Hiding Object Sizes with Constrained Padding in Multiple Settings

Andrew C. Reed

PhD Defense

Computer Science
UNC-Chapel Hill

March 5, 2025

<u>Committee</u>
Michael K. Reiter, Advisor
Jay Aikat
Jasleen Kaur
Ketan Mayer-Patel
Fabian Monrose

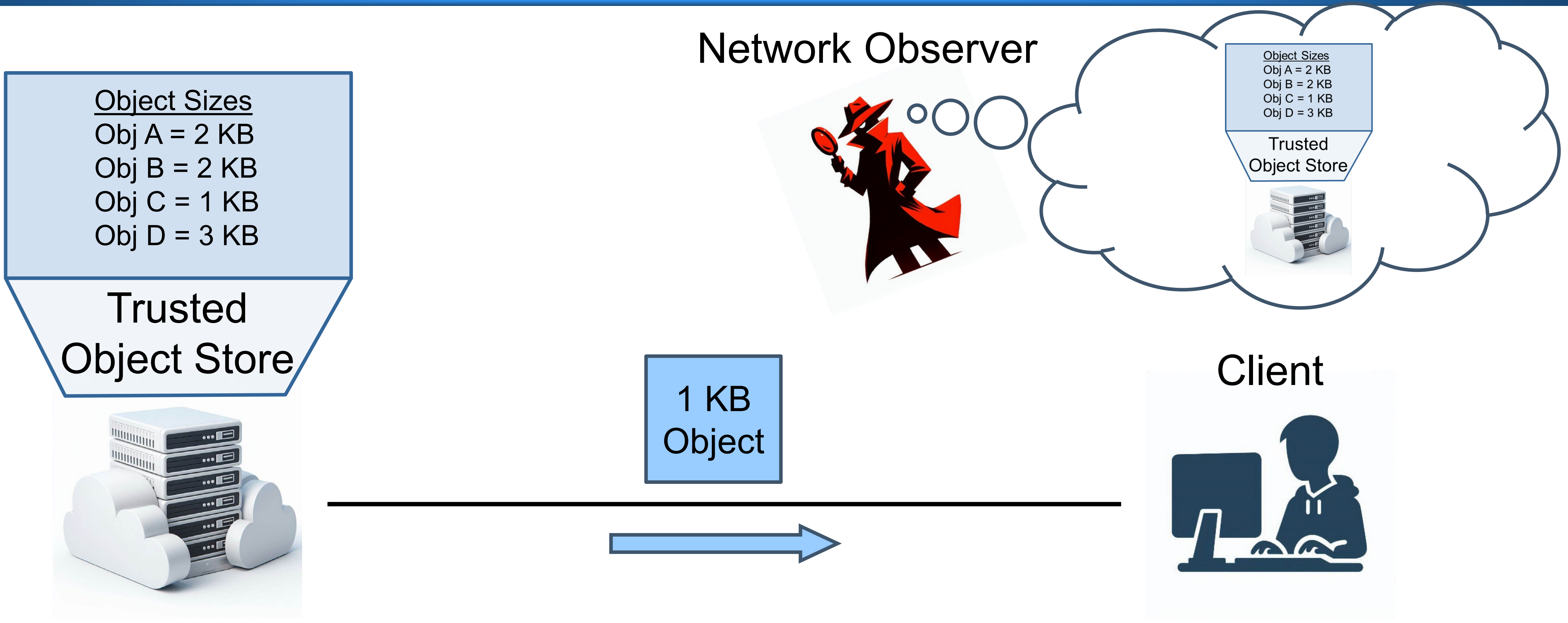
Agenda

- Objective
- Padding for Independent Object Retrievals
- Padding for Dependent Object Retrievals
- Efficiently Computing Padding for Dependent Object Retrievals using Differential Evolution
- Questions

Agenda

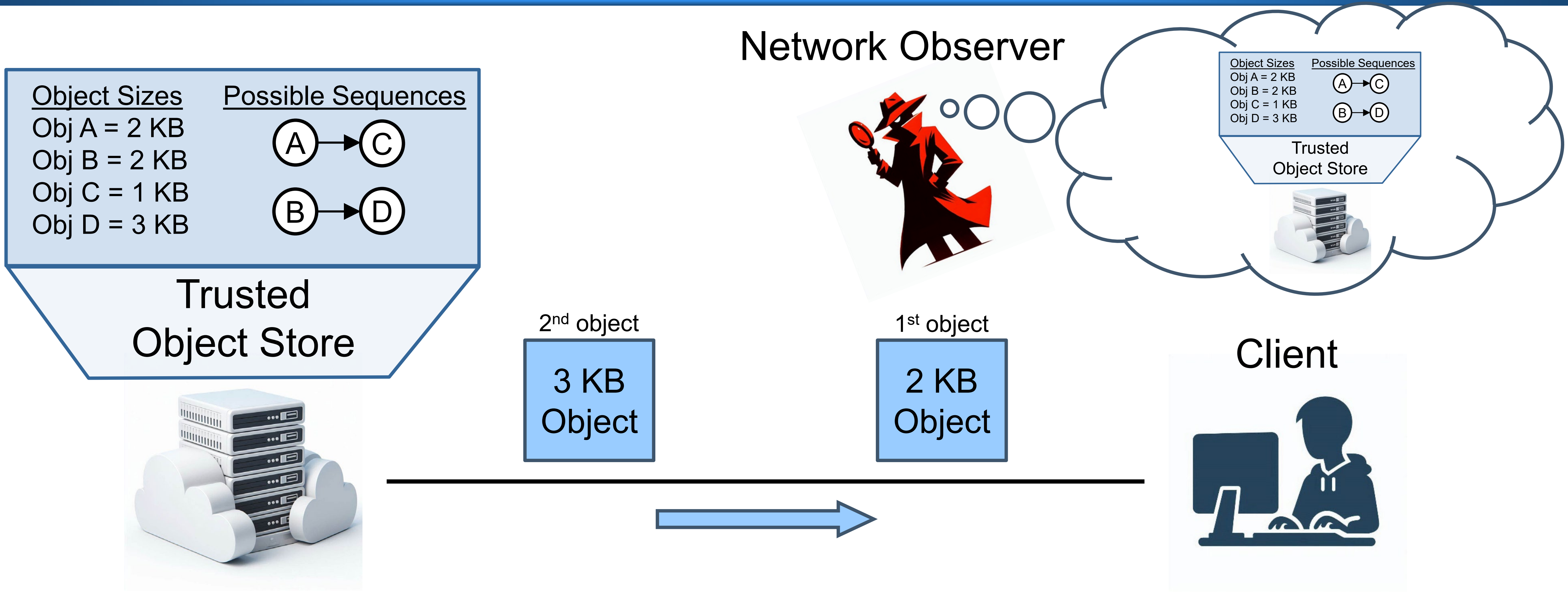
- Objective
- Padding for Independent Object Retrievals
- Padding for Dependent Object Retrievals
- Efficiently Computing Padding for Dependent Object Retrievals using Differential Evolution
- Questions

Objective: High Level – Independent Retrievals



- **Client** has retrieved an object from **Trusted Object Store**
- **Network Observer's** goal is to identify which object was requested

Objective: High Level – Dependent Retrievals



- **Client** has retrieved a **sequence** of objects from **Trusted Object Store**
- **Network Observer's** goal is to identify which objects were requested

Objective: High Level

- Threat: A network observer with the following…
  - Capability: discern the sizes of retrieved objects
  - Goal: identify which object(s) was/were retrieved
  - Knows:
    - every object's size and how often retrieved
    - all possible sequences and how often retrieved (*for the Dependent setting*)
    - the padding defense used by the object store
- Trusted Object Store's Goal: Compute a padding scheme $\lceil \cdot \rceil$ that…
  1. Uses padding to best thwart the adversary
  2. Controls the per-object overhead due to padding

Note: The object store is only willing to pad objects, i.e., it will not insert decoy objects.

Objective: Our Approach

- **Objective**: Minimize…
  - Independent Retrievals: $\mathbb{I}(S;Y)$ and $\mathbb{I}_\infty(S;Y)$
  - Dependent Retrievals: $\mathbb{I}_\infty\left(\vec{S};\vec{Y}\right)$

- **Notation**:
  - $\mathbb{I}$ = mutual information
  - $\mathbb{I}_\infty$= Sibson mutual information of order infinity, also referred to as **min-capacity**[1] and **maximal leakage**[2]
  - S = random variable for an object's **identity**
  - Y = random variable for an object's **padded size**
  - $\rightarrow$ denotes a sequence

- **Constraints**: For a given **max pad factor** $c > 1$:
  - No object is padded by more than a factor of c
  - Each object is served in full

Note: it's possible for some objects to remain isolated in our setting

1. M. Alvim, K. Chatzikokolakis, C. Palamidessi, and G. Smith, "Measuring information leakage using generalized gain functions," *25th IEEE Computer Security Foundations*, Jun. 2012.
2. I. Issa, A. B. Wagner, and S. Kamath, "An operational approach to information leakage," *IEEE Transactions on Information Theory*, vol. 66, no. 3, Mar. 2020.

Objective: Our Chosen Metrics

- $\mathbb{I}(S;Y)$ **Mutual Information**
  - $\mathbb{I}(S;Y) = \mathbb{H}(S) - \mathbb{H}(S \mid Y)$
  - Since $\mathbb{H}(S)$ is constant, minimizing $\mathbb{I}(S;Y)$ thereby maximizes $\mathbb{H}(S \mid Y)$.
  - $\mathbb{G}(S \mid Y) > \dfrac{2^{\mathbb{H}(S|Y)}}{e} + \dfrac{1}{2}$
- $\mathbb{I}_\infty(S;Y)$ **Sibson Mutual Information of Order Infinity**
  - [1] and [2] advocate for this metric because:
    - $\mathbb{I}(S;Y) \leq \mathbb{I}_\infty(S;Y)$ over all distributions of S.
    - $\mathbb{I}_\infty(S;Y)$ upper-bounds an adversary's multiplicative gain in correctly guessing any function of S after observing Y, over all distributions of S.

- **Notation**:
  - $\mathbb{H}$ = entropy
  - $\mathbb{G}$ = guessing entropy

1. M. Alvim, K. Chatzikokolakis, C. Palamidessi, and G. Smith, "Measuring information leakage using generalized gain functions," *25th IEEE Computer Security Foundations*, Jun. 2012.
2. I. Issa, A. B. Wagner, and S. Kamath, "An operational approach to information leakage," *IEEE Transactions on Information Theory*, vol. 66, no. 3, Mar. 2020.

Agenda

- Objective
- Padding for Independent Object Retrievals
- Padding for Dependent Object Retrievals
- Efficiently Computing Padding for Dependent Object Retrievals using Differential Evolution
- Questions

$\mathbb{I}(S;Y)$: Per-Object Padding (POP)

- Setting:
  - Each object is padded only once
- Key Insights:
  - $\mathbb{I}(S;Y) = \mathbb{H}(S) - \mathbb{H}(S \mid Y) = \mathbb{H}(Y) - \cancelto{0}{\mathbb{H}(Y \mid S)}$
    - Sufficient to minimize $\mathbb{H}(Y)$
  - Optimal $\lceil \cdot \rceil$ will be a partition of contiguous blocks
    - e.g., for c = 1.05 and original object sizes:

100	105	109	110	113	114	115

    - Optimal $\lceil \cdot \rceil$ **will not** be of the form:

105	105	114	115	115	114	115

    - Optimal $\lceil \cdot \rceil$ **will** be of the form:

105	105	114	114	114	114	115

- Solution:
  - Dynamic programming algorithm that runs in $O\left(\#S^2\right)$

$\mathbb{I}(S;Y)$: Per-Request Padding (PRP)

- Setting:
  - Objects are padded anew with each request
- Key Insight:
  - Special case of *rate-distortion minimization*[3]
- Solution:
  - Use the iterative algorithm "Blahut-Arimoto"[4,5] with:
    - $D(s,y) = 0$ If s can be padded to y
    - $D(s,y) = \infty$ If s cannot be padded to y

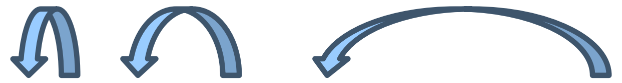
3. C. E. Shannon, "Coding theorems for a discrete source with a fidelity criterion," in *Institute of Radio Engineers, International Convention Record*, vol. 7, 1959.
4. R. Blahut, "Computation of channel capacity and rate-distortion functions," *IEEE Transactions on Information Theory*, vol. 18, no. 4, Jul. 1972.
5. S. Arimoto, "An algorithm for computing the capacity of arbitrary discrete memoryless channels," *IEEE Transactions on Information Theory*, vol. 18, no. 1, Jan. 1972.

$\mathbb{I}_\infty(\mathsf{S};\mathsf{Y})$: Padding without a Distribution (PwoD)

- Calculation:
  - $$\mathbb{I}_\infty(\mathsf{S};\mathsf{Y}) = \log_2 \sum_{y} \max_{s:\mathbb{P}(\mathsf{S}=s)>0} \mathbb{P}(\mathsf{Y}=y \mid \mathsf{S}=s)$$
  - $\mathbb{I}_\infty(\mathsf{S};\mathsf{Y})$ only requires that the object store know which objects have a nonzero probability of being retrieved
- Solution:
  - A greedy algorithm that runs in time linear in $\#S$
    - e.g., for c = 1.05 and original object sizes:

100	105	109	110	113	114	115

    - PwoD runs from right-to-left assigning objects to padding groups as follows:

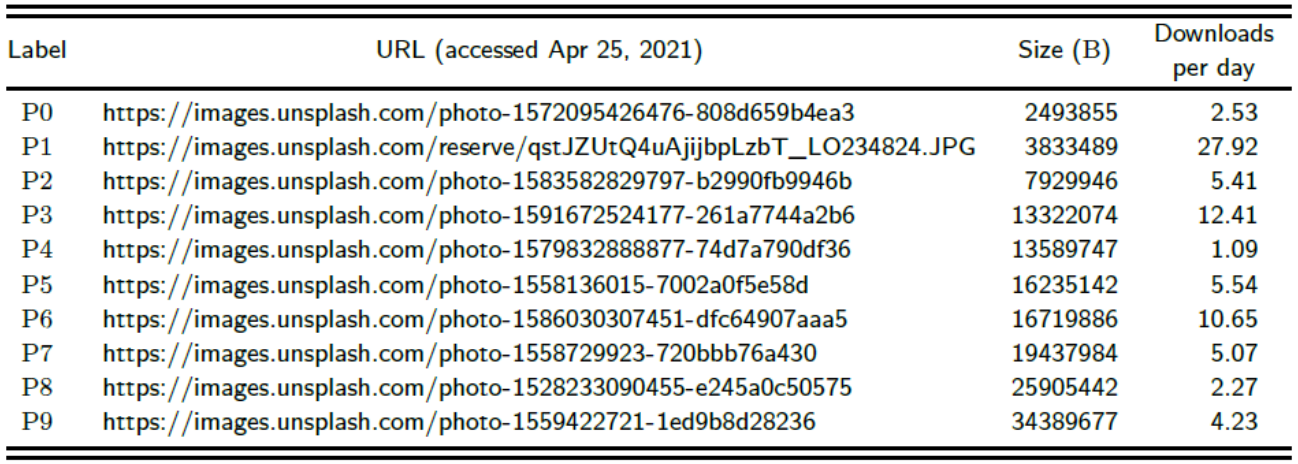


100	105	109	110	113	114	115
100	109	109	115	115	115	115

Example Padding Schemes

Inputs:

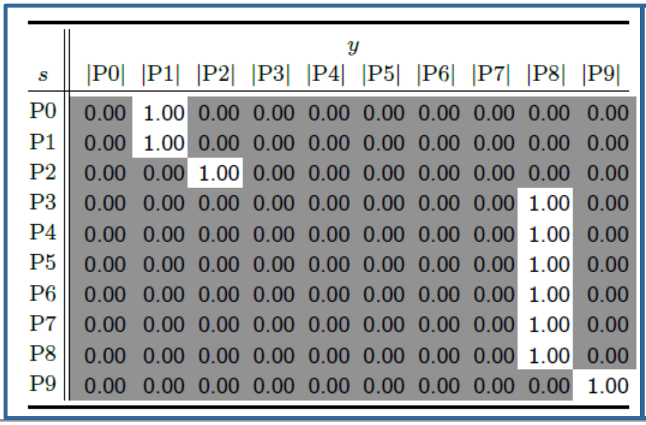
$C = 2$ &



Label	URL (accessed Apr 25, 2021)	Size (B)	Downloads per day
P0	https://images.unsplash.com/photo-1572095426476-808d659b4ea3	2493855	2.53
P1	https://images.unsplash.com/reserve/qstJZUtQ4uAjijbpLzbT_LO234824.JPG	3833489	27.92
P2	https://images.unsplash.com/photo-1583582829797-b2990fb9946b	7929946	5.41
P3	https://images.unsplash.com/photo-1591672524177-261a7744a2b6	13322074	12.41
P4	https://images.unsplash.com/photo-1579832888877-74d7a790df36	13589747	1.09
P5	https://images.unsplash.com/photo-1558136015-7002a0f5e58d	16235142	5.54
P6	https://images.unsplash.com/photo-1586030307451-dfc64907aaa5	16719886	10.65
P7	https://images.unsplash.com/photo-1558729923-720bbb76a430	19437984	5.07
P8	https://images.unsplash.com/photo-1528233090455-e245a0c50575	25905442	2.27
P9	https://images.unsplash.com/photo-1559422721-1ed9b8d28236	34389677	4.23

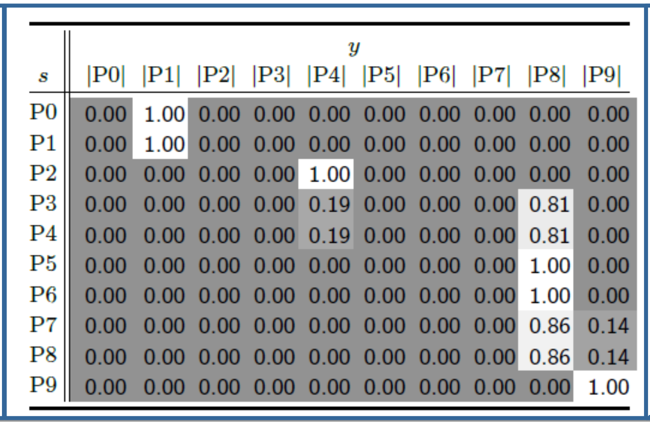
Outputs:

POP



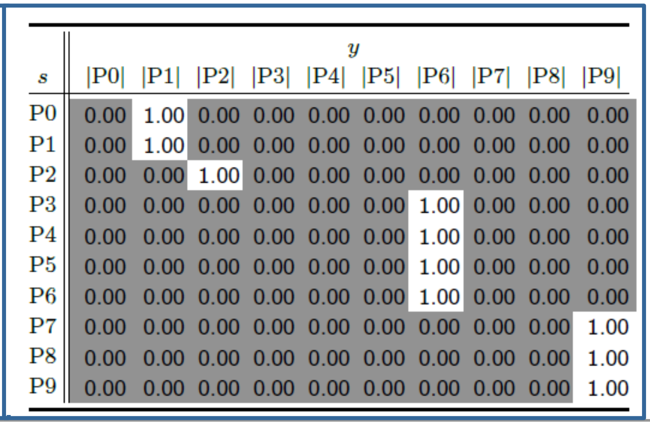
	y									
s	|P0|	|P1|	|P2|	|P3|	|P4|	|P5|	|P6|	|P7|	|P8|	|P9|
P0	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
P1	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
P2	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
P3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00
P4	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00
P5	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00
P6	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00
P7	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00
P8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00
P9	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00

PRP



	y									
s	|P0|	|P1|	|P2|	|P3|	|P4|	|P5|	|P6|	|P7|	|P8|	|P9|
P0	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
P1	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
P2	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00
P3	0.00	0.00	0.00	0.00	0.19	0.00	0.00	0.00	0.81	0.00
P4	0.00	0.00	0.00	0.00	0.19	0.00	0.00	0.00	0.81	0.00
P5	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00
P6	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00
P7	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.86	0.14
P8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.86	0.14
P9	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00

PwoD

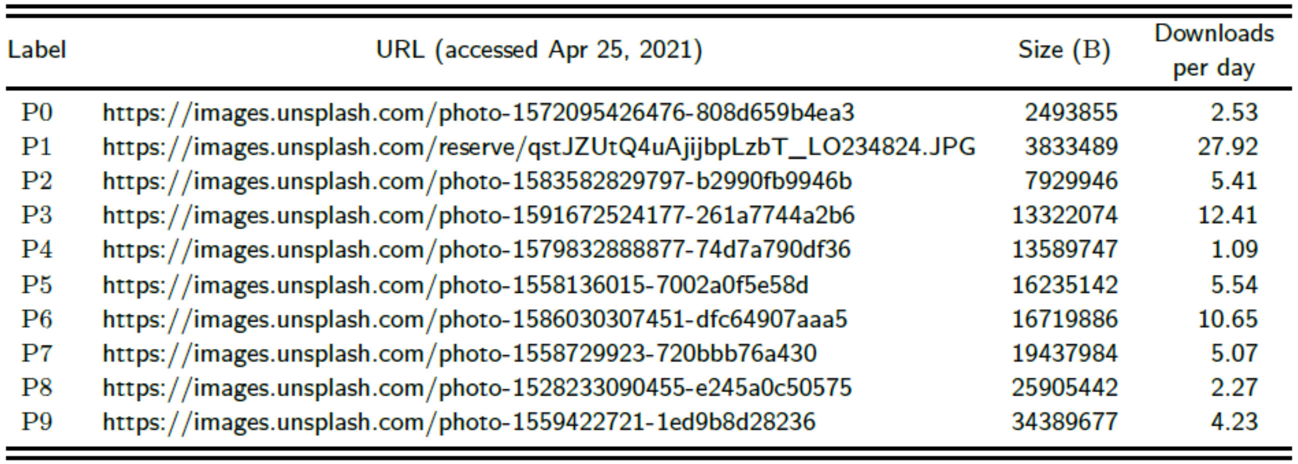


	y									
s	|P0|	|P1|	|P2|	|P3|	|P4|	|P5|	|P6|	|P7|	|P8|	|P9|
P0	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
P1	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
P2	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
P3	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00
P4	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00
P5	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00
P6	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00
P7	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00
P8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00
P9	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00

Competitors

Inputs:

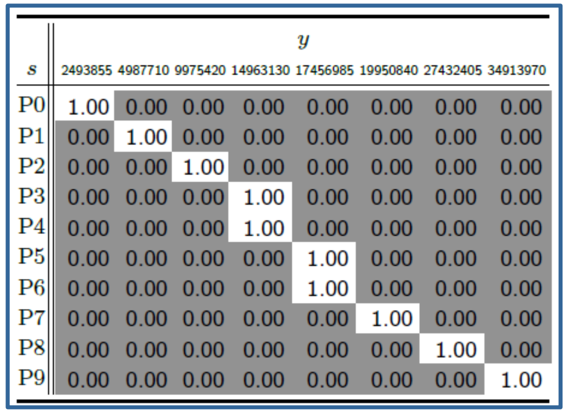
$C = 2$ &



Label	URL (accessed Apr 25, 2021)	Size (B)	Downloads per day
P0	https://images.unsplash.com/photo-1572095426476-808d659b4ea3	2493855	2.53
P1	https://images.unsplash.com/reserve/qstJZUtQ4uAjijbpLzbT_LO234824.JPG	3833489	27.92
P2	https://images.unsplash.com/photo-1583582829797-b2990fb9946b	7929946	5.41
P3	https://images.unsplash.com/photo-1591672524177-261a7744a2b6	13322074	12.41
P4	https://images.unsplash.com/photo-1579832888877-74d7a790df36	13589747	1.09
P5	https://images.unsplash.com/photo-1558136015-7002a0f5e58d	16235142	5.54
P6	https://images.unsplash.com/photo-1586030307451-dfc64907aaa5	16719886	10.65
P7	https://images.unsplash.com/photo-1558729923-720bbb76a430	19437984	5.07
P8	https://images.unsplash.com/photo-1528233090455-e245a0c50575	25905442	2.27
P9	https://images.unsplash.com/photo-1559422721-1ed9b8d28236	34389677	4.23

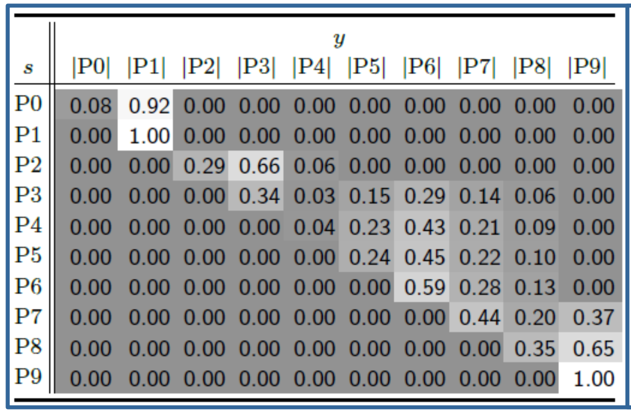
Outputs:

D-ALPaCA[6]



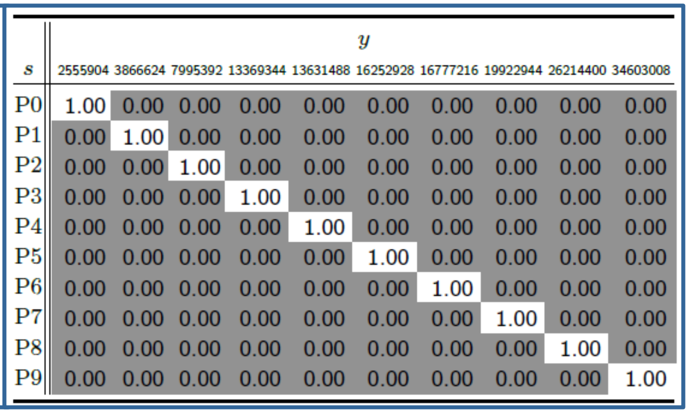
	y							
s	2493855	4987710	9975420	14963130	17456985	19950840	27432405	34913970
P0	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
P1	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00
P2	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00
P3	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00
P4	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00
P5	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00
P6	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00
P7	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00
P8	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00
P9	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00

P-ALPaCA[6]



	y									
s	|P0|	|P1|	|P2|	|P3|	|P4|	|P5|	|P6|	|P7|	|P8|	|P9|
P0	0.08	0.92	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
P1	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
P2	0.00	0.00	0.29	0.66	0.06	0.00	0.00	0.00	0.00	0.00
P3	0.00	0.00	0.00	0.34	0.03	0.15	0.29	0.14	0.06	0.00
P4	0.00	0.00	0.00	0.00	0.04	0.23	0.43	0.21	0.09	0.00
P5	0.00	0.00	0.00	0.00	0.00	0.24	0.45	0.22	0.10	0.00
P6	0.00	0.00	0.00	0.00	0.00	0.00	0.59	0.28	0.13	0.00
P7	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.44	0.20	0.37
P8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.35	0.65
P9	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00

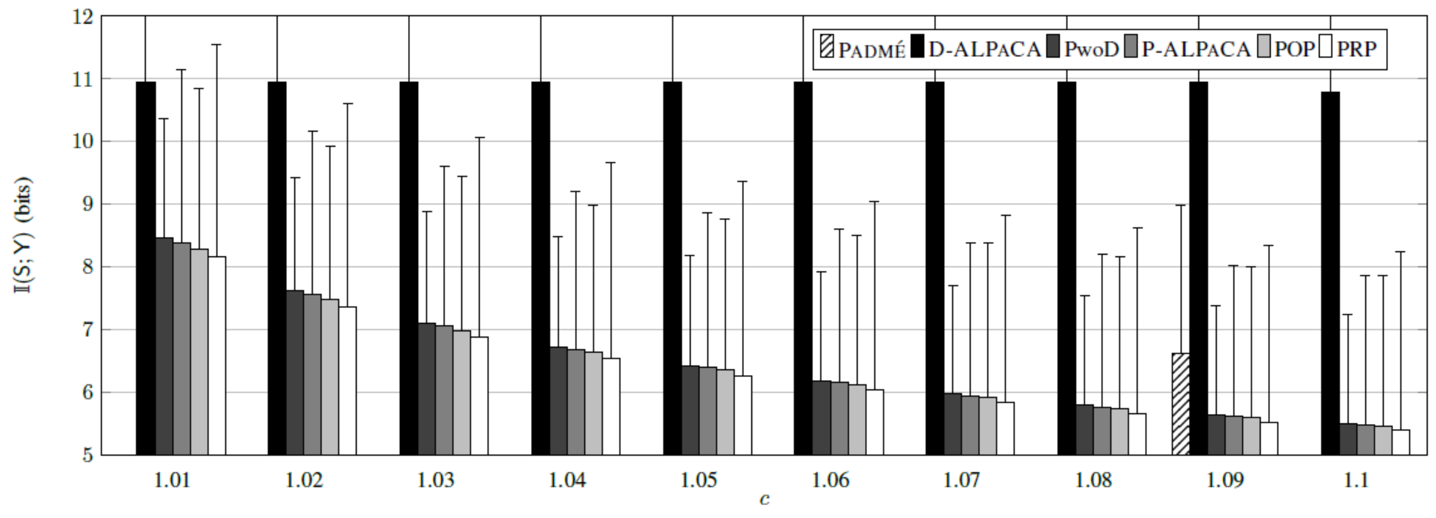
Padmé[7]



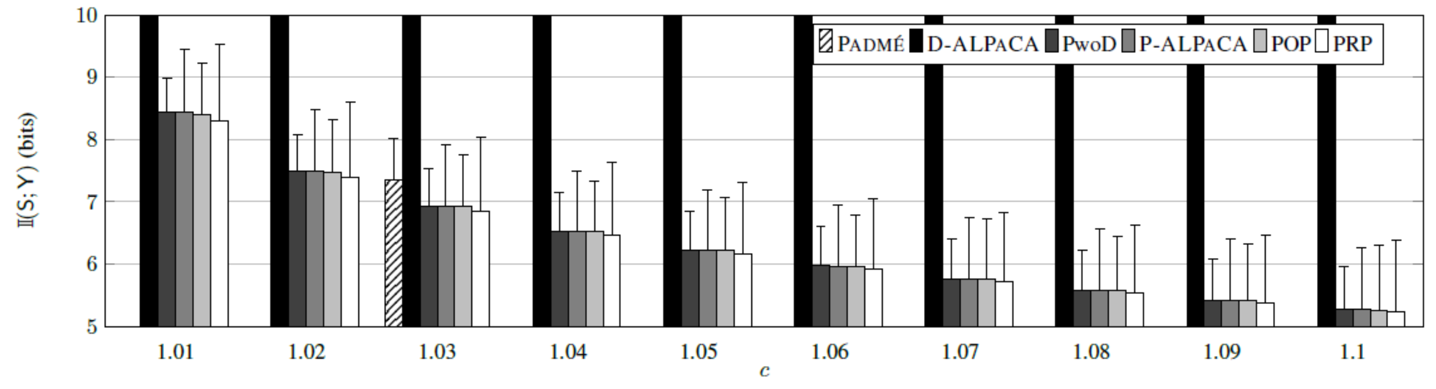
	y									
s	2555904	3866624	7995392	13369344	13631488	16252928	16777216	19922944	26214400	34603008
P0	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
P1	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
P2	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
P3	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00
P4	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00
P5	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00
P6	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00
P7	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00
P8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00
P9	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00

6. G. Cherubin, J. Hayes, and M. Juarez, "Website fingerprinting defenses at the application layer," *Proceedings on Privacy Enhancing Technologies*, vol. 2017, no. 2, 2017.
7. K. Nikitin, L. Barman, W. Lueks, M. Underwood, J.-P. Hubaux, and B. Ford, "Reducing metadata leakage from encrypted files and communication with PURBs," *Proceedings on Privacy Enhancing Technologies*, vol. 2019, no. 4, 2019.

Evaluation: $\mathbb{I}(S;Y)$ and $\mathbb{I}_\infty(S;Y)$



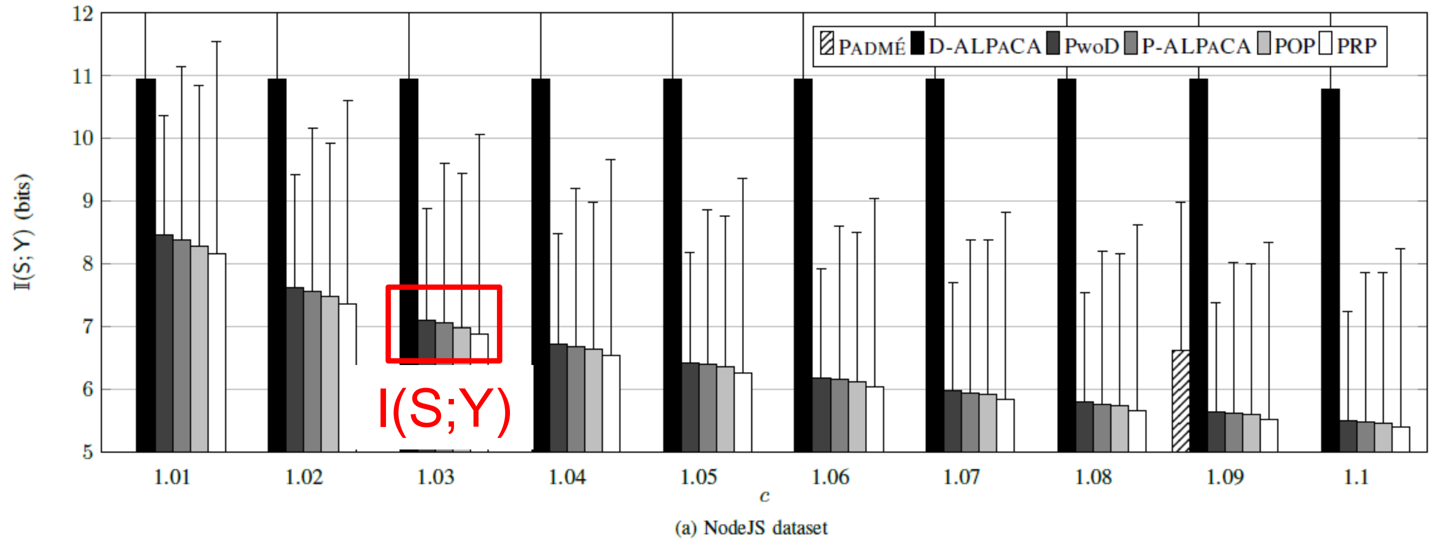
(a) NodeJS dataset



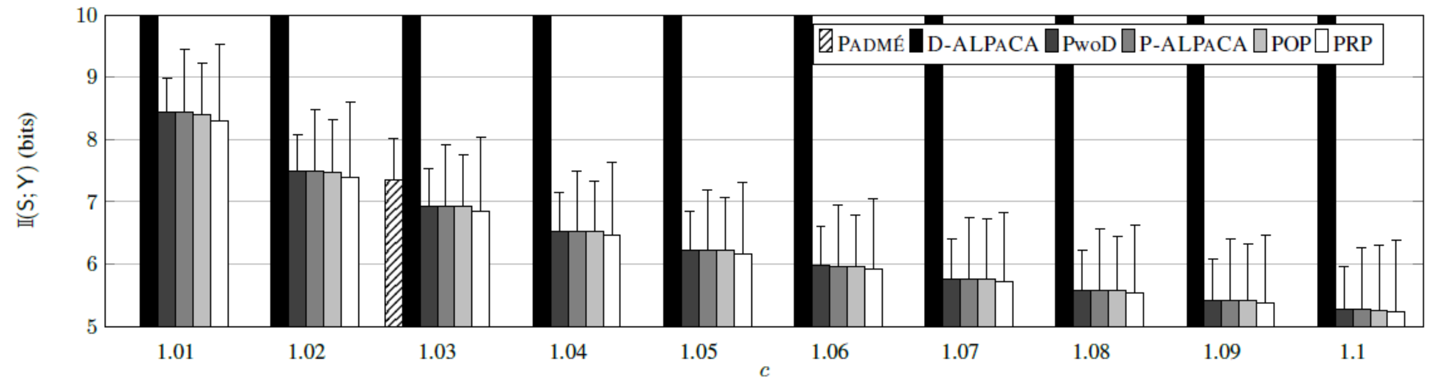
(b) Unsplash Lite dataset

Per-algorithm mutual information. Error bars extend to $\mathbb{I}_\infty(S;Y)$. Lower values indicate better security.

Evaluation: $\mathbb{I}(S;Y)$ and $\mathbb{I}_\infty(S;Y)$



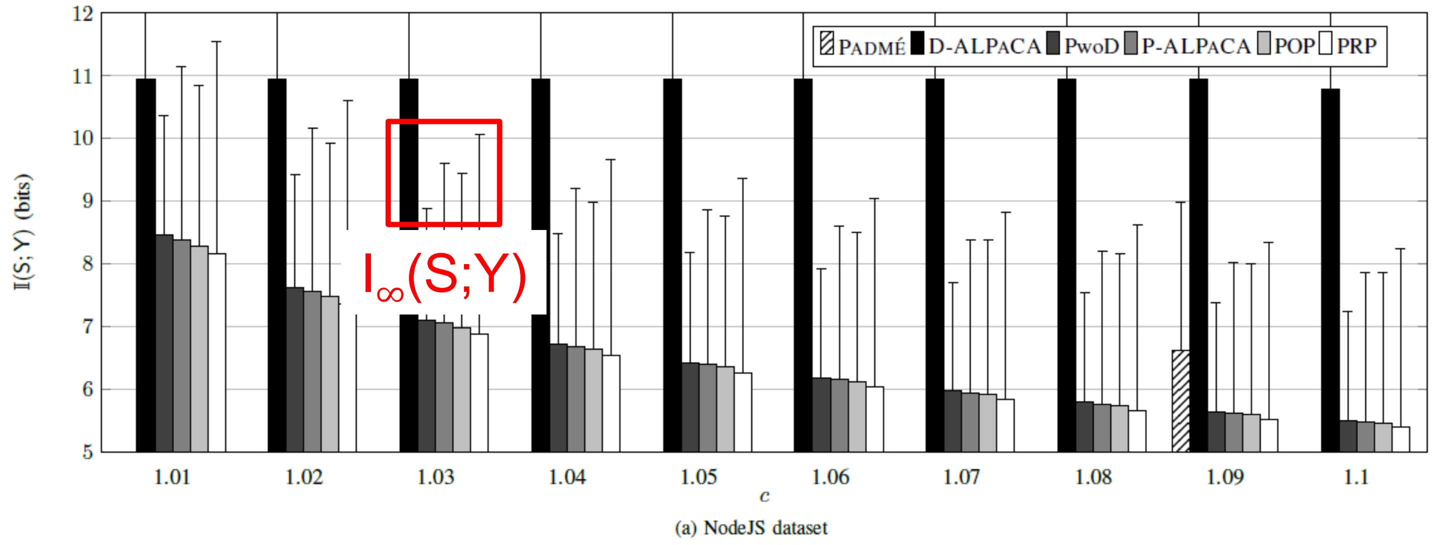
(a) NodeJS dataset



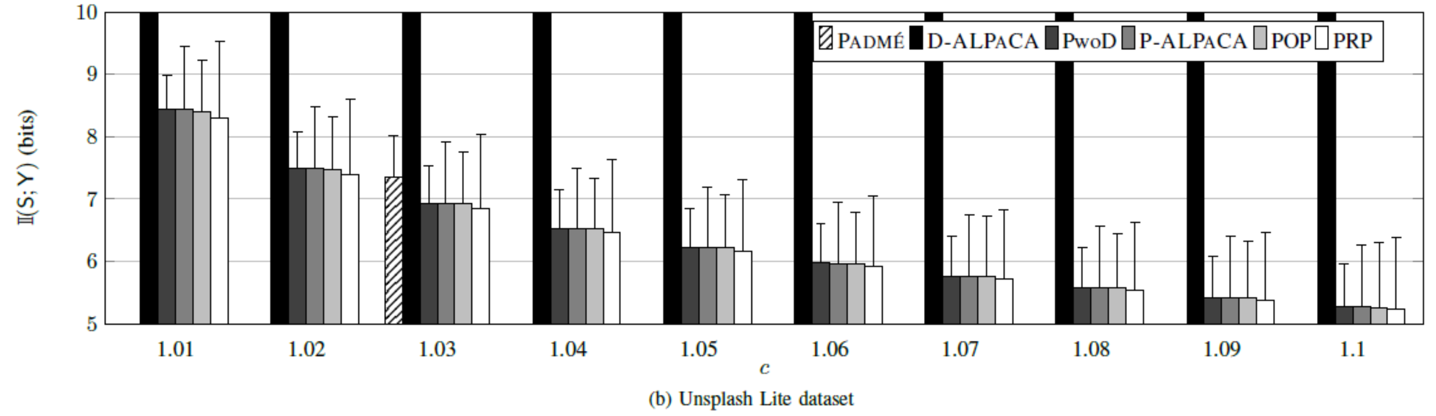
(b) Unsplash Lite dataset

Per-algorithm mutual information. Error bars extend to $\mathbb{I}_\infty(S;Y)$. Lower values indicate better security.

Evaluation: $\mathbb{I}(S;Y)$ and $\mathbb{I}_\infty(S;Y)$



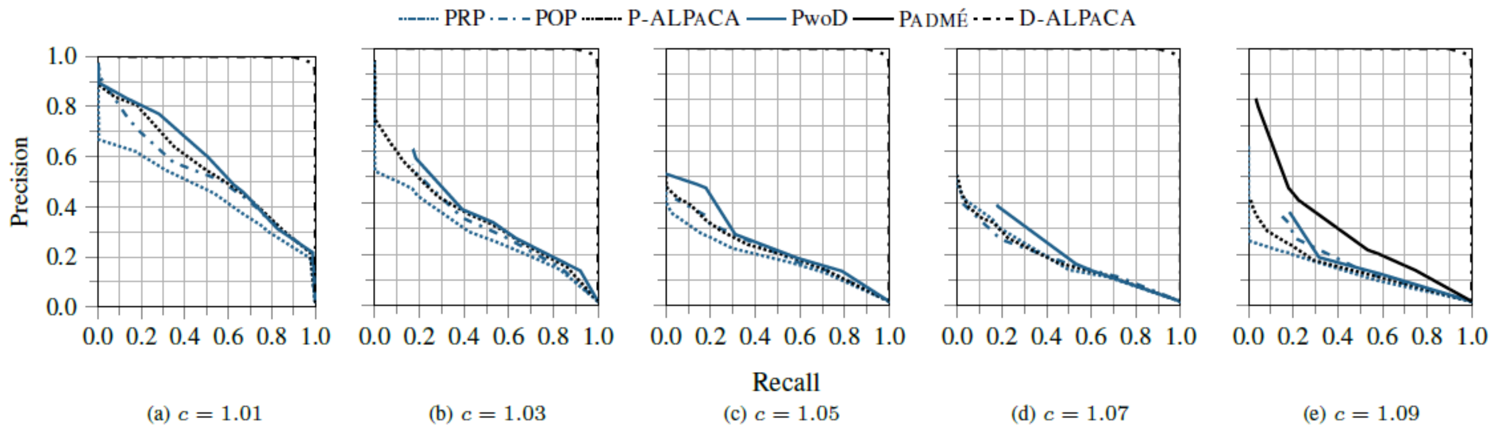
(a) NodeJS dataset



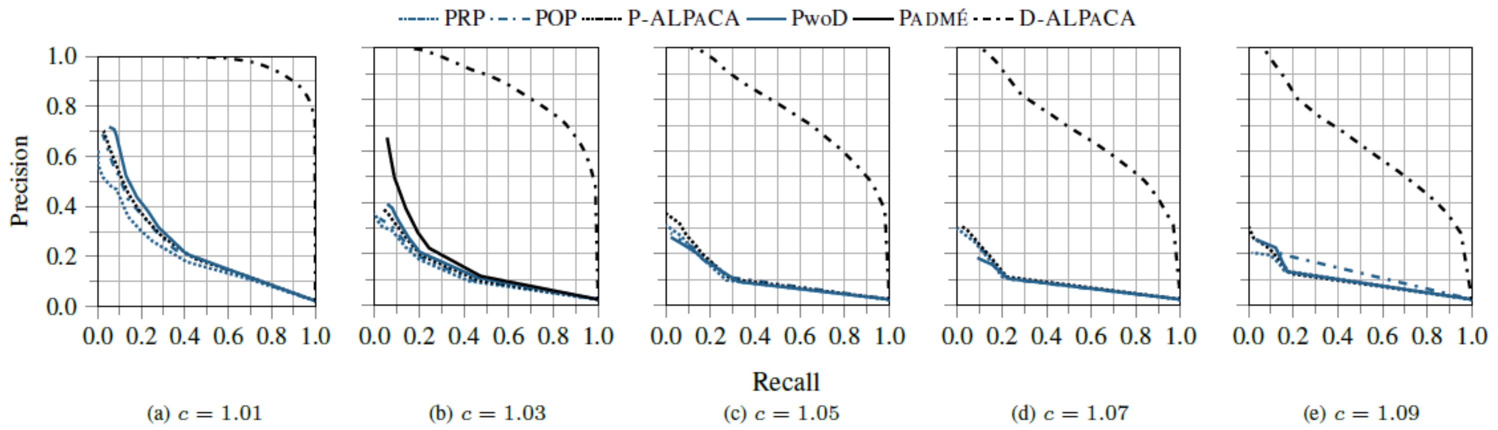
(b) Unsplash Lite dataset

Per-algorithm mutual information. Error bars extend to $\mathbb{I}_\infty(S;Y)$. Lower values indicate better security.

Evaluation: Recall & Precision

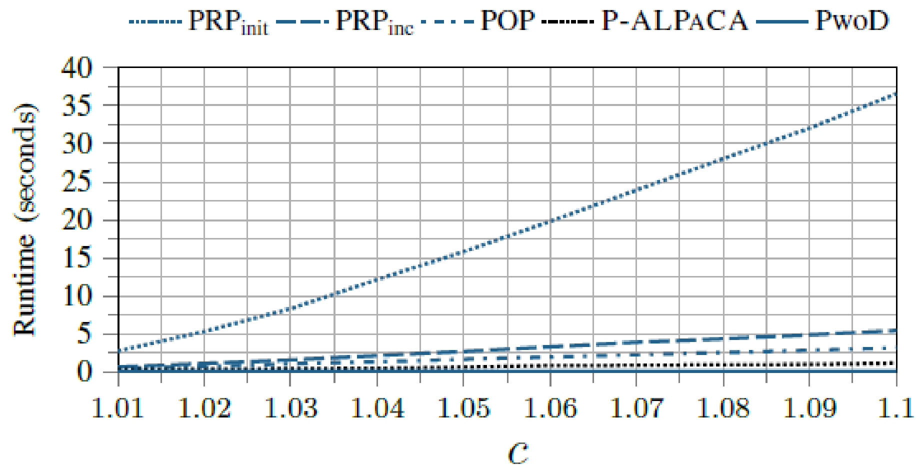


Adversary's recall and precision for detecting vulnerable NodeJS packages.

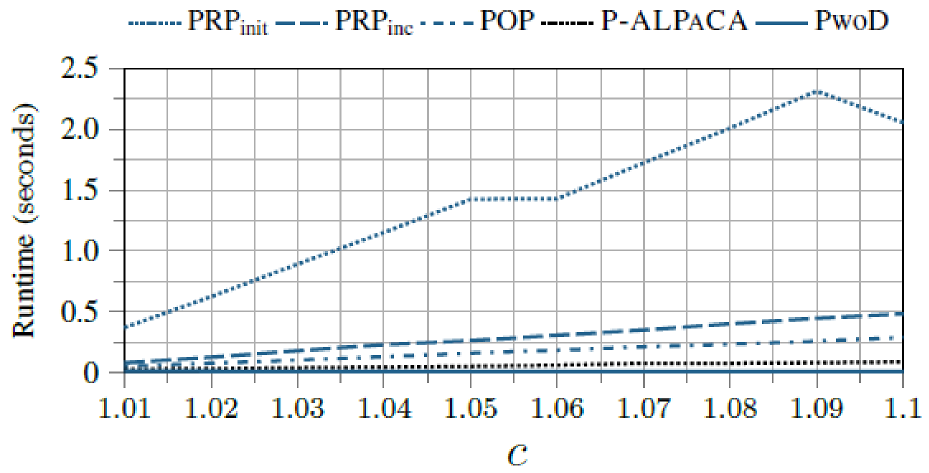


Adversary's recall and precision for detecting the Unsplash Lite *Nature* collection.

Evaluation: Runtimes



Runtimes on the NodeJS dataset.



Runtimes on the Unsplash Lite dataset.

Agenda

- Objective
- Padding for Independent Object Retrievals
- Padding for Dependent Object Retrievals
- Efficiently Computing Padding for Dependent Object Retrievals using Differential Evolution
- Questions

Algorithm: Padding For Sequences (PFS)

- Design: a linear program named Padding For Sequences (PFS)

- Inputs:
  - S = the set of objects
  - $\vec{S}$ = the set of possible sequences
  - c = max padding factor per object
  - k = an efficiency parameter (i.e., dimensionality reduction)
- Output:
  - A *memoryless* padding scheme $\lceil \cdot \rceil$ that minimizes an upper bound on $\mathbb{I}_\infty\left(\vec{\mathrm{S}};\vec{\mathrm{Y}}\right)$ and does not violate c for any object

Competitors: Overview

- BDK[8]
  - Target metric: $\mathbb{I}\left(\vec{\mathsf{S}};\vec{\mathsf{Y}}\right)$
- MVMD-D[9]
  - Target metric: ℓ-diversity
- PwoD
  - Target metric: $\mathbb{I}_\infty(\mathsf{S};\mathsf{Y})$

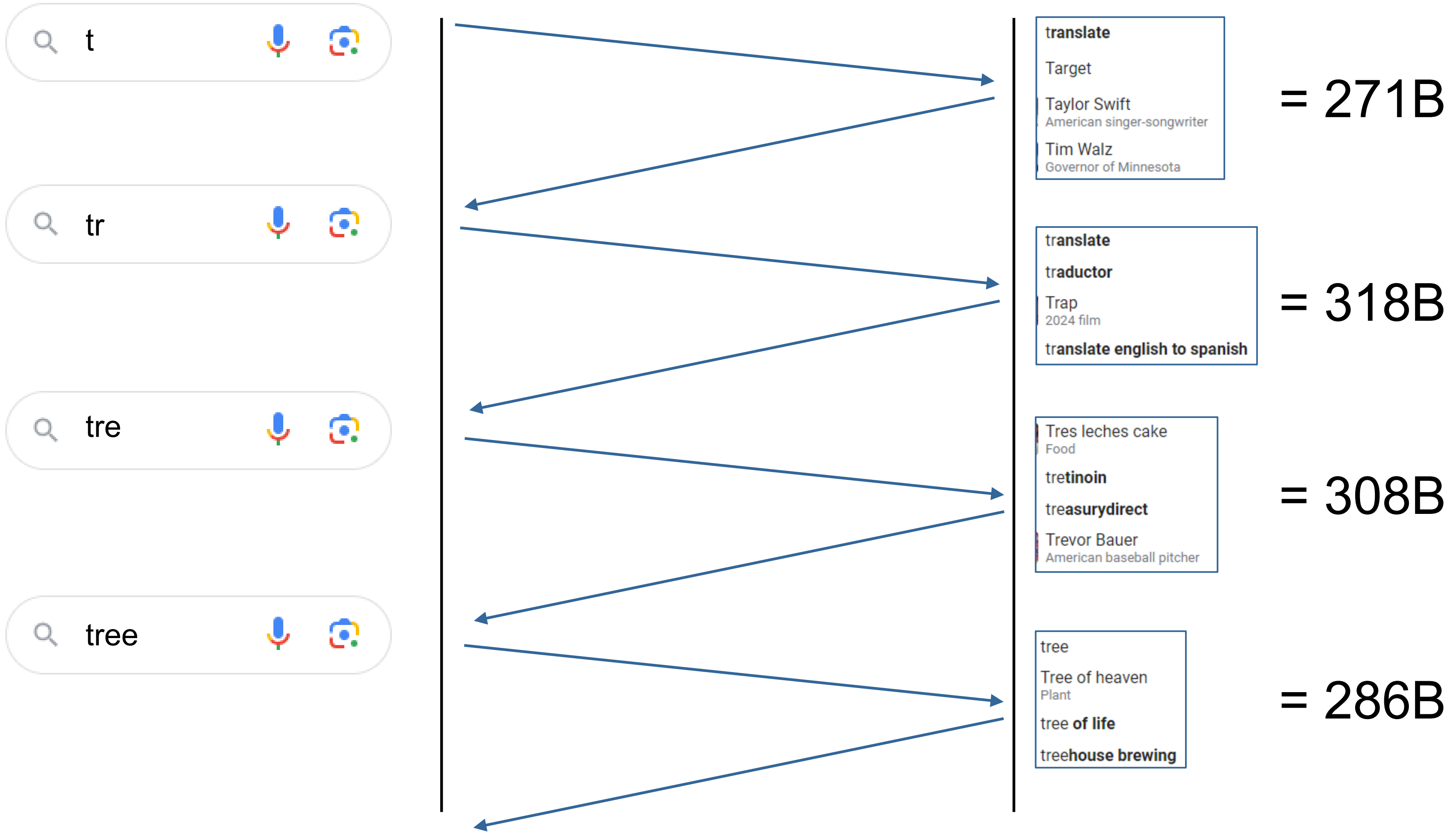
Inputs required per algorithm.

Algorithm	Sets			Distributions		
	S	$\vec{S}$	E	$\mathbb{P}(\mathsf{S}=s)$	$\mathbb{P}\left(\vec{\mathsf{S}}_{1\ldots i}=\vec{s}_{1\ldots i}\right)$	$\mathbb{P}\left(\vec{\mathsf{S}}_{i+1}=s' \mid \vec{\mathsf{S}}_i=s\right)$
BDK	✓		✓	✓		✓
MVMD-D	✓	✓			✓	
PwoD	✓					
PFS	✓	✓				

8. M. Backes, G. Doychev, and B. Kopf, "Preventing side channel leaks in web traffic: A formal approach," *20th ISOC Network and Distributed System Security Symposium*, February 2013.
9. W. M. Liu, L. Wang, P. Cheng, K. Ren, S. Zhu, and M. Debbabi, "PPTP: Privacy-preserving traffic padding in web-based applications," *IEEE Transactions on Dependable and Secure Computing*, Nov-Dec 2014.

Dataset: Autocomplete

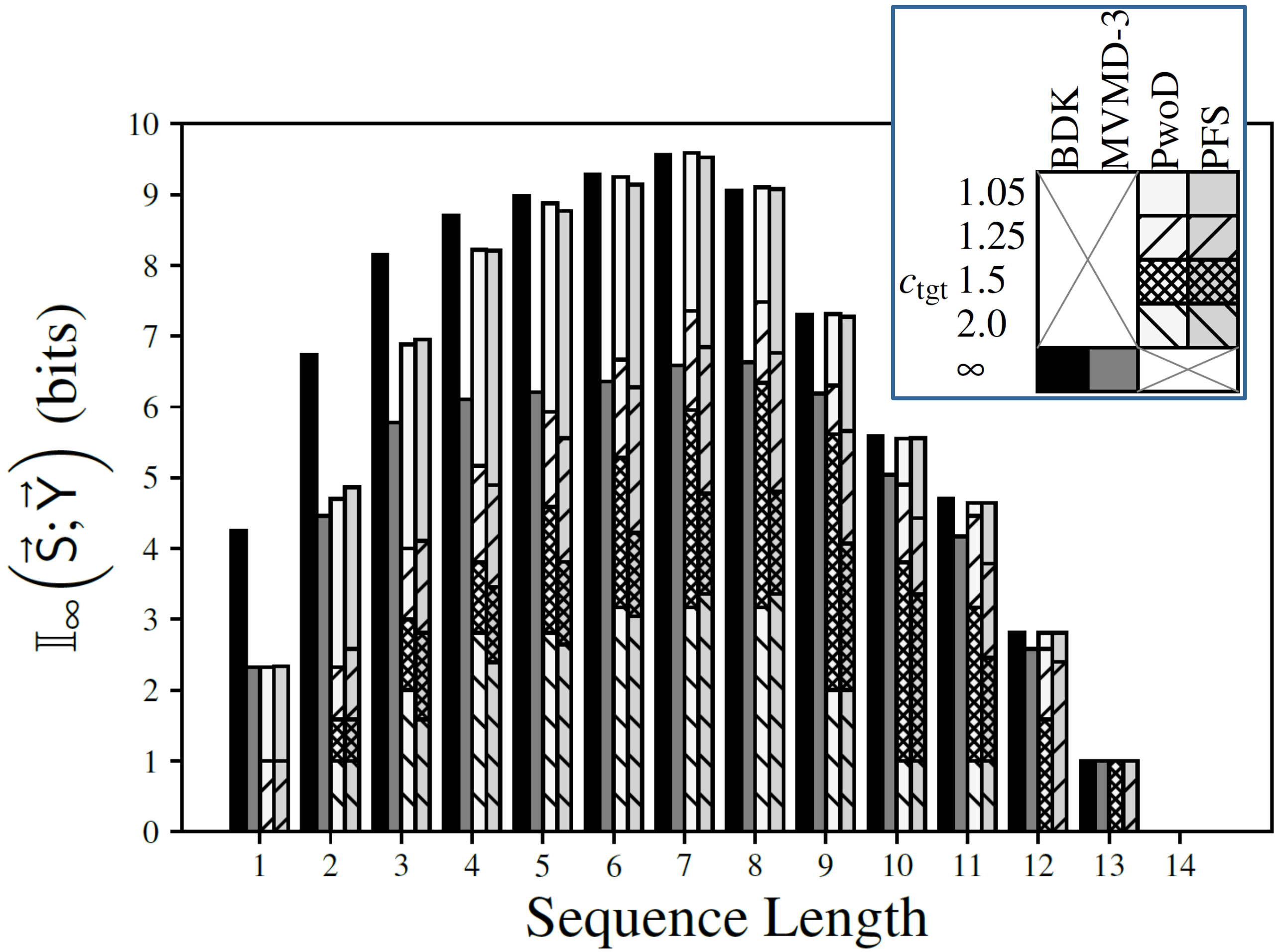
- 899 full words
- 3870 total sequences
- Models a user typing a word into the Google search bar and receiving suggested search terms after each character is typed.



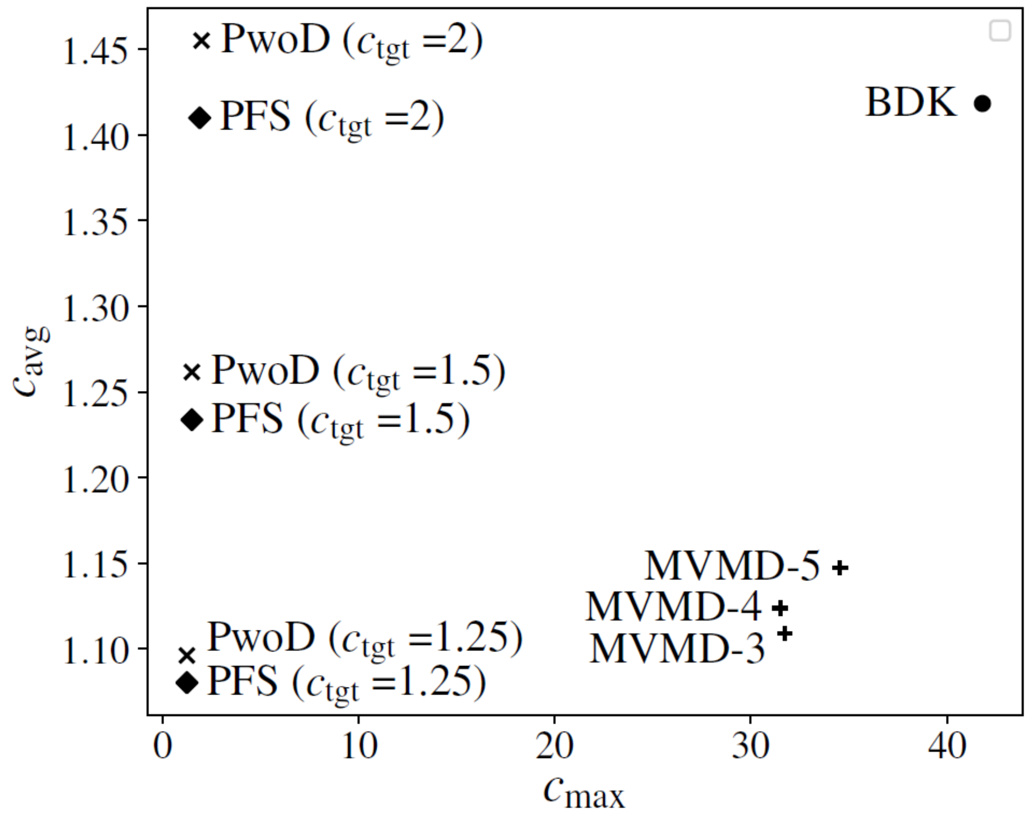
The Autocomplete sequence corresponding to the word "tree" is: (t, tr, tre, tree) = (271, 318, 308, 286)

Note: Screenshots taken Aug 8, 2024 and do not correspond to the sizes in the provided dataset. They are for illustrative purposes only.

Evaluation: Autocomplete - $\mathbb{I}_\infty\left(\vec{S};\vec{Y}\right)$

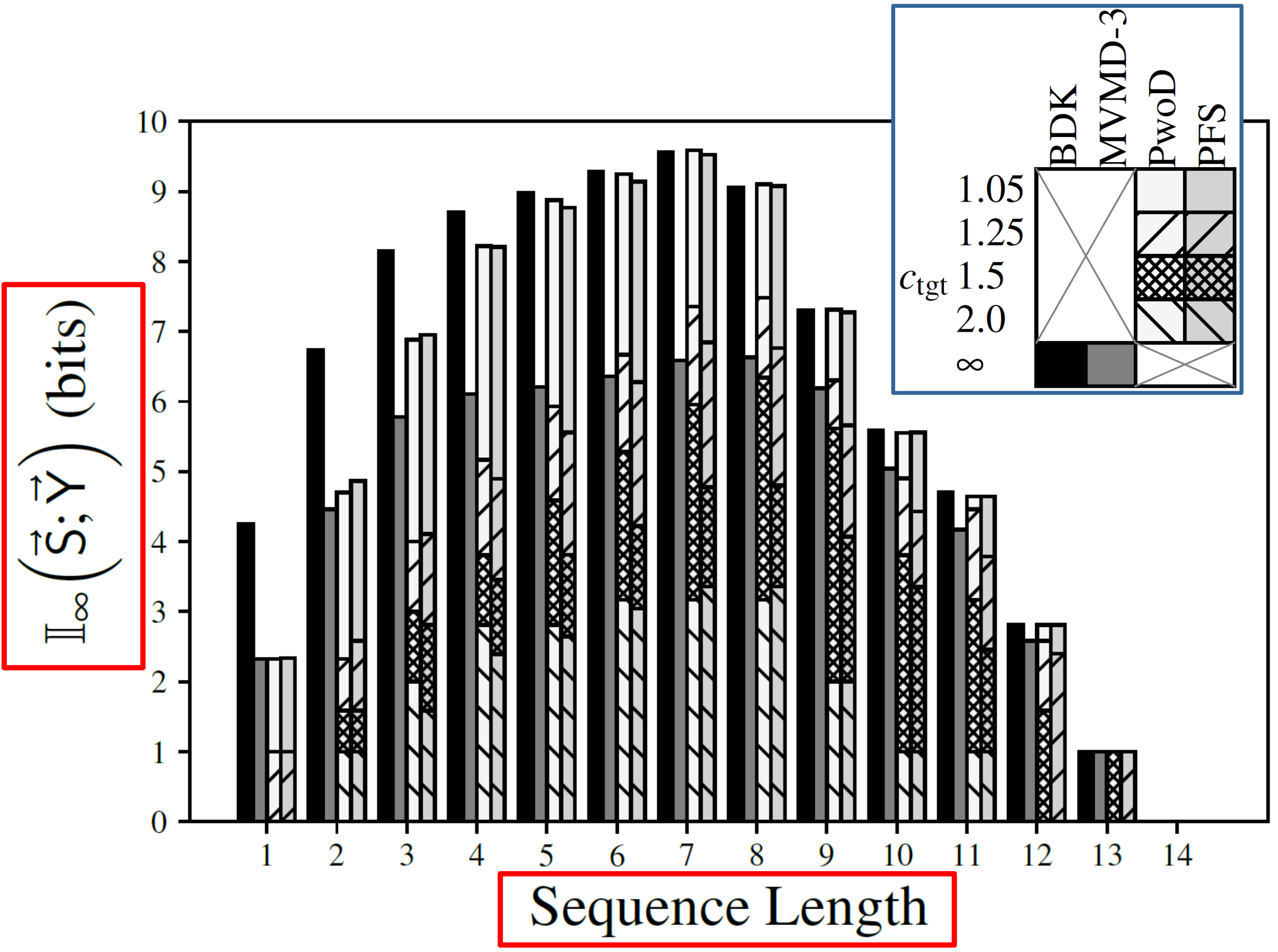


Comparing all algorithms using $\mathbb{I}_\infty\left(\vec{S};\vec{Y}\right)$ as the privacy metric.

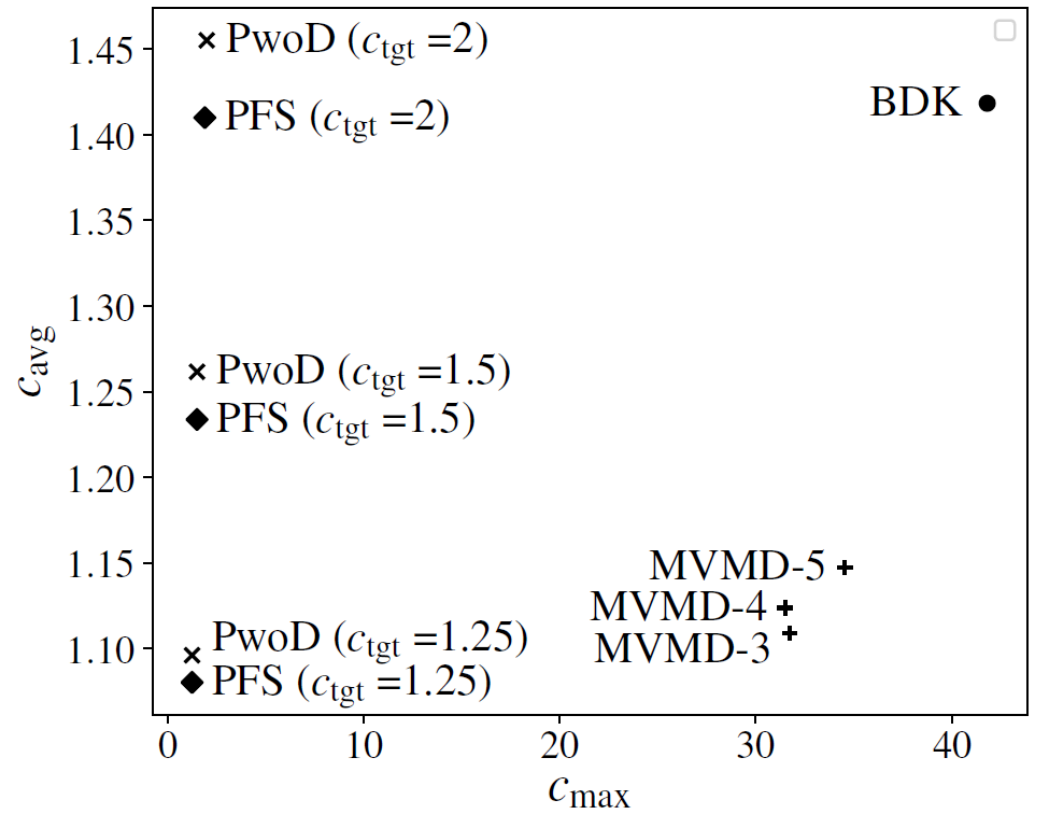


Padding overhead factors for each padding algorithm.

Evaluation: Autocomplete - $\mathbb{I}_\infty\left(\vec{S};\vec{Y}\right)$

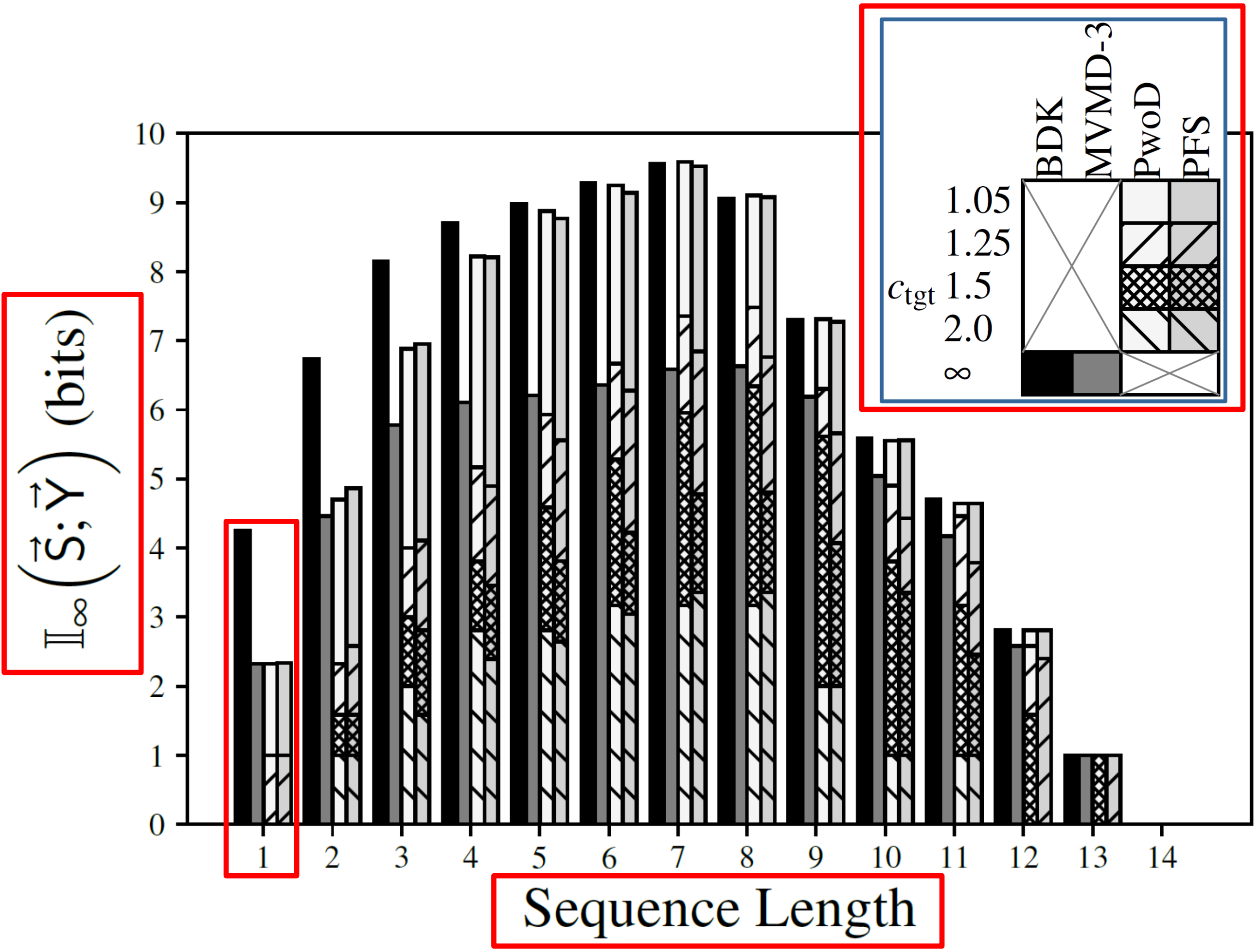


Comparing all algorithms using $\mathbb{I}_\infty\left(\vec{S};\vec{Y}\right)$ as the privacy metric.

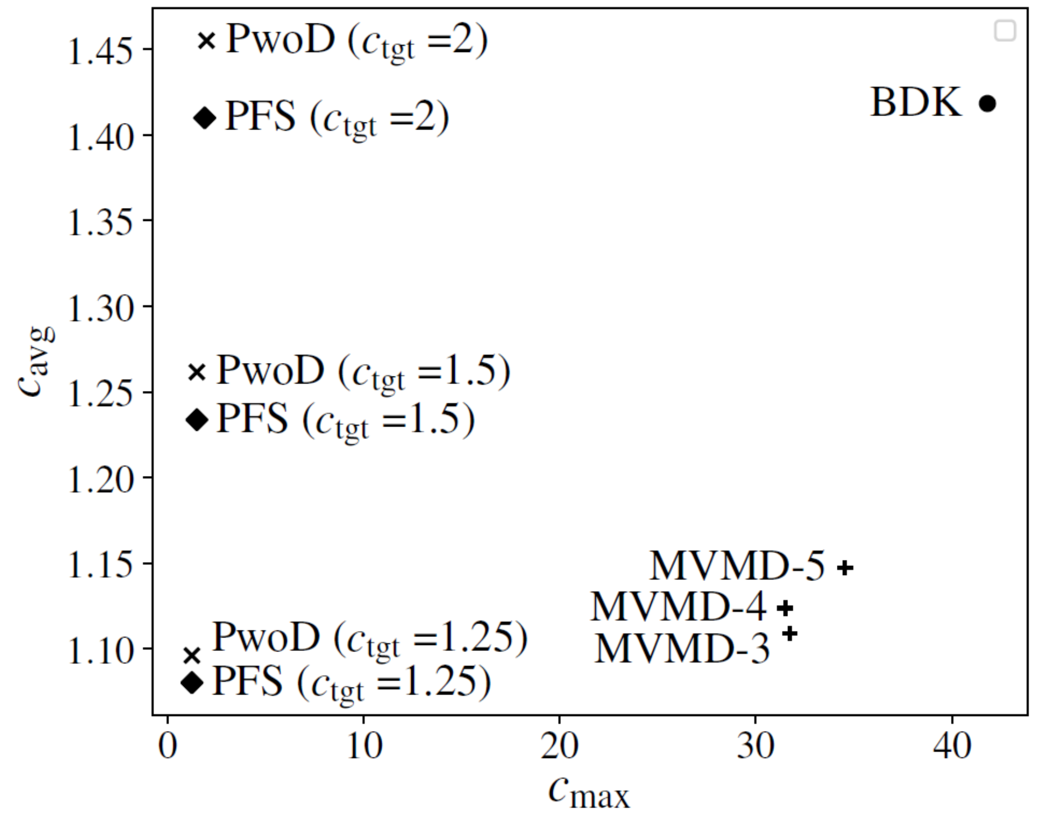


Padding overhead factors for each padding algorithm.

Evaluation: Autocomplete - $\mathbb{I}_\infty\left(\vec{S};\vec{Y}\right)$

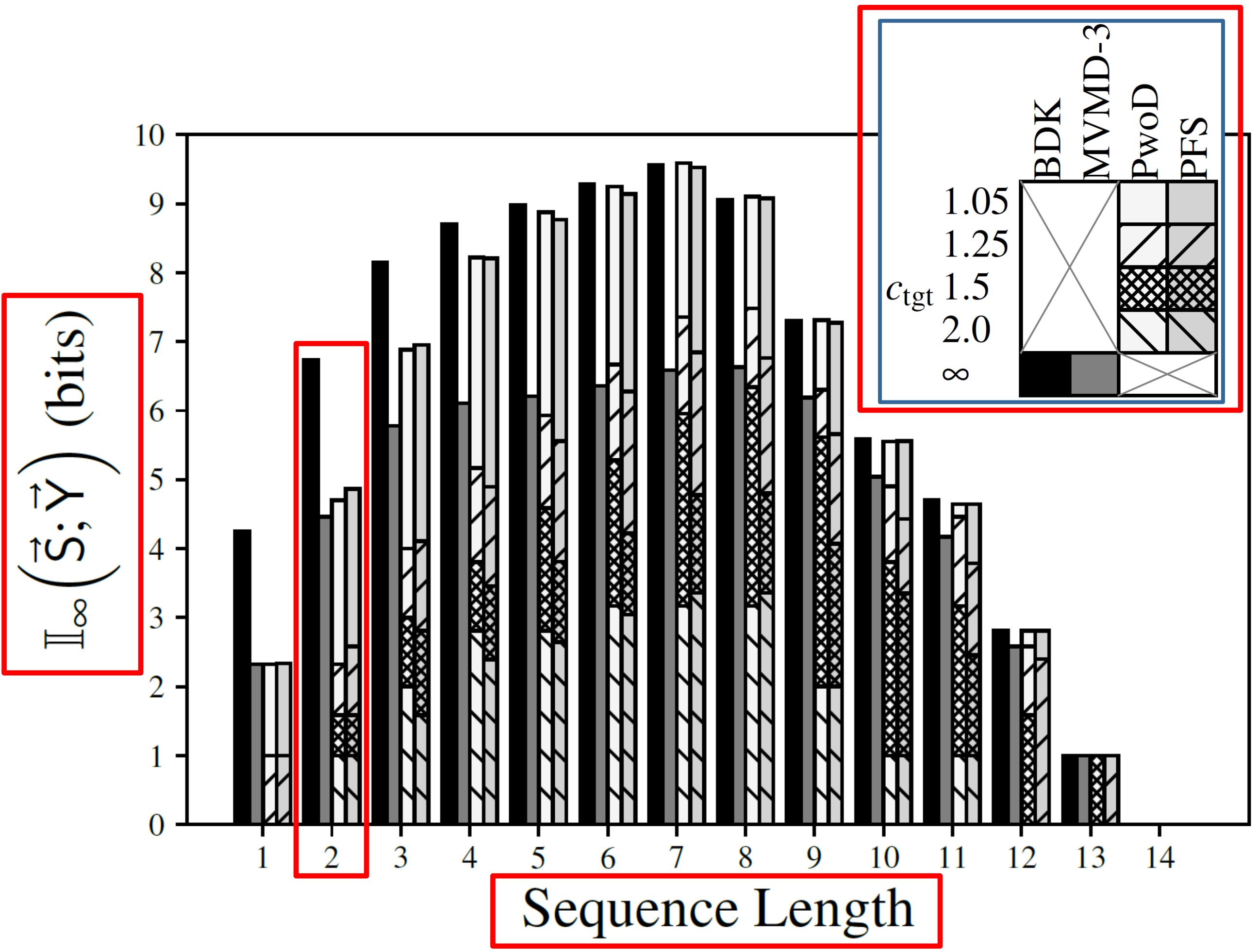


Comparing all algorithms using $\mathbb{I}_\infty\left(\vec{S};\vec{Y}\right)$ as the privacy metric.

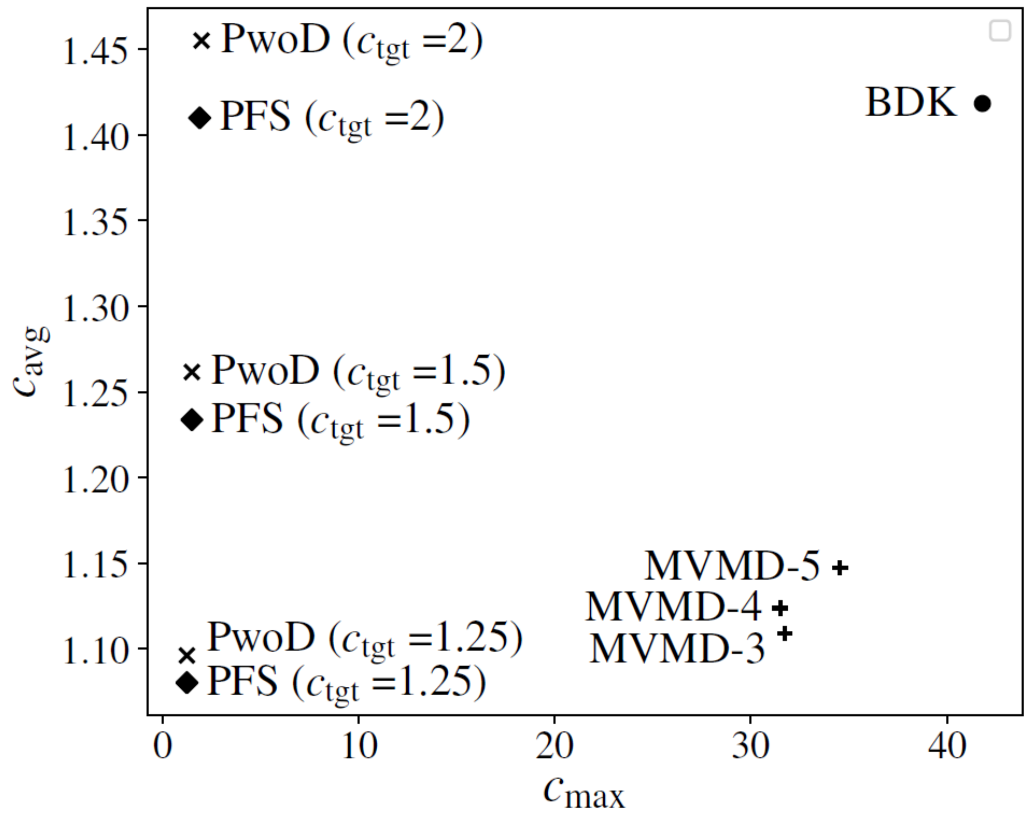


Padding overhead factors for each padding algorithm.

Evaluation: Autocomplete - $\mathbb{I}_\infty(\vec{S};\vec{Y})$

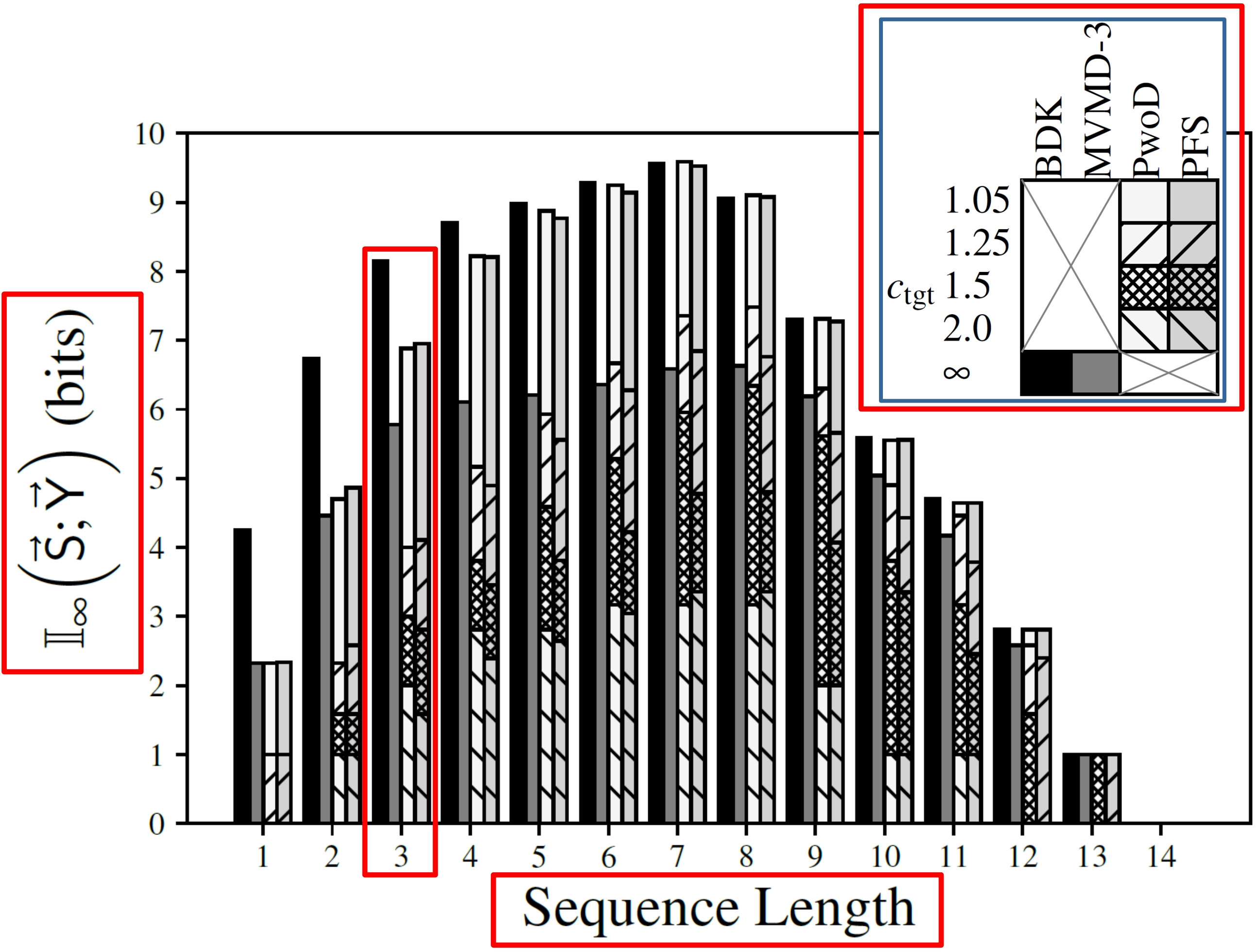


Comparing all algorithms using $\mathbb{I}_\infty(\vec{S};\vec{Y})$ as the privacy metric.

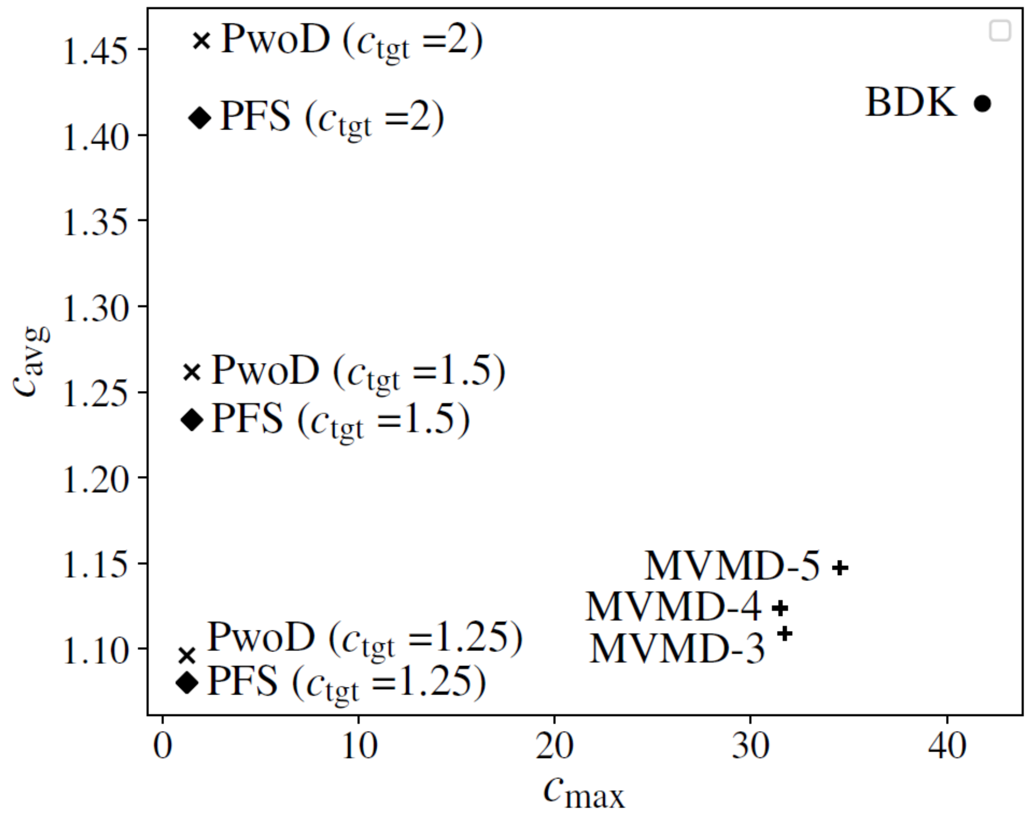


Padding overhead factors for each padding algorithm.

Evaluation: Autocomplete - $\mathbb{I}_\infty\left(\vec{S};\vec{Y}\right)$

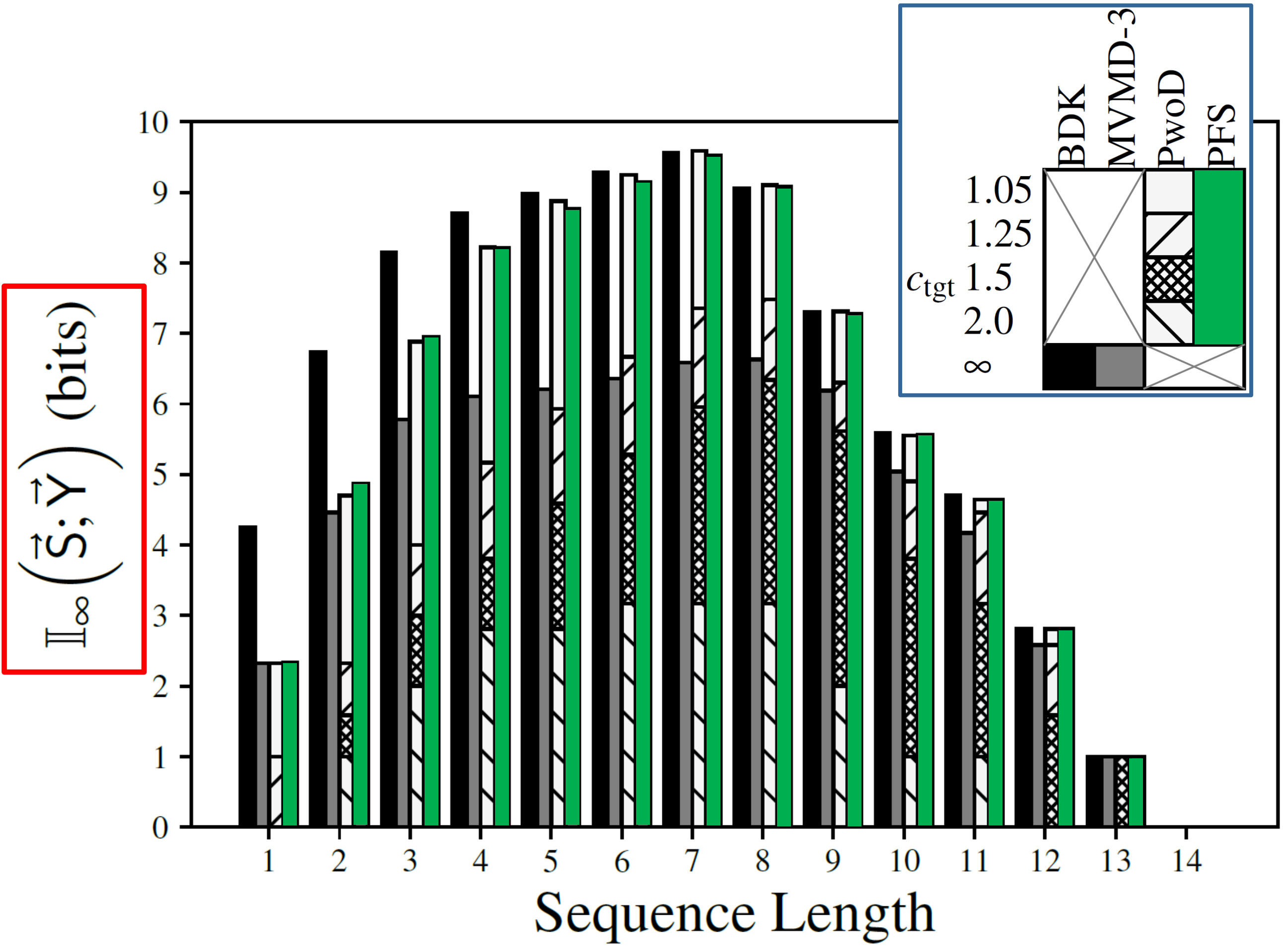


Comparing all algorithms using $\mathbb{I}_\infty\left(\vec{S};\vec{Y}\right)$ as the privacy metric.

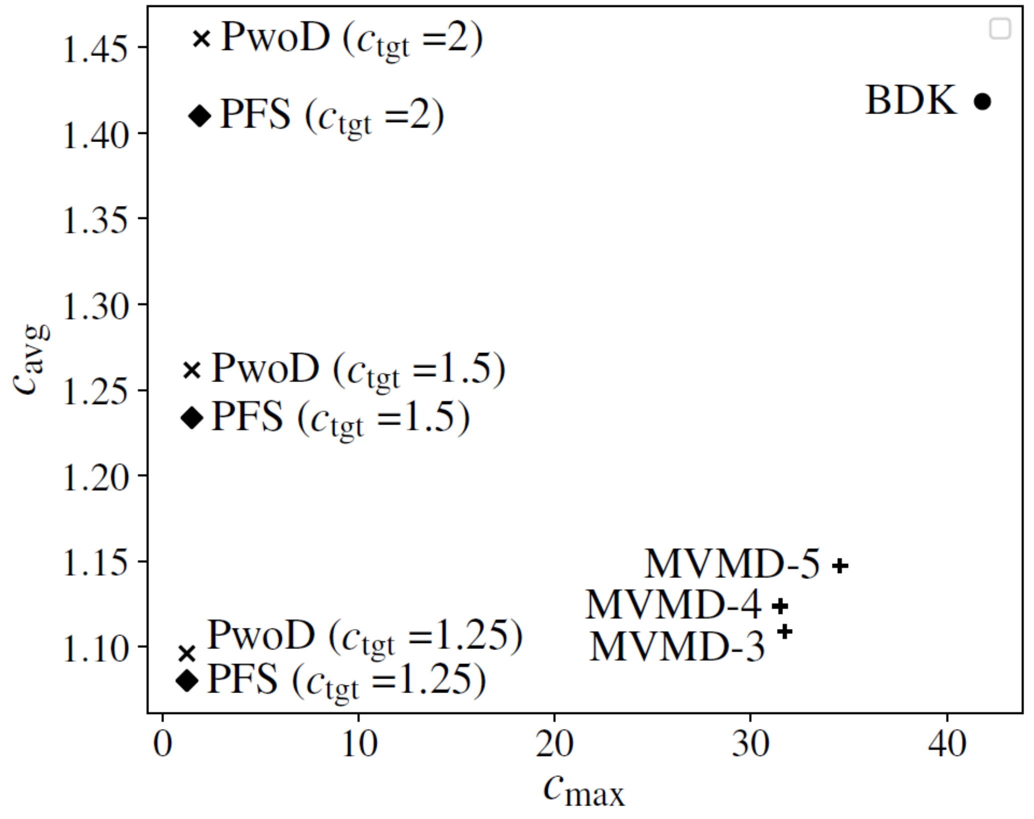


Padding overhead factors for each padding algorithm.

Evaluation: Autocomplete - $\mathbb{I}_\infty(\vec{S};\vec{Y})$

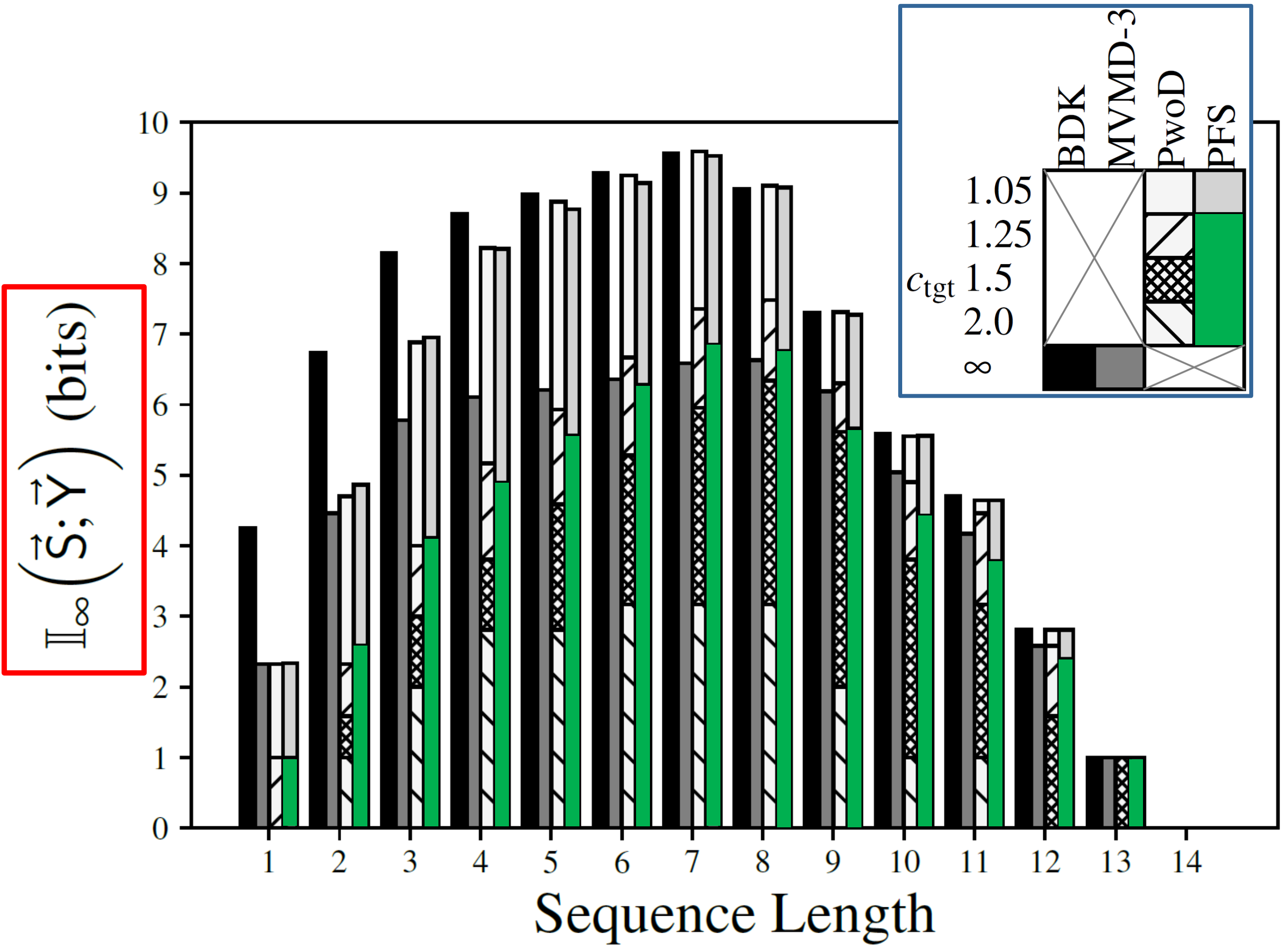


Comparing all algorithms using $\mathbb{I}_\infty(\vec{S};\vec{Y})$ as the privacy metric.

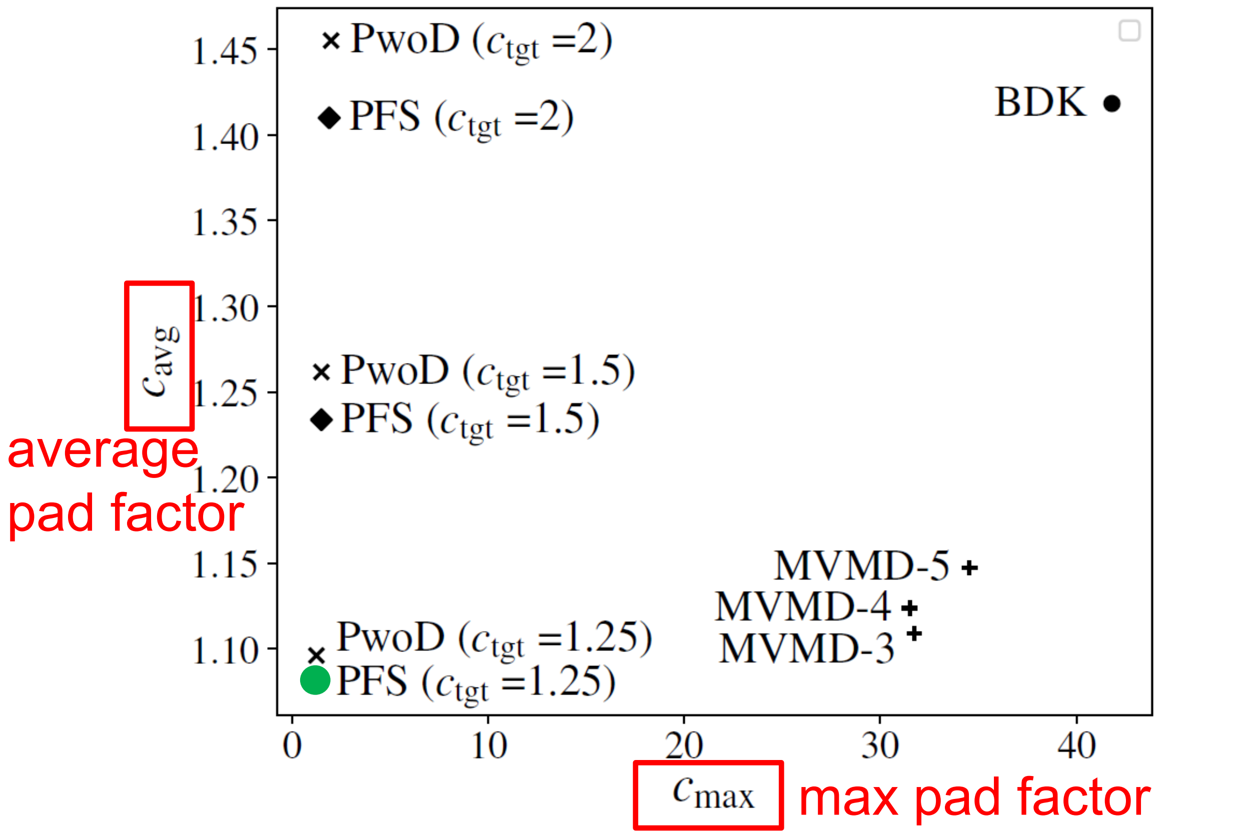


Padding overhead factors for each padding algorithm.

Evaluation: Autocomplete - $\mathbb{I}_\infty(\vec{S};\vec{Y})$

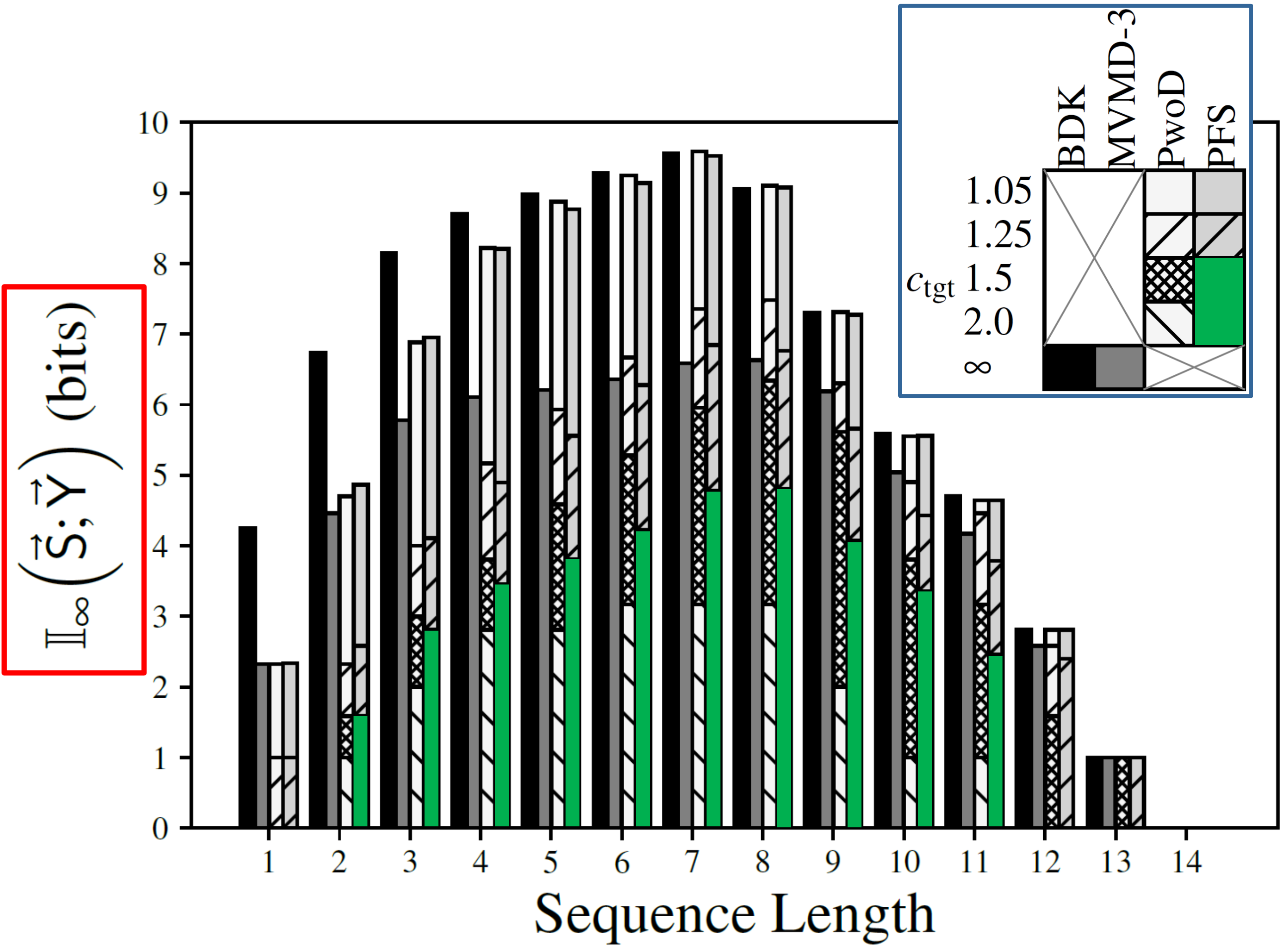


Comparing all algorithms using $\mathbb{I}_\infty(\vec{S};\vec{Y})$ as the privacy metric.

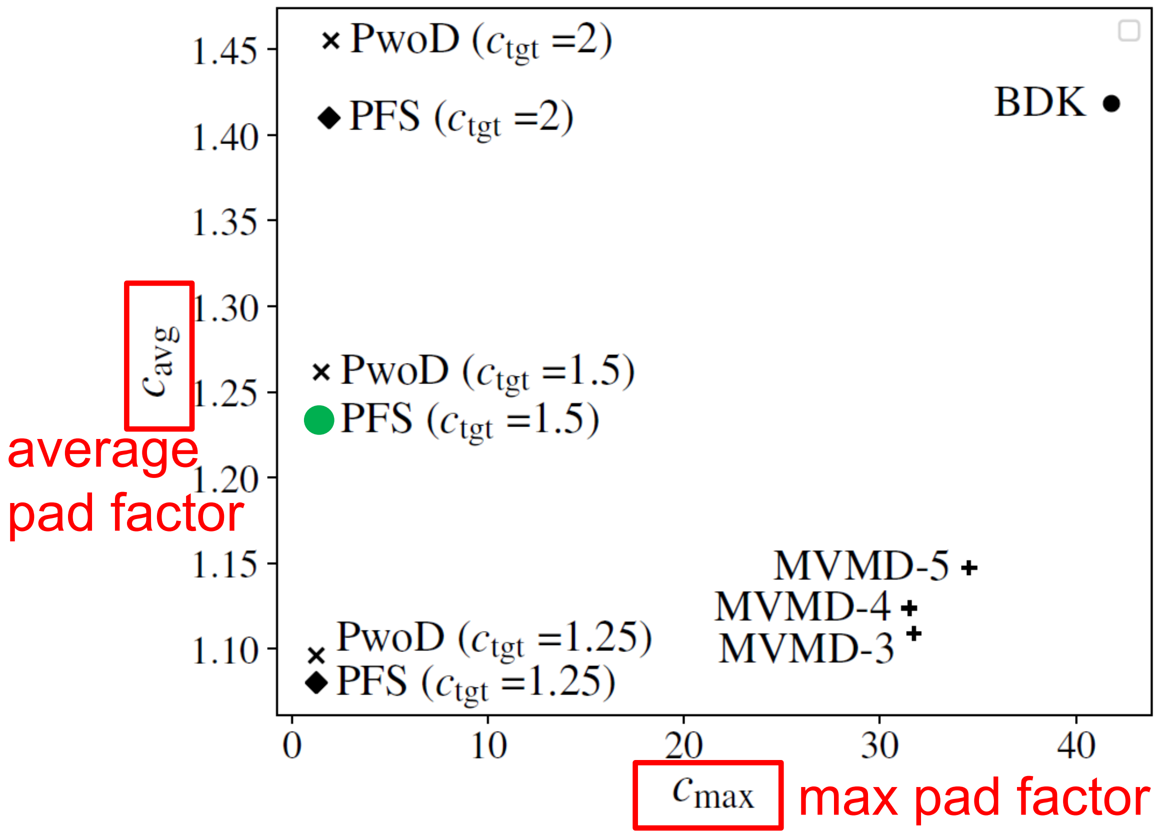


Padding overhead factors for each padding algorithm.

Evaluation: Autocomplete - $\mathbb{I}_\infty\left(\vec{S};\vec{Y}\right)$

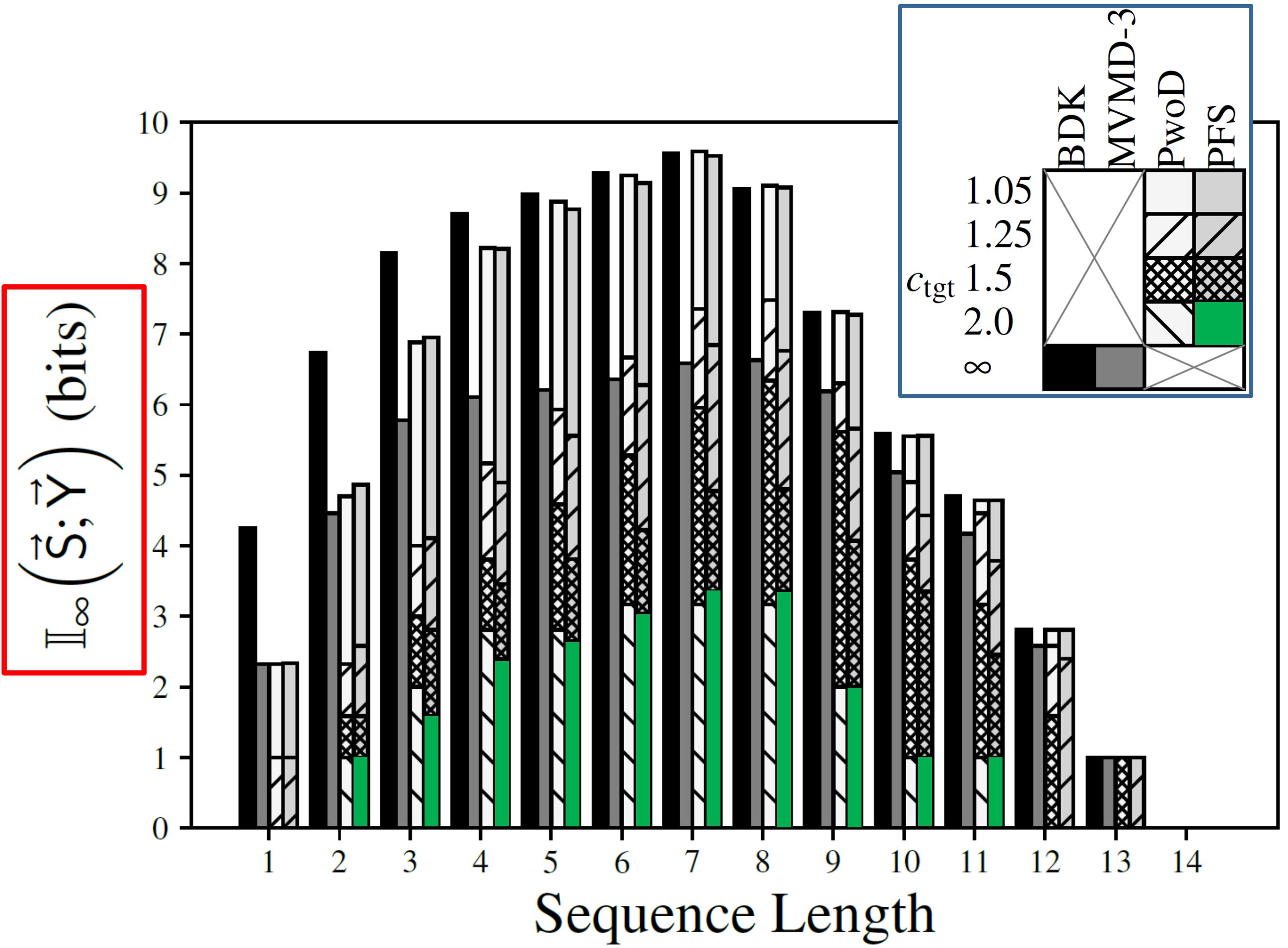


Comparing all algorithms using $\mathbb{I}_\infty\left(\vec{S};\vec{Y}\right)$ as the privacy metric.

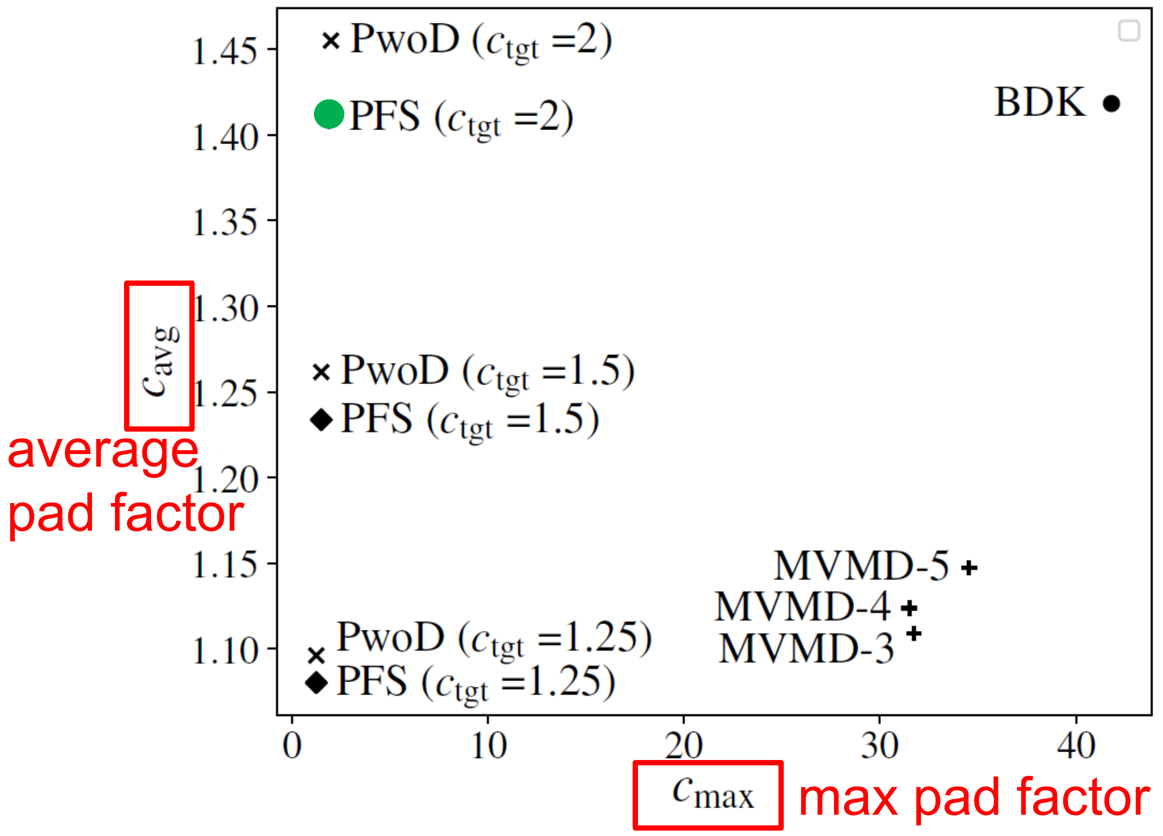


Padding overhead factors for each padding algorithm.

Evaluation: Autocomplete - $\mathbb{I}_\infty\left(\vec{S};\vec{Y}\right)$

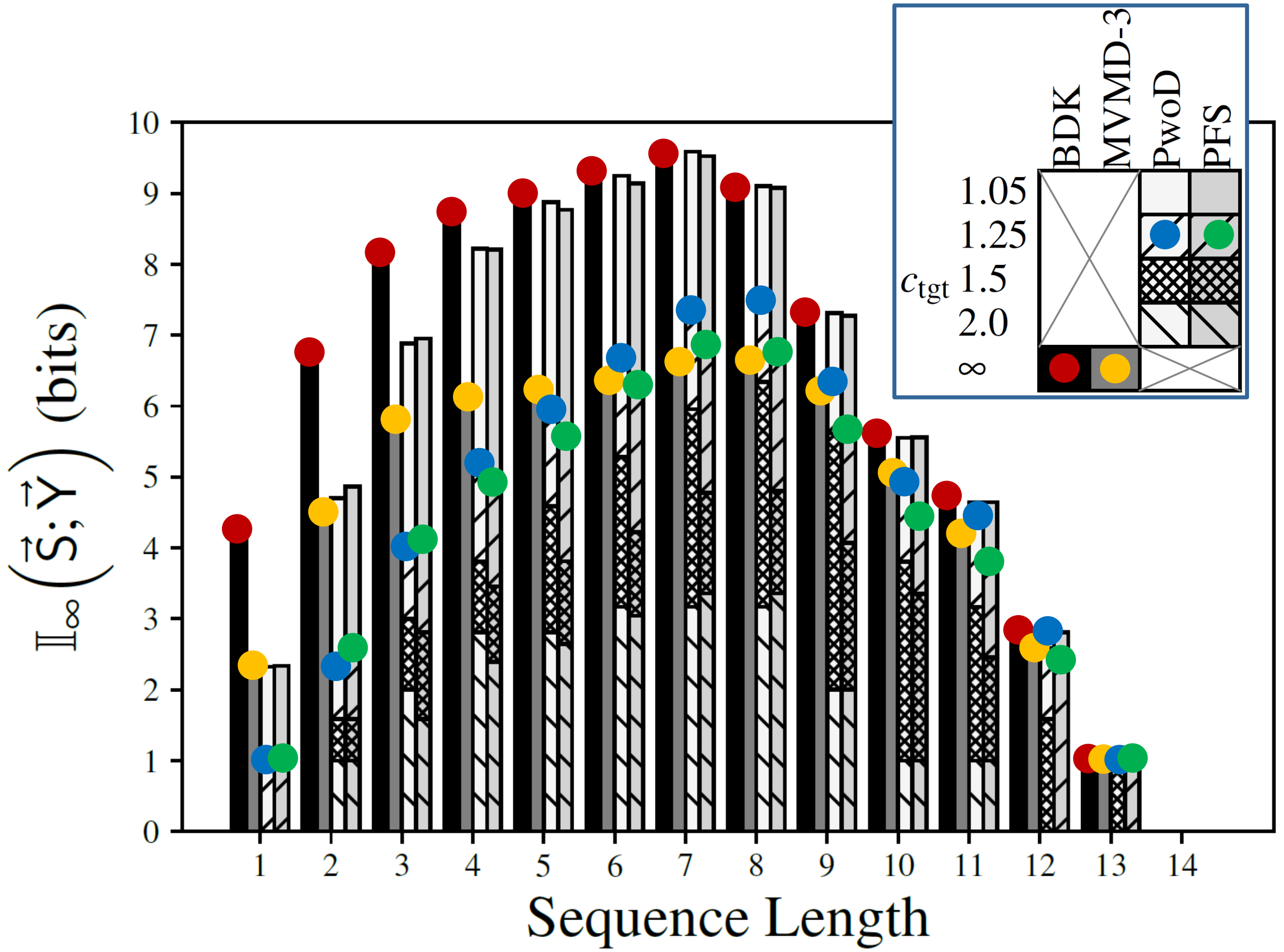


Comparing all algorithms using $\mathbb{I}_\infty\left(\vec{S};\vec{Y}\right)$ as the privacy metric.

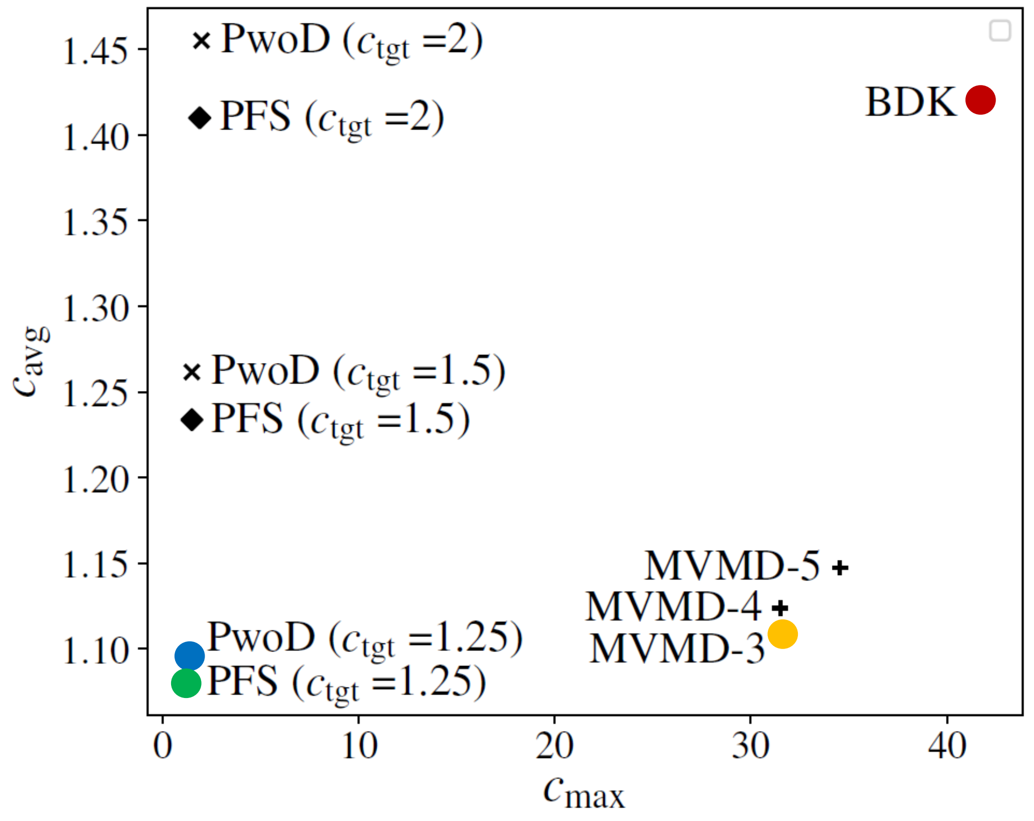


Padding overhead factors for each padding algorithm.

Evaluation: Autocomplete - $\mathbb{I}_\infty(\vec{S};\vec{Y})$



Comparing all algorithms using $\mathbb{I}_\infty(\vec{S};\vec{Y})$ as the privacy metric.



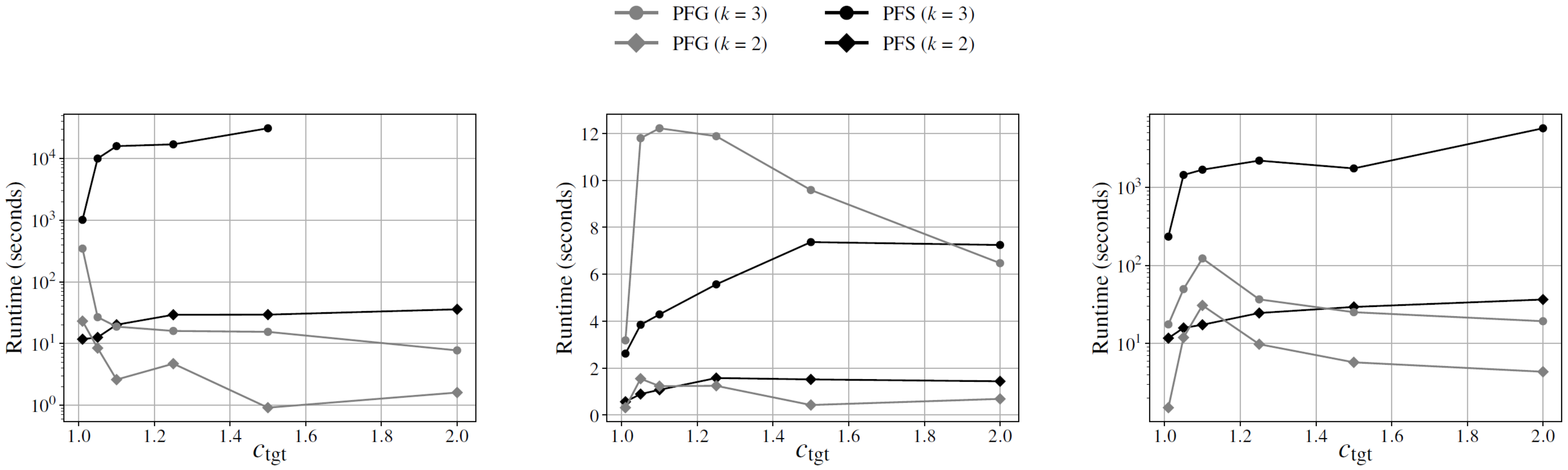
Padding overhead factors for each padding algorithm.

Algorithm: Padding For Graphs (PFG)

- Motivation: $\#\vec{S}$ increases exponentially as the sequence length grows
- Design: the same linear program used in PFS
- Inputs:
  - S = the set of objects
  - E = the "edges" between objects, i.e.,

    $$E = \left\{ (s, s') \,\middle|\, \exists \vec{s} \in \vec{S}, i \in [\mathrm{len}(\vec{s}) - 1] : \vec{s}_i = s \wedge \vec{s}_{i+1} = s' \right\}$$
  - c = max padding factor per object
  - k = an efficiency parameter (i.e., dimensionality reduction)
- Insight:
  - $\vec{S}$ is a subset of the walks in the graph = (S, E)

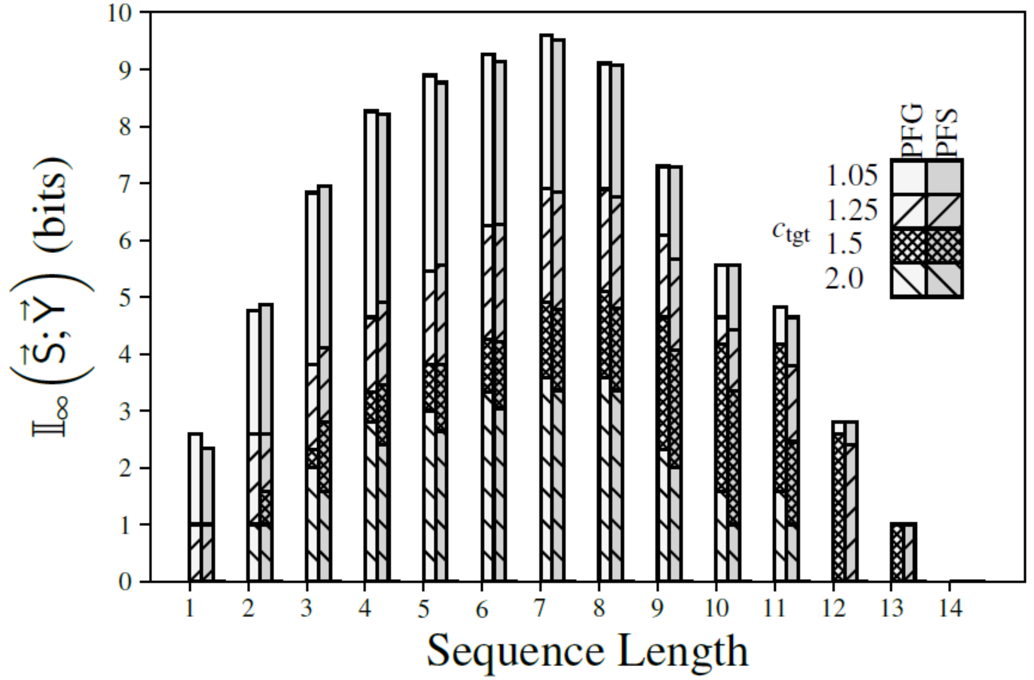
Evaluation: PFG vs. PFS - Runtime



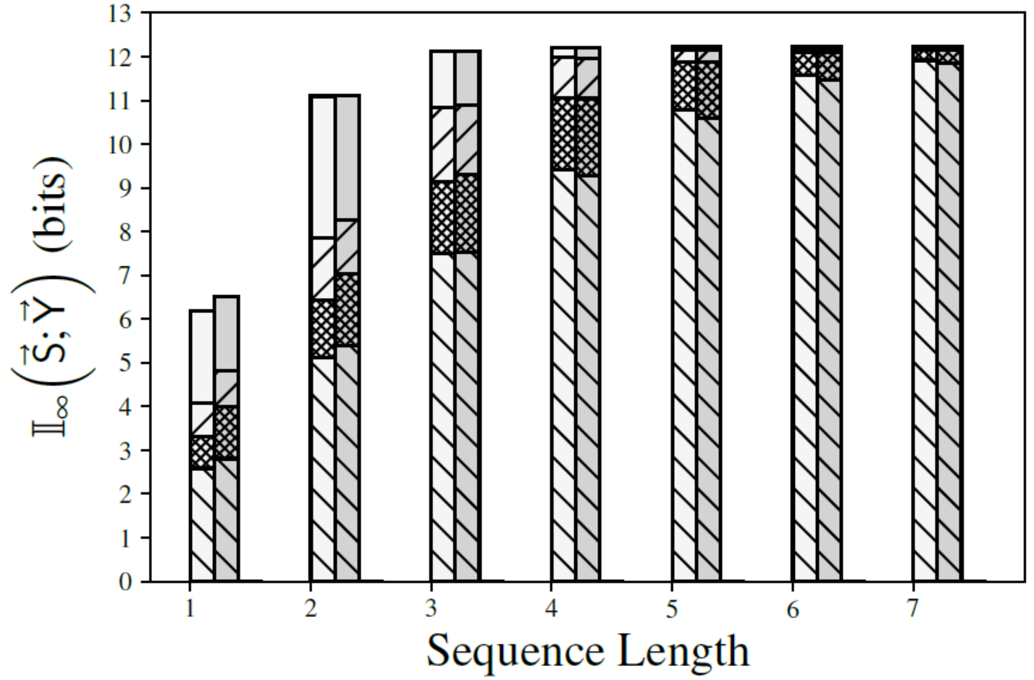
(a) Autocomplete (b) Linode (c) Wikipedia

Model runtime for PFS and PFG as a function of c_{tgt}.

Evaluation: PFG vs. PFS - $\mathbb{I}_\infty\left(\vec{S};\vec{Y}\right)$



(a) Autocomplete



(b) Wikipedia

Comparing PFG to PFS with $k = 2$.

Agenda

- Objective
- Padding for Independent Object Retrievals
- Padding for Dependent Object Retrievals
- Efficiently Computing Padding for Dependent Object Retrievals using Differential Evolution
- Questions

Algorithm: PFG-DE (Differential Evolution)

- Motivation: Design a PFG algorithm without using the LP framework
- Key Insight:

$$\mathbb{I}_\infty\left(\vec{\mathsf{S}};\vec{\mathsf{Y}}\right) \leq \log_2 \#\vec{Y}$$

$$\#\vec{Y}_{1\ldots m} \leq \frac{1}{2}\sum_{y\in Y}\left(d_{in}(y)^m + d_{out}(y)^m\right)^*$$

- Goal:
  - Use DE to produce a $\lceil\cdot\rceil$ that minimizes $\#\vec{Y}_G$
  - i.e., minimize the **number** of **vertices + edges** in the "padded graph"
- Notation:
  - d_{in} = in-degree of a vertex
  - d_{out} = out-degree
  - $\vec{S}_G$ = $S \cup E$ = $\vec{S}_{1\ldots 1} \cup \vec{S}_{1\ldots 2}$
  - $\vec{Y}_G$ = $\left\lceil \vec{S}_G \right\rceil$

* H. Täubig. The number of walks and degree powers in directed graphs. 2012.

Example $\vec{S}_G \Rightarrow \vec{Y}_G$

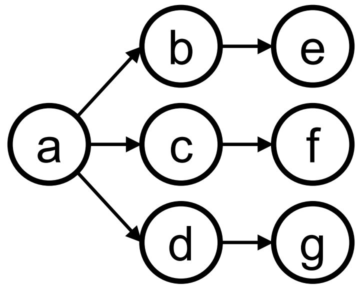
$\vec{S}_G$:

$S =$

a	b	c	d	e	f	g
100	105	109	110	113	114	115

$E = \{ (a,b), (a,c), (a,d), (b,e), (c,f), (d,g) \}$

$$\#\vec{S}_G = \#S + \#E$$
$$\#\vec{S}_G = 7 + 6 = 13$$



$$\vec{S}_{1\ldots3} \begin{cases} (a, b, e) \\ (a, c, f) \\ (a, d, g) \end{cases}$$

For C = 1.05, four example $\lceil \cdot \rceil$:

$\lceil S \rceil =$

105	105	109	115	115	115	115

$\lceil E \rceil = \{ (105,105), (105,109), (105,115), (109,115), (115,115) \}$

$$\#\vec{Y}_G = \#\lceil S \rceil + \#\lceil E \rceil$$
$$\#\vec{Y}_G = 3 + 5 = 8$$

$$\vec{Y}_{1\ldots3} \begin{cases} (105, 105, 115) \\ (105, 109, 115) \\ (105, 115, 115) \end{cases}$$

$\lceil S \rceil =$

105	105	114	114	114	114	115

$\lceil E \rceil = \{ (105,105), (105,114), (114,114), (114,115) \}$

$$\#\vec{Y}_G = \#\lceil S \rceil + \#\lceil E \rceil$$
$$\#\vec{Y}_G = 3 + 4 = 7$$

$$\vec{Y}_{1\ldots3} \begin{cases} (105, 105, 114) \\ (105, 114, 114) \\ (105, 114, 115) \end{cases}$$

$\lceil S \rceil =$

100	109	109	115	115	115	115

$\lceil E \rceil = \{ (100,109), (100,115), (109,115), (115,115) \}$

$$\#\vec{Y}_G = \#\lceil S \rceil + \#\lceil E \rceil$$
$$\#\vec{Y}_G = 3 + 4 = 7$$

$$\vec{Y}_{1\ldots3} \begin{cases} (100, 109, 115) \\ (100, 115, 115) \end{cases}$$

$\lceil S \rceil =$

100	110	110	110	115	115	115

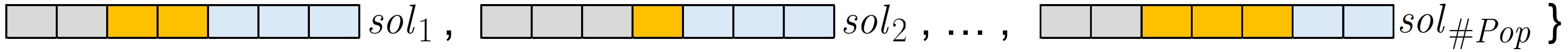
$\lceil E \rceil = \{ (100,110), (110,115) \}$

$$\#\vec{Y}_G = \#\lceil S \rceil + \#\lceil E \rceil$$
$$\#\vec{Y}_G = 3 + 2 = 5$$

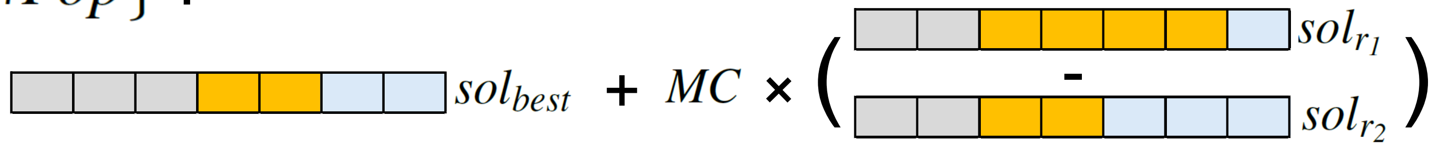
$$\vec{Y}_{1\ldots3} \{ (100, 110, 115)$$

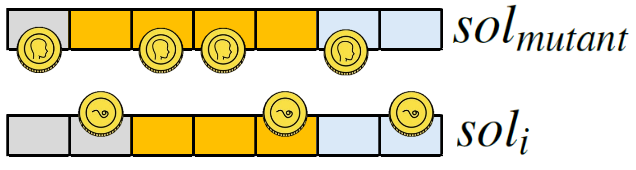
Differential Evolution – General Framework

$Pop = \{ sol_1 , sol_2 , \ldots , sol_{\#Pop} \}$



```
Loop:
    for $i \in \{1, \ldots, \#Pop\}$ :
        $sol_{mutant} = sol_{best} + MC \times ( sol_{r_1} - sol_{r_2} )$
        $sol_{trial} =$ combination of $sol_{mutant}$ and $sol_i$
        $sol_i = sol_{trial}$ if $f_{obj}(sol_{trial}) < f_{obj}(sol_i)$
    $sol_{best}$ is updated to be the current best solution
```





for each component, ● ~ U([0, 1)) …

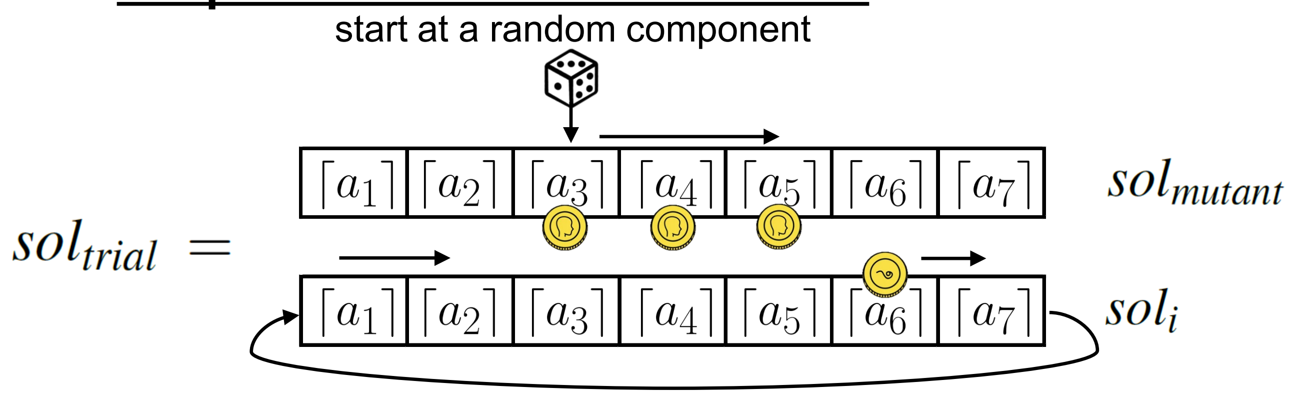
● less than CP , keep component from sol_{mutant}

○ else keep component from sol_i

- Notation:
  - Pop = population
  - sol = solution vector
  - MC = mutation constant
  - CP = crossover probability
  - f_{obj} = objective function

Differential Evolution – Our Key Design Choices

- sol Structure: Each sol is only a $\lceil \cdot \rceil$ for the "anchor sizes"
  - e.g., for $\boxed{100}\boxed{105}\boxed{109}\boxed{110}\boxed{113}\boxed{114}\boxed{115}$ and c=1.05 the anchor sizes are {100, 109, 115}
  - If sol_i = [105, 110, 115] then the full $\lceil \cdot \rceil$ = $\boxed{105}\boxed{105}\boxed{110}\boxed{110}\boxed{115}\boxed{115}\boxed{115}$
  - If sol_i = [100, 110, 115] then the full $\lceil \cdot \rceil$ = $\boxed{100}\boxed{110}\boxed{110}\boxed{110}\boxed{115}\boxed{115}\boxed{115}$
  - If sol_i = [100, 114, 115] then the full $\lceil \cdot \rceil$ = $\boxed{100}\boxed{105}\boxed{114}\boxed{114}\boxed{114}\boxed{114}\boxed{115}$
- Exponential Crossover:



start at a random component
$\lceil a_1 \rceil$ $\lceil a_2 \rceil$ $\lceil a_3 \rceil$ $\lceil a_4 \rceil$ $\lceil a_5 \rceil$ $\lceil a_6 \rceil$ $\lceil a_7 \rceil$
sol_{mutant}
$sol_{trial} =$
$\lceil a_1 \rceil$ $\lceil a_2 \rceil$ $\lceil a_3 \rceil$ $\lceil a_4 \rceil$ $\lceil a_5 \rceil$ $\lceil a_6 \rceil$ $\lceil a_7 \rceil$
sol_i

once a component in sol_i is selected, all remaining components come from sol_i

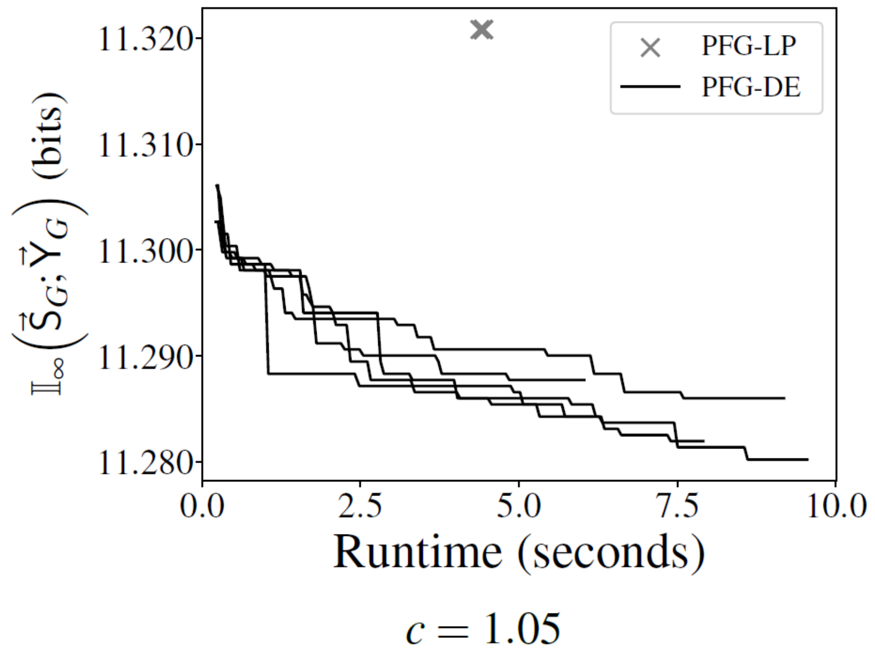
Preserves contiguous components in both sol_i and sol_{mutant}

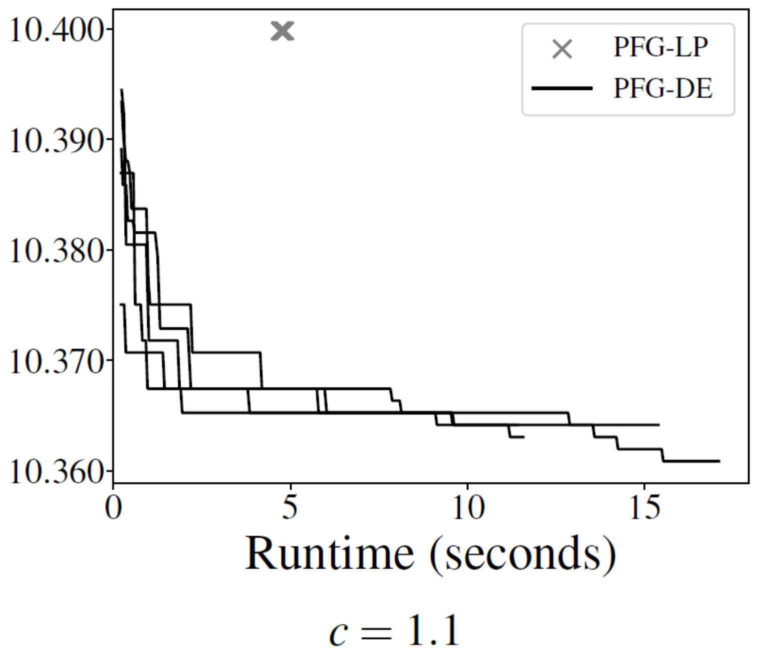
Evaluation: Datasets

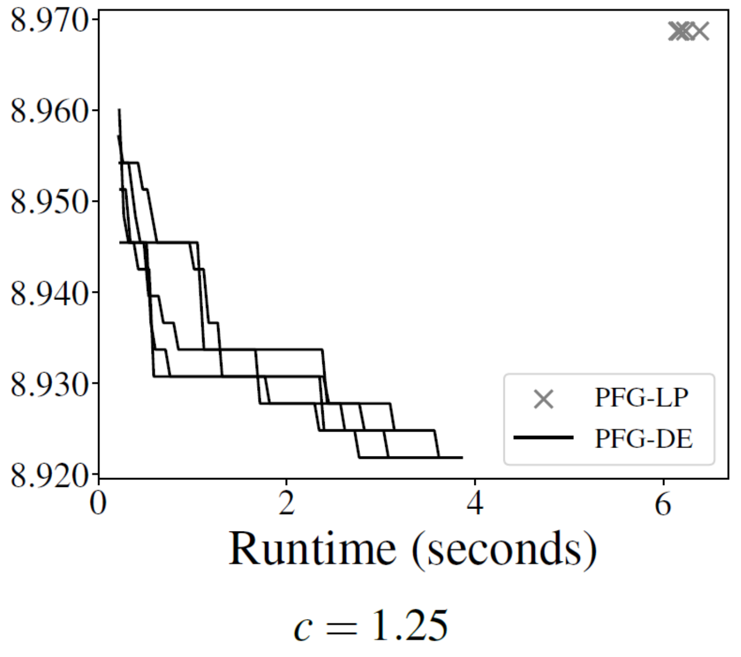
Statistic	Linode	Wikipedia	Netflix
$\#S$, i.e., $\#\vec{S}_{1...1}$	1,569	2,804	47,402
$\#E$, i.e., $\#\vec{S}_{1...2}$	13,692	14,996	130,809
$\#\vec{S}_{1...3}$	251,774	137,754	369,478
$\#\vec{S}_{1...4}$	3,615,730	1,286,227	1,050,929
$\#\vec{S}_{1...5}$	43,443,467	12,373,371	3,003,036

Number of simple paths at each sequence length.

PFG-DE vs. PFG-LP: $\mathbb{I}_\infty$ and Runtime (Linode)

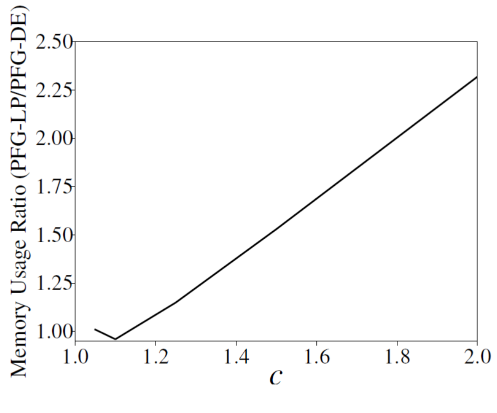




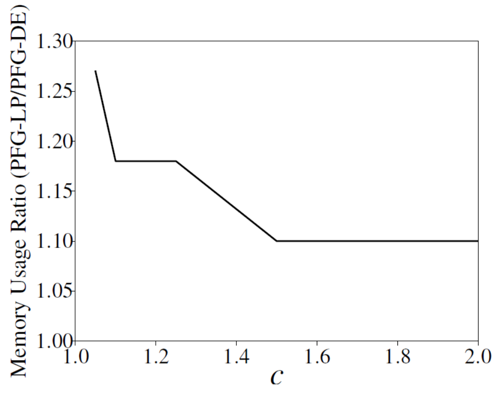


Comparing the two algorithms on $\mathbb{I}_\infty\left(\vec{S}_G;\vec{Y}_G\right)$.

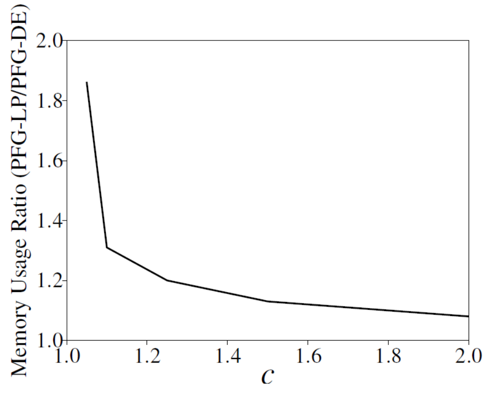
PFG-DE vs. PFG-LP: Memory Usage



(a) Linode

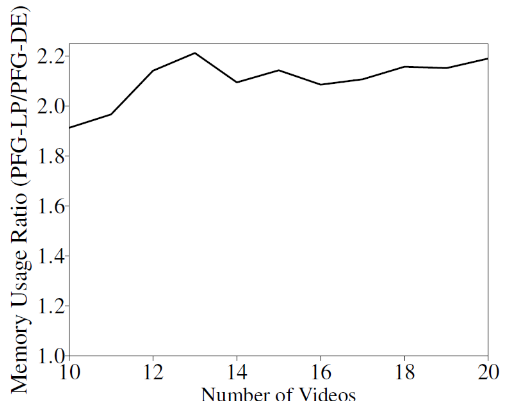


(b) Wikipedia

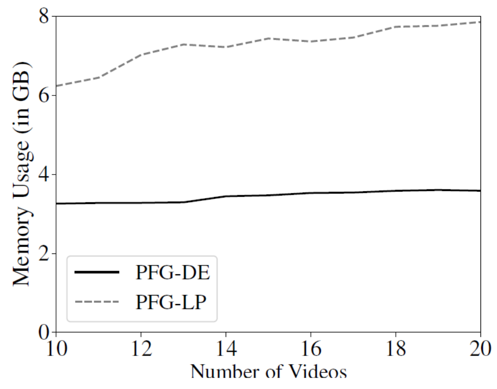


(c) Netflix

Comparing the total memory usage of PFG-DE to PFG-LP.



(a) Ratio



(b) Usage

Comparing the total memory usage of PFG-DE to PFG-LP as Netflix videos are added.

Thank You!

▪ Collaborator:	Pranay Jain
▪ Committee:	Michael K. Reiter, Advisor
	Jay Aikat
	Jasleen Kaur
	Ketan Mayer-Patel
	Fabian Monrose

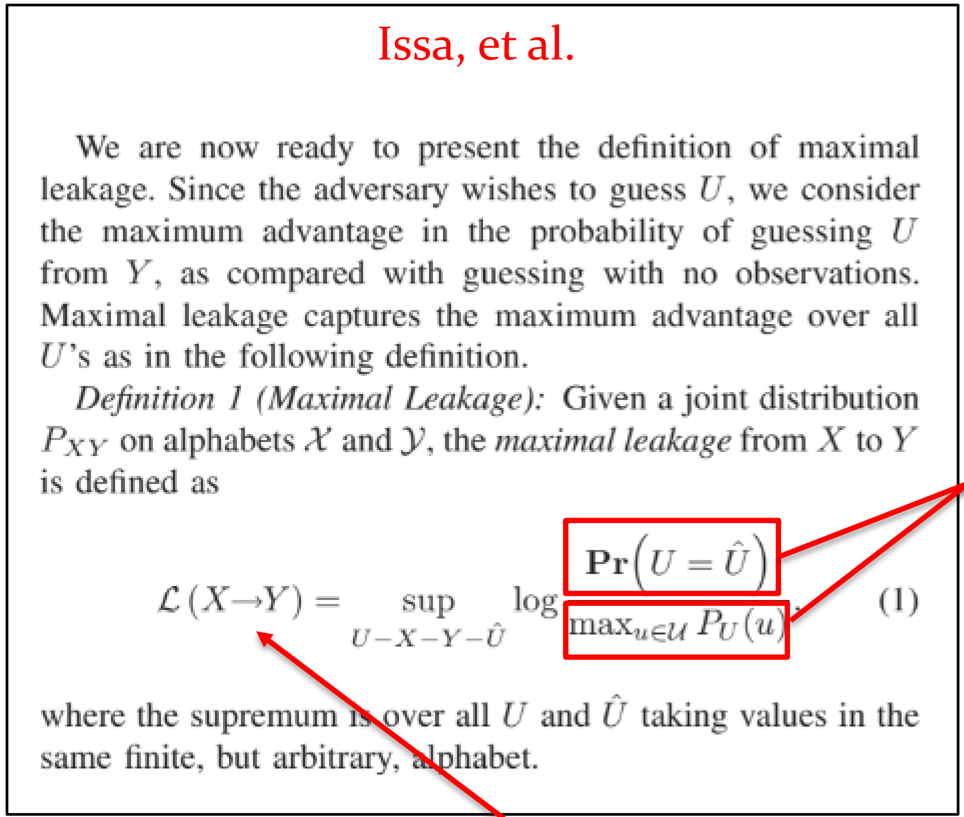
Summary

- **Objective:** Use padding to hide object sizes from a network observer
  - Minimize Metrics: $\mathbb{I}(S;Y)$, $\mathbb{I}_\infty(S;Y)$, or $\mathbb{I}_\infty\left(\vec{S};\vec{Y}\right)$
  - Constraints: (i) padding cannot exceed c x for any object and (ii) object must be served in full
- **Padding for Independent Object Retrievals**
  - Optimal algorithms for $\mathbb{I}(S;Y)$ – in both Per-Object and Per-Request settings – and for $\mathbb{I}_\infty(S;Y)$
  - Evaluated using two datasets: NodeJS packages and Unsplash Lite photos
- **Padding for Dependent Object Retrievals**
  - Linear program named Padding for Sequences (PFS)
  - Evaluated using three datasets: Linode documentation, Autocomplete results, and Wikipedia pages
- **Efficiently Computing Padding for Dep. Obj. Retrievals w/ Differential Evolution**
  - Leveraged Differential Evolution to efficiently create $\lceil \cdot \rceil$ when $\vec{S}$ is large
  - Evaluated using three datasets: Linode documentation, Wikipedia pages, and Netflix videos

Questions?

Back-up Slides

Maximal Leakage / Min-Capacity $\Big\}\ \mathbb{I}_\infty(S;Y)$



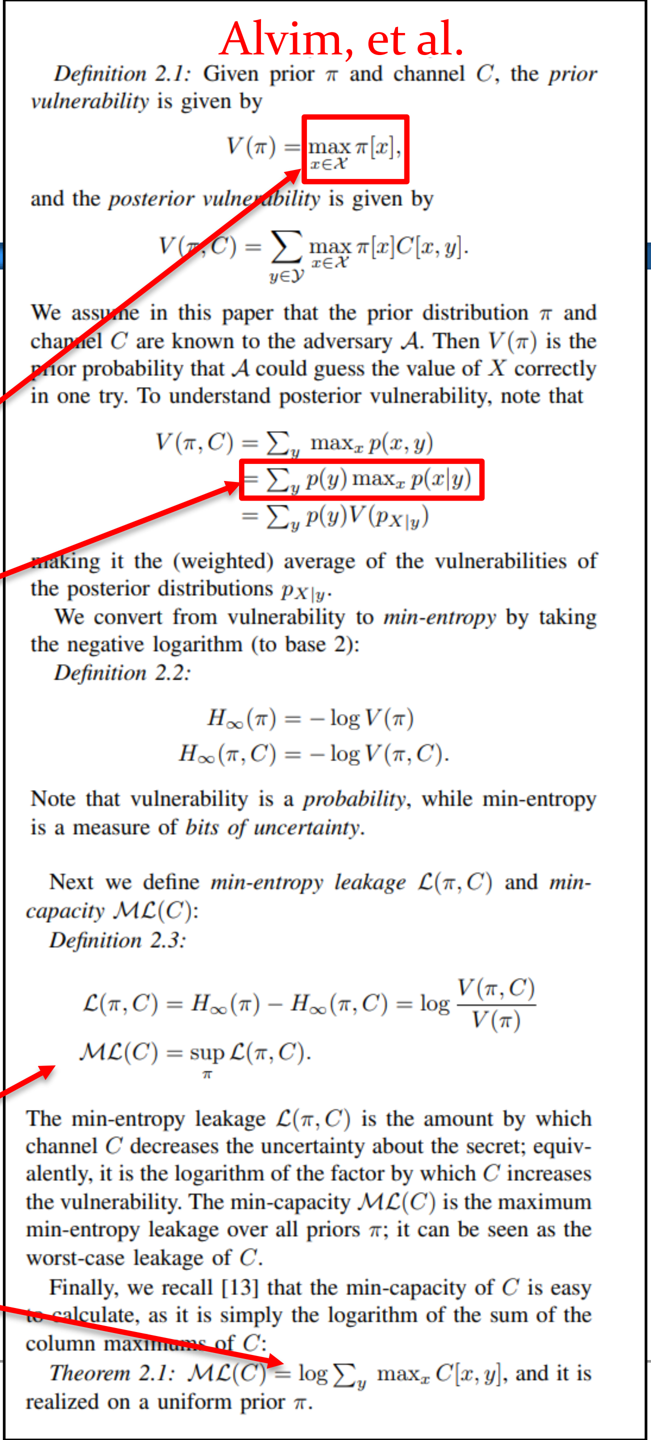
Issa, et al.

We are now ready to present the definition of maximal leakage. Since the adversary wishes to guess U, we consider the maximum advantage in the probability of guessing U from Y, as compared with guessing with no observations. Maximal leakage captures the maximum advantage over all U's as in the following definition.

Definition 1 (Maximal Leakage): Given a joint distribution P_{XY} on alphabets $\mathcal{X}$ and $\mathcal{Y}$, the *maximal leakage* from X to Y is defined as

$$\mathcal{L}(X \to Y) = \sup_{U - X - Y - \hat{U}} \log \frac{\boxed{\mathbf{Pr}\left(U = \hat{U}\right)}}{\boxed{\max_{u \in \mathcal{U}} P_U(u)}}, \qquad (1)$$

where the supremum is over all U and $\hat{U}$ taking values in the same finite, but arbitrary, alphabet.



Alvim, et al.

Definition 2.1: Given prior π and channel C, the *prior vulnerability* is given by

$$V(\pi) = \boxed{\max_{x \in \mathcal{X}} \pi[x],}$$

and the *posterior vulnerability* is given by

$$V(\pi, C) = \sum_{y \in \mathcal{Y}} \max_{x \in \mathcal{X}} \pi[x] C[x, y].$$

We assume in this paper that the prior distribution π and channel C are known to the adversary $\mathcal{A}$. Then $V(\pi)$ is the prior probability that $\mathcal{A}$ could guess the value of X correctly in one try. To understand posterior vulnerability, note that

$$\begin{aligned} V(\pi, C) &= \textstyle\sum_y \max_x p(x, y) \\ &\boxed{= \textstyle\sum_y p(y) \max_x p(x|y)} \\ &= \textstyle\sum_y p(y) V(p_{X|y}) \end{aligned}$$

making it the (weighted) average of the vulnerabilities of the posterior distributions $p_{X|y}$.

We convert from vulnerability to *min-entropy* by taking the negative logarithm (to base 2):

Definition 2.2:

$$\begin{aligned} H_\infty(\pi) &= -\log V(\pi) \\ H_\infty(\pi, C) &= -\log V(\pi, C). \end{aligned}$$

Note that vulnerability is a *probability*, while min-entropy is a measure of *bits of uncertainty*.

Next we define *min-entropy leakage* $\mathcal{L}(\pi, C)$ and *min-capacity* $\mathcal{ML}(C)$:

Definition 2.3:

$$\mathcal{L}(\pi, C) = H_\infty(\pi) - H_\infty(\pi, C) = \log \frac{V(\pi, C)}{V(\pi)}$$

$$\mathcal{ML}(C) = \sup_\pi \mathcal{L}(\pi, C).$$

The min-entropy leakage $\mathcal{L}(\pi, C)$ is the amount by which channel C decreases the uncertainty about the secret; equivalently, it is the logarithm of the factor by which C increases the vulnerability. The min-capacity $\mathcal{ML}(C)$ is the maximum min-entropy leakage over all priors π; it can be seen as the worst-case leakage of C.

Finally, we recall [13] that the min-capacity of C is easy to calculate, as it is simply the logarithm of the sum of the column maximums of C:

Theorem 2.1: $\mathcal{ML}(C) = \log \sum_y \max_x C[x, y]$, and it is realized on a uniform prior π.

Maximal Leakage = Min-Capacity = $\mathbb{I}_\infty$

POP Proposition

Proposition 1. *Let f be a function that is defined on the interval $R \subseteq \mathbb{R}$, and that has a negative second derivative. For all $z, z' \in R$ such that $z \leq z'$ and for any $\varepsilon > 0$ such that $z - \varepsilon \in R$ and $z' + \varepsilon \in R$:*

$$f(z) + f\left(z'\right) > f(z-\varepsilon) + f\left(z' + \varepsilon\right)$$

Proof. Since a negative second derivative implies a decreasing first derivative,

$$\frac{f(z) - f(z-\varepsilon)}{z - (z-\varepsilon)} > \frac{f(z'+\varepsilon) - f(z')}{(z'+\varepsilon) - z'}$$

and the result follows by rearranging terms. $\square$

Blahut-Arimoto

$$v_{t+1}(y,s) = \frac{u_t(y)\exp(-\beta \times D(s,y))}{\sum_{y'} u_t(y')\exp(-\beta \times D(s,y'))}$$

$$= \begin{cases} \frac{u_t(y)}{\sum_{y':D(s,y')=0} u_t(y')} & \text{if } D(s,y) = 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.4}$$

$$u_{t+1}(y) = \sum_{s \in S} \mathbb{P}(\mathsf{S} = s) v_{t+1}(y,s)$$

(3.4) holds since in our case, $D(s,y) \in \{0,\infty\}$ for all s, y.

D-ALPaCA

D-ALPaCA Cherubin, et al. [17] proposed padding algorithms to defend against website fingerprinting attacks and, so, that seek to address leakage resulting from the retrieval of objects hyperlinked in a webpage, subject to padding overhead constraints. Distilled down to our simpler scenario, though, their padding algorithms result in natural contenders for comparison. One of these, called D-ALPaCA, is deterministic and so is suitable as a per-object padding algorithm. In brief, D-ALPaCA sets $\lceil \mathsf{obj}_s \rceil$ to be the smallest multiple of σ that is $\geq |\mathsf{obj}_s|$, where σ is an input parameter. For our setting, we set $\sigma = \mathsf{floor}\big((c-1) \times |\mathsf{obj}_{s_{\min}}|\big)$, where $\mathsf{floor} : \mathbb{R} \to \mathbb{N}$ is the floor function and where $s_{\min}$ is the identifier of the smallest object in the object store. This, then, ensures that D-ALPaCA does not violate c for any $s \in S$. Note that D-ALPaCA is insensitive to the distribution of S.

P-ALPaCA

P-ALPaCA Cherubin, et al. [17] also proposed a randomized algorithm called P-ALPaCA that is suitable as a per-request padding algorithm. When applied to our setting, P-ALPaCA pads the response to a request s so that

$$\begin{aligned}\mathbb{P}(\lceil \mathsf{obj}_s \rceil = y) &= \mathbb{P}\big(|\mathsf{obj}_\mathsf{S}| = y \;\big|\; |\mathsf{obj}_s| \leq |\mathsf{obj}_\mathsf{S}| \leq c \times |\mathsf{obj}_s|\big) \\ &= \frac{\sum_{s':|\mathsf{obj}_{s'}|=y} \mathbb{P}(\mathsf{S} = s')}{\sum_{s':|\mathsf{obj}_s| \leq |\mathsf{obj}_{s'}| \leq c \times |\mathsf{obj}_s|} \mathbb{P}(\mathsf{S} = s')}\end{aligned}$$

for each y, $|\mathsf{obj}_s| \leq y \leq c \times |\mathsf{obj}_s|$. In particular, the most probable padded size for obj_s is the most probable unpadded size in the interval $[|\mathsf{obj}_s|, c \times |\mathsf{obj}_s|]$.

Padmé

4.4 Padmé

We now describe our padding scheme PADMÉ, which limits information leakage about the length of the plaintext for wide range of encrypted data sizes. Similarly to the previous strawman, PADMÉ also asymptotically leaks $O(\log\log M)$ bits of information, but its overhead is much lower (at most 12% and decreasing with L).

Intuition. In NEXTP2, any permissible padded length L has the form $L = 2^n$. We can therefore represent L as a binary floating-point number with a $\lfloor \log_2 n \rfloor + 1$-bit exponent and a mantissa of zero, *i.e.*, no fractional bits.

In PADMÉ, we similarly represent a permissible padded length as a binary floating-point number, but we allow a non-zero mantissa at most as long as the exponent (see Figure 6). This approach doubles the number of bits used to represent an allowed padded length – hence doubling absolute leakage via length – but allows for more fine-grained buckets, reducing overhead. PADMÉ asymptotically leaks the same number of bits as NEXTP2, differing only by a constant factor of 2, but reduces space overhead by almost 10× (from +100% to +12%). More importantly, the multiplicative expansion overhead decreases with L (see Figure 7).

Algorithm 1: PADMÉ

```
Data: length of content $L$
Result: length of padded content $L'$
$E \leftarrow \lfloor \log_2 L \rfloor$          // $L$'s floating-point exponent
$S \leftarrow \lfloor \log_2 E \rfloor + 1$      // # of bits to represent $E$
$z \leftarrow E - S$                  // # of low bits to set to 0
$m \leftarrow (1 \ll z) - 1$             // mask of $z$ 1's in LSB
    // round up using mask $m$ to clear last $z$ bits
$L' \leftarrow (L + m)$ & $\sim m$
```

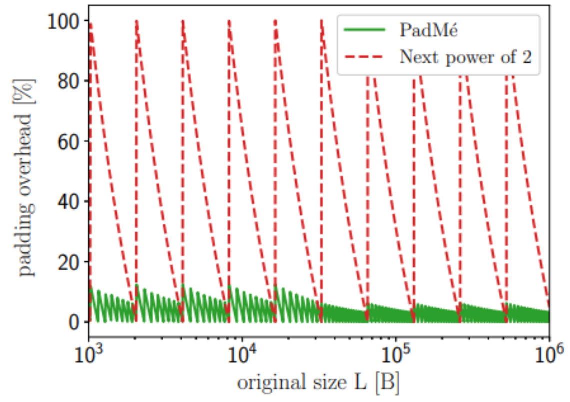
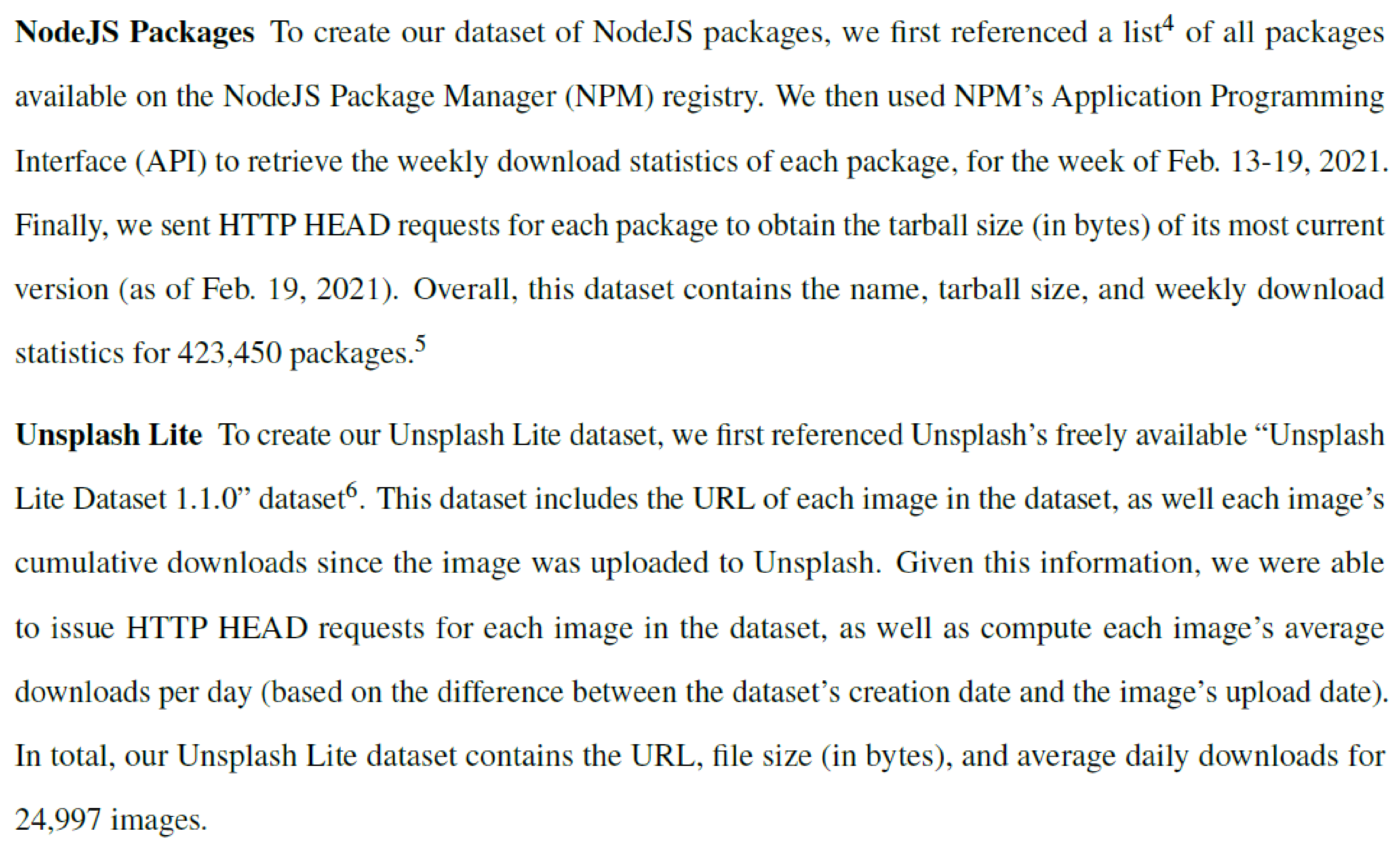


Fig. 7. Maximum multiplicative expansion overhead with respect to the plaintext size L. The naïve approach to pad to the next power of two has a constant maximum overhead of 100%, whereas PADMÉ's maximum overhead decreases with L, following $\frac{1}{2\log_2 L}$.

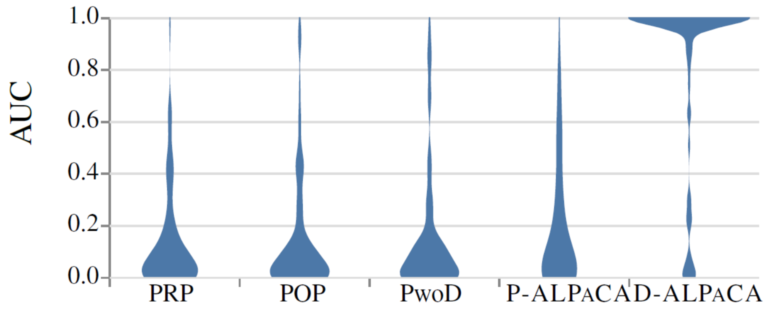
NodeJS and Unsplash Lite datasets



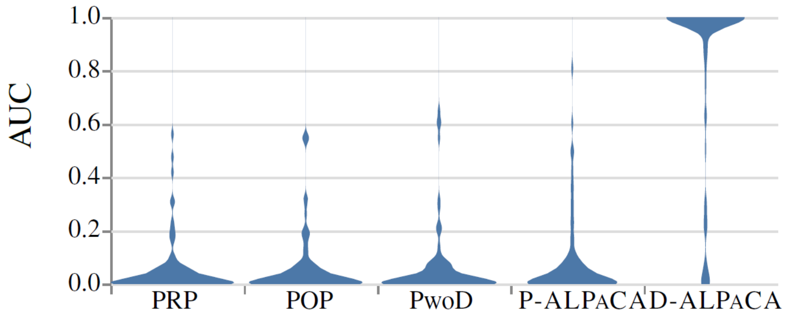
NodeJS Packages To create our dataset of NodeJS packages, we first referenced a list[4] of all packages available on the NodeJS Package Manager (NPM) registry. We then used NPM's Application Programming Interface (API) to retrieve the weekly download statistics of each package, for the week of Feb. 13-19, 2021. Finally, we sent HTTP HEAD requests for each package to obtain the tarball size (in bytes) of its most current version (as of Feb. 19, 2021). Overall, this dataset contains the name, tarball size, and weekly download statistics for 423,450 packages.[5]

Unsplash Lite To create our Unsplash Lite dataset, we first referenced Unsplash's freely available "Unsplash Lite Dataset 1.1.0" dataset[6]. This dataset includes the URL of each image in the dataset, as well each image's cumulative downloads since the image was uploaded to Unsplash. Given this information, we were able to issue HTTP HEAD requests for each image in the dataset, as well as compute each image's average downloads per day (based on the difference between the dataset's creation date and the image's upload date). In total, our Unsplash Lite dataset contains the URL, file size (in bytes), and average daily downloads for 24,997 images.

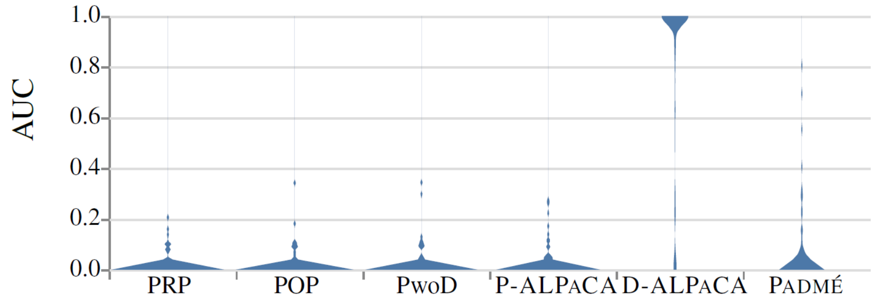
AUC: NodeJS



(a) $c = 1.01$

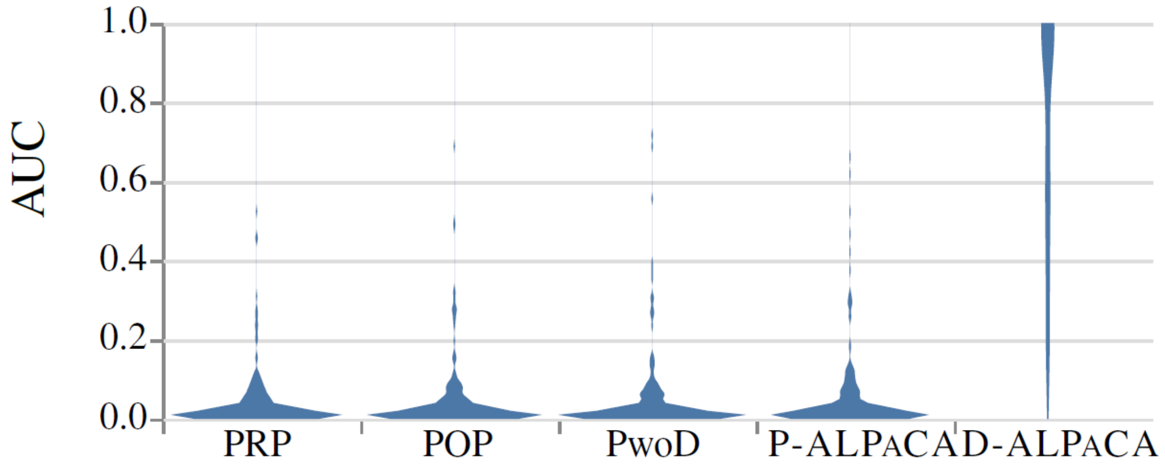


(b) $c = 1.03$

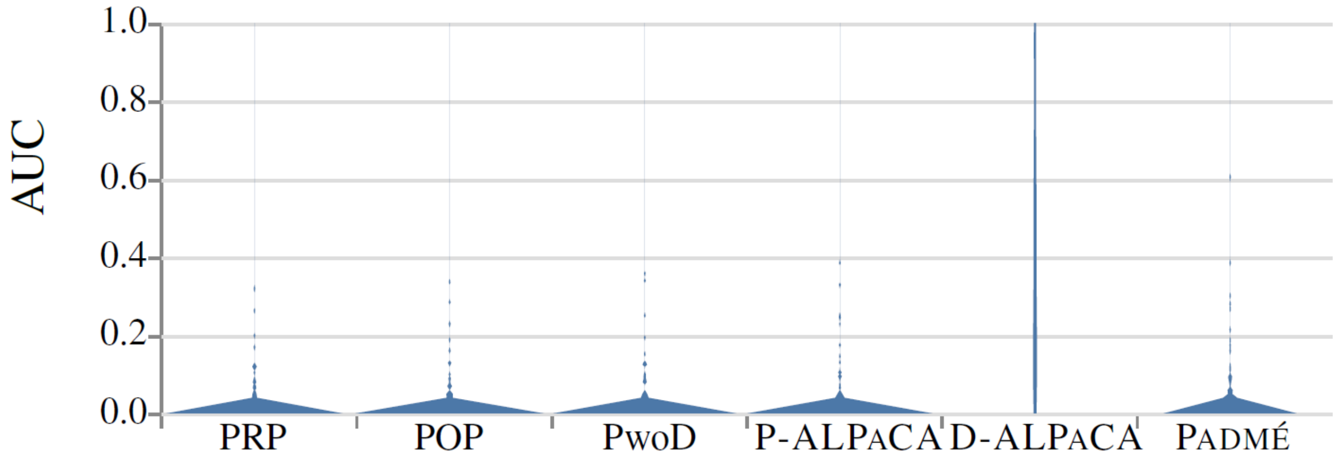


(c) $c = 1.09$

AUC: Unsplash Lite

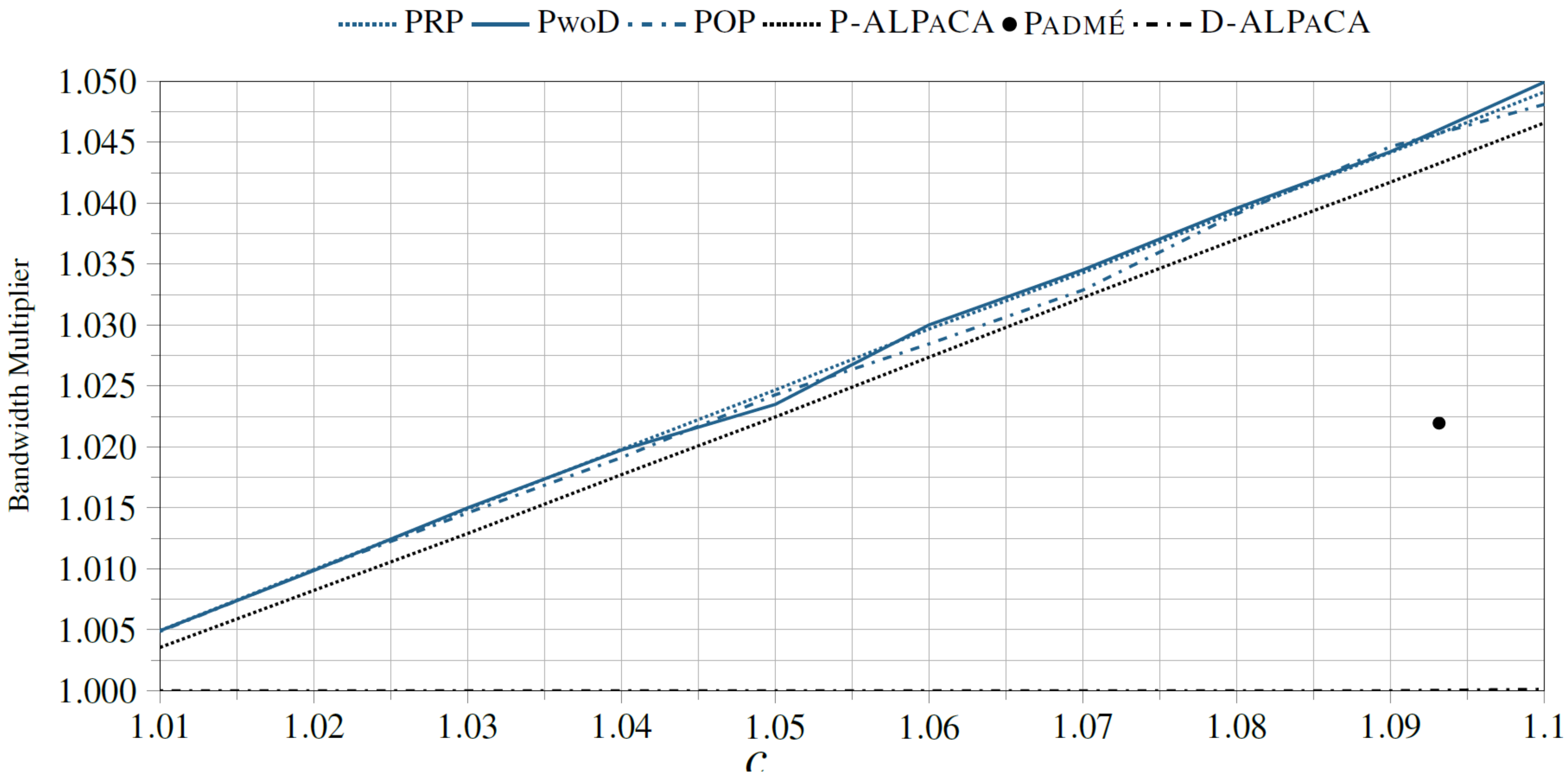


(a) $c = 1.01$

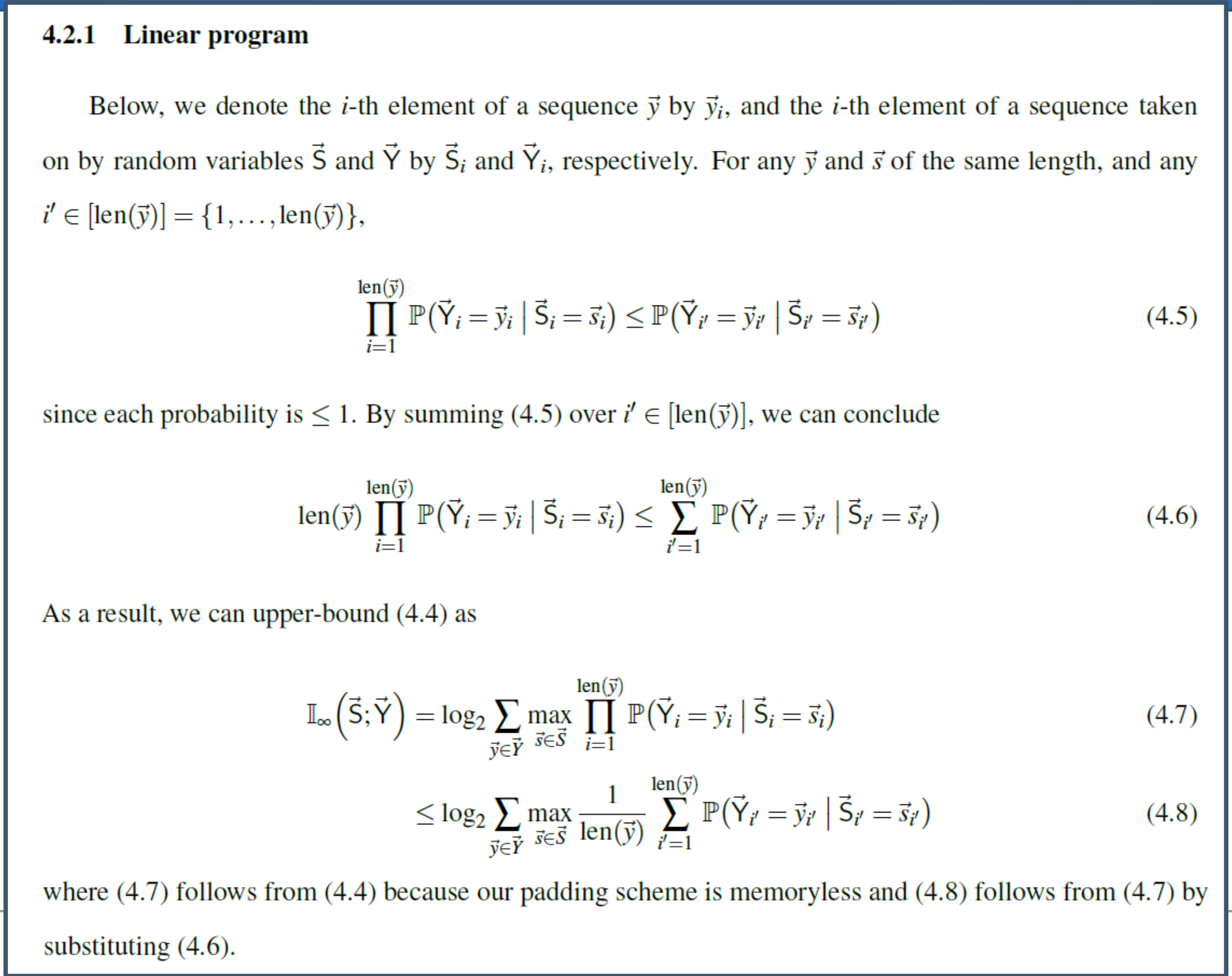


(b) $c = 1.03$

Bandwidth Increase: NodeJS



PFS: Linear Program (1 of 2)



4.2.1 Linear program

Below, we denote the i-th element of a sequence $\vec{y}$ by $\vec{y}_i$, and the i-th element of a sequence taken on by random variables $\vec{\mathsf{S}}$ and $\vec{\mathsf{Y}}$ by $\vec{\mathsf{S}}_i$ and $\vec{\mathsf{Y}}_i$, respectively. For any $\vec{y}$ and $\vec{s}$ of the same length, and any $i' \in [\mathrm{len}(\vec{y})] = \{1, \ldots, \mathrm{len}(\vec{y})\}$,

$$\prod_{i=1}^{\mathrm{len}(\vec{y})} \mathbb{P}(\vec{\mathsf{Y}}_i = \vec{y}_i \mid \vec{\mathsf{S}}_i = \vec{s}_i) \le \mathbb{P}(\vec{\mathsf{Y}}_{i'} = \vec{y}_{i'} \mid \vec{\mathsf{S}}_{i'} = \vec{s}_{i'}) \tag{4.5}$$

since each probability is ≤ 1. By summing (4.5) over $i' \in [\mathrm{len}(\vec{y})]$, we can conclude

$$\mathrm{len}(\vec{y}) \prod_{i=1}^{\mathrm{len}(\vec{y})} \mathbb{P}(\vec{\mathsf{Y}}_i = \vec{y}_i \mid \vec{\mathsf{S}}_i = \vec{s}_i) \le \sum_{i'=1}^{\mathrm{len}(\vec{y})} \mathbb{P}(\vec{\mathsf{Y}}_{i'} = \vec{y}_{i'} \mid \vec{\mathsf{S}}_{i'} = \vec{s}_{i'}) \tag{4.6}$$

As a result, we can upper-bound (4.4) as

$$\mathbb{I}_\infty\left(\vec{\mathsf{S}}; \vec{\mathsf{Y}}\right) = \log_2 \sum_{\vec{y} \in \vec{Y}} \max_{\vec{s} \in \vec{S}} \prod_{i=1}^{\mathrm{len}(\vec{y})} \mathbb{P}(\vec{\mathsf{Y}}_i = \vec{y}_i \mid \vec{\mathsf{S}}_i = \vec{s}_i) \tag{4.7}$$

$$\le \log_2 \sum_{\vec{y} \in \vec{Y}} \max_{\vec{s} \in \vec{S}} \frac{1}{\mathrm{len}(\vec{y})} \sum_{i'=1}^{\mathrm{len}(\vec{y})} \mathbb{P}(\vec{\mathsf{Y}}_{i'} = \vec{y}_{i'} \mid \vec{\mathsf{S}}_{i'} = \vec{s}_{i'}) \tag{4.8}$$

where (4.7) follows from (4.4) because our padding scheme is memoryless and (4.8) follows from (4.7) by substituting (4.6).

PFS: Linear Program (2 of 2)

$$\text{minimize} \sum_{\vec{y} \in \vec{Y}} \Pi^{\vec{y}} \quad \text{subject to}$$

$$\sum_{y \in Y_s} \pi_s^y = 1 \qquad \forall s \in S$$

$$\pi_s^y \geq 0 \qquad \forall s \in S, y \in Y_s$$

$$\pi_s^y \leq 1 \qquad \forall s \in S, y \in Y_s$$

$$\Pi^{\vec{y}} \geq \frac{1}{\text{len}(\vec{y})} \sum_{i=1}^{\text{len}(\vec{y})} \pi_{\vec{s}_i}^{\vec{y}_i} \qquad \forall \vec{s} \in \vec{S}, \vec{y} \in \vec{Y}_{\vec{s}}$$

$$\pi_s^y \Rightarrow \mathbb{P}\left(\mathsf{Y} = y \,\middle|\, \mathsf{S} = s\right)$$

$$\Pi^{\vec{y}} \Rightarrow \max_{\vec{s} \in \vec{S}} \frac{1}{\text{len}(\vec{y})} \Sigma_{i'=1}^{\text{len}(\vec{y})} \mathbb{P}\left(\vec{\mathsf{Y}}_{i'} = \vec{y}_{i'} \,\middle|\, \vec{\mathsf{S}}_{i'} = \vec{s}_{i'}\right)$$

PFS: Dataset Notation

$$\{\mathsf{obj}_s\}_{s \in S}$$

$$\vec{S}$$

$$\mathbb{P}(\mathsf{S} = s)$$

$$\mathbb{P}\left(\vec{\mathsf{S}}_{1\ldots i} = \vec{s}_{1\ldots i}\right)$$

$$\mathbb{P}\left(\vec{\mathsf{S}}_{i+1} = s' \mid \vec{\mathsf{S}}_i = s\right)$$

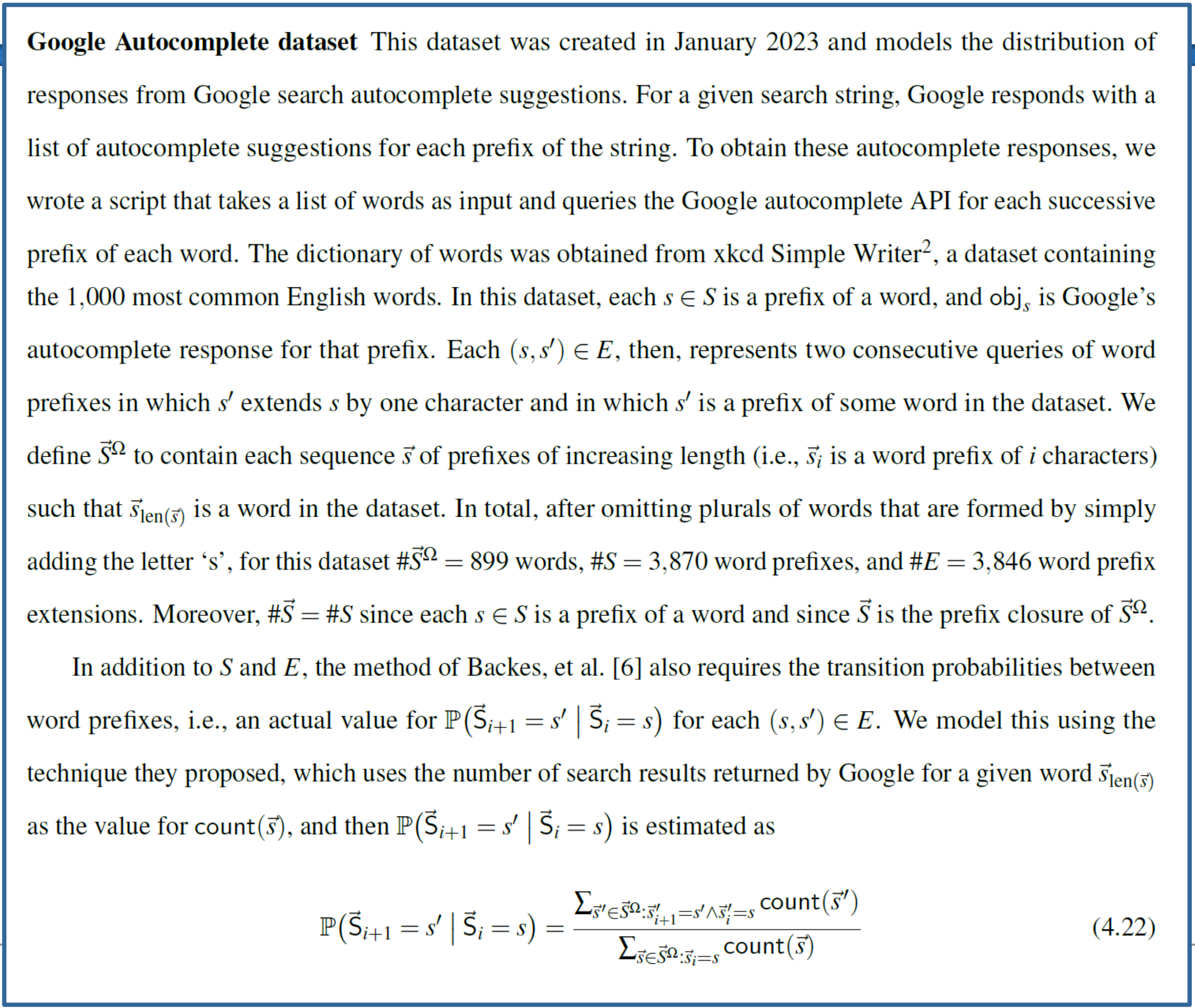
$$E = \left\{(s, s') \,\middle|\, \exists \vec{s} \in \vec{S}, i \in [\mathrm{len}(\vec{s}) - 1] : \vec{s}_i = s \wedge \vec{s}_{i+1} = s'\right\}$$

$$\vec{S} = \bigcup_{\vec{s} \in \vec{S}^{\Omega}} \; \bigcup_{i \in [\mathrm{len}(\vec{s})]} \{\vec{s}_{1\ldots i}\}$$

for each $i \in [\max_{\vec{s} \in \vec{S}^{\Omega}} \mathrm{len}(\vec{s})]$ we calculate $\mathbb{P}\left(\vec{\mathsf{S}}_{1\ldots i} = \vec{s}_{1\ldots i}\right)$ as

$$\mathbb{P}\left(\vec{\mathsf{S}}_{1\ldots i} = \vec{s}_{1\ldots i}\right) = \frac{\sum_{\vec{s}' \in \vec{S}^{\Omega} : \vec{s}'_{1\ldots i} = \vec{s}_{1\ldots i}} \mathsf{count}(\vec{s}')}{\sum_{\vec{s}'' \in \vec{S}^{\Omega} : \mathrm{len}(\vec{s}'') \geq i} \mathsf{count}(\vec{s}'')}$$

PFS: Dataset - Autocomplete



Google Autocomplete dataset This dataset was created in January 2023 and models the distribution of responses from Google search autocomplete suggestions. For a given search string, Google responds with a list of autocomplete suggestions for each prefix of the string. To obtain these autocomplete responses, we wrote a script that takes a list of words as input and queries the Google autocomplete API for each successive prefix of each word. The dictionary of words was obtained from xkcd Simple Writer[2], a dataset containing the 1,000 most common English words. In this dataset, each $s \in S$ is a prefix of a word, and obj_s is Google's autocomplete response for that prefix. Each $(s, s') \in E$, then, represents two consecutive queries of word prefixes in which s' extends s by one character and in which s' is a prefix of some word in the dataset. We define $\vec{S}^{\Omega}$ to contain each sequence $\vec{s}$ of prefixes of increasing length (i.e., $\vec{s}_i$ is a word prefix of i characters) such that $\vec{s}_{\mathrm{len}(\vec{s})}$ is a word in the dataset. In total, after omitting plurals of words that are formed by simply adding the letter 's', for this dataset $\#\vec{S}^{\Omega} = 899$ words, $\#S = 3{,}870$ word prefixes, and $\#E = 3{,}846$ word prefix extensions. Moreover, $\#\vec{S} = \#S$ since each $s \in S$ is a prefix of a word and since $\vec{S}$ is the prefix closure of $\vec{S}^{\Omega}$.

In addition to S and E, the method of Backes, et al. [6] also requires the transition probabilities between word prefixes, i.e., an actual value for $\mathbb{P}(\vec{\mathsf{S}}_{i+1} = s' \mid \vec{\mathsf{S}}_i = s)$ for each $(s, s') \in E$. We model this using the technique they proposed, which uses the number of search results returned by Google for a given word $\vec{s}_{\mathrm{len}(\vec{s})}$ as the value for $\mathsf{count}(\vec{s})$, and then $\mathbb{P}(\vec{\mathsf{S}}_{i+1} = s' \mid \vec{\mathsf{S}}_i = s)$ is estimated as

$$\mathbb{P}(\vec{\mathsf{S}}_{i+1} = s' \mid \vec{\mathsf{S}}_i = s) = \frac{\sum_{\vec{s}' \in \vec{S}^{\Omega} : \vec{s}'_{i+1} = s' \wedge \vec{s}'_i = s} \mathsf{count}(\vec{s}')}{\sum_{\vec{s} \in \vec{S}^{\Omega} : \vec{s}_i = s} \mathsf{count}(\vec{s})} \tag{4.22}$$

PFS: Dataset - Wikipedia

Wikipedia dataset This dataset was created in June 2023 and models pages retrieved during web surfing. It was created by selecting $s=$ `https://en.wikipedia.org/wiki/Cat` to obtain the HTML of the web page as obj_s. We then selected the first 50 hyperlinks (to articles) on this page and included those in S, retrieving the HTML of the web page for each. We repeated this step once more for these added pages, but increased the number of hyperlinks that we added from those pages to the first 100. We increased from 50 to 100 hyperlinks to help increase the reach of this dataset, as we observed that the articles had many common hyperlinks.

This initial step yielded a set of $\#S = 2{,}804$ articles. To then create the set of sequences $\vec{S}^{\Omega}$, for each $s \in S$, we generated two random walks of length six that both begin at s as the start vertex. This resulted in $\#\vec{S}^{\Omega} = 5{,}606$ unique sequences of webpages that a user might explore while browsing among the articles included in S with an associated $\#E = 10{,}182$ hyperlinks. Taking the prefix closure of these sequences then yielded $\#\vec{S} = 32{,}683$.

For this dataset, we used the Wikipedia Page Views API to instantiate $\mathsf{count}(\vec{s})$ for each $\vec{s} \in \vec{S}^{\Omega}$. To do so, we retrieved the total number of page views for each $s \in S$ for January 2016, which we denote as $\mathsf{pv}(s)$. For each $\vec{s} \in \vec{S}^{\Omega}$ we then set $\mathsf{count}(\vec{s})$ equal to its average page views, i.e.,

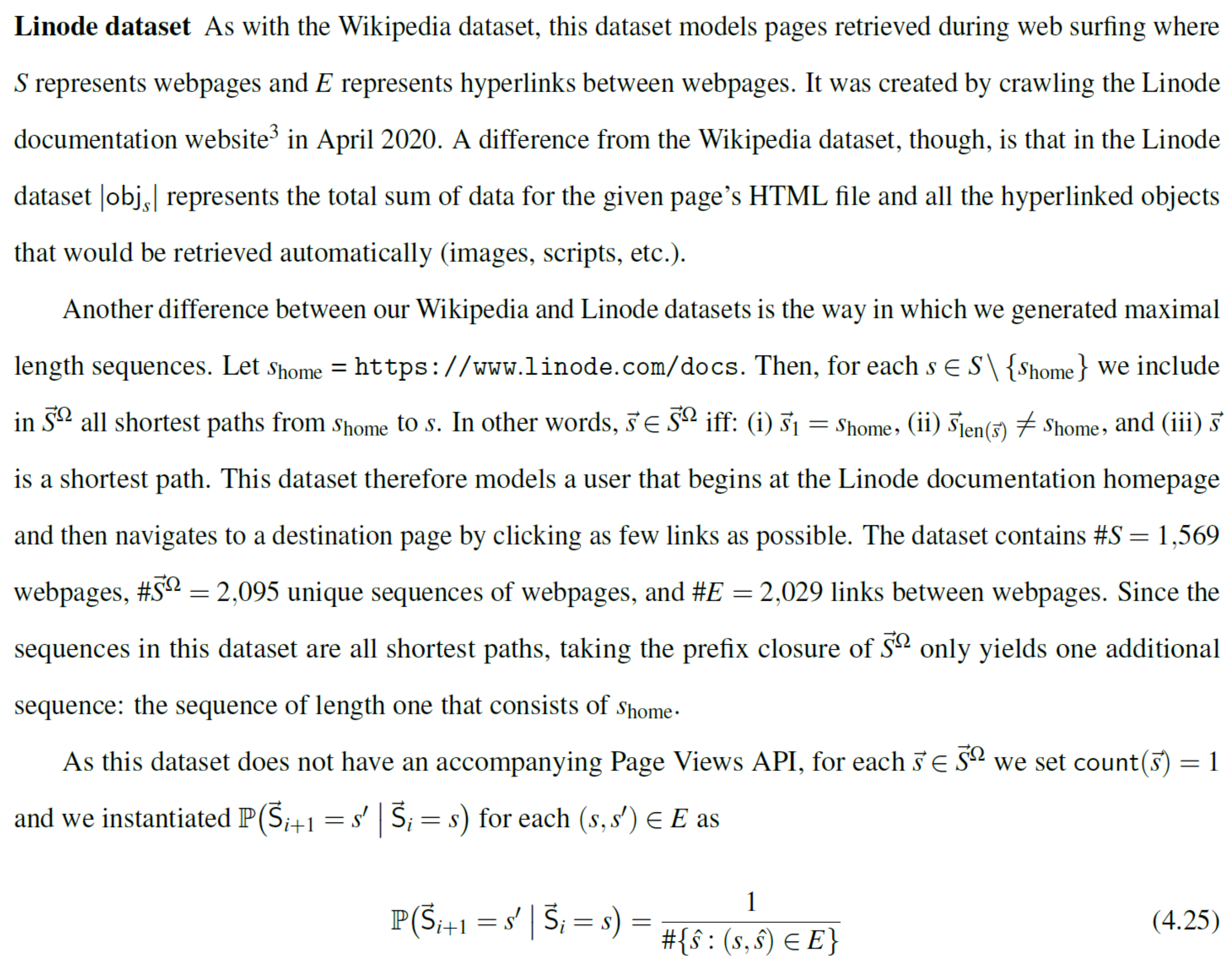
$$\mathsf{count}(\vec{s}) = \frac{\sum_{i=1}^{\mathsf{len}(\vec{s})} \mathsf{pv}(\vec{s}_i)}{\mathsf{len}(\vec{s})} \tag{4.23}$$

Finally, we instantiated $\mathbb{P}\left(\vec{\mathsf{S}}_{i+1} = s' \mid \vec{\mathsf{S}}_i = s\right)$ for each $(s, s') \in E$ as

$$\mathbb{P}\left(\vec{\mathsf{S}}_{i+1} = s' \mid \vec{\mathsf{S}}_i = s\right) = \frac{1}{\#\{\hat{s} : (s, \hat{s}) \in E\}} \tag{4.24}$$

We reiterate that this dataset only captures the size of each article's HTML file. We address this limitation with our next dataset, though we stress that our goal for all of our datasets is to enable a meaningful comparison of candidate padding algorithms (described below) based on the privacy they achieve and padding overhead they induce. Our goal is not to make specific claims about privacy in the context of Wikipedia retrievals, for example.

PFS: Dataset - Linode



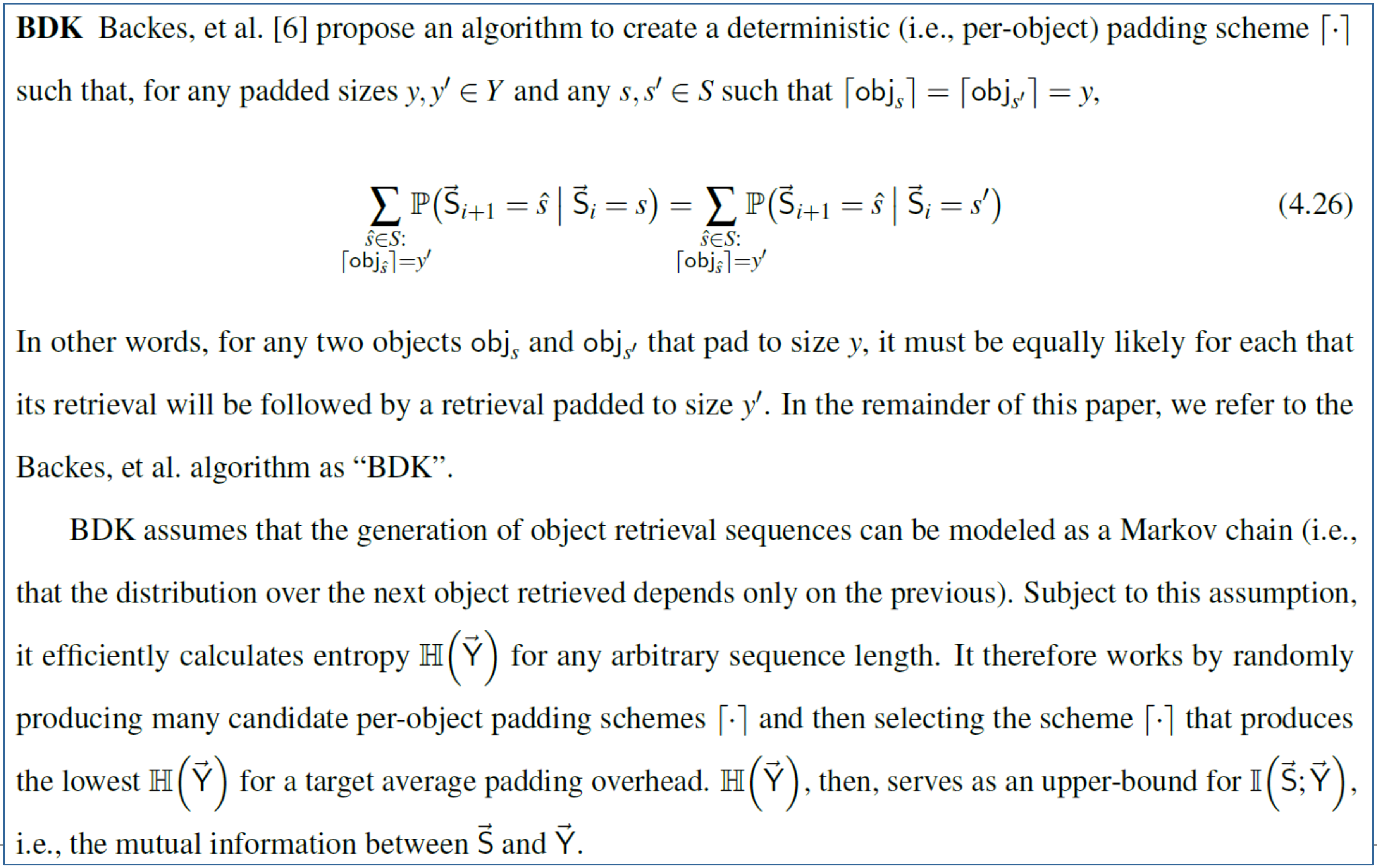
Linode dataset As with the Wikipedia dataset, this dataset models pages retrieved during web surfing where S represents webpages and E represents hyperlinks between webpages. It was created by crawling the Linode documentation website[3] in April 2020. A difference from the Wikipedia dataset, though, is that in the Linode dataset $|\mathsf{obj}_s|$ represents the total sum of data for the given page's HTML file and all the hyperlinked objects that would be retrieved automatically (images, scripts, etc.).

Another difference between our Wikipedia and Linode datasets is the way in which we generated maximal length sequences. Let s_{home} = `https://www.linode.com/docs`. Then, for each $s \in S \setminus \{s_{\text{home}}\}$ we include in $\vec{S}^{\Omega}$ all shortest paths from s_{home} to s. In other words, $\vec{s} \in \vec{S}^{\Omega}$ iff: (i) $\vec{s}_1 = s_{\text{home}}$, (ii) $\vec{s}_{\text{len}(\vec{s})} \neq s_{\text{home}}$, and (iii) $\vec{s}$ is a shortest path. This dataset therefore models a user that begins at the Linode documentation homepage and then navigates to a destination page by clicking as few links as possible. The dataset contains $\#S = 1{,}569$ webpages, $\#\vec{S}^{\Omega} = 2{,}095$ unique sequences of webpages, and $\#E = 2{,}029$ links between webpages. Since the sequences in this dataset are all shortest paths, taking the prefix closure of $\vec{S}^{\Omega}$ only yields one additional sequence: the sequence of length one that consists of s_{home}.

As this dataset does not have an accompanying Page Views API, for each $\vec{s} \in \vec{S}^{\Omega}$ we set $\mathsf{count}(\vec{s}) = 1$ and we instantiated $\mathbb{P}(\vec{\mathsf{S}}_{i+1} = s' \mid \vec{\mathsf{S}}_i = s)$ for each $(s, s') \in E$ as

$$\mathbb{P}(\vec{\mathsf{S}}_{i+1} = s' \mid \vec{\mathsf{S}}_i = s) = \frac{1}{\#\{\hat{s} : (s, \hat{s}) \in E\}} \tag{4.25}$$

BDK (Backes, et al.)



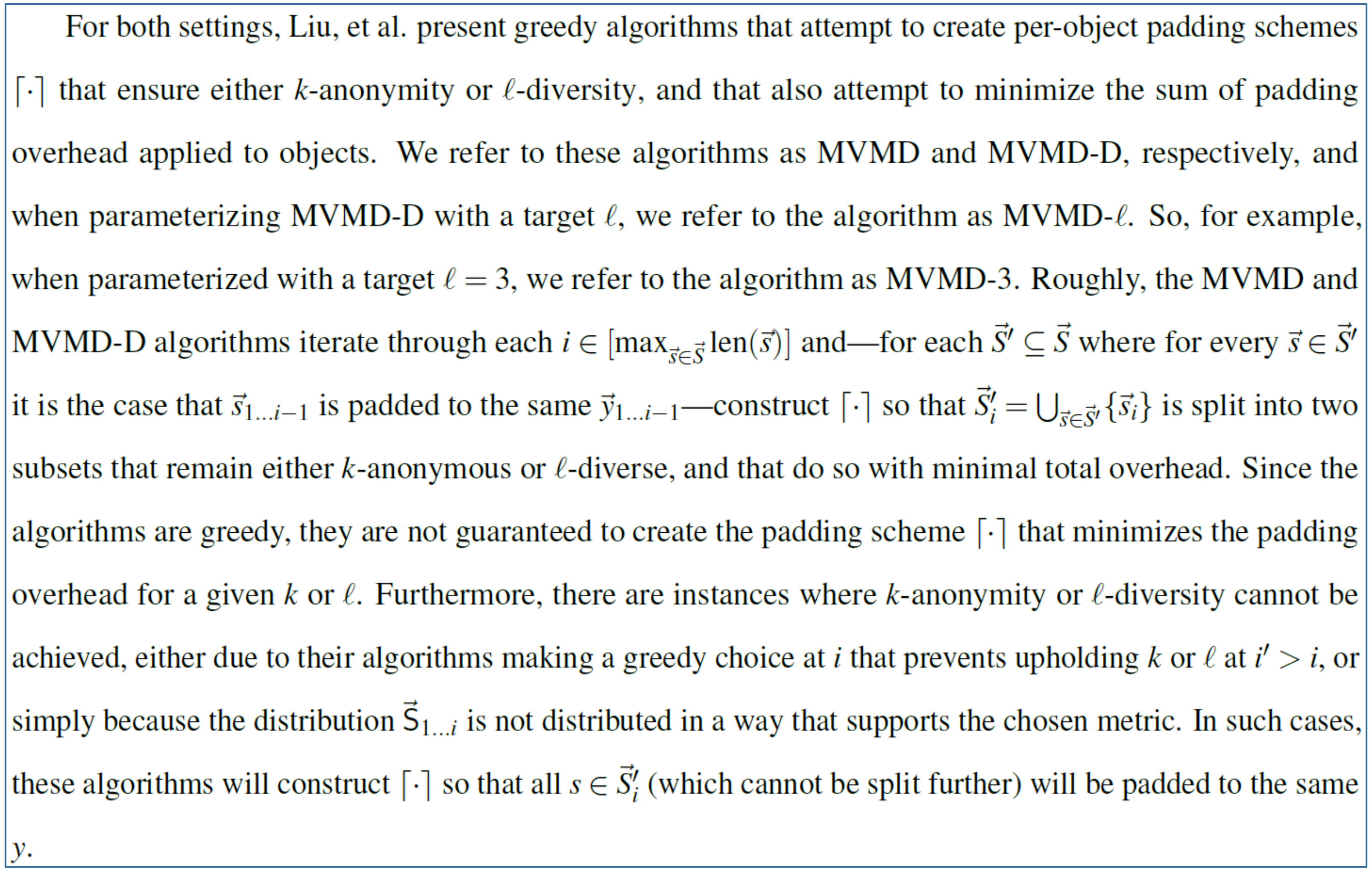
BDK Backes, et al. [6] propose an algorithm to create a deterministic (i.e., per-object) padding scheme $\lceil\cdot\rceil$ such that, for any padded sizes $y, y' \in Y$ and any $s, s' \in S$ such that $\lceil \mathsf{obj}_s \rceil = \lceil \mathsf{obj}_{s'} \rceil = y$,

$$\sum_{\substack{\hat{s}\in S:\\ \lceil \mathsf{obj}_{\hat{s}} \rceil = y'}} \mathbb{P}\left(\vec{\mathsf{S}}_{i+1} = \hat{s} \mid \vec{\mathsf{S}}_i = s\right) = \sum_{\substack{\hat{s}\in S:\\ \lceil \mathsf{obj}_{\hat{s}} \rceil = y'}} \mathbb{P}\left(\vec{\mathsf{S}}_{i+1} = \hat{s} \mid \vec{\mathsf{S}}_i = s'\right) \tag{4.26}$$

In other words, for any two objects obj_s and $\mathsf{obj}_{s'}$ that pad to size y, it must be equally likely for each that its retrieval will be followed by a retrieval padded to size y'. In the remainder of this paper, we refer to the Backes, et al. algorithm as "BDK".

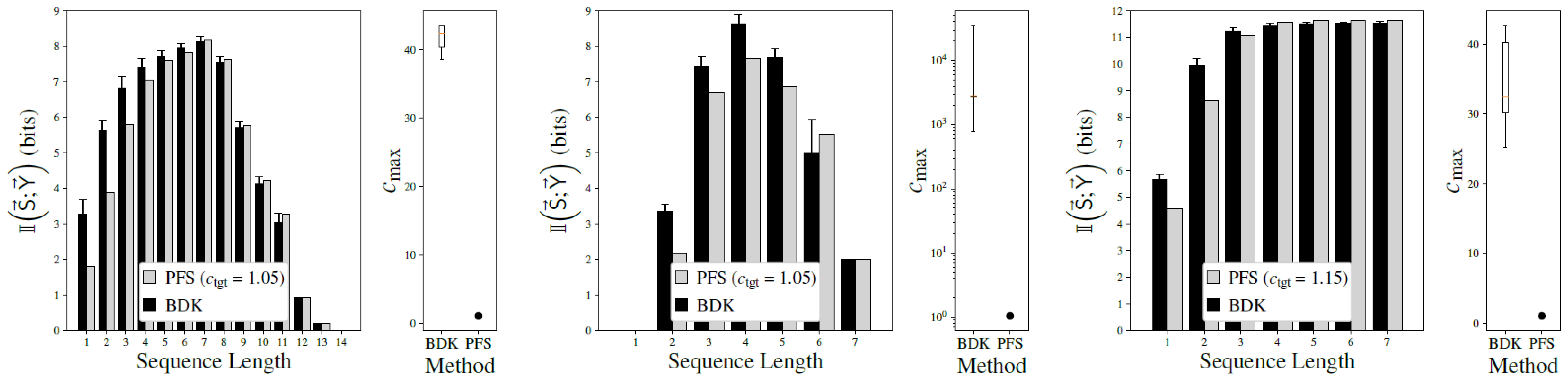
BDK assumes that the generation of object retrieval sequences can be modeled as a Markov chain (i.e., that the distribution over the next object retrieved depends only on the previous). Subject to this assumption, it efficiently calculates entropy $\mathbb{H}\left(\vec{\mathsf{Y}}\right)$ for any arbitrary sequence length. It therefore works by randomly producing many candidate per-object padding schemes $\lceil\cdot\rceil$ and then selecting the scheme $\lceil\cdot\rceil$ that produces the lowest $\mathbb{H}\left(\vec{\mathsf{Y}}\right)$ for a target average padding overhead. $\mathbb{H}\left(\vec{\mathsf{Y}}\right)$, then, serves as an upper-bound for $\mathbb{I}\left(\vec{\mathsf{S}};\vec{\mathsf{Y}}\right)$, i.e., the mutual information between $\vec{\mathsf{S}}$ and $\vec{\mathsf{Y}}$.

MVMD-D (Liu, et al.)



For both settings, Liu, et al. present greedy algorithms that attempt to create per-object padding schemes $\lceil\cdot\rceil$ that ensure either k-anonymity or ℓ-diversity, and that also attempt to minimize the sum of padding overhead applied to objects. We refer to these algorithms as MVMD and MVMD-D, respectively, and when parameterizing MVMD-D with a target ℓ, we refer to the algorithm as MVMD-ℓ. So, for example, when parameterized with a target $\ell = 3$, we refer to the algorithm as MVMD-3. Roughly, the MVMD and MVMD-D algorithms iterate through each $i \in [\max_{\vec{s}\in\vec{S}} \mathrm{len}(\vec{s})]$ and—for each $\vec{S}' \subseteq \vec{S}$ where for every $\vec{s} \in \vec{S}'$ it is the case that $\vec{s}_{1\ldots i-1}$ is padded to the same $\vec{y}_{1\ldots i-1}$—construct $\lceil\cdot\rceil$ so that $\vec{S}'_i = \bigcup_{\vec{s}\in\vec{S}'}\{\vec{s}_i\}$ is split into two subsets that remain either k-anonymous or ℓ-diverse, and that do so with minimal total overhead. Since the algorithms are greedy, they are not guaranteed to create the padding scheme $\lceil\cdot\rceil$ that minimizes the padding overhead for a given k or ℓ. Furthermore, there are instances where k-anonymity or ℓ-diversity cannot be achieved, either due to their algorithms making a greedy choice at i that prevents upholding k or ℓ at $i' > i$, or simply because the distribution $\vec{\mathsf{S}}_{1\ldots i}$ is not distributed in a way that supports the chosen metric. In such cases, these algorithms will construct $\lceil\cdot\rceil$ so that all $s \in \vec{S}'_i$ (which cannot be split further) will be padded to the same y.

Evaluation: PFS vs BDK - $\mathbb{I}\left(\vec{S};\vec{Y}\right)$



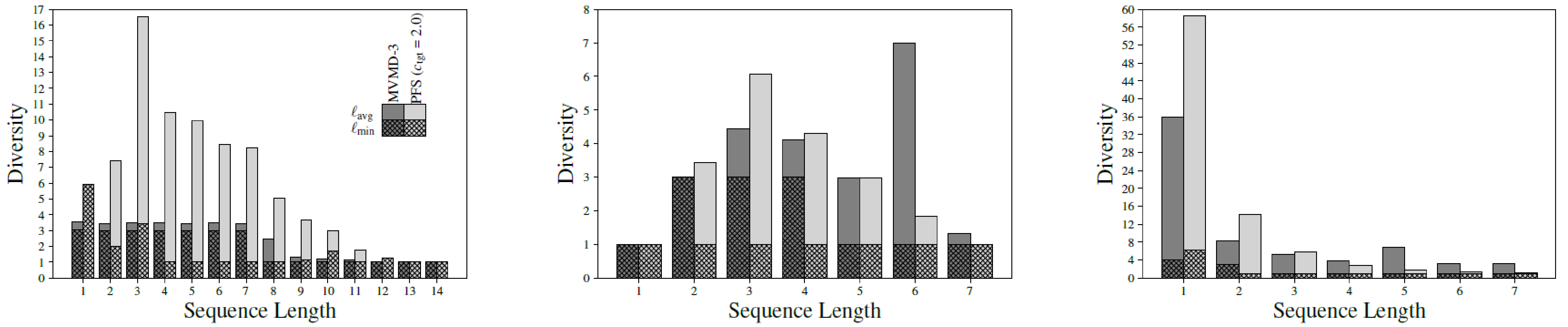
(a) Autocomplete (b) Linode (c) Wikipedia

Figure 4.2: Comparing PFS to BDK using $\mathbb{I}\left(\vec{S};\vec{Y}\right)$ as the privacy metric.

Evaluation: PFS vs MVMD-3 - ℓ-diversity

$$\ell_{\min} = \min_{\vec{y}_{1\ldots i} \in \vec{Y}_{1\ldots i}} \frac{1}{\max_{s\in S} \mathbb{P}\left(\vec{\mathsf{S}}_i = s \mid \vec{\mathsf{Y}}_{1\ldots i} = \vec{y}_{1\ldots i}\right)}$$

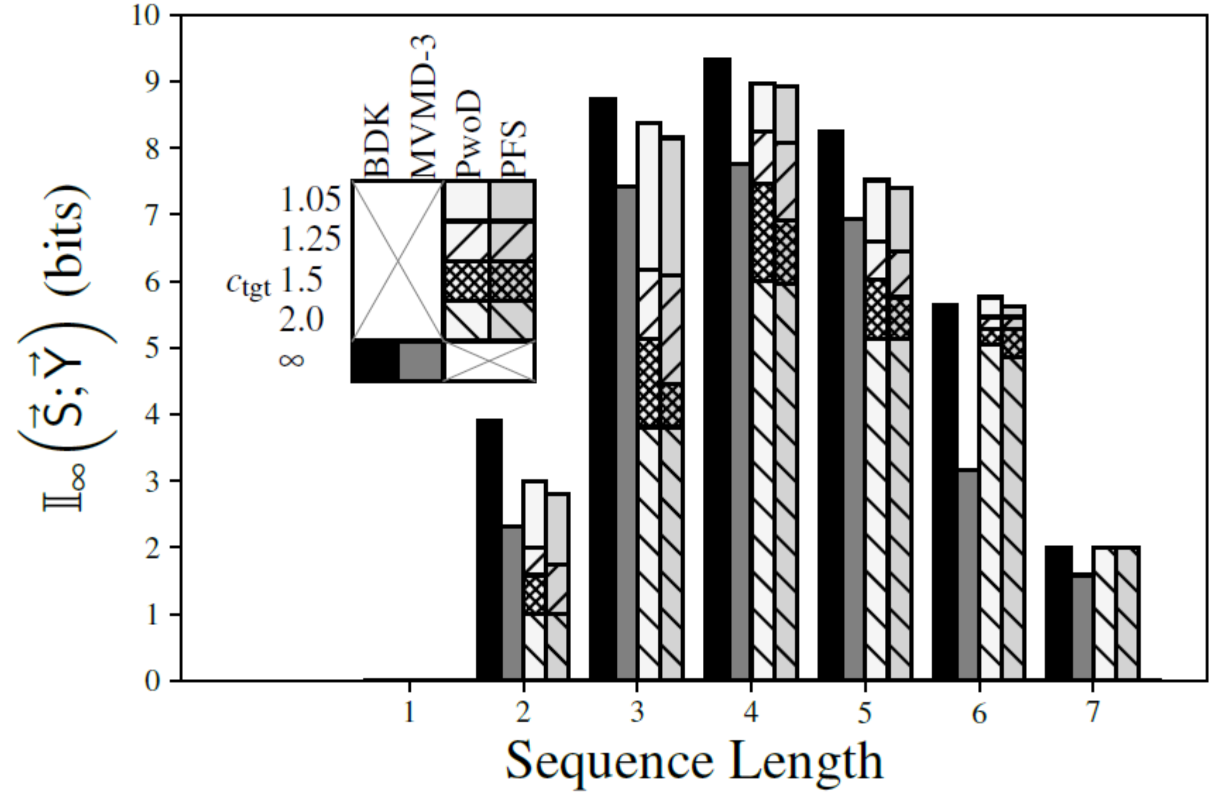
$$\ell_{\text{avg}} = \frac{\sum_{\vec{y}_{1\ldots i} \in \vec{Y}_{1\ldots i}} \frac{1}{\max_{s\in S} \mathbb{P}\left(\vec{\mathsf{S}}_i = s \mid \vec{\mathsf{Y}}_{1\ldots i} = \vec{y}_{1\ldots i}\right)}}{\# \vec{Y}_{1\ldots i}}$$



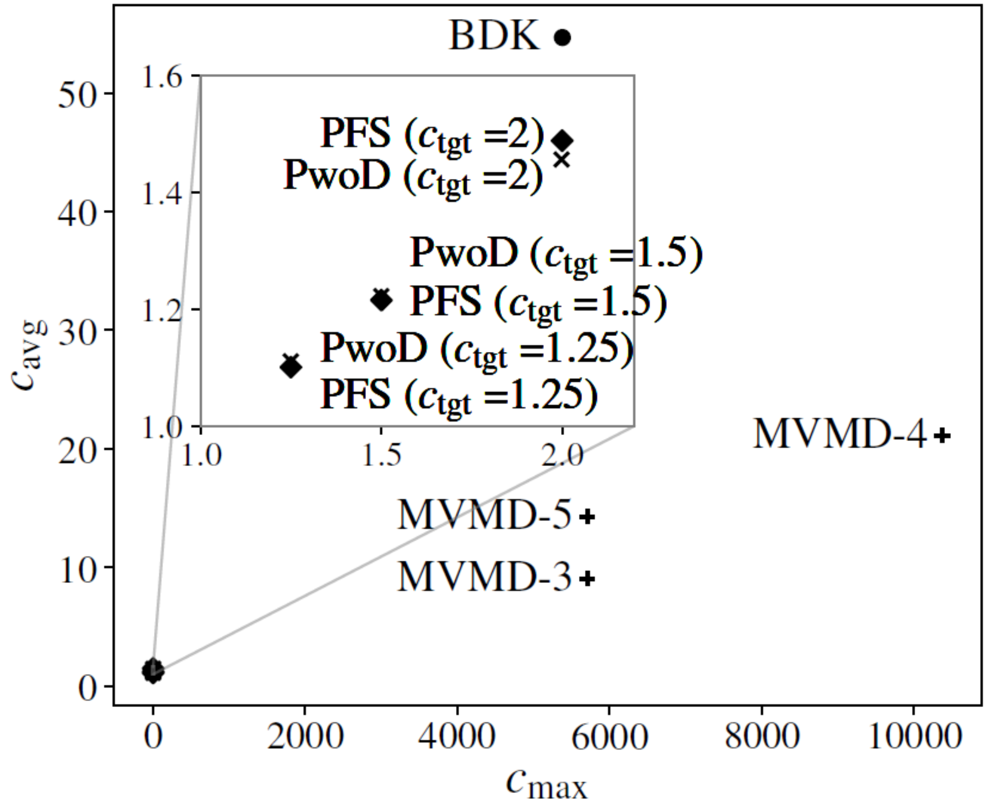
(a) Autocomplete (MVMD-3 $c_{\max} = 31.73$) (b) Linode (MVMD-3 $c_{\max} = 5707.33$) (c) Wikipedia (MVMD-3 $c_{\max} = 43.07$)

Figure 4.3: Comparing PFS to MVMD-3 using ℓ-diversity as the privacy metric.

Evaluation: Linode - $\mathbb{I}_\infty\left(\vec{S};\vec{Y}\right)$

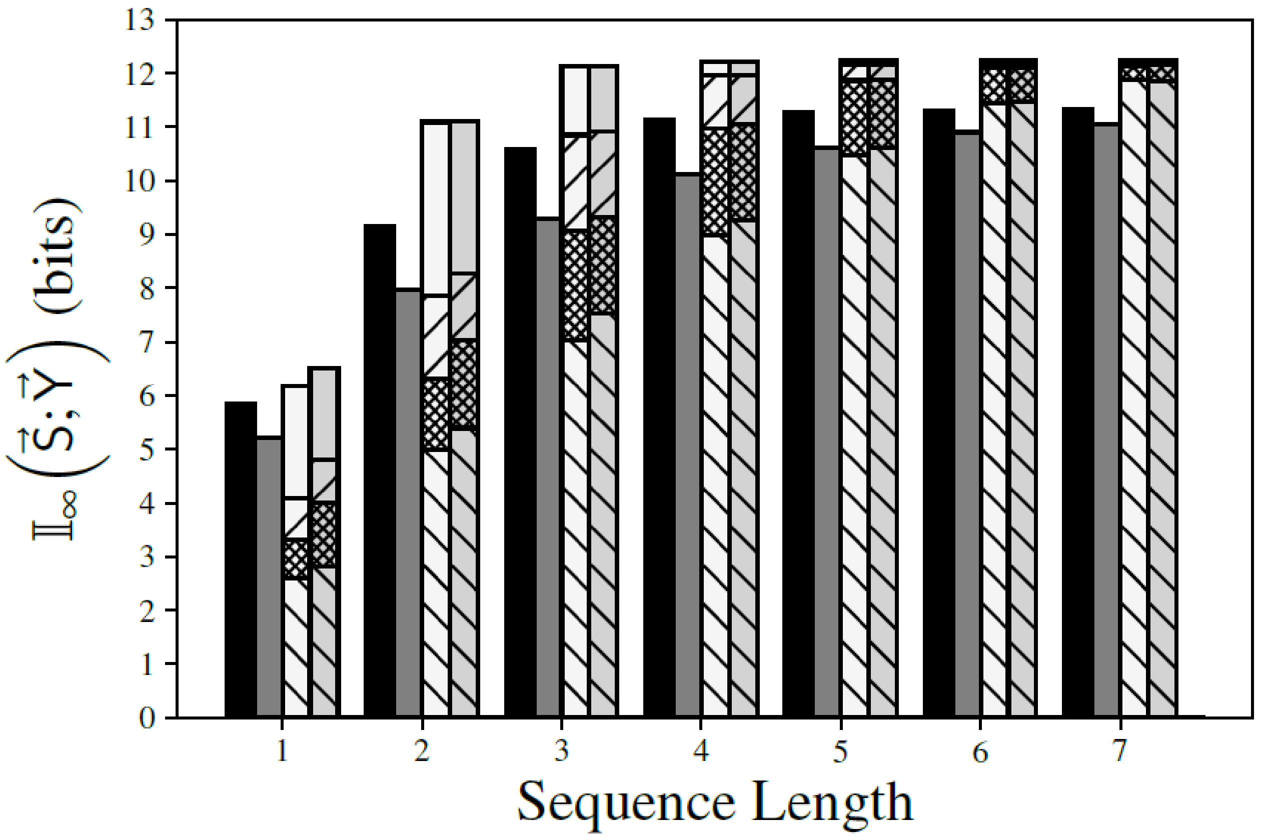


Comparing all algorithms using $\mathbb{I}_\infty\left(\vec{S};\vec{Y}\right)$ as the privacy metric.

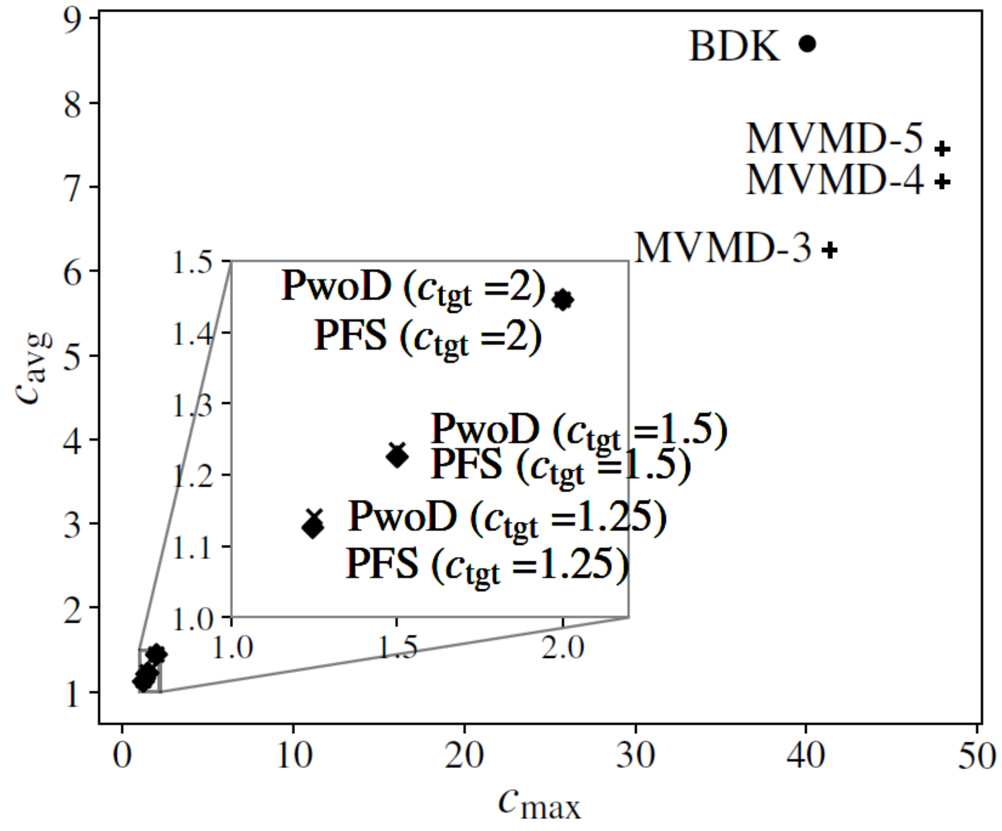


Padding overhead factors for each padding algorithm.

Evaluation: Wikipedia - $\mathbb{I}_\infty\left(\vec{S};\vec{Y}\right)$

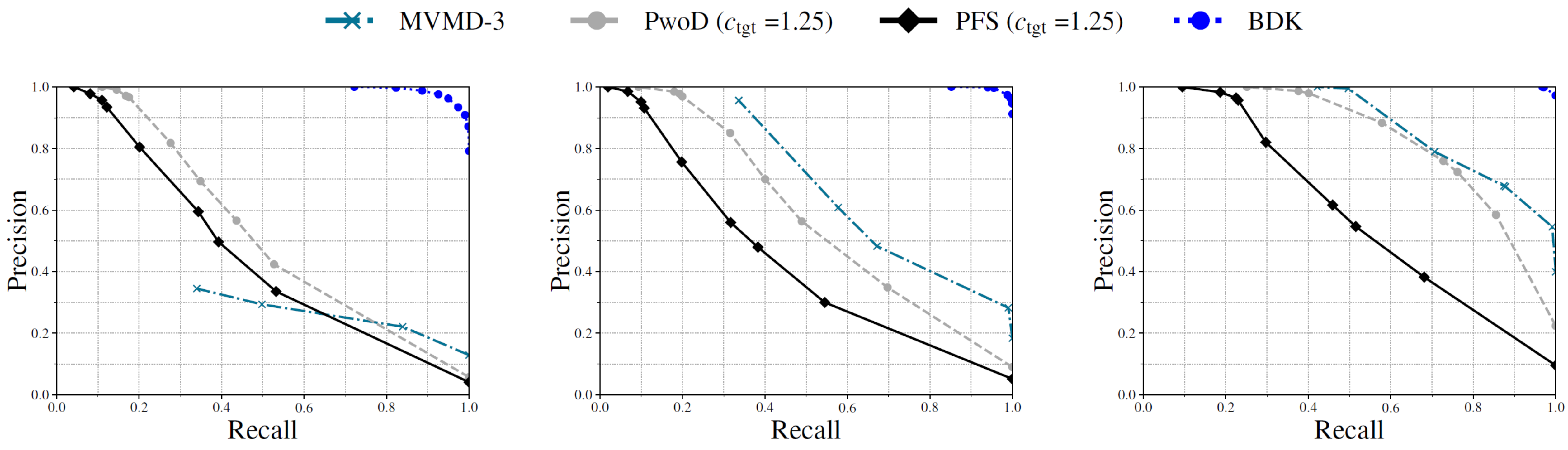


Comparing all algorithms using $\mathbb{I}_\infty\left(\vec{S};\vec{Y}\right)$ as the privacy metric.



Padding overhead factors for each padding algorithm.

PFS: Precision-Recall (Autocomplete)



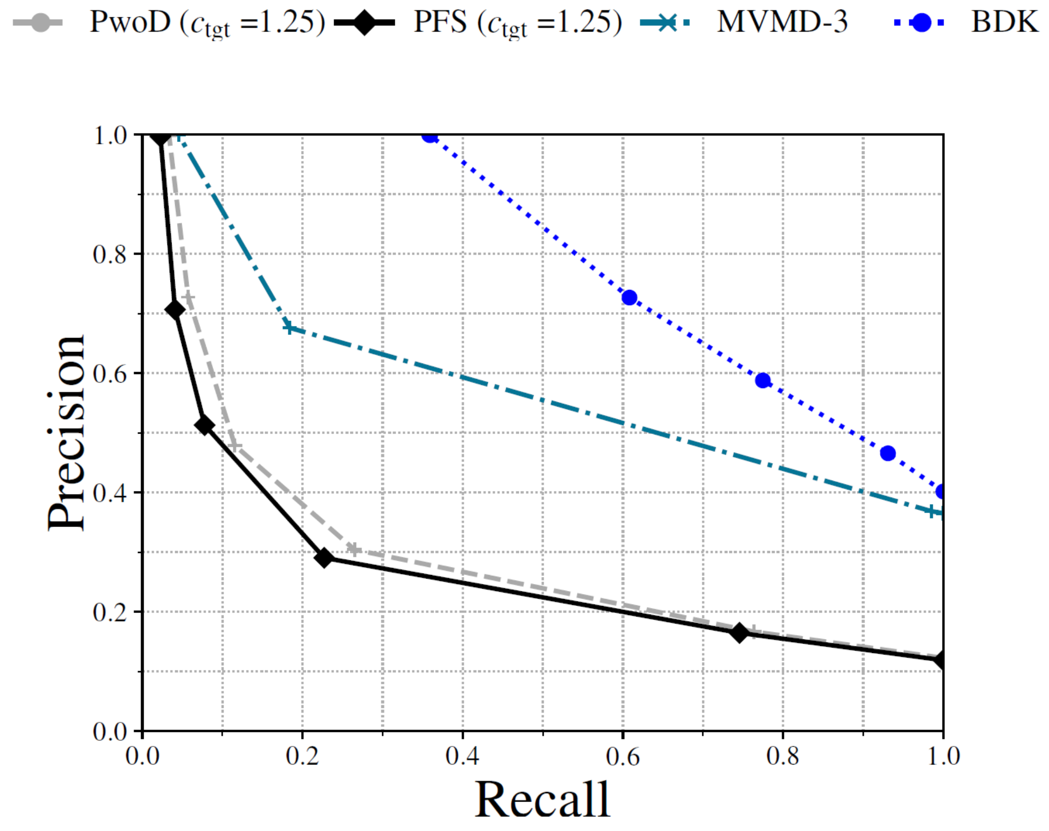
(a) Sequence length = 7

(b) Sequence length = 8

(c) Sequence length = 9

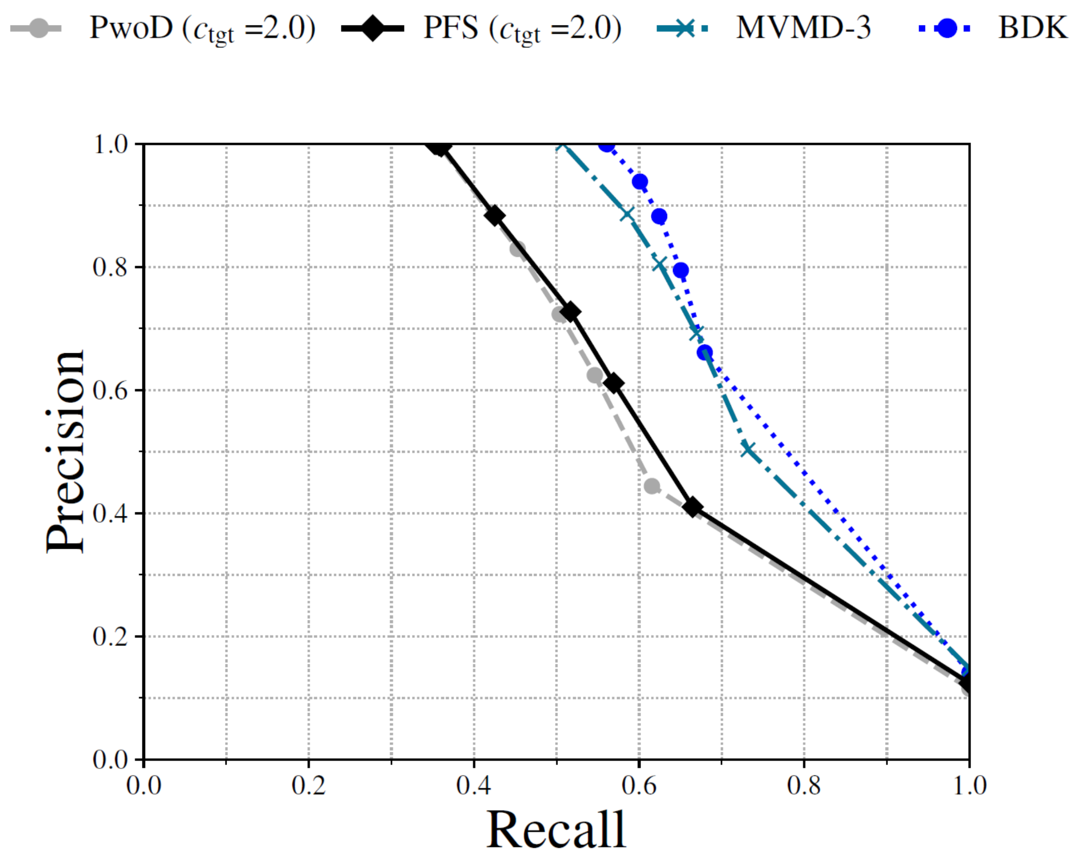
Adversary's recall and precision for detecting words from the Autocomplete dataset.

PFS: Precision-Recall (Linode)



Adversary’s recall and precision for detecting sequences of length 3 from the Linode dataset.

PFS: Precision-Recall (Wikipedia)



Adversary's recall and precision for detecting sequences of length 7 from the Wikipedia dataset.

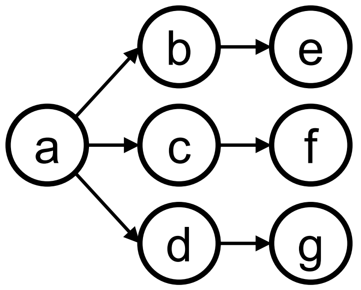
Example $\vec{S}_G \Rightarrow \vec{Y}_G$

$\vec{S}_G$:

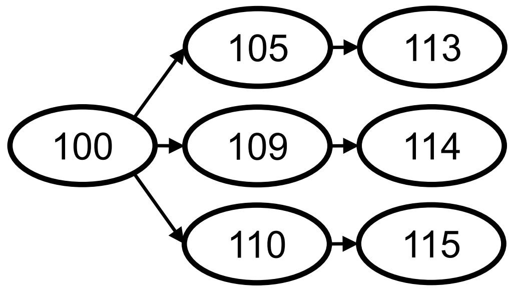
S =

a	b	c	d	e	f	g
100	105	109	110	113	114	115

E = { (a,b), (a,c), (a,d), (b,e), (c,f), (d,g) }



$\vec{S}_{1\ldots3}$: (a, b, e), (a, c, f), (a, d, g)



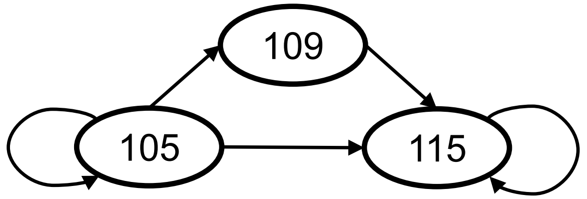
For C = 1.05, four example $\lceil\cdot\rceil$:

$\vec{Y}_G$:

105	105	109	115	115	115	115

{ (105,105), (105,109), (105,115), (109,115), (115,115) }

$\#\vec{Y}_G$ = 8



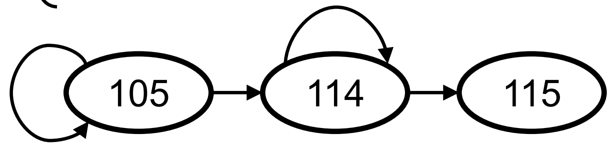
$\vec{Y}_{1\ldots3}$: (105, 105, 115), (105, 109, 115), (105, 115, 115)

$\vec{Y}_G$:

105	105	114	114	114	114	115

{ (105,105), (105,114), (114,114), (114,115) }

$\#\vec{Y}_G$ = 7



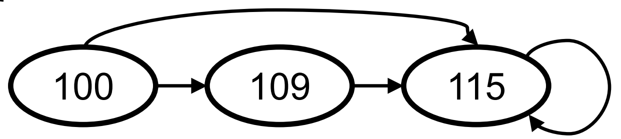
$\vec{Y}_{1\ldots3}$: (105, 105, 114), (105, 114, 114), (105, 114, 115)

$\vec{Y}_G$:

100	109	109	115	115	115	115

{ (100,109), (100,115), (109,115), (115,115) }

$\#\vec{Y}_G$ = 7



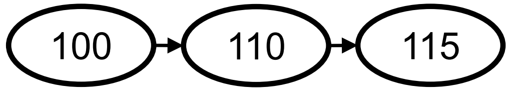
$\vec{Y}_{1\ldots3}$: (100, 109, 115), (100, 115, 115)

$\vec{Y}_G$:

100	110	110	110	115	115	115

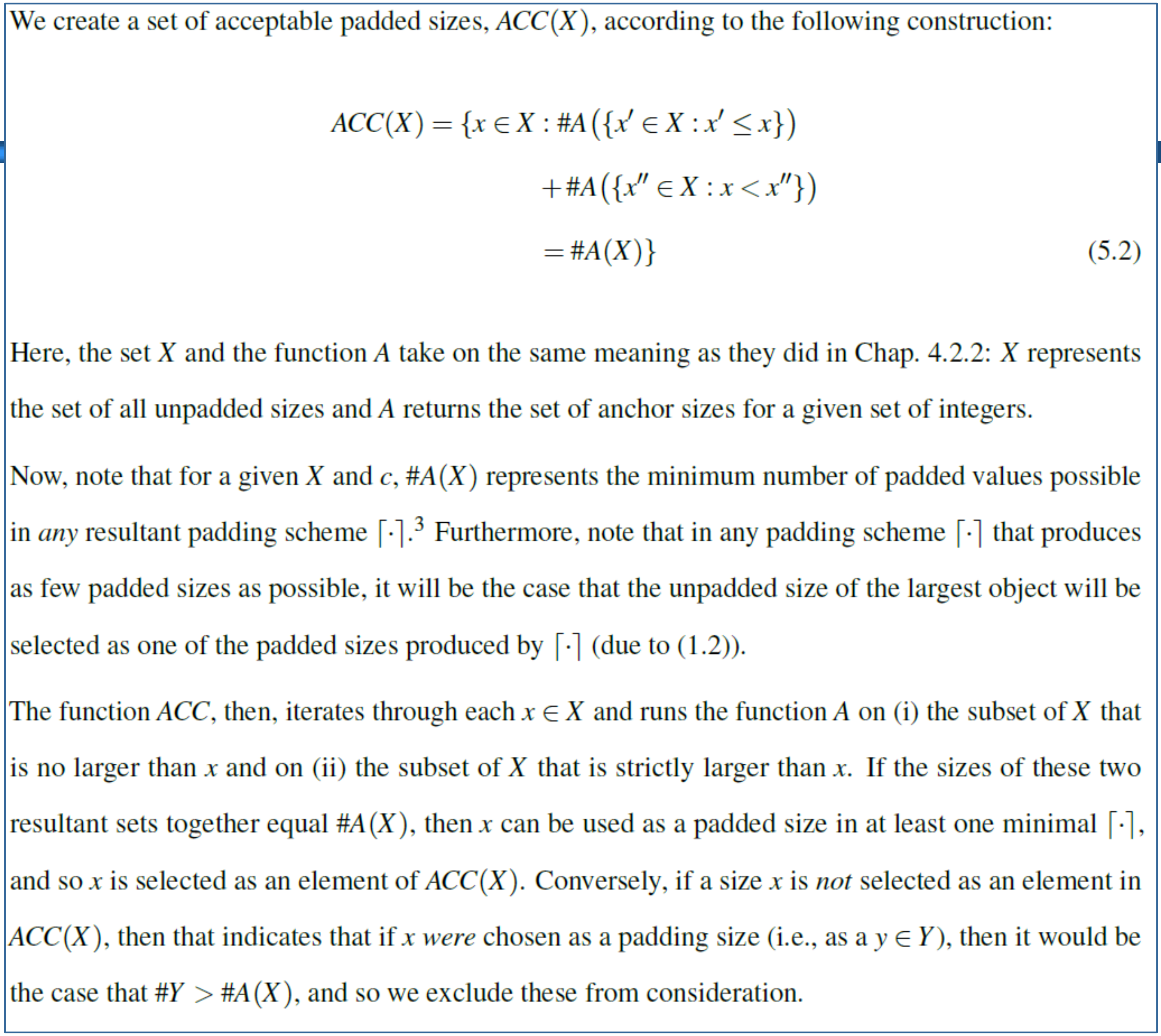
{ (100,110), (110,115) }

$\#\vec{Y}_G$ = 5



$\vec{Y}_{1\ldots3}$: (100, 110, 115)

PFG-DE: ACC(X)



We create a set of acceptable padded sizes, $ACC(X)$, according to the following construction:

$$\begin{aligned} ACC(X) = \{x \in X : \#A\left(\{x' \in X : x' \leq x\}\right) \\ + \#A\left(\{x'' \in X : x < x''\}\right) \\ = \#A(X)\} \end{aligned} \tag{5.2}$$

Here, the set X and the function A take on the same meaning as they did in Chap. 4.2.2: X represents the set of all unpadded sizes and A returns the set of anchor sizes for a given set of integers.

Now, note that for a given X and c, $\#A(X)$ represents the minimum number of padded values possible in *any* resultant padding scheme $\lceil\cdot\rceil$.[3] Furthermore, note that in any padding scheme $\lceil\cdot\rceil$ that produces as few padded sizes as possible, it will be the case that the unpadded size of the largest object will be selected as one of the padded sizes produced by $\lceil\cdot\rceil$ (due to (1.2)).

The function ACC, then, iterates through each $x \in X$ and runs the function A on (i) the subset of X that is no larger than x and on (ii) the subset of X that is strictly larger than x. If the sizes of these two resultant sets together equal $\#A(X)$, then x can be used as a padded size in at least one minimal $\lceil\cdot\rceil$, and so x is selected as an element of $ACC(X)$. Conversely, if a size x is *not* selected as an element in $ACC(X)$, then that indicates that if x *were* chosen as a padding size (i.e., as a $y \in Y$), then it would be the case that $\#Y > \#A(X)$, and so we exclude these from consideration.

PFG-DE: Mutation Strategy

- The mutation strategy was set to “randtobest1”. As mentioned in Chap. 5.2, this strategy creates each sol_{mutant} according to:

$$sol_{mutant} = sol_{r_1} + MC \times (sol_{best} - sol_{r_1} + sol_{r_2} - sol_{r_3})$$

PFG-DE: Datasets

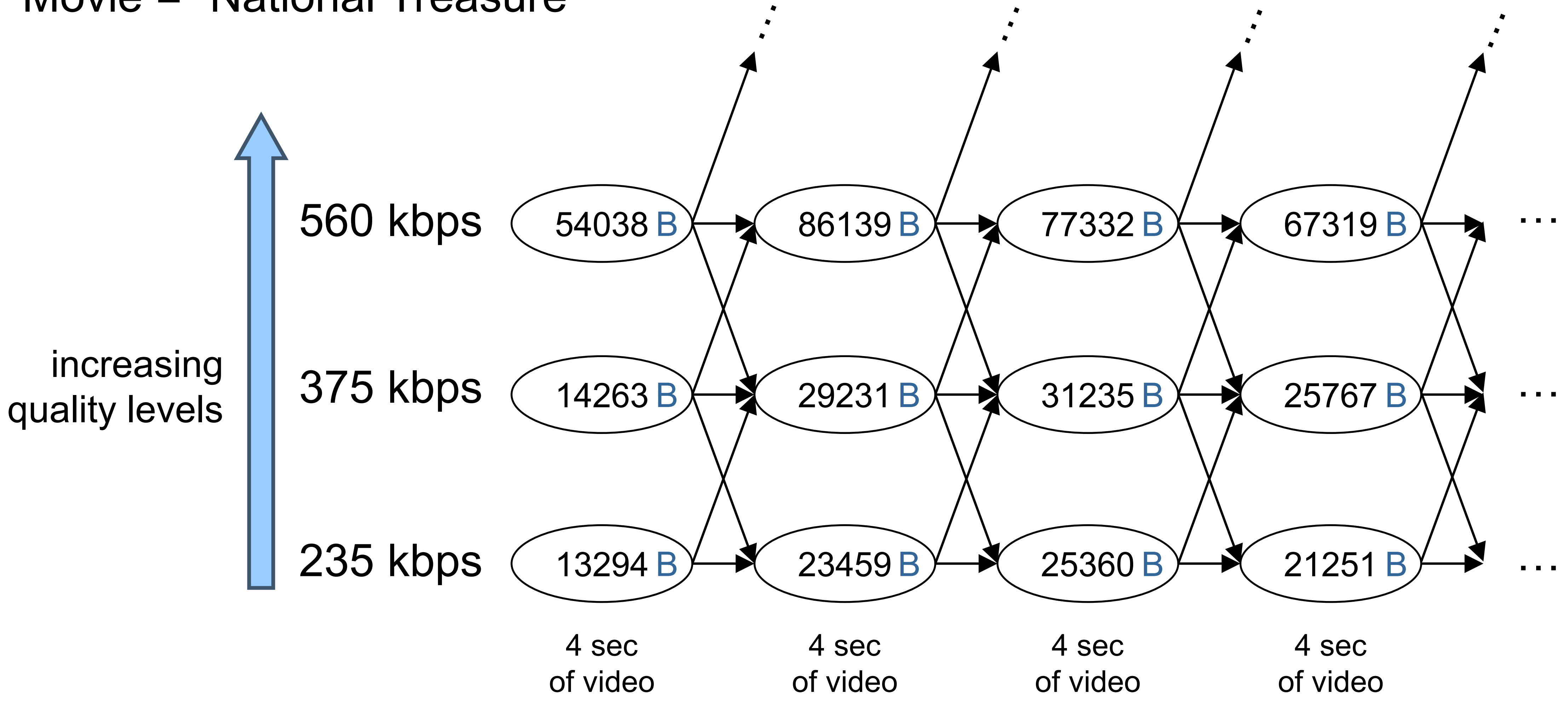
- **Linode and Wikipedia.** For our Linode and Wikipedia datasets, S remains the same as in Chap. 4. E, however, is instead comprised of every hyperlink captured during our crawls of each website. This is in contrast to the approach we took in Chap. 4, where we first created the set of maximal length sequences $\vec{S}^{\Omega}$ and then constructed E from only the subset of hyperlinks present in at least one $\vec{s} \in \vec{S}^{\Omega}$.

- **Netflix.** For our Netflix dataset, we leverage a database of Dynamic Adaptive Streaming over HTTP (DASH) videos from Reed and Kranch [67]. Reed and Kranch [67] crawled Netflix and extracted "fingerprints" for each video consisting of the size (in bytes) for each segment of each video at each available video bitrate. When this dataset was created, each segment in a Netflix video corresponded to four seconds of video.

  In this dataset that we create from these fingerprints, each $s_{b,t} \in S$ is a video segment which corresponds to a given bitrate encoding, b, at a timestamp in a video, t. Then, for each $s_{b,t} \in S$ such that $b-1$ is the next lower bitrate, $b+1$ is the next higher bitrate, and $t+1$ is the next timestamp of the video, we add the following edges to E: $(s_{b,t}, s_{b,t+1})$, $(s_{b,t}, s_{b-1,t+1})$, and $(s_{b,t}, s_{b+1,t+1})$.

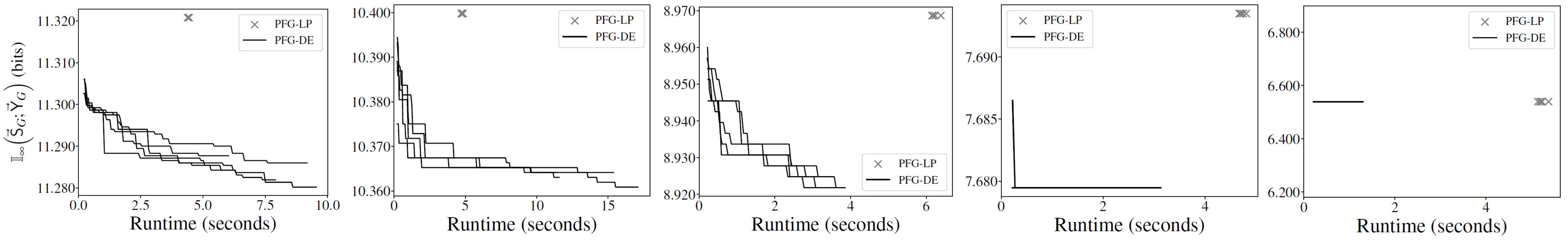
PFG-DE: Datasets – Netflix example

Movie = “National Treasure”



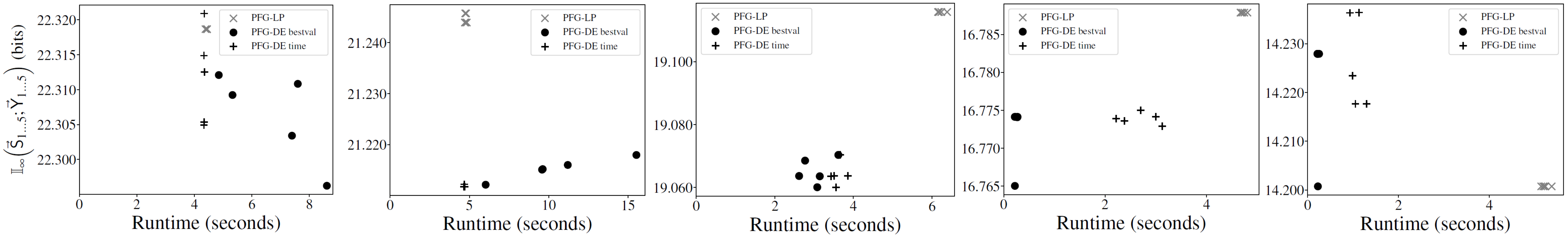
B = Byte

PFG-DE vs. PFG-LP: $\mathbb{I}_\infty$ and Runtime (Linode)



$c = 1.05$ $c = 1.1$ $c = 1.25$ $c = 1.5$ $c = 2.0$

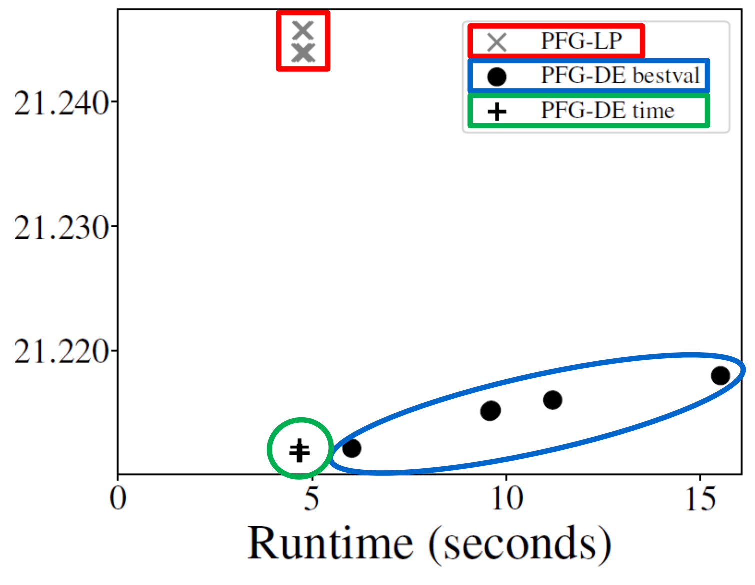
Comparing the two algorithms on $\mathbb{I}_\infty\left(\vec{S}_G;\vec{Y}_G\right)$.



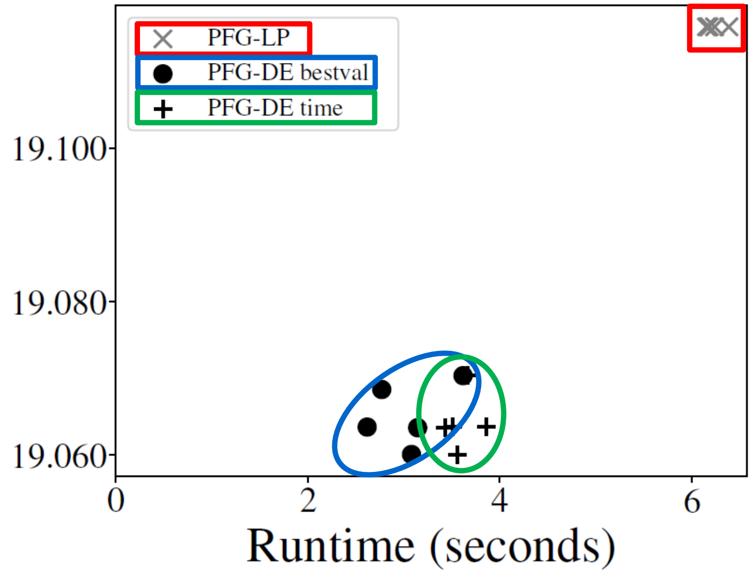
$c = 1.05$ $c = 1.1$ $c = 1.25$ $c = 1.5$ $c = 2.0$

Comparing the two algorithms on sequences of length 5.

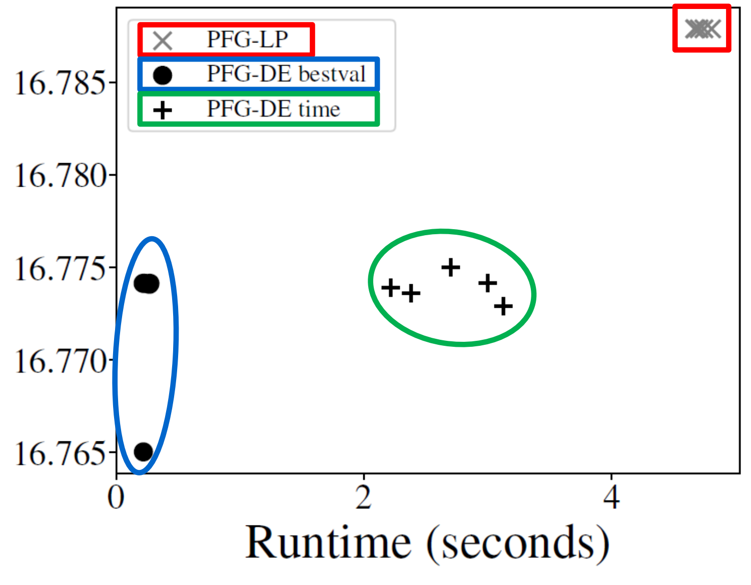
PFG-DE vs. PFG-LP: $\mathbb{I}_\infty$ and Runtime (Linode)



$c = 1.1$



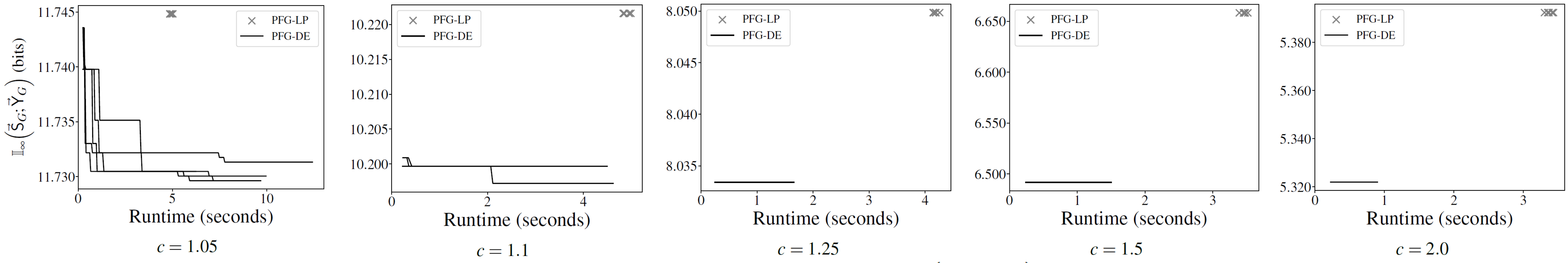
$c = 1.25$



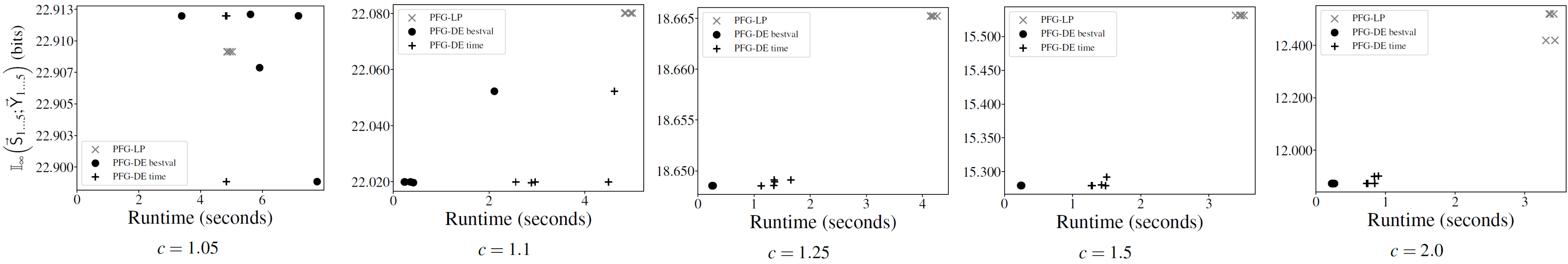
$c = 1.5$

Comparing the two algorithms on sequences of length 5.

PFG-DE vs. PFG-LP: $\mathbb{I}_\infty$ and Runtime (Wikipedia)

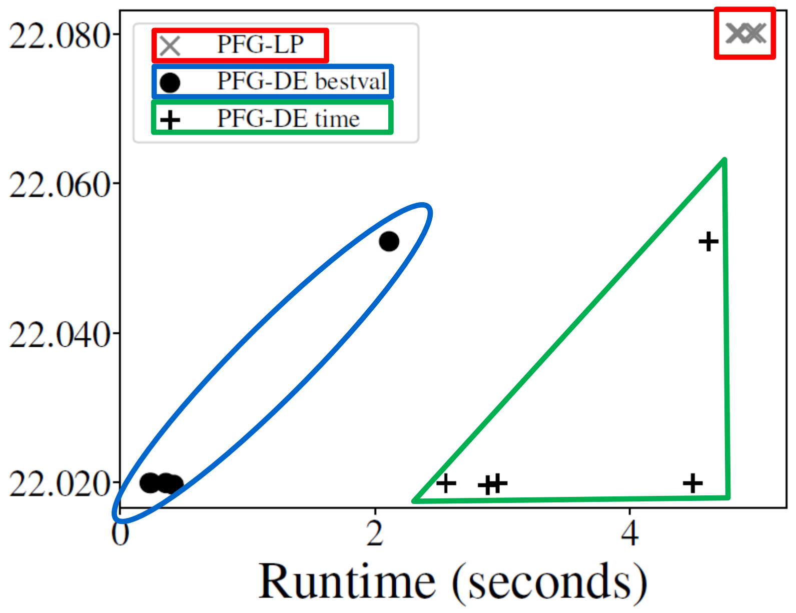


Comparing the two algorithms on $\mathbb{I}_\infty\left(\vec{\mathrm{S}}_G;\vec{\mathrm{Y}}_G\right)$.

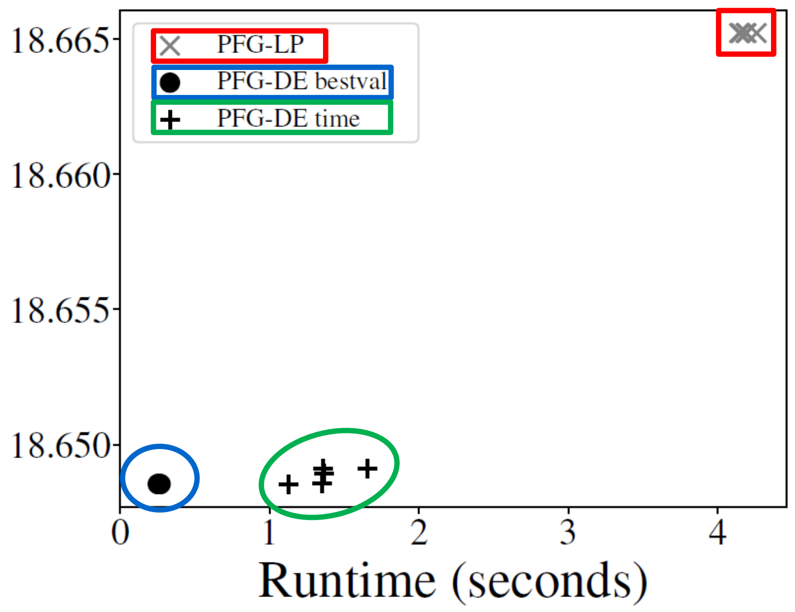


Comparing the two algorithms on sequences of length 5.

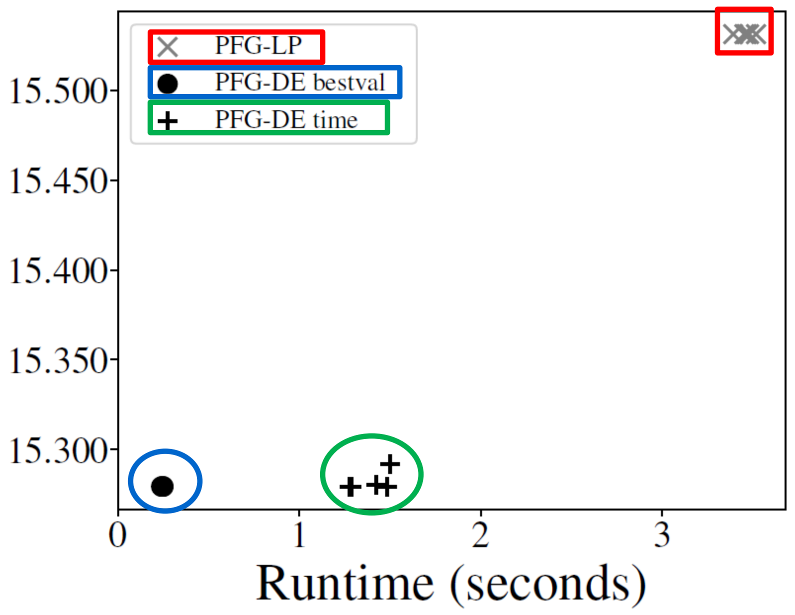
PFG-DE vs. PFG-LP: $\mathbb{I}_\infty$ and Runtime (Wikipedia)



$c = 1.1$



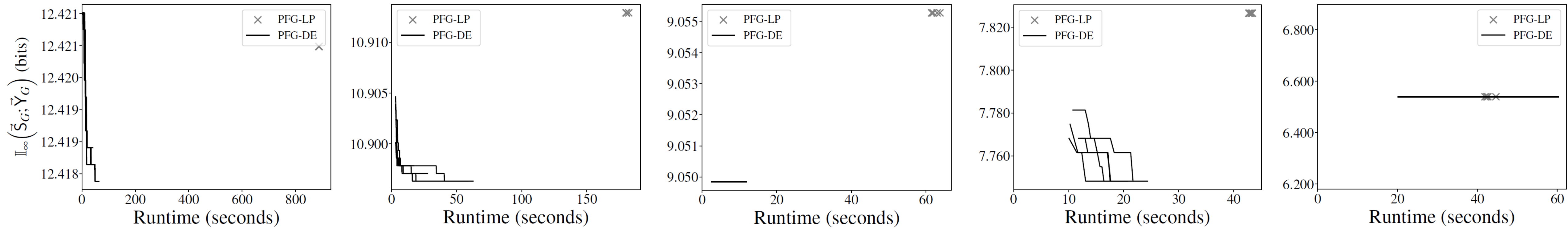
$c = 1.25$



$c = 1.5$

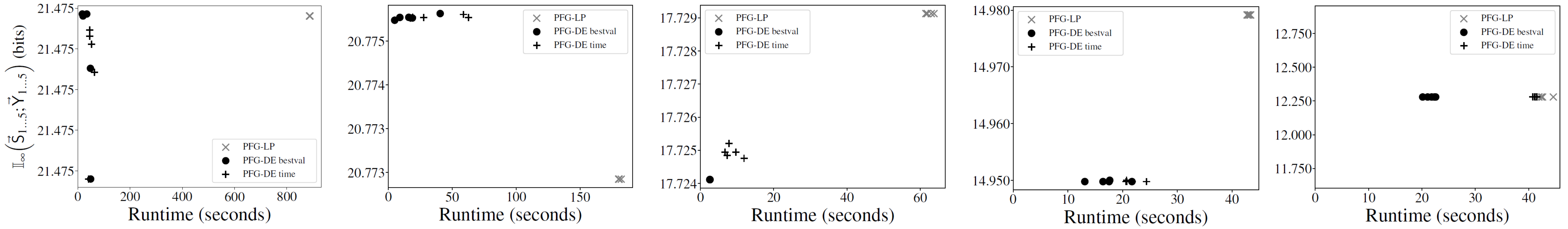
Comparing the two algorithms on sequences of length 5.

PFG-DE vs. PFG-LP: $\mathbb{I}_\infty$ and Runtime (Netflix)



$c = 1.05$ $c = 1.1$ $c = 1.25$ $c = 1.5$ $c = 2.0$

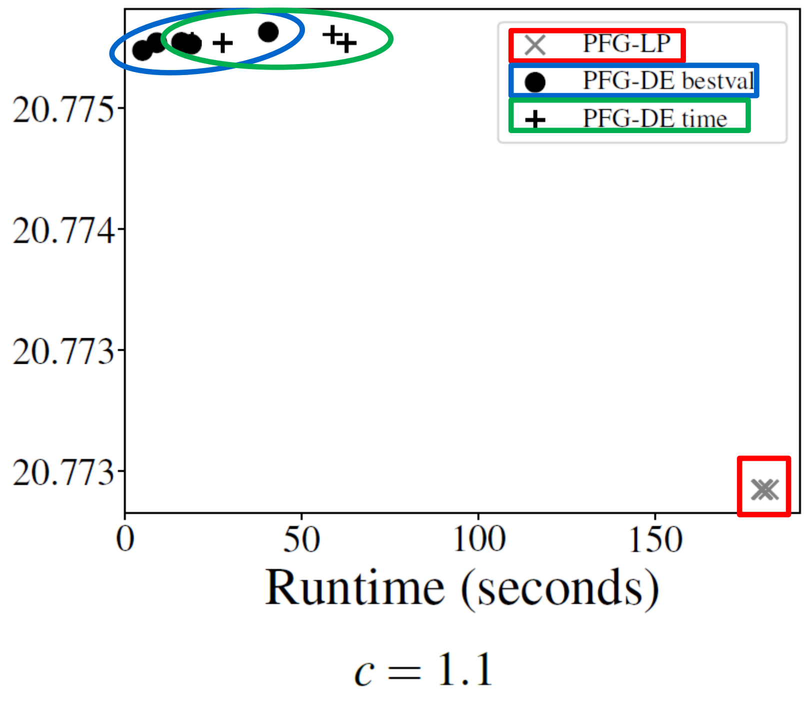
Comparing the two algorithms on $\mathbb{I}_\infty\left(\vec{S}_G;\vec{Y}_G\right)$.

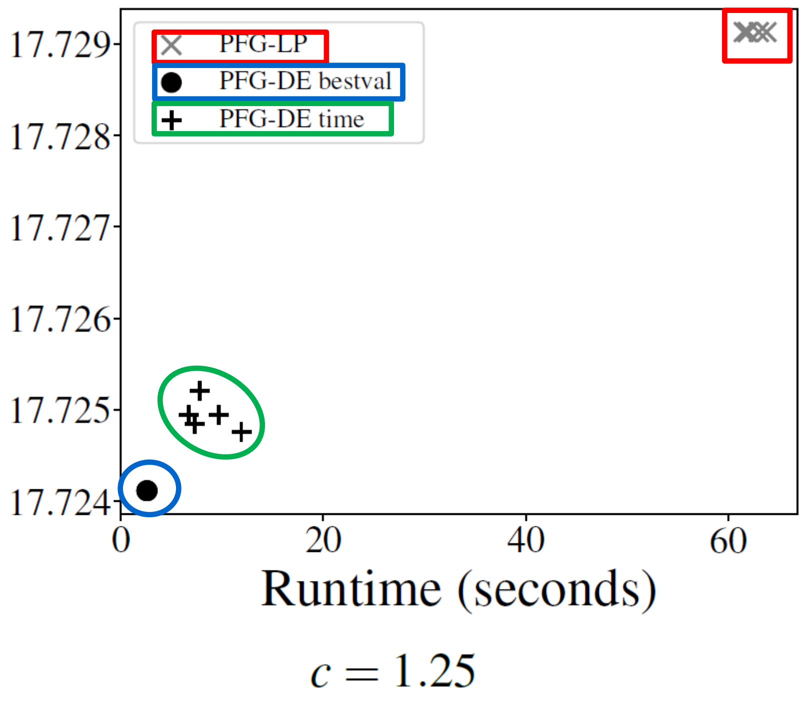


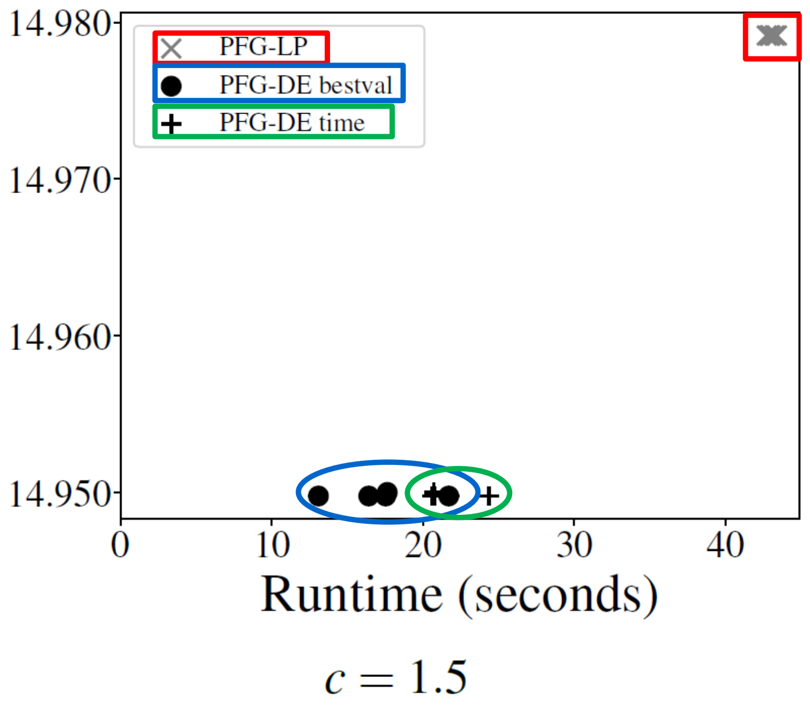
$c = 1.05$ $c = 1.1$ $c = 1.25$ $c = 1.5$ $c = 2.0$

Comparing the two algorithms on sequences of length 5.

PFG-DE vs. PFG-LP: $\mathbb{I}_\infty$ and Runtime (Netflix)







Comparing the two algorithms on sequences of length 5.