BuildHON^2: A Scalable Method for Growing Higher Order Networks

STEVEN KRIEG

SCALABLE GRAPH ALGORITHMS

UNIVERSITY OF NOTRE DAME

Presentation Overview

1. **Background & Motivation**
2. Design & Implementation
3. Evaluation
4. Conclusions & Future Work

Higher Order Networks

How do we **represent** big data as a network, while accurately preserving dependencies?

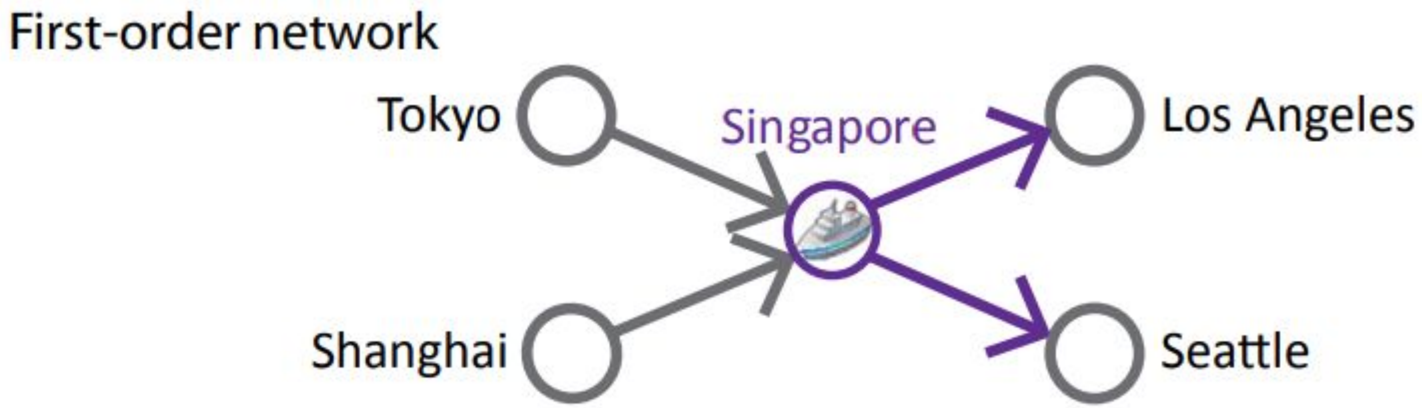
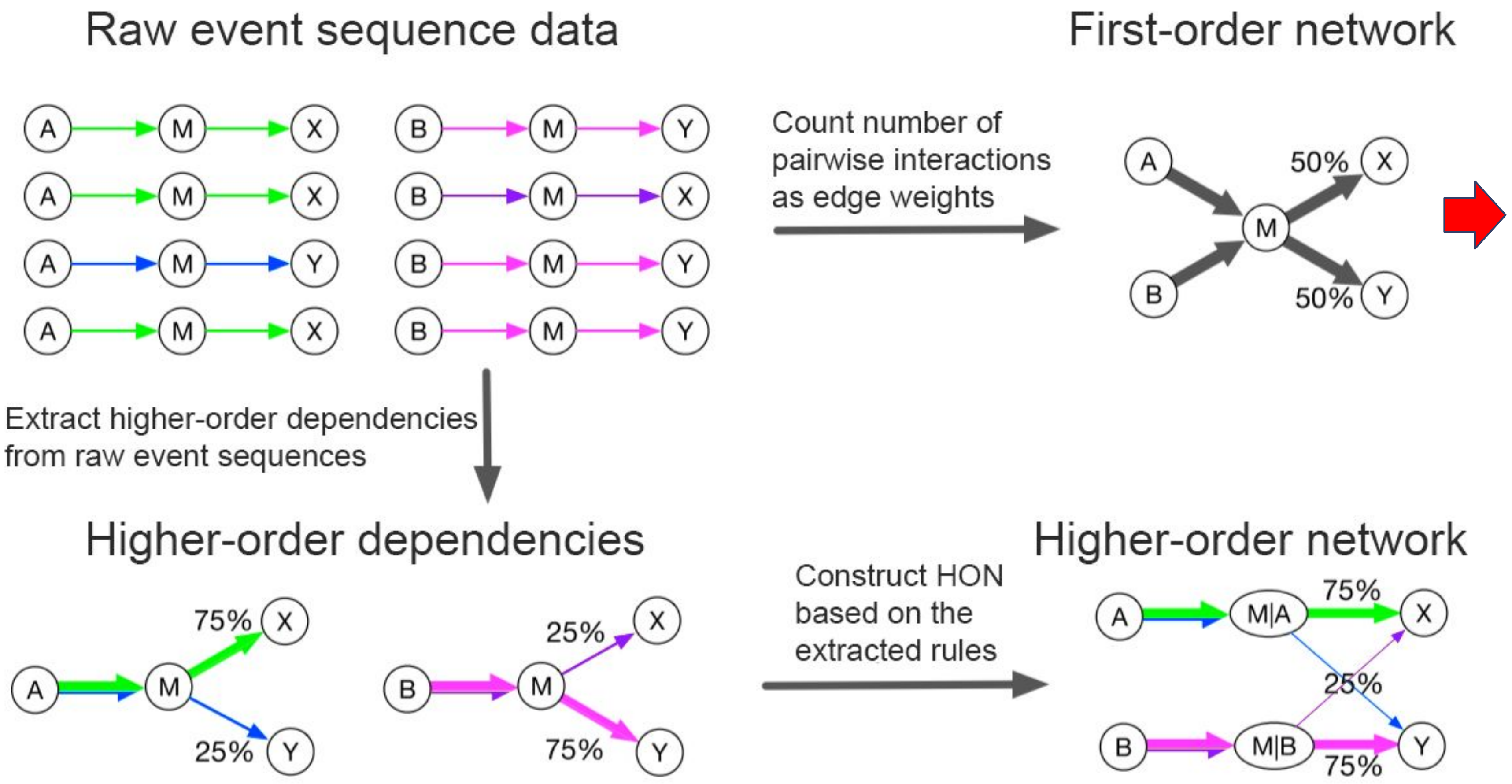


Figure from [2]

Higher Order Networks



(Markov model)

Figure from [2]

Sequential Implementation

Published version written in Python [3]:

1. Extracts all 2 node sequences
2. Checks to see if adding a **prefix** to a sequence "significantly" changes the movement confidence
3. If so, preserve the 3 node sequence as a **rule** and try extending further
4. Repeat until convergence or max order is reached

Sequential Implementation

My version written in C++:

- Scaled horribly due to repeated searches of sequence data
- Python version uses an "indexing cache" to store the locations of sequences
    - Better performance
    - Large memory footprint

Enter Fp-Trees [7]

TID	Items
1	{a, b}
2	{b, c, d}
3	{a, c, d, e}
4	{a, d, e}
5	{a, b, c}
6	{a, b, c, d}
7	{a}
8	{a, b, c}
9	{a, b, d}
10	{b, c, e}

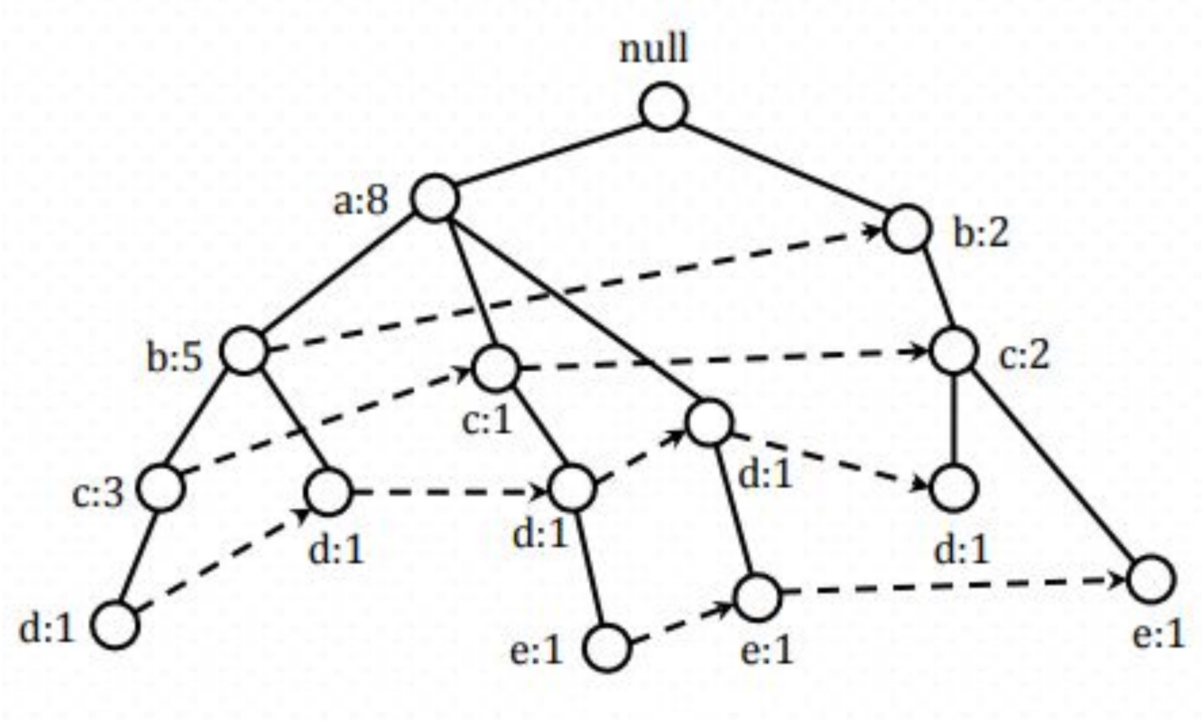
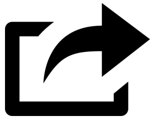


Figure from [9]

Fp-Trees

	Algorithmic Component	Helpful to HON?
	Growth*	
	Extraction**	

* No way to efficiently find rules of higher/lower order

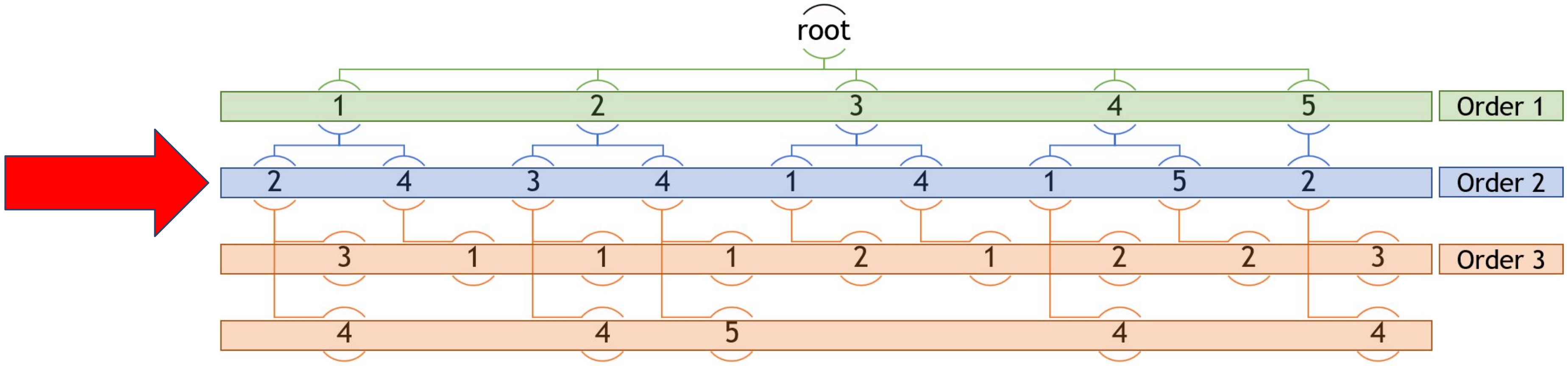
** Extraction does not account for sequential ordering

Design & Implementation

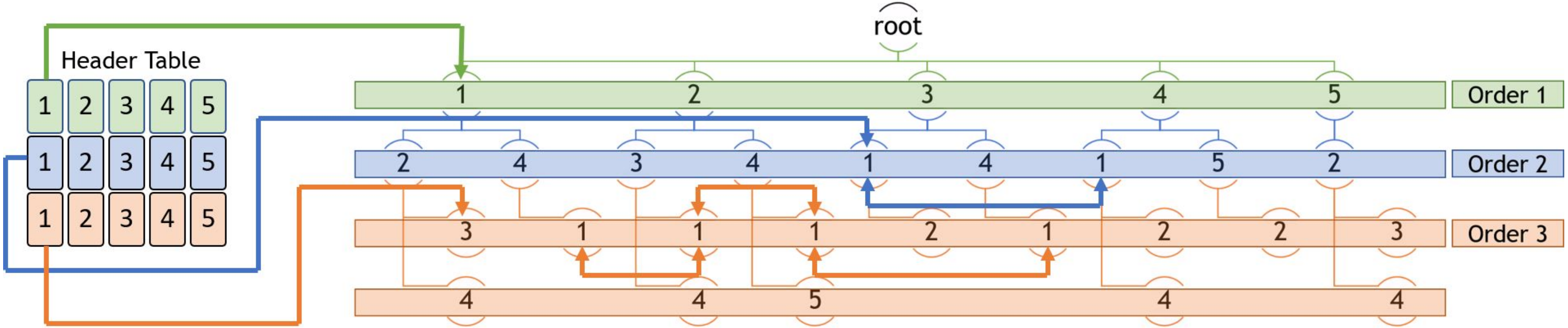
1. Background & Motivation
2. **Design & Implementation**
3. Evaluation
4. Conclusions & Future Work

Modified Fp-Tree

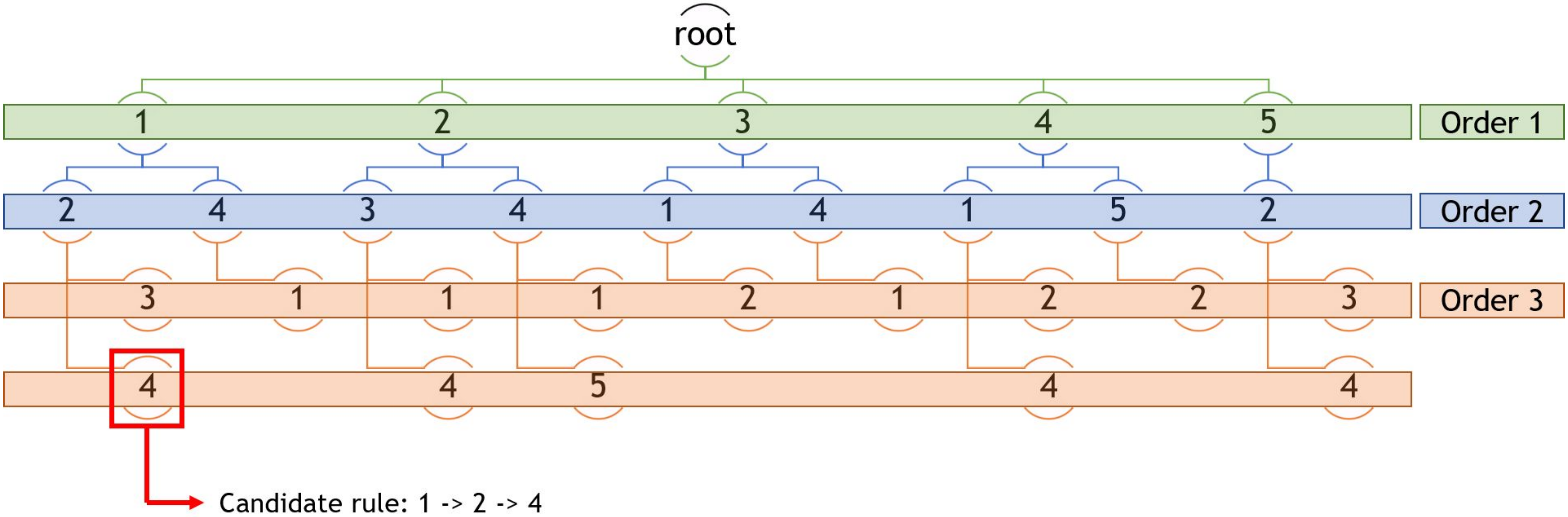
TID	Sequence
1	{1, 2, 3, 1, 2, 3}
2	{1, 2, 3, 1, 2, 3}
3	{1, 2, 4, 1, 2, 4}
4	{1, 2, 3, 1, 2, 3}
5	{2, 3, 4, 1, 4, 1}
6	{5, 2, 4, 5, 2, 4}
7	{5, 2, 4, 5, 2, 4}
8	{5, 2, 4, 5, 2, 3}



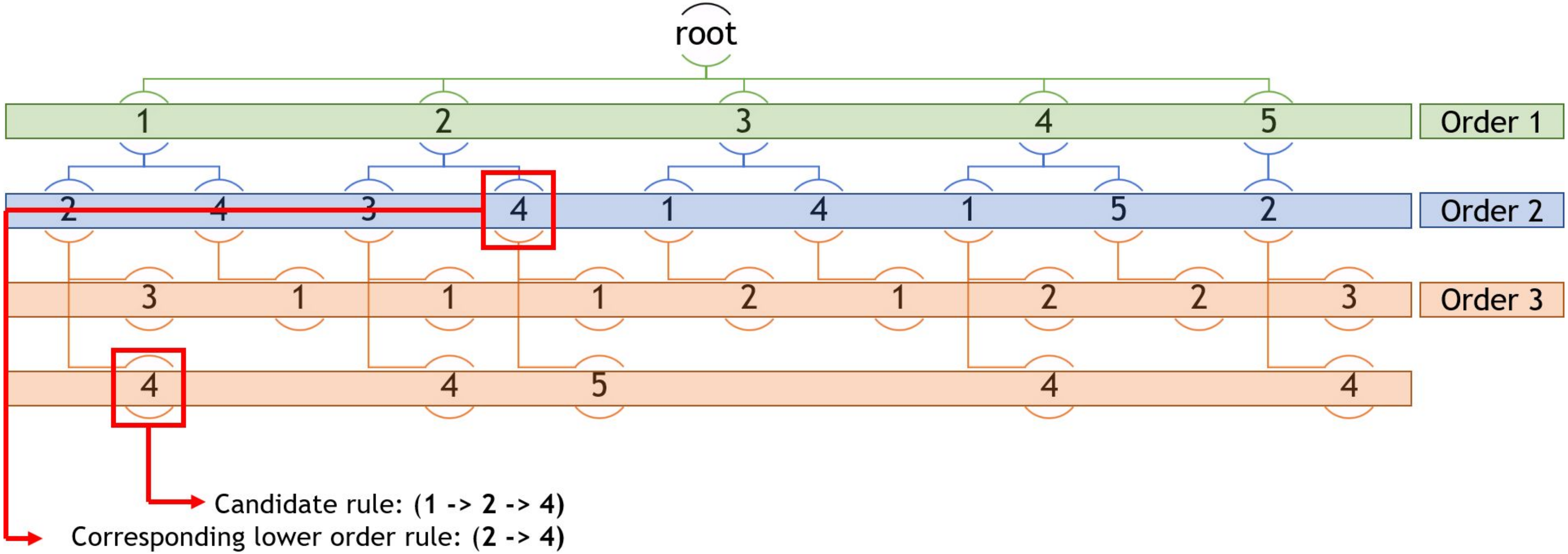
2D Header Table + Cousin Links



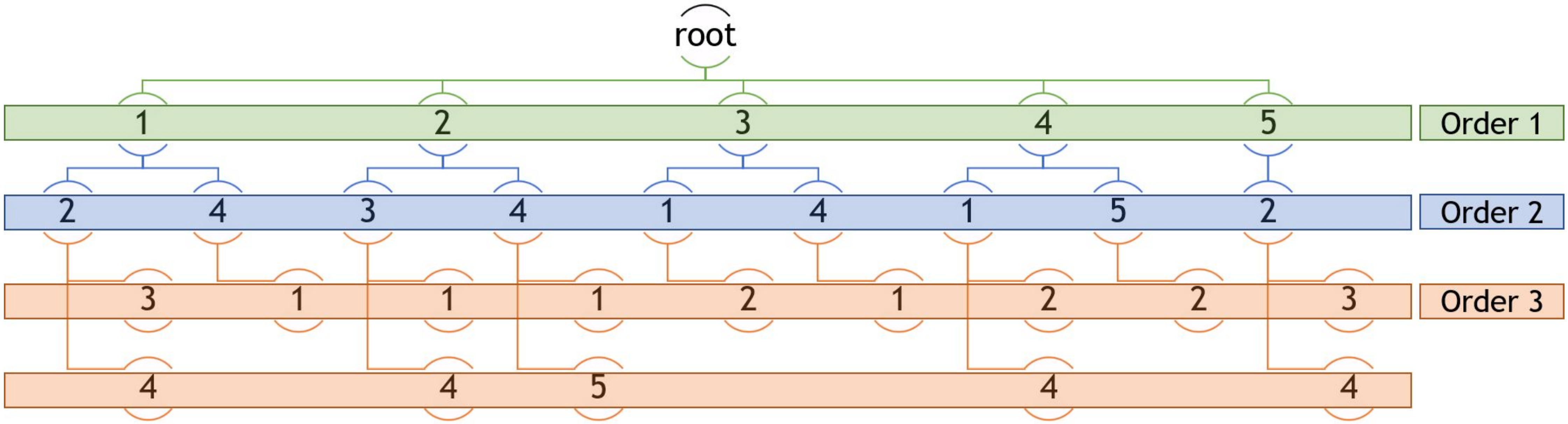
Pruning the Tree



Pruning the Tree



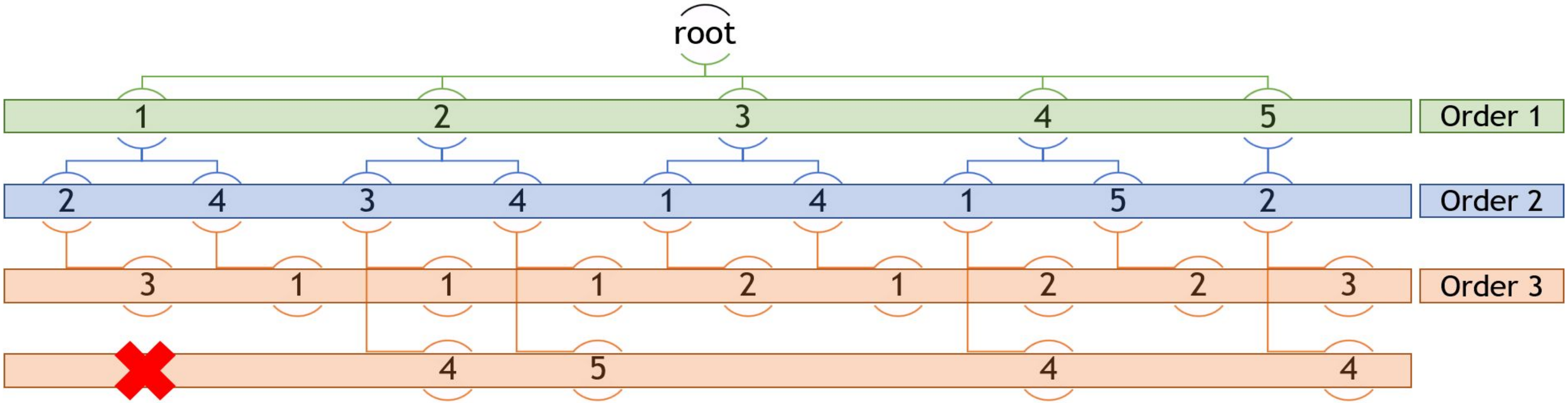
Pruning the Tree



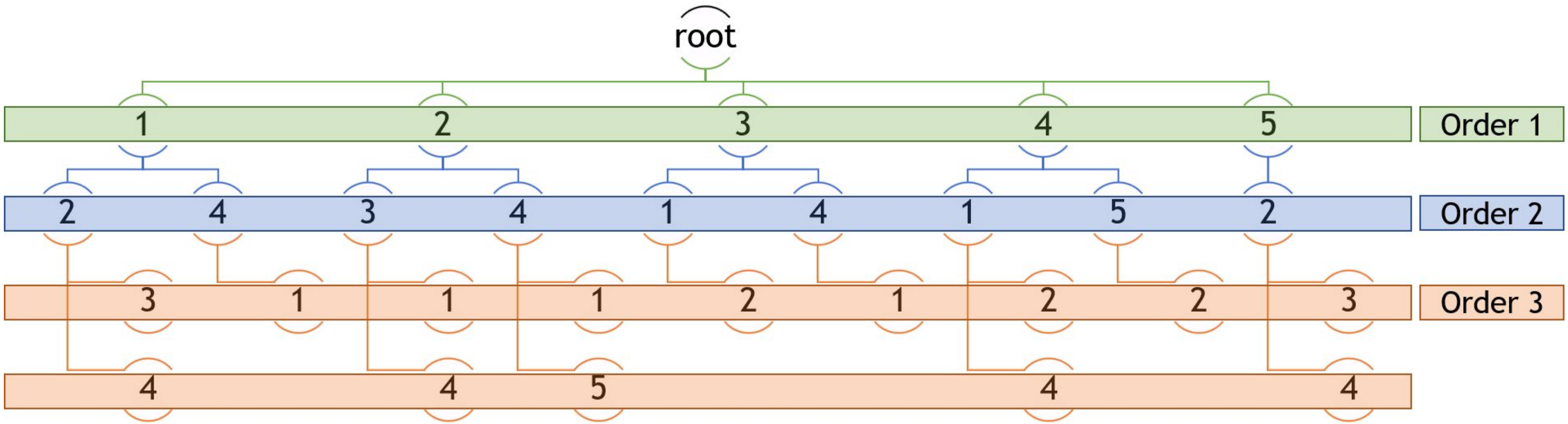
Candidate rule: (**1** -> **2** -> **4**)
Corresponding lower order rule: (**2** -> **4**)

if conf (**1** -> **2** -> **4**) !~ conf (**2** -> **4**):
prune (**1** -> **2** -> **4**)

Pruning the Tree



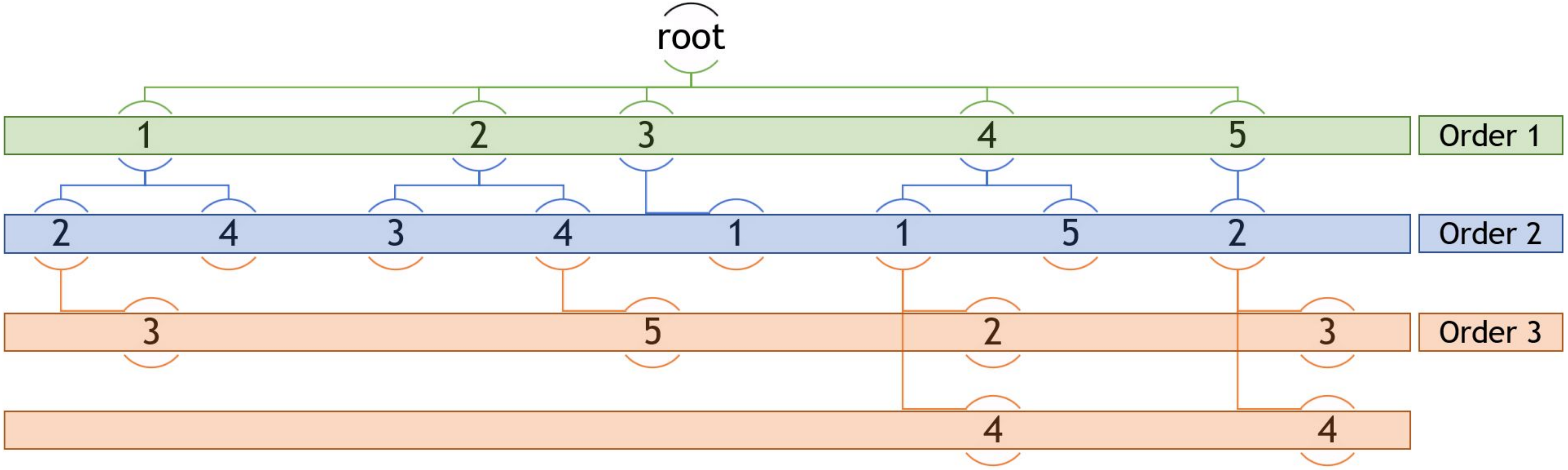
Pruning the Tree



Candidate rule: (**1 -> 2 -> 4**)
Corresponding lower order rule: (**2 -> 4**)

else:
<u>prune (2 -> 4)</u>

Pruned Tree



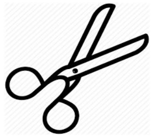
Driver Code

```
#include "hon_includes.h"

int main(int argc, char** argv) {
    std::string inf_path = (argc > 1) ? argv[1] : DEF_INF_PATH;
    int num_seq_prefixes = argc > 2 ? std::stoi(argv[2]) : DEF_NUM_SEQ_PREFIXES;
    int max_order = argc > 3 ? std::stoi(argv[3]) : DEF_MAX_ORDER;
    bool verbose = argc > 4 ? std::stoi(argv[4]) : DEF_VERBOSE;

    if (verbose) std::cout << "Building tree from file: " << inf_path << std::endl;
    FpTree fpt = FpTree(inf_path, num_seq_prefixes, max_order, verbose);
    if (verbose) std::cout << "Tree build complete!" << std::endl;
    fpt.prune();
    fpt.dump_rules();
    return 0;
}
```

Complexity Analysis

	Algorithm	Time	Space
	Growing	O(k * m)	O(n ^ m) - dense O(n * m) - sparse
	Pruning	O(n ^ m) - dense O(n * m) - sparse	--

k = total length of data set
m = max order
n = number of unique nodes
Density/sparsity refers to the number of first-order (pairwise) relationships between nodes

Implementation

- C++11 using only the standard library (lots of unordered maps - possible reallocation issue?)
- Two custom classes:
  - FpTree: ~300 lines of code
  - FpNode.cpp: ~100 lines of code
- ~480 total lines of code; remaining 80 are driver / header definitions

Evaluation

1. Background & Motivation
2. Design & Implementation
3. **Evaluation**
4. Conclusions & Future Work

Synthetic Data Sets

	Data Set	Number of Records	Record Length	Total Size	Unique Nodes
CSV	SyntheticFull	10,000	100	1,000,000	100
CSV	SyntheticLarge	100,000	100	10,000,000	1,000
CSV	SyntheticGiant	1,000,000	100	100,000,000	1,000
CSV	SyntheticBalrog	10,000,000	100	1,000,000,000	1,000

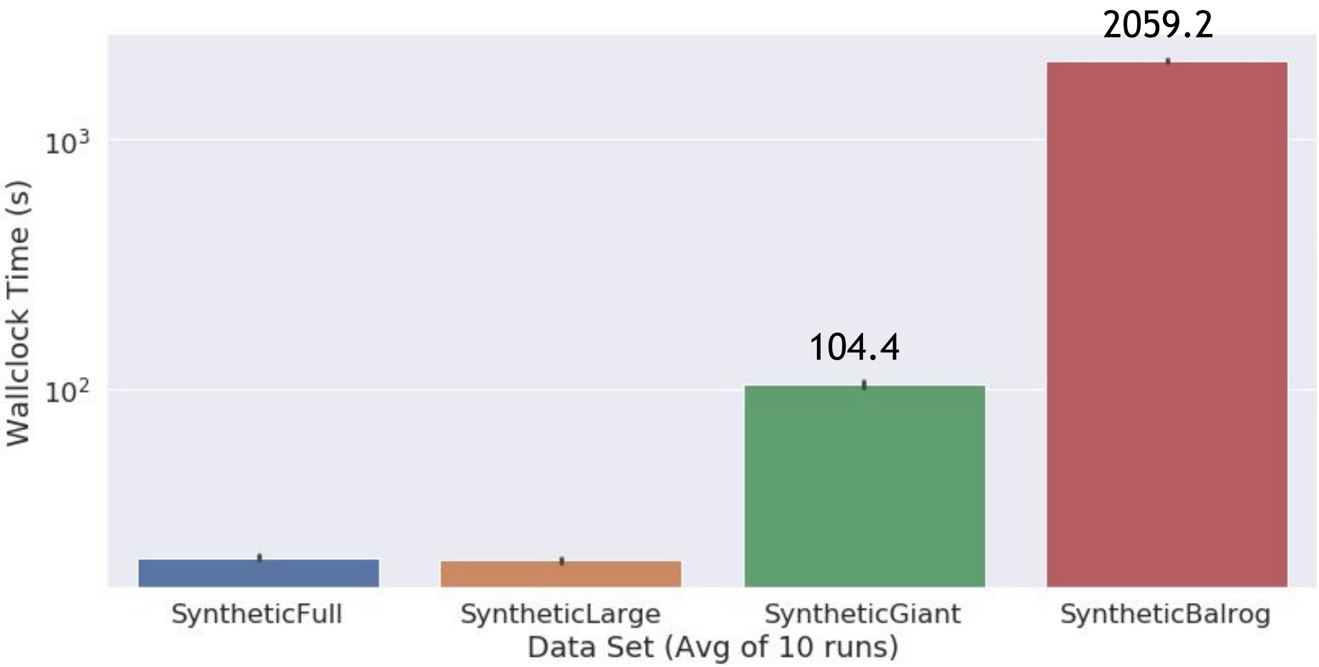
Scalability Experiment

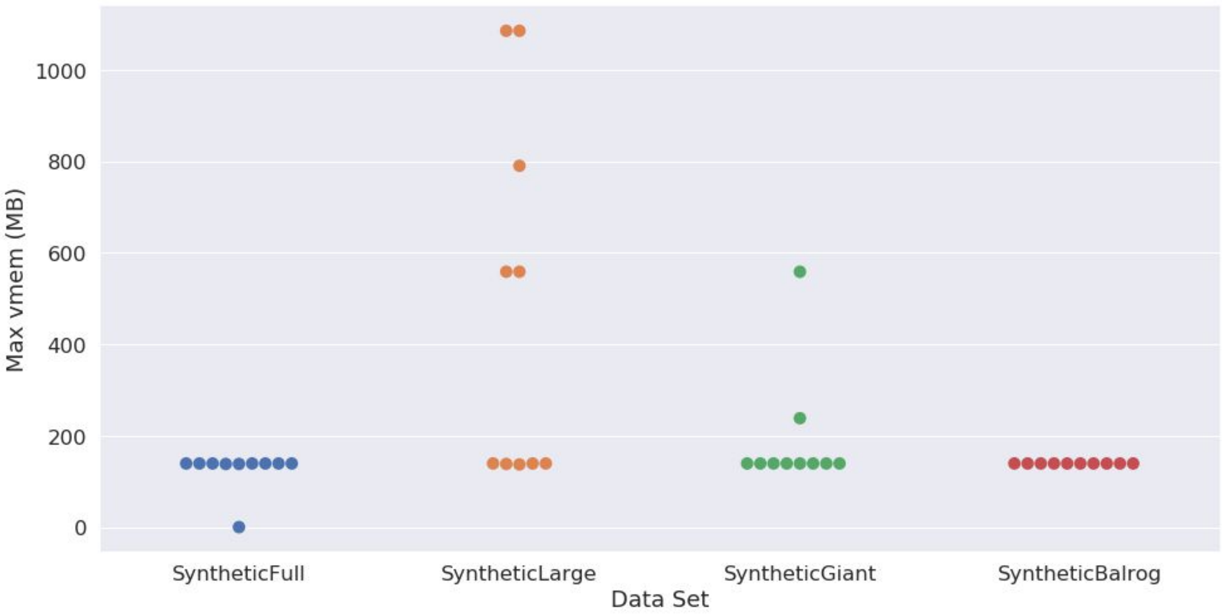
	Application	Data Collection	Runtime Parameters	Iterations
	HON^2 (CHON)	Base Synthetic	max_order = 5	40 (10 runs * 4 data sets)
	HON+ (Python)	Base Synthetic	max_order = 99	40 (10 runs * 4 data sets)
	HON^2 (CHON)	Extended Synthetic	max_order = 5	40 (10 runs * 4 data sets)
	HON^2 (CHON)	SyntheticGiant only	max_order = [3, 4, 5, 6, 7, 8, 9]	70 (10 runs * 7 parameters)

Key metrics:

- Runtime (wallclock)
- Max vmem usage

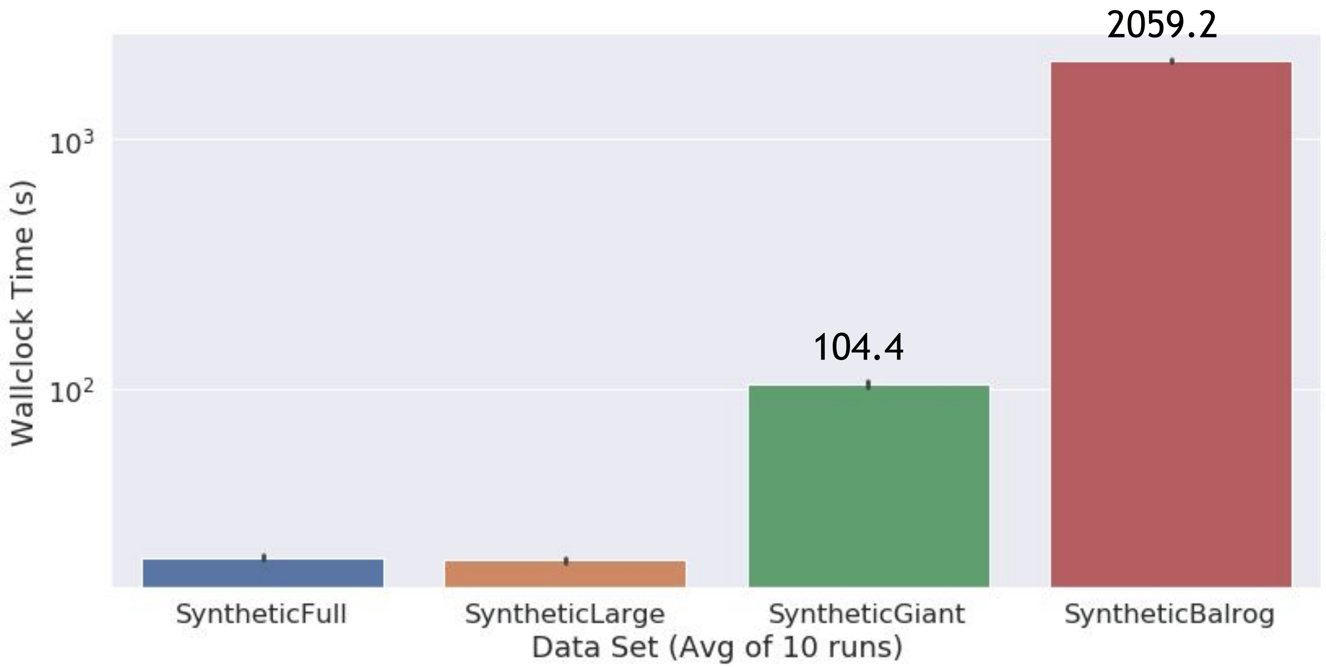
Results: Increasing Total Size

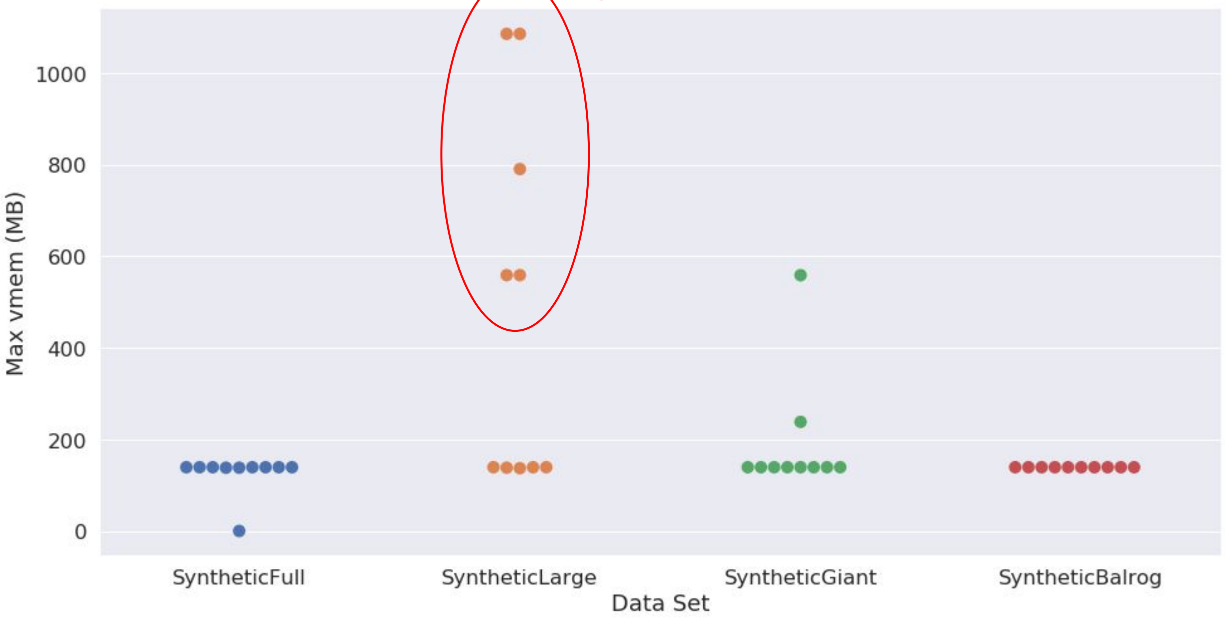




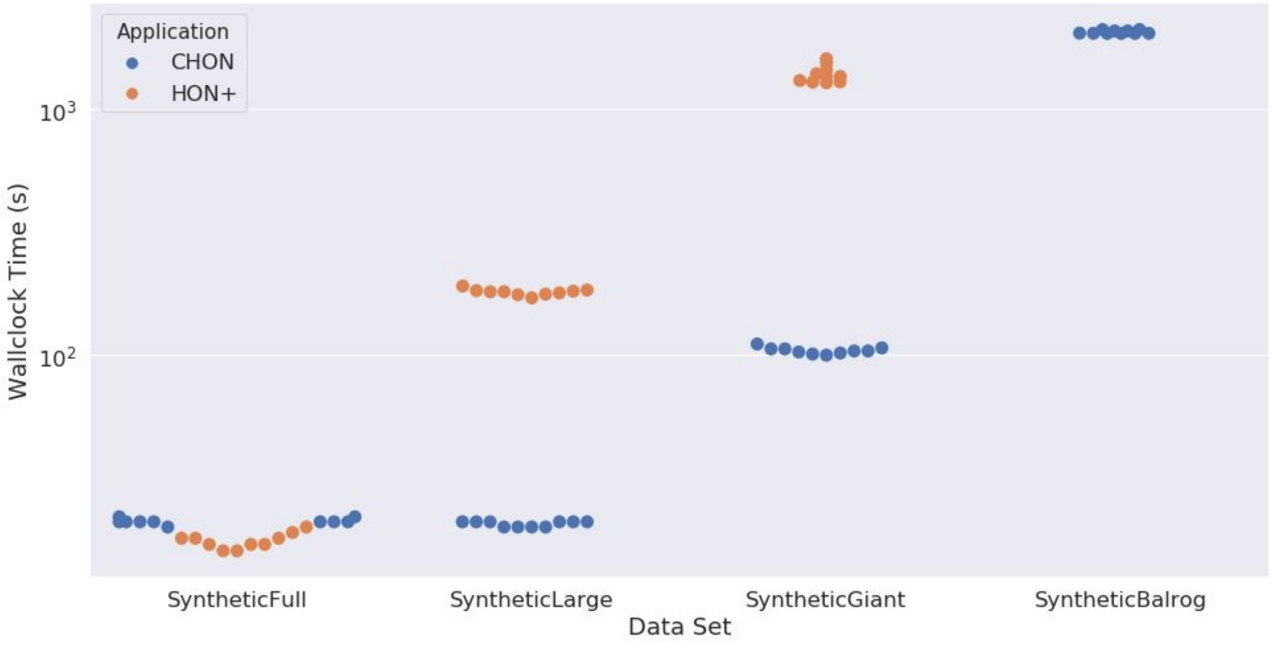
Results: Increasing Total Size

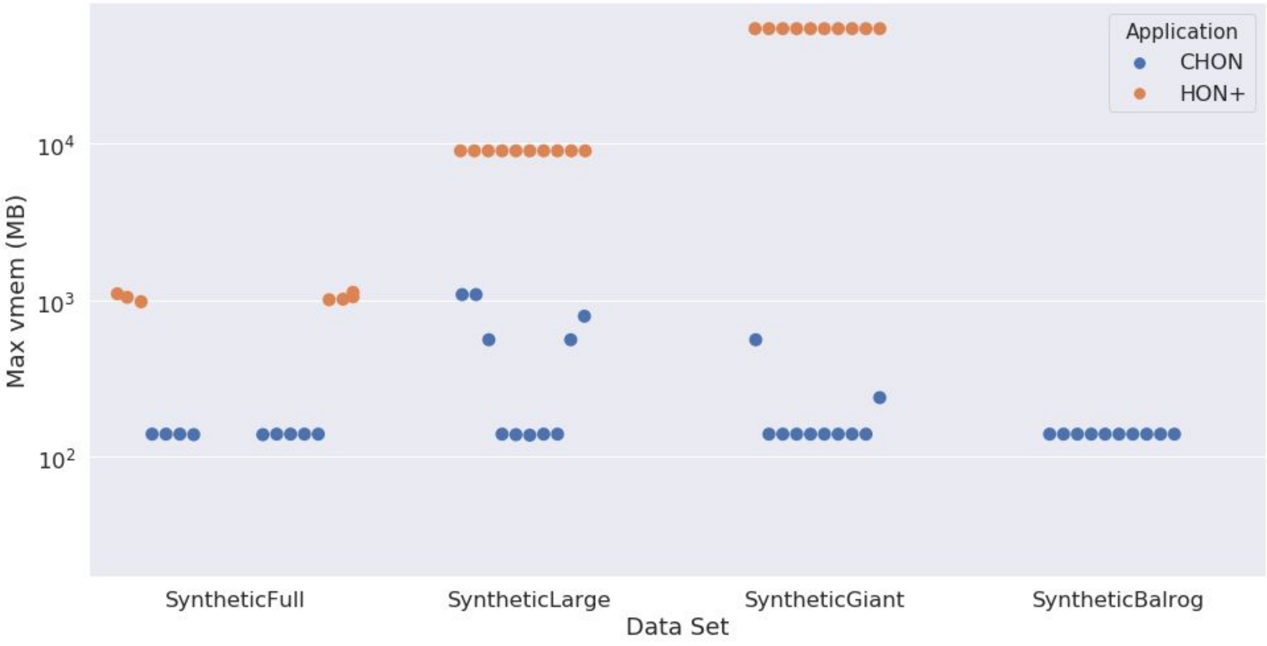
CRC shenaniganry?





Results: Baseline Comparison

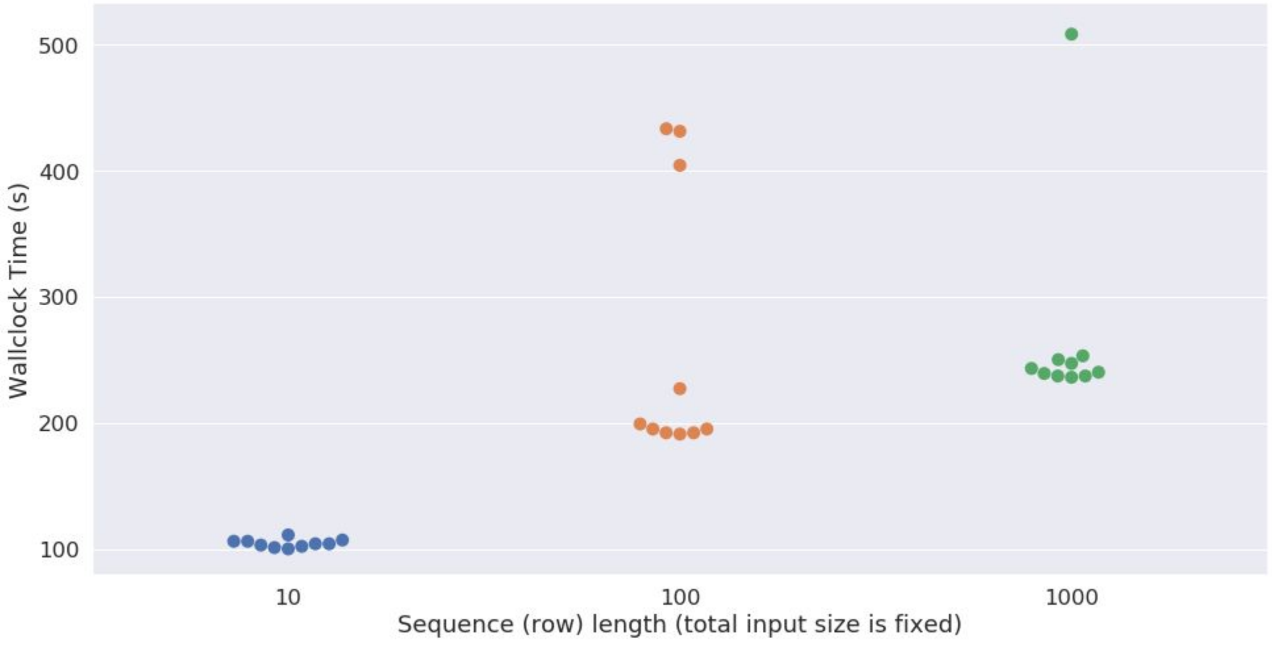


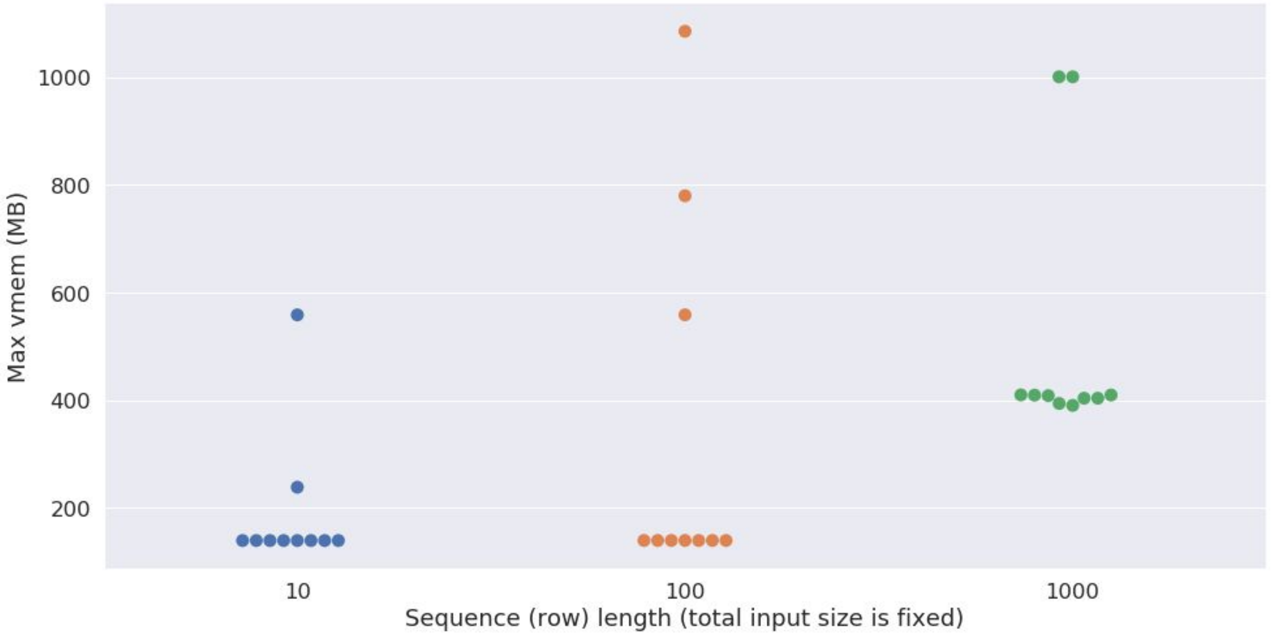


Synthetic Data Set Extensions

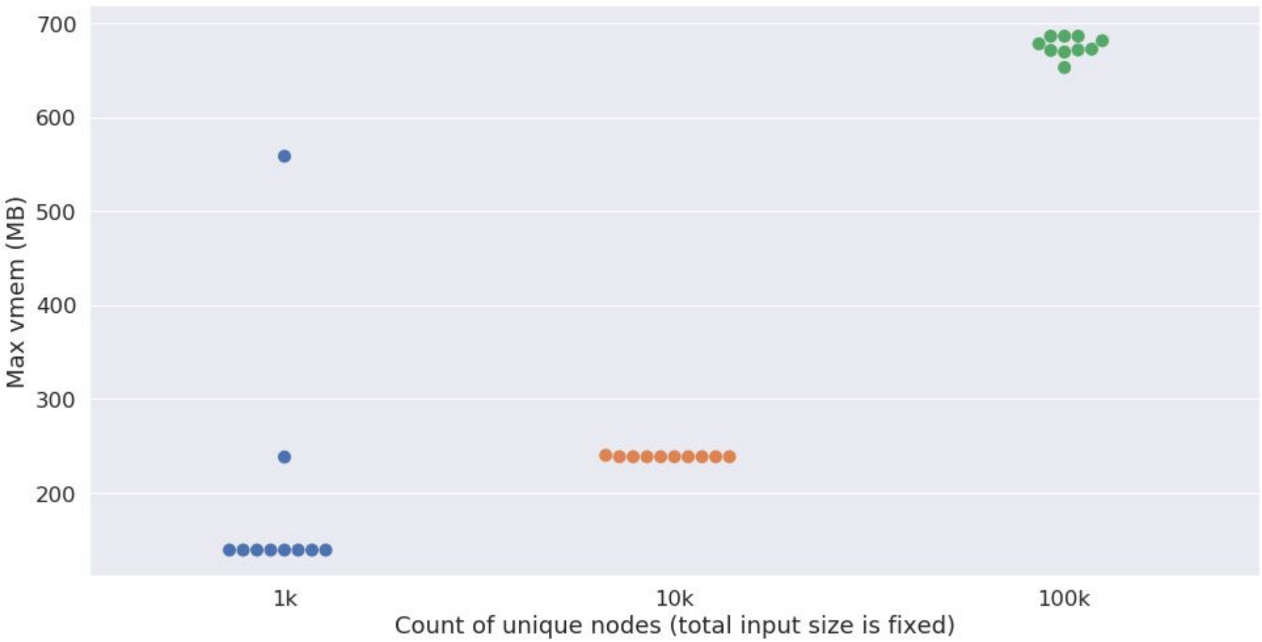
	Base Data Set	Number of Records	Record Length	Total Size	Unique Nodes
CSV	SyntheticGiant	10,000	100	1,000,000	**10,000 (x10)**
CSV	SyntheticGiant	100,000	100	10,000,000	**100,000 (x100)**
CSV	SyntheticBalrog	1,000,000	**10 (x0.1)**	100,000,000	1,000
CSV	SyntheticLarge	100,000	**1000 (x10)**	100,000,000	1,000

Results: Varying Sequence Length

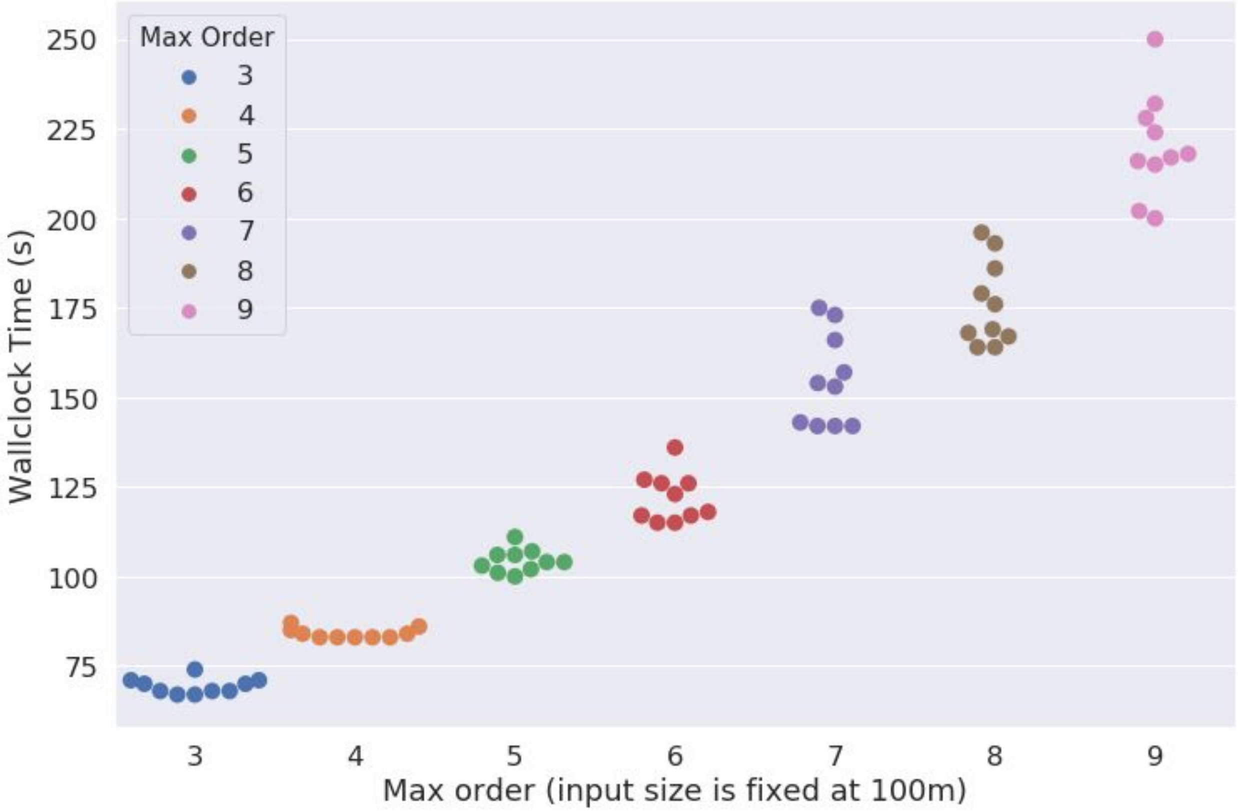


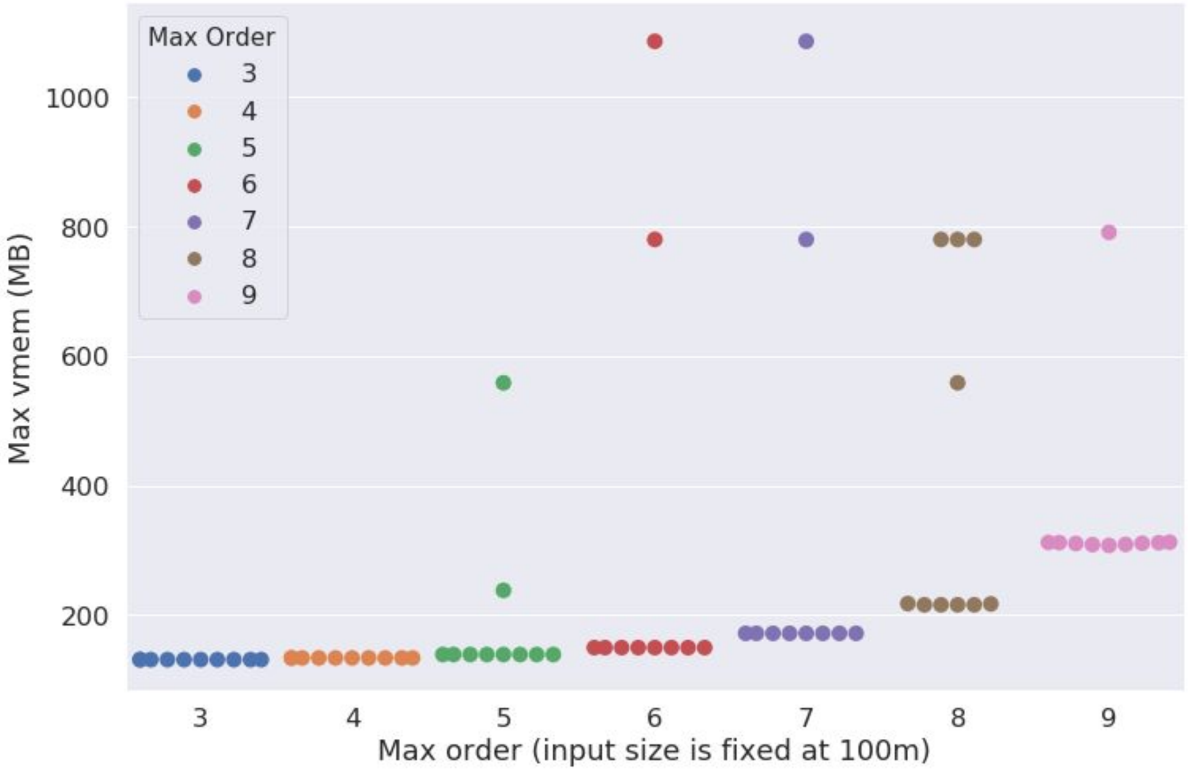


Results: Varying Unique Count



Results: Varying Max Order





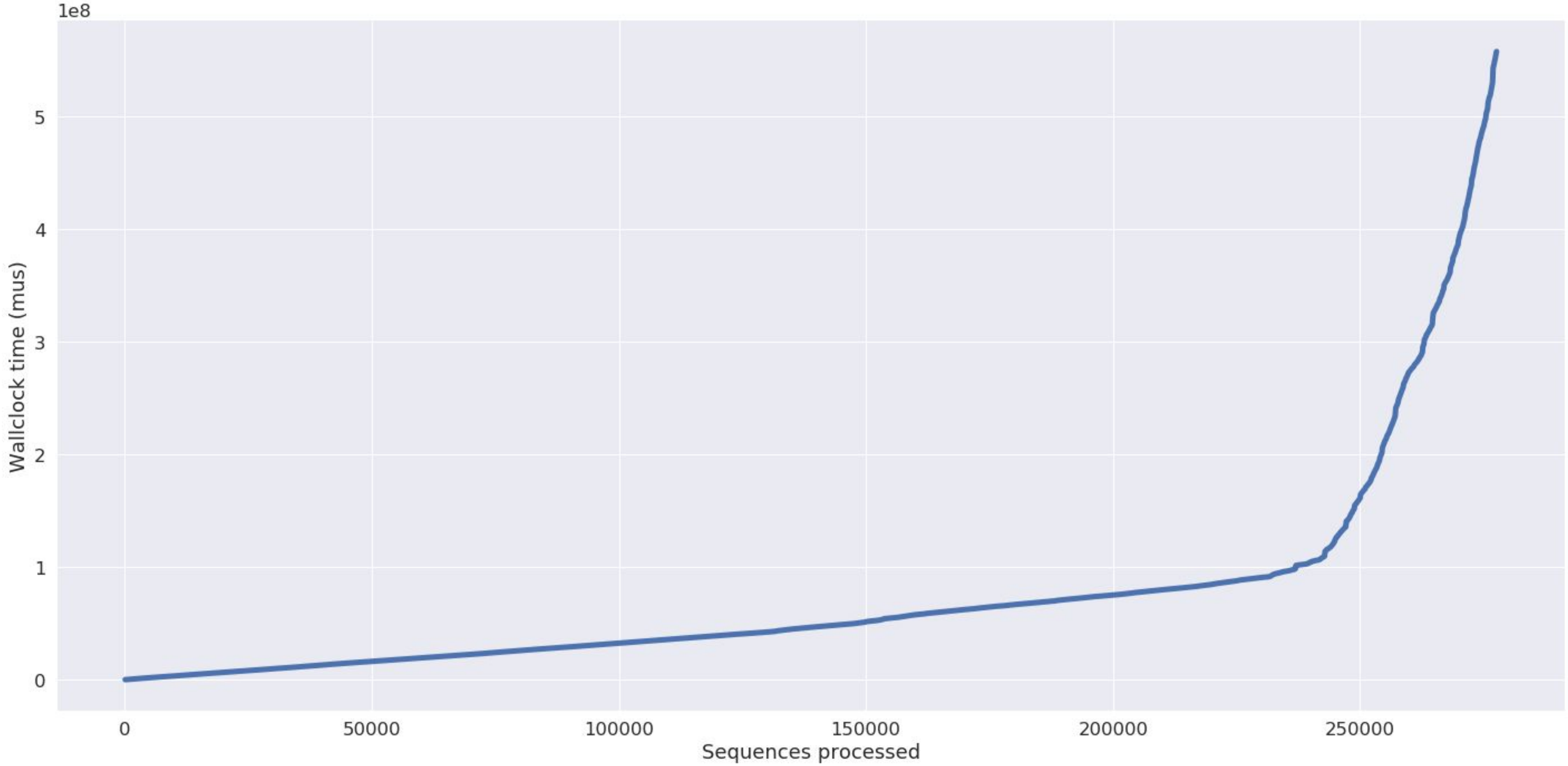
Real-world* Data

	Data Set	Number of Records	Record Length	Total Size	Unique Nodes
	2018 NY Taxi Data	4,306,477	52	223,387,351	266

Each item in the sequence is a “Taxi Zone” in which a trip begins or ends (http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml)

* I had to artificially link sequences :-(

Results: NY Taxi Data

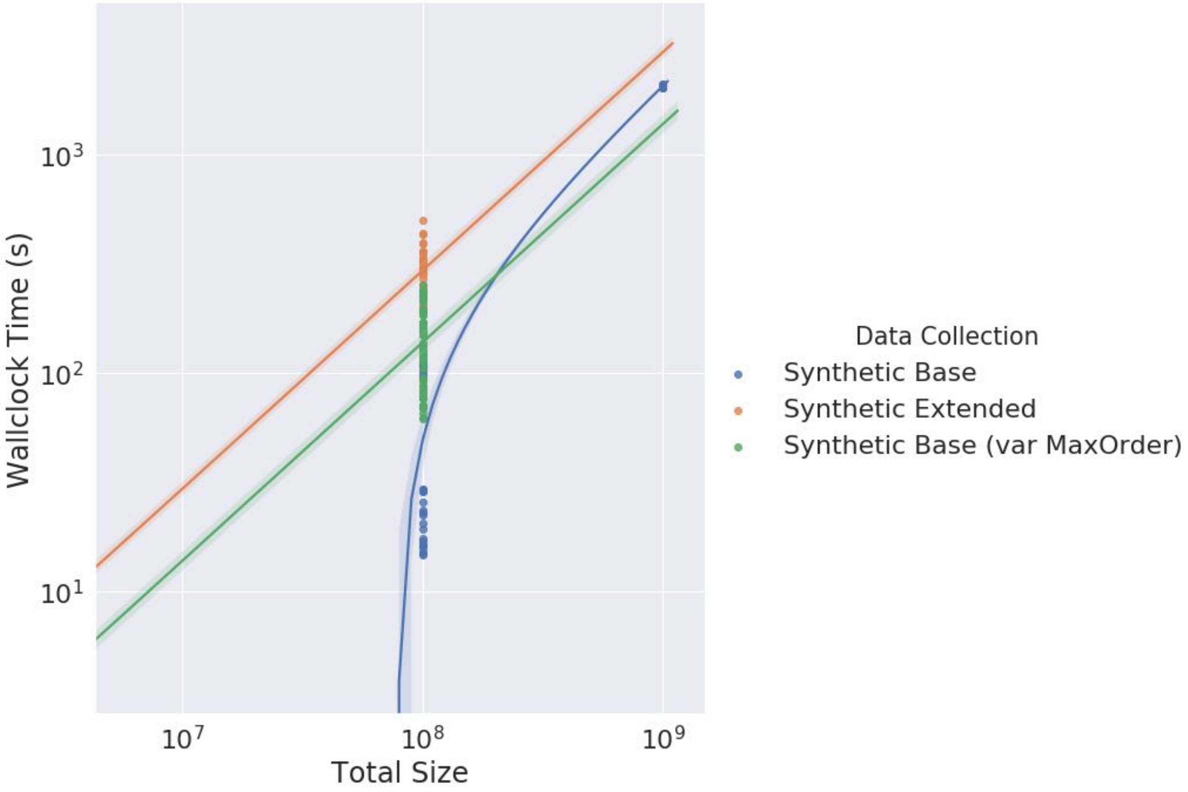


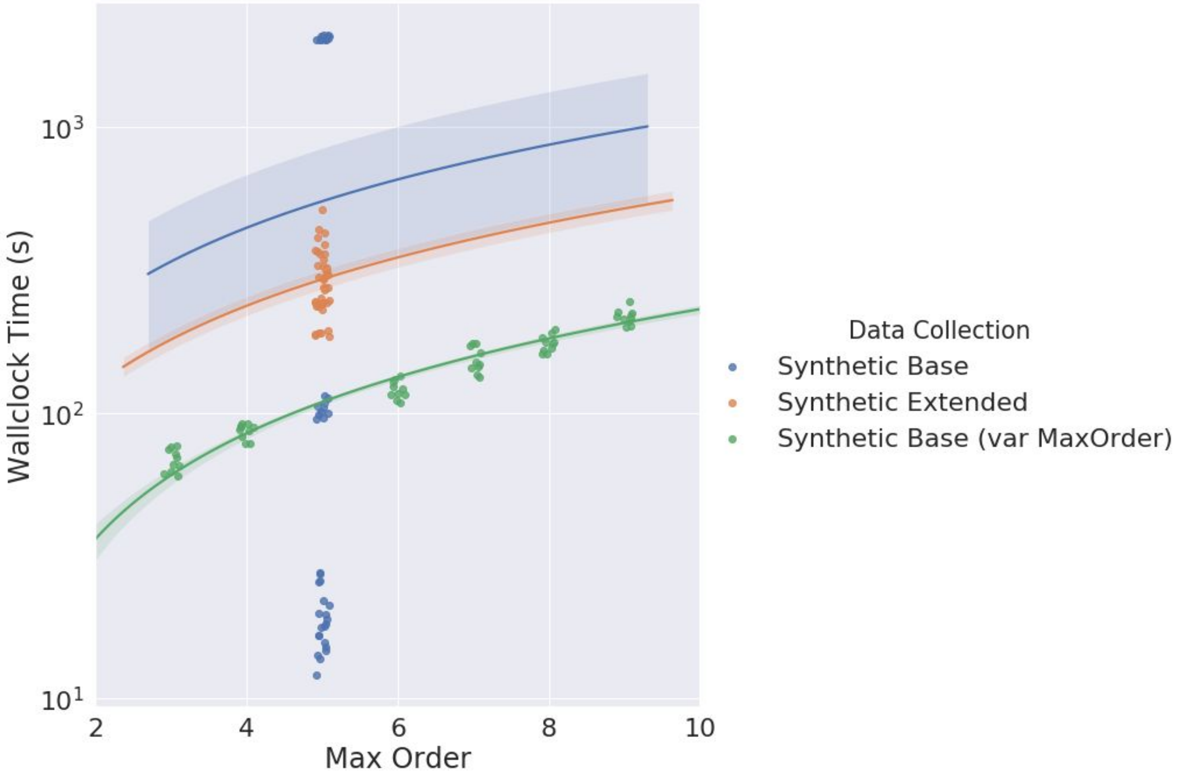
Conclusions & Future Work

1. Background & Motivation
2. Design & Implementation
3. Evaluation
4. **Conclusions & Future Work**

Conclusions

Scalability achieved 👍



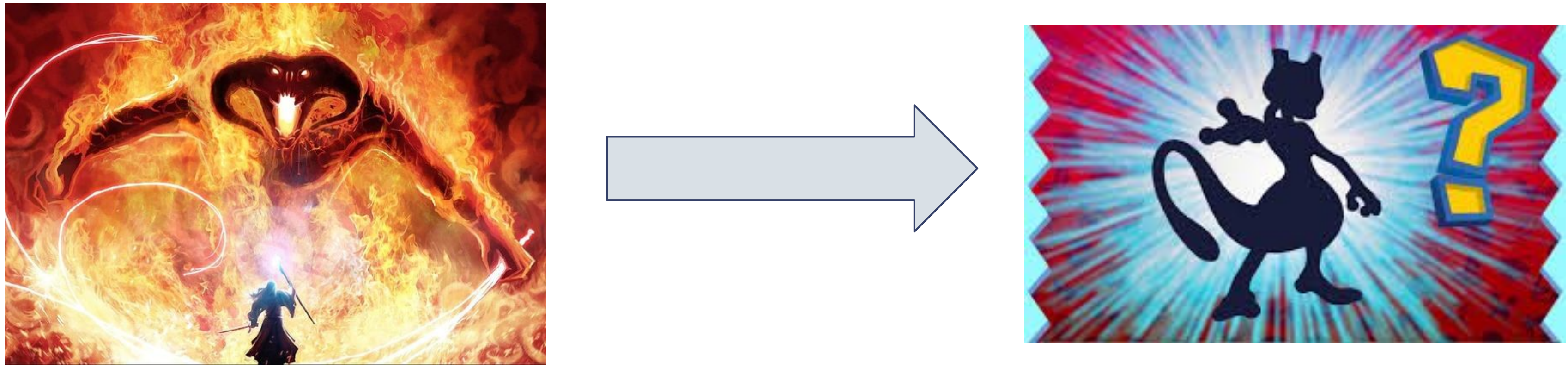


Future Work

- Fix runtime wall (collisions?)
- Fix pruning accuracy (currently over-prunes)
- Vary graph density
- Parallelize

Future Work

- Fix runtime wall (collisions?)
- Fix pruning accuracy (currently over-prunes)
- Vary graph density
- Parallelize



References

[1] Xu, J., Wickramarathne, T., & Chawla, N. (2016). Representing higher-order dependencies in networks. Science Advances, 2(5), E1600028.
[2] http://www.higherordernetwork.com/
[3] https://github.com/xyjprc/hon
[4] Xu, J., Saebi, M., Ribeiro, B., Kaplan, L., & Chawla, N. (2017). Detecting Anomalies in Sequential Data with Higher-order Networks.
[5] Cui Jiao, Guo Jun, Zhang Cangsong, & Chang Xiaojun. (2012). Implementation of random walk algorithm by parallel computing. Fuzzy Systems and Knowledge Discovery (FSKD), 2012 9th International Conference on, 2477-2481.
[6] Fournier-Viger, P., Nkambou, R., & Tseng, V. S. M. (2011, March). RuleGrowth: mining sequential rules common to several sequences by pattern-growth. In Proceedings of the 2011 ACM symposium on applied computing (pp. 956-961). ACM.
[7] Han, Jiawei, Jian Pei, and Yiwen Yin. "Mining frequent patterns without candidate generation." ACM sigmod record. Vol. 29. No. 2. ACM, 2000.
[8] Fournier-Viger, Faghihi, Nkambou, and Nguifo. "CMRules: Mining Sequential Rules Common to Several Sequences." Knowledge-Based Systems 25, no. 1 (2011): 63-76.
[9] Johnson, Reid. “Mining Association Rules” from *Data Mining* course at the University of Notre Dame. https://www3.nd.edu/~rjohns15/cse40647.sp14/www/content/lectures/11%20-%20FP-Growth%20&%20Evaluation.pdf