y_2), ..., \sigma(y_m))$

Projection of a function

Definition

A function $f(x_1, x_2, ..., x_n)$ is called a projection of a function $g(y_1, y_2, ..., y_m)$ if

- There is a mapping $\sigma : \{y_1, ..., y_m\} \rightarrow \{x_1, \overline{x_1}, ..., x_n, \overline{x_n}, 0, 1\}$
- Where $f(x_1, x_2, ..., x_n) = g(\sigma(y_1), \sigma(y_2), ..., \sigma(y_m))$

Comments

Projection of a function

Definition

A function $f(x_1, x_2, ..., x_n)$ is called a projection of a function $g(y_1, y_2, ..., y_m)$ if

- There is a mapping $\sigma : \{y_1, ..., y_m\} \rightarrow \{x_1, \overline{x_1}, ..., x_n, \overline{x_n}, 0, 1\}$
- Where $f(x_1, x_2, ..., x_n) = g(\sigma(y_1), \sigma(y_2), ..., \sigma(y_m))$

Comments

- Used by [Skyum and Valiant 1981] to define reduction.

Projection of a function

Definition

A function $f(x_1, x_2, ..., x_n)$ is called a projection of a function $g(y_1, y_2, ..., y_m)$ if

- There is a mapping $\sigma : \{y_1, ..., y_m\} \rightarrow \{x_1, \overline{x_1}, ..., x_n, \overline{x_n}, 0, 1\}$
- Where $f(x_1, x_2, ..., x_n) = g(\sigma(y_1), \sigma(y_2), ..., \sigma(y_m))$

Comments

- Used by [Skyum and Valiant 1981] to define reduction.
- Even though g is derived from f we get exactly how many bits affected.

Reduction based on Projection

Definition

$\pi_1 \leq_{proj} \pi_2$

π_1 is projection reducible to π_2 if there is a polynomial $p(n)$ and a polynomially computable family of mappings $\sigma = \{\sigma_n\}_{n\geq 1}$

$$\sigma_n : \{y_1, ..., y_{p(n)}\} \rightarrow \{x_1, \overline{x_1}, ..., x_n, \overline{x_n}, 0, 1\}$$

Reduction based on Projection

Definition

$\pi_1 \leq_{proj} \pi_2$

π_1 is projection reducible to π_2 if there is a polynomial $p(n)$ and a polynomially computable family of mappings $\sigma = \{\sigma_n\}_{n\geq 1}$

$$\sigma_n : \{y_1, ..., y_{p(n)}\} \rightarrow \{x_1, \overline{x_1}, ..., x_n, \overline{x_n}, 0, 1\}$$

- n bit instance of π_1 is $x_1, x_2, ..., x_n$.

Reduction based on Projection

Definition

$\pi_1 \leq_{proj} \pi_2$

π_1 is projection reducible to π_2 if there is a polynomial $p(n)$ and a polynomially computable family of mappings $\sigma = \{\sigma_n\}_{n \geq 1}$

$$\sigma_n : \{y_1, \ldots, y_{p(n)}\} \rightarrow \{x_1, \overline{x_1}, \ldots, x_n, \overline{x_n}, 0, 1\}$$

- n bit instance of π_1 is $x_1, x_2, \ldots, x_n$.
- $p(n)$ bit instance of π_2 is $\sigma_n(y_1), \sigma_n(y_2), \ldots, \sigma_n(y_{p(n)})$.

Reduction based on Projection

Definition

$\pi_1 \leq_{proj} \pi_2$

π_1 is projection reducible to π_2 if there is a polynomial $p(n)$ and a polynomially computable family of mappings $\sigma = \{\sigma_n\}_{n \geq 1}$

$$\sigma_n : \{y_1, ..., y_{p(n)}\} \rightarrow \{x_1, \overline{x_1}, ..., x_n, \overline{x_n}, 0, 1\}$$

- n bit instance of π_1 is $x_1, x_2, ..., x_n$.
- $p(n)$ bit instance of π_2 is $\sigma_n(y_1), \sigma_n(y_2), ..., \sigma_n(y_{p(n)})$.
- $\pi_1(X) = 1$ iff $pi_2(\sigma(Y)) = 1$

Reduction based on Projection

Definition

$\pi_1 \leq_{proj} \pi_2$

π_1 is projection reducible to π_2 if there is a polynomial $p(n)$ and a polynomially computable family of mappings $\sigma = \{\sigma_n\}_{n \geq 1}$

$$\sigma_n : \{y_1, ..., y_{p(n)}\} \rightarrow \{x_1, \overline{x_1}, ..., x_n, \overline{x_n}, 0, 1\}$$

- n bit instance of π_1 is $x_1, x_2, ..., x_n$.
- $p(n)$ bit instance of π_2 is $\sigma_n(y_1), \sigma_n(y_2), ..., \sigma_n(y_{p(n)})$.
- $\pi_1(X) = 1$ iff $pi_2(\sigma(Y)) = 1$
- For each y_i the corresponding bit on instance of π_2 is either some constant or one of x_i or $\overline{x_i}$

Reduction based on Projection

Comments

Reduction based on Projection

Comments

- Intuitively π_1 is a projection of π_2.

Reduction based on Projection

Comments

- Intuitively π_1 is a projection of π_2.
- Almost same as Karp-Reduction except gives a notion of bits of I' being directly influenced by bits of I.

Reduction based on Projection

Comments

- Intuitively π_1 is a projection of π_2.
- Almost same as Karp-Reduction except gives a notion of bits of I' being directly influenced by bits of I.

Definition

A problem π is $<_{proj}$ complete for a class C, if

Reduction based on Projection

Comments

- Intuitively π_1 is a projection of π_2.
- Almost same as Karp-Reduction except gives a notion of bits of I' being directly influenced by bits of I.

Definition

A problem π is $<_{proj}$ complete for a class C, if

- π is in C.

Reduction based on Projection

Comments

- Intuitively π_1 is a projection of π_2.
- Almost same as Karp-Reduction except gives a notion of bits of I' being directly influenced by bits of I.

Definition

A problem π is $<_{proj}$ complete for a class C, if

- π is in C.
- There is a function $p(n)$ bounded above by a polynomial in n.

Reduction based on Projection

Comments

- Intuitively π_1 is a projection of π_2.
- Almost same as Karp-Reduction except gives a notion of bits of I' being directly influenced by bits of I.

Definition

A problem π is $<_{proj}$ complete for a class C, if

- π is in C.
- There is a function $p(n)$ bounded above by a polynomial in n.
- $\forall \pi_1 \in$ C, $\pi_1 <_{proj} \pi$ by a projection $\sigma = \{\sigma_n\}_{n \geq 1}$ bounded by polynomial p.

Non-Redundant Projection Completeness

Definition

Let π_1 and π be two decision problems where $\pi_1 \leq_{proj} \pi$
We say π is non-redundant w.r.t. π_1 if there is poly time computable family $\sigma = \{\sigma_n\}_{n\geq 1}$ of mappings and a number $k \in \mathbb{N}$ such that $\forall x_i, |\sigma_n^{-1}(x_i, \overline{x_i})| = O(\log^k n)$

Non-Redundant Projection Completeness

Definition

Let π_1 and π be two decision problems where $\pi_1 \leq_{proj} \pi$
We say π is non-redundant w.r.t. π_1 if there is poly time computable family $\sigma = \{\sigma_n\}_{n \geq 1}$ of mappings and a number $k \in \mathbb{N}$ such that $\forall x_i, |\sigma_n^{-1}(x_i, \overline{x_i})| = O(\log^k n)$

Comments

Non-Redundant Projection Completeness

Definition

Let π_1 and π be two decision problems where $\pi_1 \leq_{proj} \pi$
We say π is non-redundant w.r.t. π_1 if there is poly time computable family $\sigma = \{\sigma_n\}_{n\geq 1}$ of mappings and a number $k \in \mathbb{N}$
such that $\forall x_i, |\sigma_n^{-1}(x_i, \overline{x_i})| = O(\log^k n)$

Comments

- Intuitively how many bits of Y are affected by single bit x_i.

Non-Redundant Projection Completeness

Definition

Let π_1 and π be two decision problems where $\pi_1 \leq_{proj} \pi$
We say π is non-redundant w.r.t. π_1 if there is poly time computable family $\sigma = \{\sigma_n\}_{n \geq 1}$ of mappings and a number $k \in \mathbb{N}$ such that $\forall x_i, |\sigma_n^{-1}(x_i, \overline{x_i})| = O(\log^k n)$

Comments

- Intuitively how many bits of Y are affected by single bit x_i.
- Non-Redundant if bounded by poly logarithmic in n.

Non-Redundant Projection Completeness

Definition

Let π_1 and π be two decision problems where $\pi_1 \leq_{proj} \pi$
We say π is non-redundant w.r.t. π_1 if there is poly time computable family $\sigma = \{\sigma_n\}_{n \geq 1}$ of mappings and a number $k \in \mathbb{N}$ such that $\forall x_i, |\sigma_n^{-1}(x_i, \overline{x_i})| = O(\log^k n)$

Comments

- Intuitively how many bits of Y are affected by single bit x_i.
- Non-Redundant if bounded by poly logarithmic in n.
- All NRP-Complete are *incr*-POLYLOGTIME-Complete.

Non-Redundant Projection Completeness

Definition

Let π_1 and π be two decision problems where $\pi_1 \leq_{proj} \pi$
We say π is non-redundant w.r.t. π_1 if there is poly time computable family $\sigma = \{\sigma_n\}_{n\geq 1}$ of mappings and a number $k \in \mathbb{N}$
such that $\forall x_i, |\sigma_n^{-1}(x_i, \overline{x_i})| = O(\log^k n)$

Comments

- Intuitively how many bits of Y are affected by single bit x_i.
- Non-Redundant if bounded by poly logarithmic in n.
- All NRP-Complete are *incr*-POLYLOGTIME-Complete.
  1. In preprocessing we calculate this projection map.

Non-Redundant Projection Completeness

Definition

Let π_1 and π be two decision problems where $\pi_1 \leq_{proj} \pi$
We say π is non-redundant w.r.t. π_1 if there is poly time computable family $\sigma = \{\sigma_n\}_{n \geq 1}$ of mappings and a number $k \in \mathbb{N}$ such that $\forall x_i, |\sigma_n^{-1}(x_i, \overline{x_i})| = O(\log^k n)$

Comments

- Intuitively how many bits of Y are affected by single bit x_i.
- Non-Redundant if bounded by poly logarithmic in n.
- All NRP-Complete are *incr*-POLYLOGTIME-Complete.
    1. In preprocessing we calculate this projection map.
    2. Hence one bit change can easily be updated using the map.

Outline

For Computation from Scratch

Theorem

$\mathbf{NSPACE}[s(n)] \subseteq \mathbf{DTIME}[k^{\log(n)+s(n)}] = \mathbf{DTIME}[n.2^{s(n)}]$

For Computation from Scratch

Theorem

$\mathbf{NSPACE}[s(n)] \subseteq \mathbf{DTIME}[k^{\log(n)+s(n)}] = \mathbf{DTIME}[n.2^{s(n)}]$

Proof

Given a k string NDTM M with input and output that decides L in space $s(n)$.

For Computation from Scratch

Theorem

$\mathbf{NSPACE}[s(n)] \subseteq \mathbf{DTIME}[k^{\log(n)+s(n)}] = \mathbf{DTIME}[n.2^{s(n)}]$

Proof

Given a k string NDTM M with input and output that decides L in space $s(n)$.

- Configuration depends on <State,I/O Head, Work Tapes, Work Tape Head>

For Computation from Scratch

Theorem

$\textbf{NSPACE}[s(n)] \subseteq \textbf{DTIME}[k^{\log(n)+s(n)}] = \textbf{DTIME}[n.2^{s(n)}]$

Proof

Given a k string NDTM M with input and output that decides L in space $s(n)$.

- Configuration depends on <State,I/O Head, Work Tapes, Work Tape Head>
- Number of configurations $States * (n+1) * \Sigma^{k*s(n)} = O(n.c^{s(n)})$.

For Computation from Scratch

Theorem

$\textbf{NSPACE}[s(n)] \subseteq \textbf{DTIME}[k^{\log(n)+s(n)}] = \textbf{DTIME}[n.2^{s(n)}]$

Proof

Given a k string NDTM M with input and output that decides L in space $s(n)$.

- Configuration depends on <State,I/O Head, Work Tapes, Work Tape Head>
- Number of configurations $States * (n+1) * \Sigma^{k*s(n)} = O(n.c^{s(n)})$.
- Create a configuration graph, $x \in L$ if there is a path to accepting configuration.

For Incremental Computation

Theorem

Given $s(n)$ is computible in $O(n^{O(1)})$ time such that $s(n) = O(\log n)$
$\textbf{NSPACE}[s(n)] \subseteq incr\text{-}\textbf{TIME}[\log n.2^{s(n)}]$

For Incremental Computation

Theorem

Given $s(n)$ is computible in $O(n^{O(1)})$ time such that $s(n) = O(\log n)$ $\mathbf{NSPACE}[s(n)] \subseteq incr\text{-}\mathbf{TIME}[\log n.2^{s(n)}]$

Construction

Consider an NDTM M with read only input tape
$x_0 = \#, x_1, ..., x_n, x_{n+1} = \#$, such that

For Incremental Computation

Theorem

Given $s(n)$ is computible in $O(n^{O(1)})$ time such that $s(n) = O(\log n)$
$\textbf{NSPACE}[s(n)] \subseteq \textit{incr}\text{-}\textbf{TIME}[\log n.2^{s(n)}]$

Construction

Consider an NDTM M with read only input tape
$x_0 = \#, x_1, ..., x_n, x_{n+1} = \#$, such that

- M accepts X only if input head leaves the input tape part and rejects otherwise.

For Incremental Computation

Theorem

Given $s(n)$ is computible in $O(n^{O(1)})$ time such that $s(n) = O(\log n)$
$\textbf{NSPACE}[s(n)] \subseteq incr\text{-}\textbf{TIME}[\log n.2^{s(n)}]$

Construction

Consider an NDTM M with read only input tape
$x_0 = \#, x_1, ..., x_n, x_{n+1} = \#$, such that

- M accepts X only if input head leaves the input tape part and rejects otherwise.
- Semi-Configuration S of M is description excluding input head.

For Incremental Computation

Theorem

Given $s(n)$ is computible in $O(n^{O(1)})$ time such that $s(n) = O(\log n)$
$\mathbf{NSPACE}[s(n)] \subseteq \mathit{incr}\text{-}\mathbf{TIME}[\log n.2^{s(n)}]$

Construction

Consider an NDTM M with read only input tape
$x_0 = \#, x_1, ..., x_n, x_{n+1} = \#$, such that

- M accepts X only if input head leaves the input tape part and rejects otherwise.
- Semi-Configuration S of M is description excluding input head.
- Current configuration thus depends on (S, x_i).

For Incremental Computation

Theorem

Given $s(n)$ is computible in $O(n^{O(1)})$ time such that $s(n) = O(\log n)$
$\mathbf{NSPACE}[s(n)] \subseteq incr\text{-}\mathbf{TIME}[\log n.2^{s(n)}]$

Construction

Consider an NDTM M with read only input tape
$x_0 = \#, x_1, ..., x_n, x_{n+1} = \#$, such that

- M accepts X only if input head leaves the input tape part and rejects otherwise.
- Semi-Configuration S of M is description excluding input head.
- Current configuration thus depends on (S, x_i).
- Consider binary relation of form $R_{i,j} : S \times \{l, r\} \rightarrow S \times \{L, R\}$ $< u, l > R_{ij} < v, R >$: If M enters input tape region $x_i ... x_j$ from left with state u it leaves the region for first time from right with state v.

For Incremental Computation

Proof

Clearly R_{ij} can be recursively defined in terms of R_{ik} and R_{k+1j} by transitive closure

For Incremental Computation

Proof

Clearly R_{ij} can be recursively defined in terms of R_{ik} and R_{k+1j} by transitive closure

- We maintain R in form of binary tree with root R_{0n+1}.

For Incremental Computation

Proof

Clearly R_{ij} can be recursively defined in terms of R_{ik} and R_{k+1j} by transitive closure

- We maintain R in form of binary tree with root R_{0n+1}.
- $R_{i,j}$ has two children R_{ik} and R_{k+1j} where $k = \lfloor \frac{i+j}{2} \rfloor$.

For Incremental Computation

Proof

Clearly R_{ij} can be recursively defined in terms of R_{ik} and R_{k+1j} by transitive closure

- We maintain R in form of binary tree with root R_{0n+1}.
- $R_{i,j}$ has two children R_{ik} and R_{k+1j} where $k = \lfloor \frac{i+j}{2} \rfloor$.
- Thus we have a binary tree of height $O(\log n)$

For Incremental Computation

Proof

Clearly R_{ij} can be recursively defined in terms of R_{ik} and R_{k+1j} by transitive closure

- We maintain R in form of binary tree with root R_{0n+1}.
- $R_{i,j}$ has two children R_{ik} and R_{k+1j} where $k = \lfloor \frac{i+j}{2} \rfloor$.
- Thus we have a binary tree of height $O(\log n)$
- Query can be made at root R_{0n+1}.

For Incremental Computation

Proof

Clearly R_{ij} can be recursively defined in terms of R_{ik} and R_{k+1j} by transitive closure

- We maintain R in form of binary tree with root R_{0n+1}.
- $R_{i,j}$ has two children R_{ik} and R_{k+1j} where $k = \lfloor \frac{i+j}{2} \rfloor$.
- Thus we have a binary tree of height $O(\log n)$
- Query can be made at root R_{0n+1}.
- An update updates exactly $O(\log n)$ nodes.

For Incremental Computation

Proof

Clearly R_{ij} can be recursively defined in terms of R_{ik} and R_{k+1j} by transitive closure

- We maintain R in form of binary tree with root R_{0n+1}.
- $R_{i,j}$ has two children R_{ik} and R_{k+1j} where $k = \lfloor \frac{i+j}{2} \rfloor$.
- Thus we have a binary tree of height $O(\log n)$
- Query can be made at root R_{0n+1}.
- An update updates exactly $O(\log n)$ nodes.
- Each update done by transitive closure on set of size $O(2^{O(s(n))})$.

For Incremental Computation

Proof

Clearly R_{ij} can be recursively defined in terms of R_{ik} and R_{k+1j} by transitive closure

- We maintain R in form of binary tree with root R_{0n+1}.
- $R_{i,j}$ has two children R_{ik} and R_{k+1j} where $k = \lfloor \frac{i+j}{2} \rfloor$.
- Thus we have a binary tree of height $O(\log n)$
- Query can be made at root R_{0n+1}.
- An update updates exactly $O(\log n)$ nodes.
- Each update done by transitive closure on set of size $O(2^{O(s(n))})$.
- Hence total time is $O(\log n 2^{O(s(n))})$.