Near-Optimal Constrained Padding for Object Retrievals with Dependencies

Pranay Jain, *Duke University;* Andrew C. Reed, *United States Military Academy;* Michael K. Reiter, *Duke University*

https://www.usenix.org/conference/usenixsecurity24/presentation/jain

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 33rd USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 33rd USENIX Security Symposium.

August 14–16, 2024 • Philadelphia, PA, USA

978-1-939133-44-1

USENIX Security '24 Artifact Appendix: Near-Optimal Constrained Padding for Object Retrievals with Dependencies

Pranay Jain
Duke University

Andrew C. Reed
United States Military Academy

Michael K. Reiter
Duke University

A Artifact Appendix

A.1 Abstract

This artifact contains (i) our three datasets; (ii) the implementation code for PFS as well as the algorithms to which we compare, BDK, MVMD-D, and PwoD; and (iii) the code needed to reproduce the results presented in our paper. The source code for each algorithm is provided in separate Python scripts and each experiment is provided as a Jupyter notebook. This document describes how to obtain our artifact, setup the needed packages, and reproduce our results.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

PFS leverages the Gurobi Optimizer package for Python, which requires a license to activate after downloading. If an evaluator does not already have access to a licensed instance of Gurobi, then a free academic license can be obtained after registering an account with Gurobi. Aside from this potential privacy consideration, our artifact has no other security, privacy, or ethical concerns.

A.2.2 How to access

Our artifact can be accessed on GitHub at `https://github.com/pranay-jain/constrained-padding-sequences/releases/tag/1.0.0` or via DOI at `https://doi.org/10.5281/zenodo.13119687`.

A.2.3 Hardware dependencies

We recommend that our artifact be run on a computer with a multi-core CPU and at least 16GB of RAM. As a point of reference, this artifact was developed on an Apple MacBook Pro with an M1 CPU (8 cores) and 16GB of RAM.

A.2.4 Software dependencies

All of the Python packages that are required to use our datasets and code are listed in the `requirements.txt` file that is included in our artifact. This file is formatted so that it can be run with either the `pip` or `pip3` command to install all needed packages (see Sec. A.3.1 for more information).

A.2.5 Benchmarks

The datasets required to reproduce our results are provided as part of this artifact.

A.3 Set-up

A.3.1 Installation

We recommend the following steps to install our artifact.

1. Create and activate a new Python virtual environment, e.g., `conda`, `virtualenv`, or `venv`.
2. Navigate to our artifact (see Sec. A.2.2), download the ZIP file, and extract all files. Alternatively, you can clone the entire GitHub repository for an up-to-date version.
3. Navigate into `constrained-padding-sequences/` and install the required Python packages using the following command (replace `pip` with `pip3` for `python3` environments):
   `pip install -r requirements.txt`
4. If you do not already have access to Gurobi, then you will need to activate the instance of `gurobipy` with a license. In that case, our recommendation is to obtain an *Academic Named-User License*.[1]

A.3.2 Basic Test

Perform the following steps to verify that installation was successful.

[1] Instructions for how to obtain a Gurobi academic license can be found at `https://www.gurobi.com/academia/academic-program-and-licenses/`. Note that the final step of obtaining a Gurobi license requires you to activate the license while connected to your academic network. The README included with our artifact provides additional details for this step.

1. From a command line, navigate into:
   `constrained-padding-sequences/experiments/`
2. Start Jupyter Notebook with the following command:
   `jupyter notebook`
3. Within Jupyter Notebook, open the following notebook:
   `fig2.ipynb`
4. Once the notebook opens, run the notebook with:
   `Kernel > Restart Kernel and Run All Cells`
5. Verify that the resultant figures match Fig. 2 in our paper.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): Compared to BDK, PFS provides better $\mathbb{I}\left(\vec{S};\vec{Y}\right)$ for a much lower $c_{\max}$. This is proven by experiment (E1) described in Sec. 5.3 whose results are illustrated in Fig. 2.

(C2): Compared to MVMD-3, PFS provides roughly the same, and in some cases much better, ℓ_{avg} for a much lower $c_{\max}$. This is proven by experiment (E2) described in Sec. 5.3 whose results are illustrated in Fig. 3.

(C3): Compared to both BDK and MVMD-3, PFS provides better $\mathbb{I}_{\infty}\left(\vec{S};\vec{Y}\right)$ for a much lower $c_{\max}$. Compared to PwoD, PFS generally provides better $\mathbb{I}_{\infty}\left(\vec{S};\vec{Y}\right)$ for the same c_{tgt} (on the Wikipedia dataset, PwoD and PFS provide comparable $\mathbb{I}_{\infty}\left(\vec{S};\vec{Y}\right)$). This is proven by experiment (E3) described in Sec. 5.4 whose results are illustrated in Fig. 4 and Fig. 5.

(C4): When assessing an adversary's precision and recall as it predicts sequences of objects by observing their padded sizes, PFS is consistently either among the top performers or beats the other algorithms by a wide margin. This is proven by experiment (E4) described in Sec. 5.5 whose results are illustrated in Fig. 6, Fig. 7, and Fig. 8.

(C5): PFG is a competitive alternative to PFS in those instances where an object store is unable to generate $\vec{S}$ or doing so is too costly. This is proven by experiment (E5) described in Sec. 6.2 whose results are illustrated in Fig. 11.

A.4.2 Experiments

(E1): [2 human-minutes + 5 compute-minutes]:
How to: Run `fig2.ipynb` within Jupyter notebook.
Preparation: None.
Execution: Once the notebook opens, run it with:
`Kernel > Restart Kernel and Run All Cells`
Results: The resultant figures should match Fig. 2.

(E2): [10 human-minutes + 20 compute-minutes]:
How to: Run `fig3.ipynb` three times within Jupyter notebook (once for each dataset).
Preparation: Prior to running the notebook, locate the cell where datasets are set. Uncomment only the dataset you plan to evaluate.
Execution: After the correct dataset has been uncommented, run the notebook with:
`Kernel > Restart Kernel and Run All Cells`
Results: The resultant figures should match Fig. 3.

(E3): [10 human-minutes + 20 compute-minutes]:
How to: Run `fig4-5-11.ipynb` four times within Jupyter notebook (once for each dataset).
Preparation: Prior to running the notebook, locate the cell where datasets are set. Uncomment only the dataset you plan to evaluate.
Execution: After the correct dataset has been uncommented, run the notebook with:
`Kernel > Restart Kernel and Run All Cells`
Results: The resultant figures should match Fig. 4 and Fig. 5.

(E4): [2 human-minutes + 10 compute-minutes]:
How to: Run `fig6-7-8.ipynb` within Jupyter notebook.
Preparation: None.
Execution: Once the notebook opens, run it with:
`Kernel > Restart Kernel and Run All Cells`
Results: The resultant figures should match Fig. 6, Fig. 7, and Fig. 8.

(E5): [5 human-minutes + 10 compute-minutes]:
How to: Run `fig4-5-11.ipynb` two times within Jupyter notebook (once for each dataset).
Preparation: Prior to running the notebook, locate the cell where datasets are set. Uncomment only the dataset you plan to evaluate. For this experiment, use the Autocomplete and Wikipedia datasets.
Execution: After the correct dataset has been uncommented, run the notebook with:
`Kernel > Restart Kernel and Run All Cells`
Results: The resultant figures should match Fig. 11.

A.5 Notes on Reusability

Our artifact contains the files `load_dataset.py` and `pfs.py` which provide access to our three datasets and to PFS, respectively. We encourage others to use them in their own research.[2]

A.6 Version

Based on the LaTeX template for Artifact Evaluation V20231005. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2024/.

[2] Our artifact is licensed under the MIT License.