Multi-threaded Caching Problem

Haifeng XU

Advisors: Frédéric WAGNER Denis TRYSTRAM Guochuan ZHANG

INPG & Zhejiang University

June 5, 2009

Outline

what to talk ...

- A Practical Problem: Hypercarte
- Caching Problem
- Multi-threaded Caching Problem
  - Complexity
  - Algorithms
- Summary

Outline

what to talk ...

- A Practical Problem: Hypercarte
- Caching Problem
- Multi-threaded Caching Problem
  - Complexity
  - Algorithms
- Summary

Outline

what to talk ...

- A Practical Problem: Hypercarte
- Caching Problem
- Multi-threaded Caching Problem
    - Complexity
    - Algorithms
- Summary

Outline

what to talk . . .

- A Practical Problem: Hypercarte
- Caching Problem
- Multi-threaded Caching Problem
  - Complexity
  - Algorithms
- Summary

Outline

what to talk ...

- A Practical Problem: Hypercarte
- Caching Problem
- Multi-threaded Caching Problem
  - Complexity
  - Algorithms
- Summary

Outline

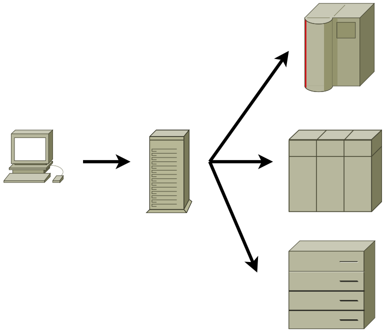
what to talk . . .

- A Practical Problem: Hypercarte
- Caching Problem
- Multi-threaded Caching Problem
  - Complexity
  - Algorithms
- Summary

Architecture of Hypercarte Problem

- client/server architecture
- parallel machines
- parallel tasks

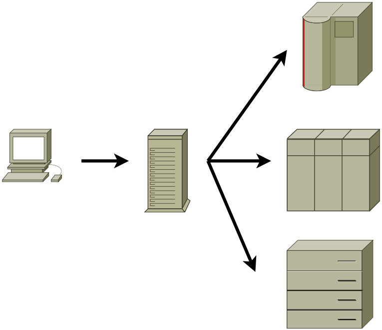
Observation: Some tasks may appear many times.



Architecture of Hypercarte Problem

- client/server architecture
- parallel machines
- parallel tasks

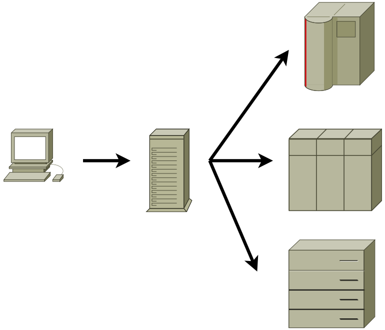
Observation: Some tasks may appear many times.



Architecture of Hypercarte Problem

- client/server architecture
- parallel machines
- parallel tasks

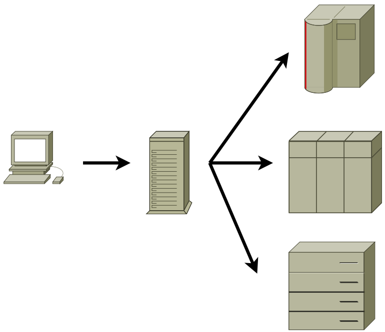
Observation: Some tasks may appear many times.



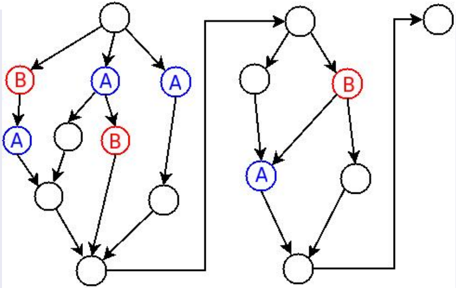
Architecture of Hypercarte Problem

- client/server architecture
- parallel machines
- parallel tasks

Observation: Some tasks may appear many times.

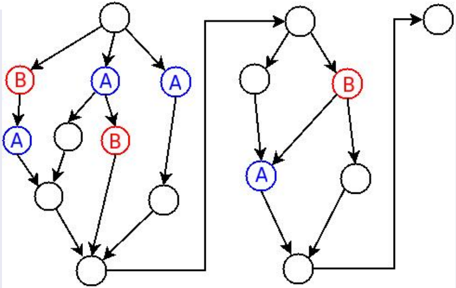


A model of Hypercarte Project: Hypercarte Problem



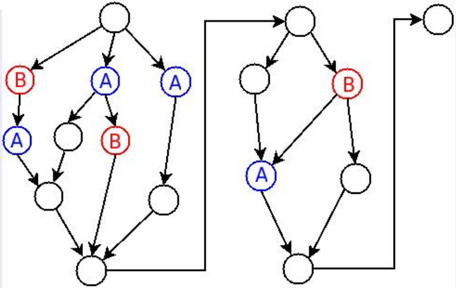
- Requests consist of DAG (Directed Acyclic Graph)
- Some of the requests are same
- m parallel machines
- Objective: C_{max} (Scheduling Problem)
- Store the results of some tasks to improve the performance.

A model of Hypercarte Project: Hypercarte Problem



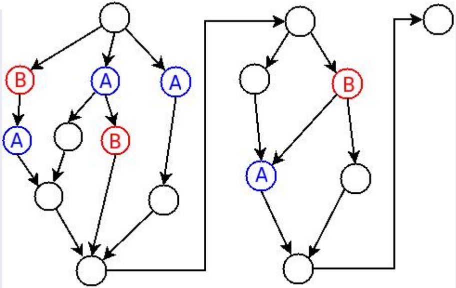
- Requests consist of DAG (Directed Acyclic Graph)
- Some of the requests are same
- m parallel machines
- Objective: C_{max} (Scheduling Problem)
- Store the results of some tasks to improve the performance.

A model of Hypercarte Project: Hypercarte Problem



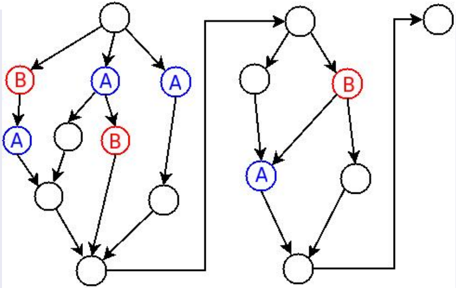
- Requests consist of DAG (Directed Acyclic Graph)
- Some of the requests are same
- m parallel machines
- Objective: C_{max} (Scheduling Problem)
- Store the results of some tasks to improve the performance.

A model of Hypercarte Project: Hypercarte Problem



- Requests consist of DAG (Directed Acyclic Graph)
- Some of the requests are same
- m parallel machines
- Objective: C_{max} (Scheduling Problem)
- Store the results of some tasks to improve the performance.

A model of Hypercarte Project: Hypercarte Problem



- Requests consist of DAG (Directed Acyclic Graph)
- Some of the requests are same
- m parallel machines
- Objective: C_{max} (Scheduling Problem)
- Store the results of some tasks to improve the performance.

Simplification of the Original Problem

We simplify the Hypercarte problem a little bit ...

- DAG
- m machines
- C_{max}
- Cache

- one chain
- one machine
- C_{max}
- Cache

Thus, we get a problem: Caching (Paging) Problem.

Simplification of the Original Problem

We simplify the Hypercarte problem a little bit ...

- DAG
- m machines
- C_{max}
- Cache

- one chain
- one machine
- C_{max}
- Cache

Thus, we get a problem: Caching (Paging) Problem.

Simplification of the Original Problem

We simplify the Hypercarte problem a little bit . . .

- DAG
- m machines
- C_{max}
- Cache

- one chain
- one machine
- C_{max}
- Cache

Thus, we get a problem: Caching (Paging) Problem.

Simplification of the Original Problem

We simplify the Hypercarte problem a little bit . . .

- DAG
- m machines
- C_{max}
- Cache

- one chain
- one machine
- C_{max}
- Cache

Thus, we get a problem: Caching (Paging) Problem.

Outline

what to talk ...

- A Practical Problem: Hypercarte
- Caching Problem
- Multi-threaded Caching Problem
    - Complexity
    - Algorithms
- Summary

Description of Off-line Caching Problem

Input

- A set of tasks
  for task T_i,
  - processing time: p_i
  - size of result: s_i
- One request chain: σ
- A cache of capacity K

$$S_{task} = \{T_1, T_2, \dots, T_L\}$$

$$\sigma : Z^+ \rightarrow \{T_1, \dots, T_L\}$$

$$\sigma_1 \rightarrow \sigma_2 \rightarrow \sigma_3 \rightarrow \sigma_N$$

$$\boxed{\sum_{T_i \in Cache} S_i \leq K}$$

$$\min : \quad \sum_{i=1}^{N} p(\sigma_i) \cdot x(\sigma_i)$$

$$x(\sigma_i) = \begin{cases} 0 & \text{if the task } \sigma_i \text{ is in the cache at the } i_{th} \text{ iteration} \\ 1 & \text{otherwise} \end{cases}$$

Description of Off-line Caching Problem

Input

- A set of tasks
  for task T_i,
  - processing time: p_i
  - size of result: s_i
- One request chain: σ
- A cache of capacity K

$$S_{task} = \{T_1, T_2, \ldots, T_L\}$$

$$\sigma : Z^+ \to \{T_1, \ldots, T_L\}$$

$$\sigma_1 \to \sigma_2 \to \sigma_3 \to \sigma_N$$

$$\boxed{\sum_{T_i \in Cache} S_i \leq K}$$

$$\min : \quad \sum_{i=1}^{N} p(\sigma_i) \cdot x(\sigma_i)$$

$$x(\sigma_i) = \begin{cases} 0 & \text{if the task } \sigma_i \text{ is in the cache at the } i_{th} \text{ iteration} \\ 1 & \text{otherwise} \end{cases}$$

Description of Off-line Caching Problem

Input

- A set of tasks
  for task T_i,
  - processing time: p_i
  - size of result: s_i
- One request chain: σ
- A cache of capacity K

$$S_{task} = \{T_1, T_2, \ldots, T_L\}$$

$$\sigma : Z^+ \to \{T_1, \ldots, T_L\}$$

$$\sigma_1 \to \sigma_2 \to \sigma_3 \to \sigma_N$$

$$\boxed{\sum_{T_i \in Cache} S_i \leq K}$$

$$\min : \quad \sum_{i=1}^{N} p(\sigma_i) \cdot x(\sigma_i)$$

$$x(\sigma_i) = \begin{cases} 0 & \text{if the task } \sigma_i \text{ is in the cache at the } i_{th} \text{ iteration} \\ 1 & \text{otherwise} \end{cases}$$

Description of Off-line Caching Problem

Input

- A set of tasks
  for task T_i,
  - processing time: p_i
  - size of result: s_i
- One request chain: σ
- A cache of capacity K

$S_{task} = \{T_1, T_2, \dots, T_L\}$

$\sigma : Z^+ \rightarrow \{T_1, \dots, T_L\}$

$\sigma_1 \rightarrow \sigma_2 \rightarrow \sigma_3 \rightarrow \sigma_N$

$\boxed{\sum_{T_i \in Cache} S_i \leq K}$

$$\min : \quad \sum_{i=1}^{N} p(\sigma_i) \cdot x(\sigma_i)$$

$$x(\sigma_i) = \begin{cases} 0 & \text{if the task } \sigma_i \text{ is in the cache at the } i_{th} \text{ iteration} \\ 1 & \text{otherwise} \end{cases}$$

Description of Off-line Caching Problem

Input

- A set of tasks
  for task T_i,
  - processing time: p_i
  - size of result: s_i
- One request chain: σ
- A cache of capacity K

$$S_{task} = \{T_1, T_2, \ldots, T_L\}$$

$$\sigma : Z^+ \rightarrow \{T_1, \ldots, T_L\}$$

$$\sigma_1 \rightarrow \sigma_2 \rightarrow \sigma_3 \rightarrow \sigma_N$$

$$\boxed{\sum_{T_i \in Cache} S_i \leq K}$$

$$\min : \quad \sum_{i=1}^{N} p(\sigma_i) \cdot x(\sigma_i)$$

$$x(\sigma_i) = \begin{cases} 0 & \text{if the task } \sigma_i \text{ is in the cache at the } i_{th} \text{ iteration} \\ 1 & \text{otherwise} \end{cases}$$

Description of Off-line Caching Problem

Input

- A set of tasks
  for task T_i,
  - processing time: p_i
  - size of result: s_i
- One request chain: σ
- A cache of capacity K

$$S_{task} = \{T_1, T_2, \ldots, T_L\}$$

$$\sigma : Z^+ \rightarrow \{T_1, \ldots, T_L\}$$

$$\sigma_1 \rightarrow \sigma_2 \rightarrow \sigma_3 \rightarrow \sigma_N$$

$$\boxed{\sum_{T_i \in Cache} S_i \leq K}$$

$$\text{min :} \quad \sum_{i=1}^{N} p(\sigma_i) \cdot x(\sigma_i)$$

$$x(\sigma_i) = \begin{cases} 0 & \text{if the task } \sigma_i \text{ is in the cache at the } i_{th} \text{ iteration} \\ 1 & \text{otherwise} \end{cases}$$

What should we do in the caching problem?

An Example

TASK	SIZE	TIME
A	1	2
B	1	1
C	2	1

We have a cache of capacity 2

$A \rightarrow C \rightarrow B \rightarrow C \rightarrow A \rightarrow C \rightarrow B$

To Do . . .

Which task should should be removed from the cache if the cache is full.

What should we do in the caching problem?

An Example

TASK	SIZE	TIME
A	1	2
B	1	1
C	2	1

We have a cache of capacity 2

$$A \to C \to B \to C \to A \to C \to B$$

To Do ...

Which task should should be removed from the cache if the cache is full.

What should we do in the caching problem?

An Example

TASK	SIZE	TIME
A	1	2
B	1	1
C	2	1

We have a cache of capacity 2

$$A \to C \to B \to C \to A \to C \to B$$

To Do ...

Which task should should be removed from the cache if the cache is full.

Previous Results of Caching Problem

Offline

The complexity depends on *the size of results* and *the processing time*.

	SIZE	TIME	Complexity
Uniform Model	1	1	P
Cost Model	1	$\mathbb{Z}^+$	P
Fault Model	$\mathbb{Z}^+$	1	?
General Model	$\mathbb{Z}^+$	$\mathbb{Z}^+$	NP-hard

- There is a 4-approximation algorithm for the general model (c.f. [Amotz Bar-Noy et al. 1991]).

online

- for the general model
- $(\frac{K}{size_{min}} + 1)$ - competitive deterministic online algorithm
- best

Previous Results of Caching Problem

Offline

The complexity depends on *the size of results* and *the processing time*.

	SIZE	TIME	Complexity
Uniform Model	1	1	P
Cost Model	1	$\mathbb{Z}^+$	P
Fault Model	$\mathbb{Z}^+$	1	?
General Model	$\mathbb{Z}^+$	$\mathbb{Z}^+$	NP-hard

- There is a 4-approximation algorithm for the general model (c.f. `[Amotz Bar-Noy et al. 1991]`).

online

- for the general model
- $\left(\frac{K}{size_{min}} + 1\right)$ - competitive deterministic online algorithm
- best

Previous Results of Caching Problem

Offline

The complexity depends on *the size of results* and *the processing time*.

	SIZE	TIME	Complexity
Uniform Model	1	1	P
Cost Model	1	$\mathbb{Z}^+$	P
Fault Model	$\mathbb{Z}^+$	1	?
General Model	$\mathbb{Z}^+$	$\mathbb{Z}^+$	NP-hard

- There is a 4-approximation algorithm for the general model (c.f. `[Amotz Bar-Noy et al. 1991]`).

online

- for the general model
- $(\frac{K}{size_{min}} + 1)$ - competitive deterministic online algorithm
- best

Extend Caching Problem

We extend caching problem a bit, because it is a little far away from our original model.

- **DAG**
- m machines
- C_{max}
- Cache

- **One** chain
- one machine
- C_{max}
- Cache

- **Several** Chains
- one machine
- C_{max}
- Cache

Having more than one request chain . . .

Extend Caching Problem

We extend caching problem a bit, because it is a little far away from our original model.

- **DAG**
- m machines
- C_{max}
- Cache

- **One** chain
- one machine
- C_{max}
- Cache

- **Several** Chains
- one machine
- C_{max}
- Cache

Having more than one request chain ...

Extend Caching Problem

We extend caching problem a bit, because it is a little far away from our original model.

- **DAG**
- m machines
- C_{max}
- Cache

- **One** chain
- one machine
- C_{max}
- Cache

- **Several** Chains
- one machine
- C_{max}
- Cache

Having more than one request chain . . .

Extend Caching Problem

We extend caching problem a bit, because it is a little far away from our original model.

- **DAG**
- m machines
- C_{max}
- Cache

- **One** chain
- one machine
- C_{max}
- Cache

- **Several** Chains
- one machine
- C_{max}
- Cache

Having more than one request chain ...

Extend Caching Problem

We extend caching problem a bit, because it is a little far away from our original model.

• **DAG**	• **One** chain	• **Several** Chains
• m machines	• one machine	• one machine
• C_{max}	• C_{max}	• C_{max}
• Cache	• Cache	• Cache

Having more than one request chain ...

Outline

what to talk ...

- A Practical Problem: Hypercarte
- Caching Problem
- Multi-threaded Caching Problem
  - Complexity
  - Algorithms
- Summary

Description of Multi-threaded Caching Problem

Input

- A set of tasks
  for task T_i,
  - processing time: p_i
  - size of result: s_i
- A set of request chains: σ^i $(1 \leq i \leq Q)$
- A cache of capacity K

$\sigma^i : Z^+ \rightarrow \{T_1, \ldots, T_L\}$

$\sigma_1^1 \rightarrow \sigma_2^1 \rightarrow \sigma_3^1 \longrightarrow \sigma_{N_1}^1$

$\ldots$

$\sigma_1^Q \rightarrow \sigma_2^Q \rightarrow \sigma_3^Q \longrightarrow \sigma_{N_Q}^Q$

$\boxed{\sum_{T_i \in Cache} S_i \leq K}$

To do ...

- Which chain should be served at each iteration?
- Which task should be removed from the cache?

Description of Multi-threaded Caching Problem

Input

- A set of tasks
  for task T_i,
  - processing time: p_i
  - size of result: s_i
- A set of request chains: σ^i $(1 \leq i \leq Q)$
- A cache of capacity K

$$\sigma^i : Z^+ \rightarrow \{T_1, \ldots, T_L\}$$

$$\sigma_1^1 \rightarrow \sigma_2^1 \rightarrow \sigma_3^1 \longrightarrow \sigma_{N_1}^1$$

$$\cdots$$

$$\sigma_1^Q \rightarrow \sigma_2^Q \rightarrow \sigma_3^Q \longrightarrow \sigma_{N_Q}^Q$$

$$\boxed{\sum_{T_i \in Cache} S_i \leq K}$$

To do . . .

- Which chain should be served at each iteration?
- Which task should be removed from the cache?

Description of Multi-threaded Caching Problem

Input

- A set of tasks
  for task T_i,
  - processing time: p_i
  - size of result: s_i
- A set of request chains: σ^i $(1 \leq i \leq Q)$
- A cache of capacity K

$\sigma^i : Z^+ \longrightarrow \{T_1, \ldots, T_L\}$

$\sigma_1^1 \rightarrow \sigma_2^1 \rightarrow \sigma_3^1 \longrightarrow \sigma_{N_1}^1$

$\cdots$

$\sigma_1^Q \rightarrow \sigma_2^Q \rightarrow \sigma_3^Q \longrightarrow \sigma_{N_Q}^Q$

$\boxed{\sum_{T_i \in Cache} S_i \leq K}$

To do ...

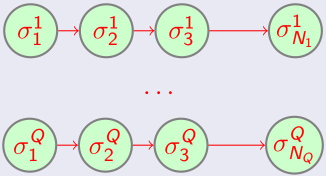
- Which chain should be served at each iteration?
- Which task should be removed from the cache?

Description of Multi-threaded Caching Problem

Input

- A set of tasks
  for task T_i,
  - processing time: p_i
  - size of result: s_i
- A set of request chains:
  σ^i $(1 \leq i \leq Q)$
- A cache of capacity K

$\sigma^i : Z^+ \rightarrow \{T_1, \ldots, T_L\}$



$\boxed{\sum_{T_i \in Cache} S_i \leq K}$

To do ...

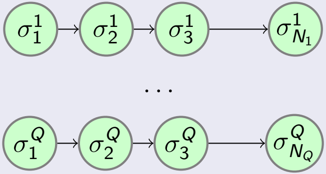
- Which chain should be served at each iteration?
- Which task should be removed from the cache?

Description of Multi-threaded Caching Problem

Input

- A set of tasks
  for task T_i,
  - processing time: p_i
  - size of result: s_i
- A set of request chains: σ^i $(1 \leq i \leq Q)$
- A cache of capacity K

$$\sigma^i : Z^+ \rightarrow \{T_1, \ldots, T_L\}$$



$$\boxed{\sum_{T_i \in Cache} S_i \leq K}$$

To do ...

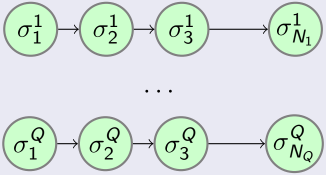
- Which chain should be served at each iteration?
- Which task should be removed from the cache?

Description of Multi-threaded Caching Problem

Input

- A set of tasks
  for task T_i,
  - processing time: p_i
  - size of result: s_i
- A set of request chains: σ^i $(1 \leq i \leq Q)$
- A cache of capacity K

$$\sigma^i : Z^+ \rightarrow \{T_1, \ldots, T_L\}$$



$$\boxed{\sum_{T_i \in Cache} S_i \leq K}$$

To do ...

- Which chain should be served at each iteration?
- Which task should be removed from the cache?

Description of Multi-threaded Caching Problem

Input

- A set of tasks
  for task T_i,
  - processing time: p_i
  - size of result: s_i
- A set of request chains: σ^i $(1 \leq i \leq Q)$
- A cache of capacity K

$$\sigma^i : Z^+ \to \{T_1, \ldots, T_L\}$$

$$\sigma_1^1 \to \sigma_2^1 \to \sigma_3^1 \longrightarrow \sigma_{N_1}^1$$

$$\ldots$$

$$\sigma_1^Q \to \sigma_2^Q \to \sigma_3^Q \longrightarrow \sigma_{N_Q}^Q$$

$$\boxed{\sum_{T_i \in Cache} S_i \leq K}$$

To do ...

- Which chain should be served at each iteration?
- Which task should be removed from the cache?

Previous Results for Multi-threaded Caching Problem

Offline

As far as we know, there is no result about it.

Online

[Feuerstein 1996] showed:

- In the uniform model, for each task t_i
  - size of result: $s_i = 1$
  - processing time: $p_i = 1$
- KQ-competitive deterministic online algorithm
- the universal lower bound is $(K + 1 - \frac{1}{Q})$

Previous Results for Multi-threaded Caching Problem

Offline

As far as we know, there is no result about it.

Online

[Feuerstein 1996] showed:

- In the uniform model, for each task t_i
  - size of result: $s_i = 1$
  - processing time: $p_i = 1$
- KQ-competitive deterministic online algorithm
- the universal lower bound is $(K + 1 - \frac{1}{Q})$

Outline

what to talk . . .

- A Practical Problem: Hypercarte
- Caching Problem
- Multi-threaded Caching Problem
    - Complexity
    - Algorithms
- Summary

Complexity

- $K = 1$, $Q \in \mathbb{Z}^+$, `uniform model`
  This special case is NP-hard, we can get a reduction from the *shortest common supersequence* problem.

Shortest Common Supersequence

Definition:

Given two sequences $w = w_1 \cdots w_m$ and $x = x_1 \cdots x_n$, we say that w is a supersequence of x, or x is a subsequence of w, if we can get x by deleting some symbols from w.

A	B	C	D	E

C	C	E	B

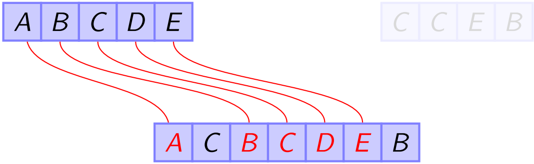
A	C	B	C	D	E	B

We say a sequence is perfect if no consecutive symbols are same.

Shortest Common Supersequence

Definition:

Given two sequences $w = w_1 \cdots w_m$ and $x = x_1 \cdots x_n$, we say that w is a supersequence of x, or x is a subsequence of w, if we can get x by deleting some symbols from w.

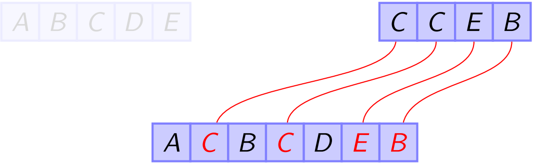


We say a sequence is perfect if no consecutive symbols are same.

Shortest Common Supersequence

Definition:

Given two sequences $w = w_1 \cdots w_m$ and $x = x_1 \cdots x_n$, we say that w is a supersequence of x, or x is a subsequence of w, if we can get x by deleting some symbols from w.



We say a sequence is perfect if no consecutive symbols are same.

Shortest Common Supersequence

Definition:

Given two sequences $w = w_1 \cdots w_m$ and $x = x_1 \cdots x_n$, we say that w is a supersequence of x, or x is a subsequence of w, if we can get x by deleting some symbols from w.

A	B	C	D	E

C	C	E	B

A	C	B	C	D	E	B

We say a sequence is perfect if no consecutive symbols are same.

Shortest Common Supersequence

Input: Finite alphabet $\mathbb{A}$, finite set $\mathbb{X} = \{x \colon x \in \mathbb{A}^*\}$ and a positive integer M.

Output: A sequence $w \in \mathbb{A}^*$ with $|w| \leq M$, such that w is a supersequence of x, $\forall x \in \mathbb{X}$.

Previous results

- SCS is *NP-complete problem* `[Maier 1978]`
- SCS(2,3) means each sequence is of length 2, and each symbol appears at most 3 times in the whole sequences.
  - SCS(2,3) is *MAX SNP-hard* `[Timkovskii 1989]`
- PTAS for a *MAX SNP-hard* problem $\Rightarrow$ $P = NP$ `[Arora et al.1992]`
- Perfect SCS means every sequence in $\mathbb{X}$ is perfect.
  - Constant approximation algo. for perfect SCS $\Rightarrow$ PTAS for SCS(2,3) `[Tao Jiang, Ming Li 1991]`

Shortest Common Supersequence

Input: Finite alphabet $\mathbb{A}$, finite set $\mathbb{X} = \{x : x \in \mathbb{A}^*\}$ and a positive integer M.

Output: A sequence $w \in \mathbb{A}^*$ with $|w| \leq M$, such that w is a supersequence of x, $\forall x \in \mathbb{X}$.

Previous results

- SCS is *NP-complete problem* [Maier 1978]
- SCS(2,3) means each sequence is of length 2, and each symbol appears at most 3 times in the whole sequences.
  - SCS(2,3) is *MAX SNP-hard* [Timkovskii 1989]
- PTAS for a *MAX SNP-hard* problem $\Rightarrow$ $P = NP$ [Arora et al.1992]
- Perfect SCS means every sequence in $\mathbb{X}$ is perfect.
  - Constant approximation algo. for perfect SCS $\Rightarrow$ PTAS for SCS(2,3) [Tao Jiang, Ming Li 1991]

Shortest Common Supersequence

Input: Finite alphabet $\mathbb{A}$, finite set $\mathbb{X} = \{x : x \in \mathbb{A}^*\}$ and a positive integer M.

Output: A sequence $w \in \mathbb{A}^*$ with $|w| \leq M$, such that w is a supersequence of x, $\forall x \in \mathbb{X}$.

Previous results

- SCS is *NP-complete problem* [Maier 1978]
- SCS(2,3) means each sequence is of length 2, and each symbol appears at most 3 times in the whole sequences.
  - SCS(2,3) is *MAX SNP-hard* [Timkovskii 1989]
- PTAS for a *MAX SNP-hard* problem $\Rightarrow$ $P = NP$ [Arora et al.1992]
- Perfect SCS means every sequence in $\mathbb{X}$ is perfect.
  - Constant approximation algo. for perfect SCS $\Rightarrow$ PTAS for SCS(2,3) [Tao Jiang, Ming Li 1991]

Shortest Common Supersequence

Input: Finite alphabet $\mathbb{A}$, finite set $\mathbb{X} = \{x : x \in \mathbb{A}^*\}$ and a positive integer M.

Output: A sequence $w \in \mathbb{A}^*$ with $|w| \le M$, such that w is a supersequence of x, $\forall x \in \mathbb{X}$.

Previous results

- SCS is *NP-complete problem* [Maier 1978]
- SCS(2,3) means each sequence is of length 2, and each symbol appears at most 3 times in the whole sequences.
  - SCS(2,3) is *MAX SNP-hard* [Timkovskii 1989]
- PTAS for a *MAX SNP-hard* problem $\Rightarrow$ $P = NP$ [Arora et al.1992]
- Perfect SCS means every sequence in $\mathbb{X}$ is perfect.
  - Constant approximation algo. for perfect SCS $\Rightarrow$ PTAS for SCS(2,3) [Tao Jiang, Ming Li 1991]

Shortest Common Supersequence

Input: Finite alphabet $\mathbb{A}$, finite set $\mathbb{X} = \{x \colon x \in \mathbb{A}^*\}$ and a positive integer M.

Output: A sequence $w \in \mathbb{A}^*$ with $|w| \leq M$, such that w is a supersequence of x, $\forall x \in \mathbb{X}$.

Previous results

- SCS is *NP-complete problem* `[Maier 1978]`
- SCS(2,3) means each sequence is of length 2, and each symbol appears at most 3 times in the whole sequences.
    - SCS(2,3) is *MAX SNP-hard* `[Timkovskii 1989]`
- PTAS for a *MAX SNP-hard* problem $\Rightarrow$ $P = NP$ `[Arora et al.1992]`
- Perfect SCS means every sequence in $\mathbb{X}$ is perfect.
    - Constant approximation algo. for perfect SCS $\Rightarrow$ PTAS for SCS(2,3) `[Tao Jiang, Ming Li 1991]`

Shortest Common Supersequence

Input: Finite alphabet $\mathbb{A}$, finite set $\mathbb{X} = \{x : x \in \mathbb{A}^*\}$ and a positive integer M.

Output: A sequence $w \in \mathbb{A}^*$ with $|w| \leq M$, such that w is a supersequence of x, $\forall x \in \mathbb{X}$.

Previous results

- SCS is *NP-complete problem* [Maier 1978]
- SCS(2,3) means each sequence is of length 2, and each symbol appears at most 3 times in the whole sequences.
  - SCS(2,3) is *MAX SNP-hard* [Timkovskii 1989]
- PTAS for a *MAX SNP-hard* problem $\Rightarrow$ $P = NP$ [Arora et al.1992]
- Perfect SCS means every sequence in $\mathbb{X}$ is perfect.
  - Constant approximation algo. for perfect SCS $\Rightarrow$ PTAS for SCS(2,3) [Tao Jiang, Ming Li 1991]

Shortest Common Supersequence

Input: Finite alphabet $\mathbb{A}$, finite set $\mathbb{X} = \{x : x \in \mathbb{A}^*\}$ and a positive integer M.

Output: A sequence $w \in \mathbb{A}^*$ with $|w| \leq M$, such that w is a supersequence of x, $\forall x \in \mathbb{X}$.

Previous results

- SCS is *NP-complete problem* [Maier 1978]
- SCS(2,3) means each sequence is of length 2, and each symbol appears at most 3 times in the whole sequences.
  - SCS(2,3) is *MAX SNP-hard* [Timkovskii 1989]
- PTAS for a *MAX SNP-hard* problem $\Rightarrow$ $P = NP$ [Arora et al.1992]
- Perfect SCS means every sequence in $\mathbb{X}$ is perfect.
  - Constant approximation algo. for perfect SCS $\Rightarrow$ PTAS for SCS(2,3) [Tao Jiang, Ming Li 1991]

Non-approximability of perfect SCS

SCS(2,3)		perfect SCS
$\mathbb{A}$	$\rightsquigarrow$	$\mathbb{A} \cup \{y^i_j\}$ $(1 \le i \le n, 1 \le j \le 2)$
$\mathbb{X} = \{\ x^1_1 x^1_2\ ,\ \ldots\ ,\ x^n_1 x^n_2\ \}$	$\rightsquigarrow$	$\mathbb{X}' = \{\ x^1_1 y^1_1 x^1_2 y^1_2\ ,\ \ldots,\ x^n_1 y^n_1 x^n_2 y^n_2\ \}$

- y^i_j is unique, which means $y^i_j = y^{i'}_{j'} \Leftrightarrow i = i'$ and $j = j'$
- Const. appro. algo. for $\mathbb{X}' \Rightarrow$ PTAS for $\mathbb{X}$

Complexity of MTC

Theorem

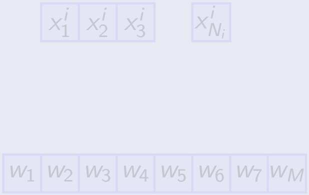
Multi-threaded caching problem is NP-hard for the uniform model even if the cache capacity $K = 1$, and it assumes no constant approximation algorithm unless $P = NP$.

Reduction

Perfect SCS		MTC
$\mathbb{A}, \mathbb{X} = \{x : x \in \mathbb{A}^*\}$	$\rightsquigarrow$	$S_{task} = \mathbb{A}$, $\sigma = \mathbb{X}$
A common sequence $\lvert w \rvert = M$	$\Longleftrightarrow$	A schedule with processing time M

Complexity of MTC

Perfect SCS



Multi-threaded Caching

Proof

Claim: The common supersequence of $\mathbb{X}$ is a feasible schedule of σ and vice versa.

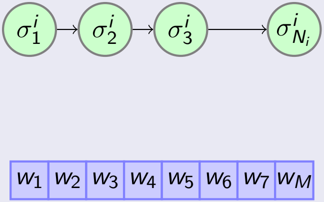
($\Rightarrow$) Let $w = w_1 \cdots w_M$ be a common supersequence of $\mathbb{X}$ with $|w| = M$, then sequence $x^i \in \mathbb{X}$ is a subsequence of w. In other words, we can schedule the request chain σ^i due to the precedence constraint.

($\Leftarrow$) Let w be a feasible schedule of σ with $|w| = M$, then each chain σ^i is a subsequence of w.

Complexity of MTC

Perfect SCS

Multi-threaded Caching



Proof

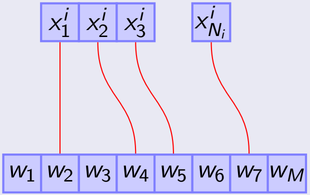
Claim: The common supersequence of $\mathbb{X}$ is a feasible schedule of σ and vice versa.

($\Rightarrow$) Let $w = w_1 \cdots w_M$ be a common supersequence of $\mathbb{X}$ with $|w| = M$, then sequence $x^i \in \mathbb{X}$ is a subsequence of w. In other words, we can schedule the request chain σ^i due to the precedence constraint.

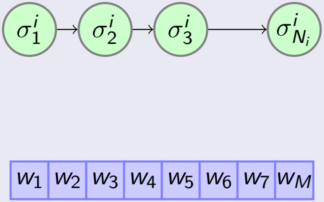
($\Leftarrow$) Let w be a feasible schedule of σ with $|w| = M$, then each chain σ^i is a subsequence of w.

Complexity of MTC

Perfect SCS



Multi-threaded Caching



Proof

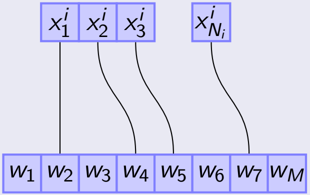
Claim: The common supersequence of $\mathbb{X}$ is a feasible schedule of σ and vice versa.

($\Rightarrow$) Let $w = w_1 \cdots w_M$ be a common supersequence of $\mathbb{X}$ with $|w| = M$, then sequence $x^i \in \mathbb{X}$ is a subsequence of w. In other words, we can schedule the request chain σ^i due to the precedence constraint.

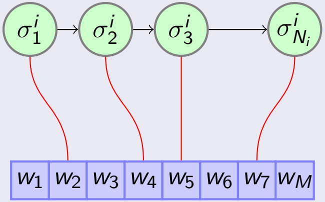
($\Leftarrow$) Let w be a feasible schedule of σ with $|w| = M$, then each chain σ^i is a subsequence of w.

Complexity of MTC

Perfect SCS



Multi-threaded Caching



Proof

Claim: The common supersequence of $\mathbb{X}$ is a feasible schedule of σ and vice versa.

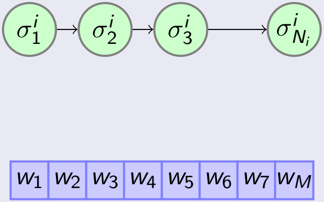
($\Rightarrow$) Let $w = w_1 \cdots w_M$ be a common supersequence of $\mathbb{X}$ with $|w| = M$, then sequence $x^i \in \mathbb{X}$ is a subsequence of w. In other words, we can schedule the request chain σ^i due to the precedence constraint.

($\Leftarrow$) Let w be a feasible schedule of σ with $|w| = M$, then each chain σ^i is a subsequence of w.

Complexity of MTC

Perfect SCS

Multi-threaded Caching



Proof

Claim: The common supersequence of $\mathbb{X}$ is a feasible schedule of σ and vice versa.

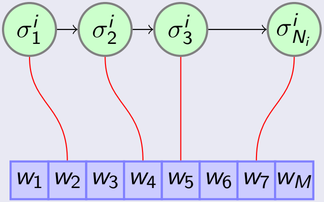
($\Rightarrow$) Let $w = w_1 \cdots w_M$ be a common supersequence of $\mathbb{X}$ with $|w| = M$, then sequence $x^i \in \mathbb{X}$ is a subsequence of w. In other words, we can schedule the request chain σ^i due to the precedence constraint.

($\Leftarrow$) Let w be a feasible schedule of σ with $|w| = M$, then each chain σ^i is a subsequence of w.

Complexity of MTC

Perfect SCS

Multi-threaded Caching



Proof

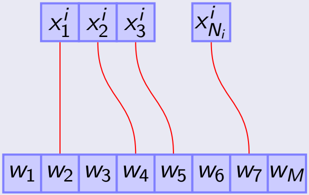
Claim: The common supersequence of $\mathbb{X}$ is a feasible schedule of σ and vice versa.

($\Rightarrow$) Let $w = w_1 \cdots w_M$ be a common supersequence of $\mathbb{X}$ with $|w| = M$, then sequence $x^i \in \mathbb{X}$ is a subsequence of w. In other words, we can schedule the request chain σ^i due to the precedence constraint.

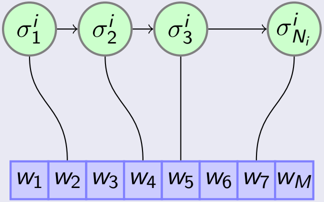
($\Leftarrow$) Let w be a feasible schedule of σ with $|w| = M$, then each chain σ^i is a subsequence of w.

Complexity of MTC

Perfect SCS



Multi-threaded Caching



Proof

Claim: The common supersequence of $\mathbb{X}$ is a feasible schedule of σ and vice versa.

($\Rightarrow$) Let $w = w_1 \cdots w_M$ be a common supersequence of $\mathbb{X}$ with $|w| = M$, then sequence $x^i \in \mathbb{X}$ is a subsequence of w. In other words, we can schedule the request chain σ^i due to the precedence constraint.

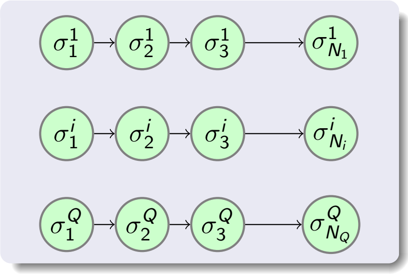
($\Leftarrow$) Let w be a feasible schedule of σ with $|w| = M$, then each chain σ^i is a subsequence of w.

Outline

what to talk ...

- A Practical Problem: Hypercarte
- Caching Problem
- Multi-threaded Caching Problem
  - Complexity
  - Algorithms
- Summary

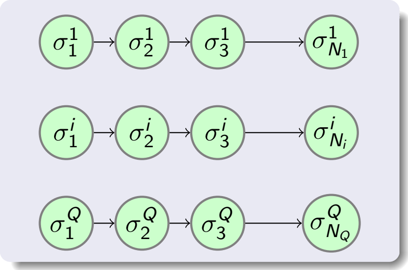
A straightforward approach for the cost model



Let OPT_i be the minimum processing time for chain i, and C_{max} be the minimum processing time for all the chains.

$$OPT_i \leq C_{max} \quad (1 \leq i \leq Q) \qquad \Rightarrow \qquad \frac{\sum_{i=1}^{Q} OPT_i}{C_{max}} \leq Q$$

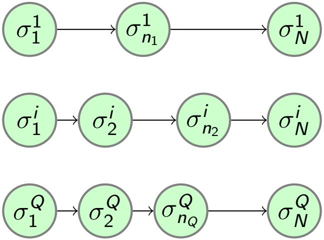
A straightforward approach for the cost model



Let OPT_i be the minimum processing time for chain i, and C_{max} be the minimum processing time for all the chains.

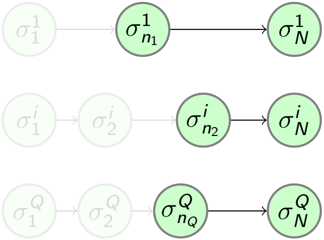
$$OPT_i \leq C_{max} \quad (1 \leq i \leq Q) \qquad \Rightarrow \qquad \frac{\sum_{i=1}^{Q} OPT_i}{C_{max}} \leq Q$$

Dynamic Programming



Obejective function: $f(n_1, \ldots, n_Q \mid cache)$

Dynamic Programming



Obejective function: $f(n_1, \ldots, n_Q \mid cache)$

Dynamic Programming

Find C_{max} with running time:

$$O(\ Q \times L \times N^Q \times \binom{L}{K}\ ) = O(\ Q \times L \times N^Q \times \binom{L}{L-K}\ )$$

Remarks:

- K, Q are constant $\Rightarrow$ P problem even for general model
- $L - K$, Q are constant $\Rightarrow$ P problem even for general model

Dynamic Programming

Find C_{max} with running time:

$$O(\ Q \times L \times N^{Q} \times \binom{L}{K}\ ) = O(\ Q \times L \times N^{Q} \times \binom{L}{L-K}\ )$$

Remarks:

- K, Q are constant $\Rightarrow$ P problem even for general model
- $L-K$, Q are constant $\Rightarrow$ P problem even for general model

Dynamic Programming

Find C_{max} with running time:

$$O(\ Q \times L \times N^Q \times \binom{L}{K}\ ) = O(\ Q \times L \times N^Q \times \binom{L}{L-K}\ )$$

Remarks:

- K, Q are constant $\Rightarrow$ P problem even for general model
- $L-K$, Q are constant $\Rightarrow$ P problem even for general model

Outline

what to talk . . .

- A Practical Problem: Hypercarte
- Caching Problem
- Multi-threaded Caching Problem
    - Complexity
    - Algorithms
- Summary

Our contributions

- *Multi-threaded caching* problem is NP-hard for the uniform model even if $K = 1$
- There is no constant approximation algorithm for it unless $P = NP$.
- Q-approximation algorithm for the cost model
- If both K (or $L - K$) and Q are constant, it becomes P problem even for the general model

Open problems

- The complexity of fault model when $Q = 1$
- Algorithm for uniform model when $Q(\geq 2)$ is constant
- A better lower/upper bound for the online case

Our contributions

- *Multi-threaded caching* problem is NP-hard for the uniform model even if $K = 1$
- There is no constant approximation algorithm for it unless $P = NP$.
- Q-approximation algorithm for the cost model
- If both K (or $L - K$) and Q are constant, it becomes P problem even for the general model

Open problems

- The complexity of fault model when $Q = 1$
- Algorithm for uniform model when $Q(\geq 2)$ is constant
- A better lower/upper bound for the online case

Thank you!