TKG: Efficient Mining of Top-K Frequent Subgraphs

Philippe Fournier-Viger[1], Chao Cheng[1],
Jerry C.W. Lin[2], Unil Yun[3], R. Uday Kiran[4]

[1]Harbin institute of technology (Shenzhen), China
[2]Western Norway University, Norway
[3]Sejong University, South Korea
[4]University of Tokyo & NIICT, Japan

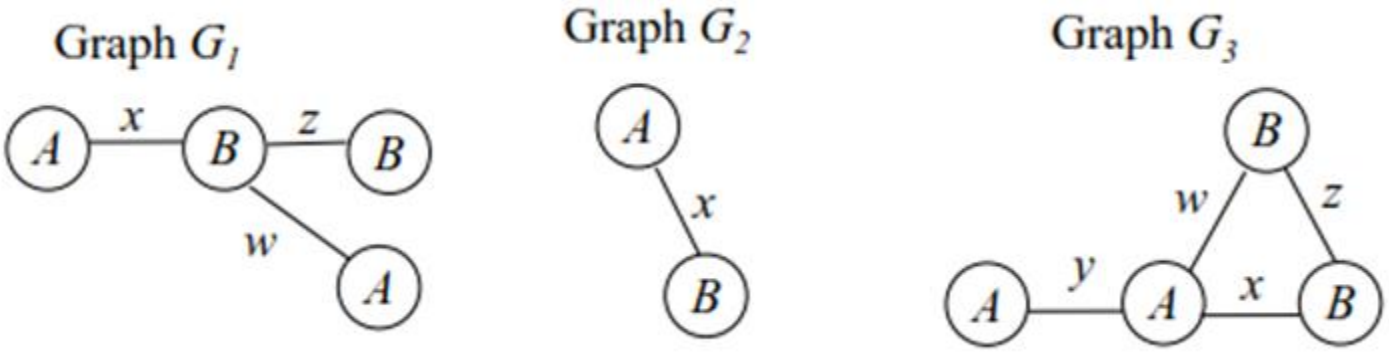
BDA 2019 – 2019/12/

Introduction

- **Graph data**
  - **e.g.** social network, chemical molecules, flights between cities, networks
- **Frequent subgraph mining**
  - discovering interesting patterns in graphs
  - unsupervised algorithms: gSpan, FSM, Gaston…
  - applications such as: finding common submolecules [10], graph indexing [22]

Frequent subgraph mining

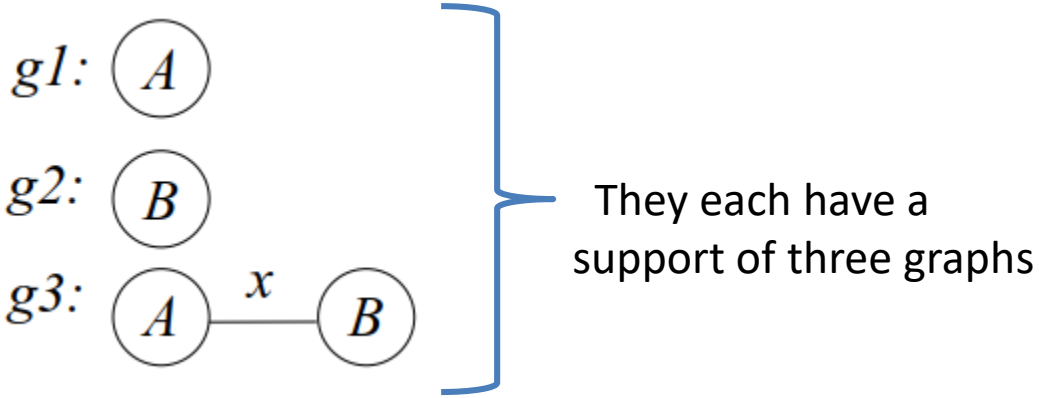
Input: a database of connected labelled graphs



a minimum support threshold

e.g. *minsup = 3*

Output: all subgraphs appearing in at least *minsup* graphs



How to choose the *minsup* threshold?

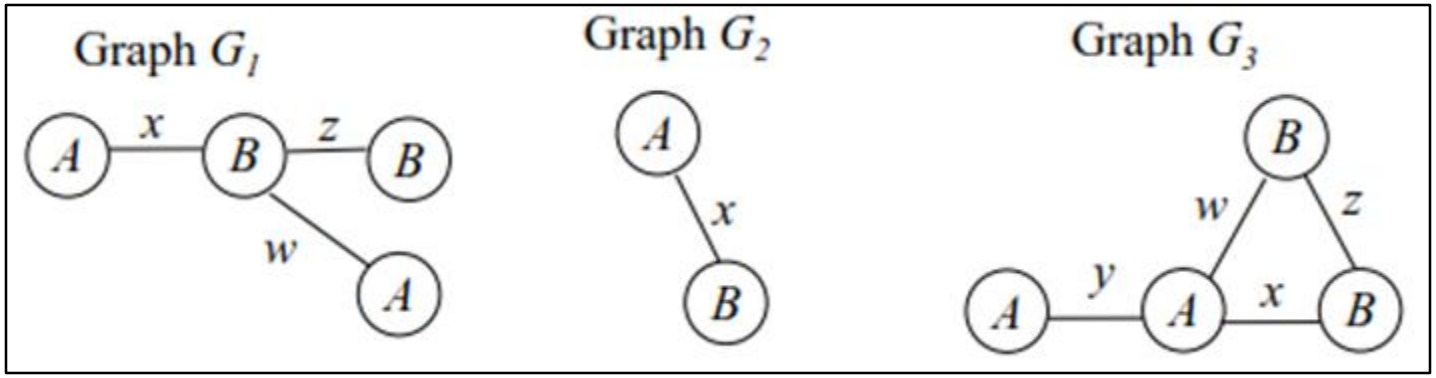
- **How ?**
  - too high, too few results
  - too low, too many results,
    performance often exponentially degrades
- **In real-life:**
  - time/storage limitation,
  - the user cannot analyze too many patterns,
  - fine tuning parameters is time-consuming (optimal values are dataset dependent)

A solution

- Redefining the problem as **top-*k* frequent subgraph mining**
  - **Input:** *k,* the number of patterns to be generated.
- **Output:**
  - the *k* most frequent subgraphs

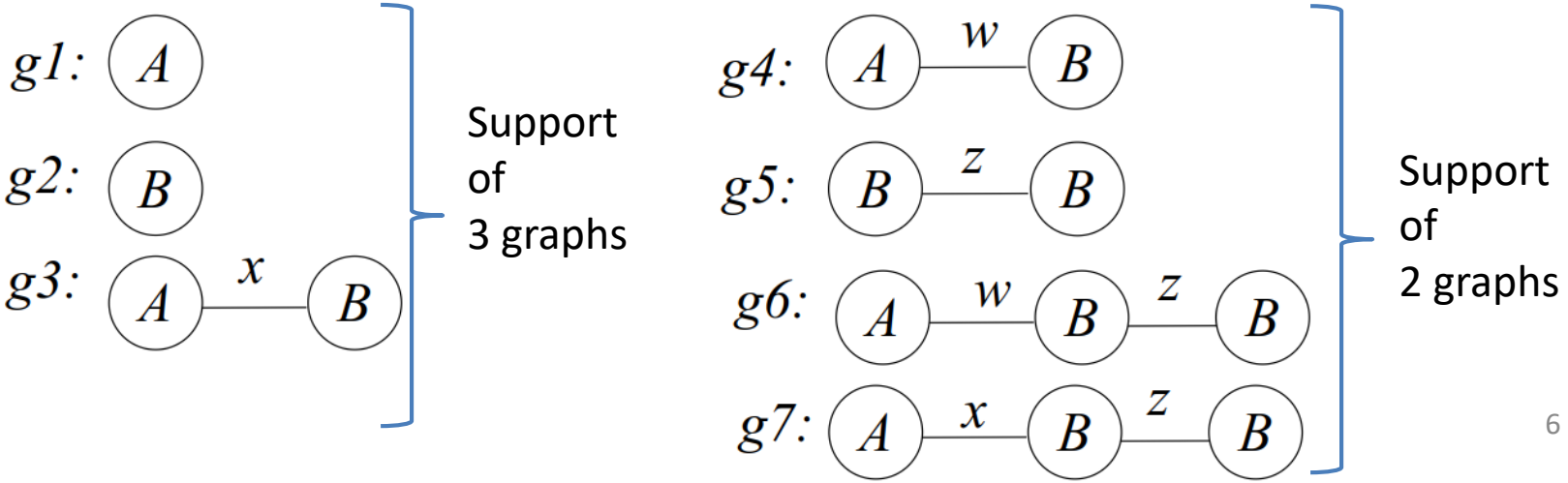
Top-*k* frequent subgraph mining

Input: a database of connected labelled graphs



the number of patterns *k*. **e.g.** *k* = 7

Output: the top *k* most frequent subgraphs



Challenges

- An algorithm for **top-k subgraph mining** cannot use a fixed *minsup* threshold to reduce the search space.
- Therefore, the problem is **more difficult**.
- Large search space

Current algorithms

- The **algorithm** of **Li et al [13]** **is inefficient**. It generates all patterns to then select the top-k subgraphs.
  (it cannot terminate on the Chemical340 dataset)
- The **algorithms** of **Duong et al. [3,4]** **are approximate** (they can miss some frequent subgraphs).

- **Our proposal:** an efficient and exact algorithm named **TKG** for top-*k* subgraph mining

Graph

*A **graph** $G = (V, E, L^V, L^E, \varphi^V, \varphi^E)$*

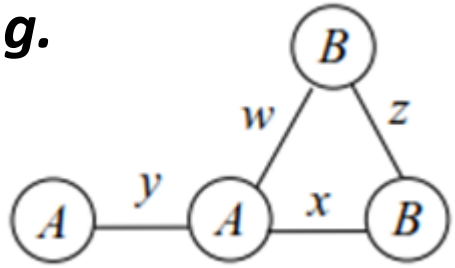
- *V is the set of vertices*
- *E is the set of edges*
- *L^V is the set of vertex labels*
- *L^E is the set of edge labels.*

A **graph database** *$GD = \{G^1, G^2 \ldots G^n\}$ is a set of labeled graphs.*

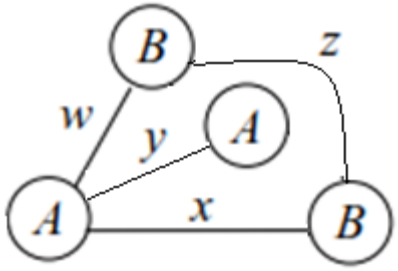
Graph isomorphism

- Let there be **two graphs**:
  - $G^x = (V^x, E^x, L^{xV}, L^{xE}, \varphi^{xV}, \varphi^{xE})$
  - $G^y = (V^y, E^y, L^{yV}, L^{yE}, \varphi^{yV}, \varphi^{yE})$
- G^x ***is isomorphic to*** G^y *if there is a bijective mapping* $f : V^x \rightarrow V^y$ *such that:*
  - *for any vertex* $v \in V^x$, *it follows that* $L^{xV}(v) = L^{yV}(f(v))$
  - *for any pair* $(u, v) \in E^x$, *it follows that* $(f(u), f(v)) \in E^y$ *and* $L^{xE}(u, v) = L^{yE}(f(u), f(v))$.

e.g.



is isomorphic to

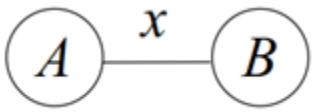


(**i.e.** the two graphs have the same structure with the same labels)

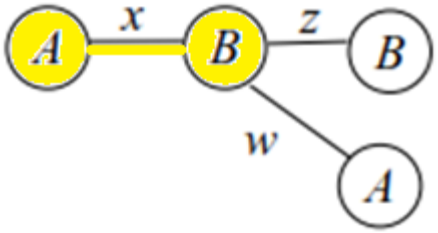
Subgraph isomorphism

- *Let there be* ***two graphs***
  - $G^x = (V^x, E^x, L^{xV}, L^{xE}, \varphi^{xV}, \varphi^{xE})$
  - $G^z = (V^z, E^z, L^{zV}, L^{zE}, \varphi^{zV}, \varphi^{zE})$.
- G^x ***appears in the graph*** G^z*, or equivalently that* G^x *is a subgraph isomorphism of* G^z*, if* G^x *is isomorphic to a subgraph* $G^y \subseteq G^z$.

e.g.

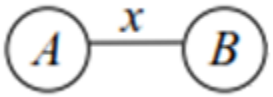


appears in this graph:

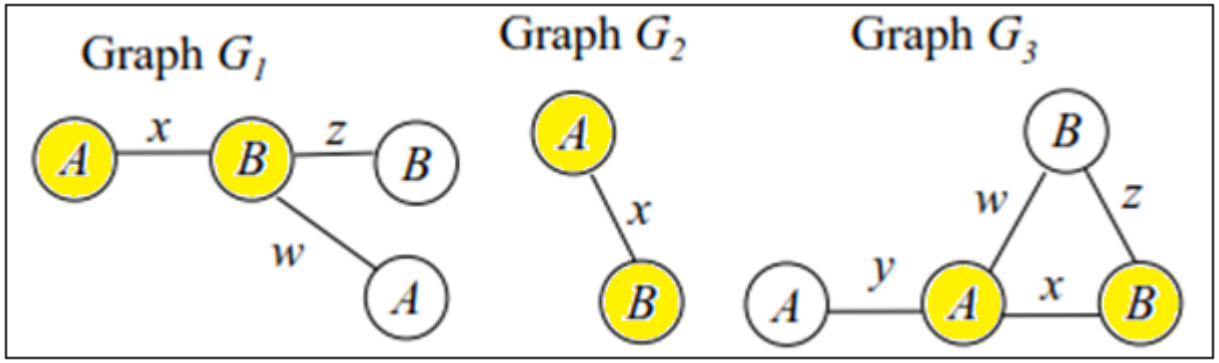


Support of a subgraph

- *Let there be a **graph database** GD.*
- *The **support** (occurrence frequency) **of a subgraph** G^x in GD is the number of graphs where G^x appears, that is:*
  $sup(G^x) = |\{g | g \in GD \wedge G^x \sqsubseteq g\}|$

e.g. 

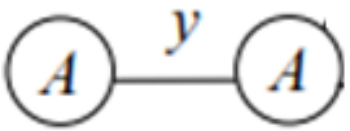
has a support of 3 in this graph database:



Search space pruning property

- There can be **millions of subgraphs** in a database!
- To avoid looking at all of them, **frequent subgraph mining algorithms** apply the "**Apriori" property**:
  - If a subgraph G^x is infrequent, then all its supergraphs are infrequent.

- **e.g.**



Is infrequent (support = 1).
Thus all its supergraphs are infrequent

Problem definition

- *Let there be a user-defined parameter $k \geq 1$ and a graph database GD.*
- ***Problem (Top-k frequent subgraph mining):*** *Finding a set T of k subgraphs such that their support is greater or equal to that of any other subgraphs not in T.*

Note*:* in some cases, more than *k* patterns could be included in the set *T*.

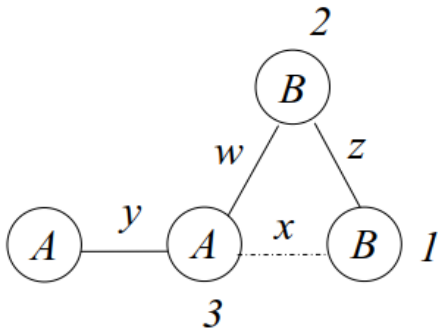
The TKG algorithm

Basic idea

1. Start with *minsup* = 1.
2. Perform a dept-first search starting from single edges to explore the search space.
3. Keep a queue Q_k that contains the current top-*k* subgraphs found until now.
4. When *k* patterns are found, raise *minsup* to the support of the least frequent subgraph in Q_k.
5. After that, for each subgraph added to Q_k, raise the *minsup* threshold.
6. When the algorithm terminates, the top-*k* subgraphs have been found.

More details…

- Each **subgraph** is represented by a **DFS code**.



<1, 2, B, B, z>
<2, 3, B, A, w>
<3, 1, A, B, x>
<3, 4, A, A, y>

- When a subgraph is extended with an edge, we calculate its support and check if its **DFS code is canonical. If not,** it means that this subgraph is a duplicate and we can ignore it.
- This technique was proposed in **gSpan** to avoid processing duplicate subgraphs.

The TKG algorithm

- The resulting algorithm finds the correct result.
- However, it has **poor execution time** because the **search space is too large**.
- **Solution**: three strategies →

Strategy 1 – dynamic search

- **Observation:**
  - if we can find subgraphs having a high support first, we can raise *minsup* more quickly to prune reduce the space.
- **Strategy**
  - We store **subgraphs** that can be extended to generate more subgraphs in **a priority queue** Q_c , ordered by decreasing support.
  - Instead of using a depth-first search, we always extend the subgraph having the highest support in Q_c first.

Strategy 2 – skip strategy

- To find **extensions of a subgraph** *g*, the algorithm scans all input graphs of the database that contain *g.*
- The algorithm counts the support of each extension.
- After processing a graph G^j, let *hsup* be the highest support among the found extensions and *rn* be the number of remaining graphs.
- If $hsup + rn < minsup$, it indicates that the subgraph *g* **has no frequent extension**.
- This strategy decreases runtime

Strategy 3

- Initially, the algorithm scans the database to calculate the support of all single edge graphs.
- Then, use this information to update Q^k, *minsup* and Q^c, before performing the dynamic search.
- Doing so decreases the processing time for single edge graphs.

Algorithm 1: The TKG algorithm

input : GD: a graph database, k: a user-specified number of patterns
output: the top-k frequent subgraphs

```
1  Initialize a priority queue Q_K for storing the current top-k frequent
     subgraphs, where subgraphs with smaller support have higher priority.
2  Initialize a priority queue Q_c for storing candidate subgraphs for next
     extension, where subgraphs with higher support have higher priority.
     Initially, contains an empty graph.
3  minsup = 1
4  while Q_c is not empty do
5      g ← pop highest priority subgraph from Q_c
6      ε ← rightMostPathExtensions(g, GD) // Finds edges that can extend g
         and compute their support values.
7      foreach (t, sup(t)) ∈ ε do
8          g' ← g ∪ {t} // Add the edge t to the DFS code of graph g
9          sup(g') ← sup(t)
10         if sup(g') ≥ minsup and isCanonical(g') then
11             // Save pattern g' in list of current top-k patterns
12             Insert g' into Q_K
13             if Q_K.size() ≥ k then
14                 // Raise the internal threshold
15                 if Q_K.size() > k then pop the highest priority (least support)
                     subgraph from Q_K;
16                 minsup = sup(Q_K.peek())
17             end
18             // Save g' as a candidate for future extension instead of doing a
                 depth-first search
19             Insert g' into Q_c
20         end
21     end
22 end
23 Return Q_K
```

Experiments

Datasets

Dataset	$\|GD\|$	Avg. nodes	Avg. edges	$\|L_V\|$	$\|L_E\|$
protein	1113	39.05	72.82	3	1
nci1	4110	29.87	32.3	37	3
enzymes	600	32.63	62.13	3	1
Chemical340	340	27.02	27.40	66	4

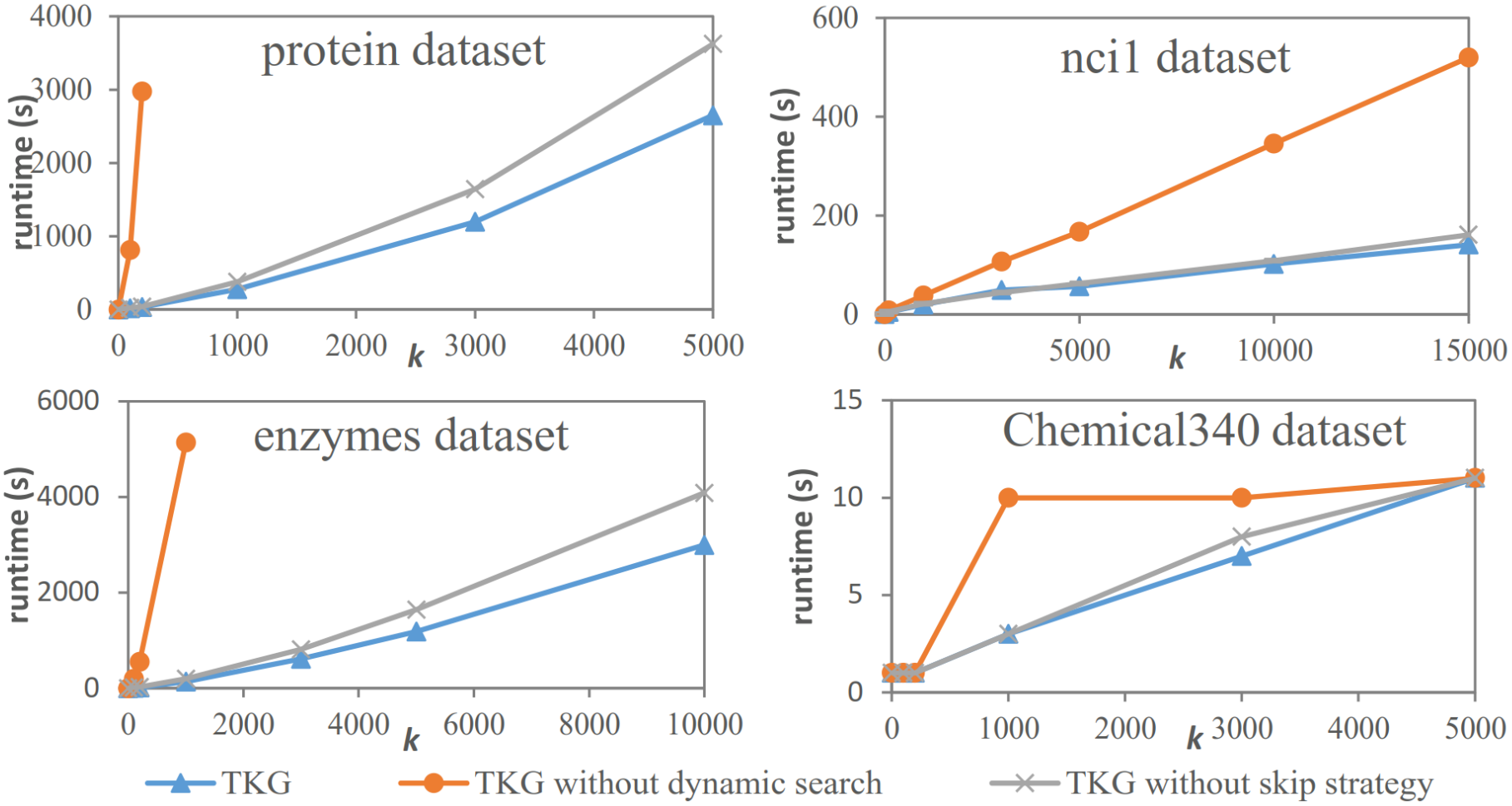
$|GD|$ = graph count $|L_v|$ = vertex label count

$|L_E|$ = edge label count

Algorithms

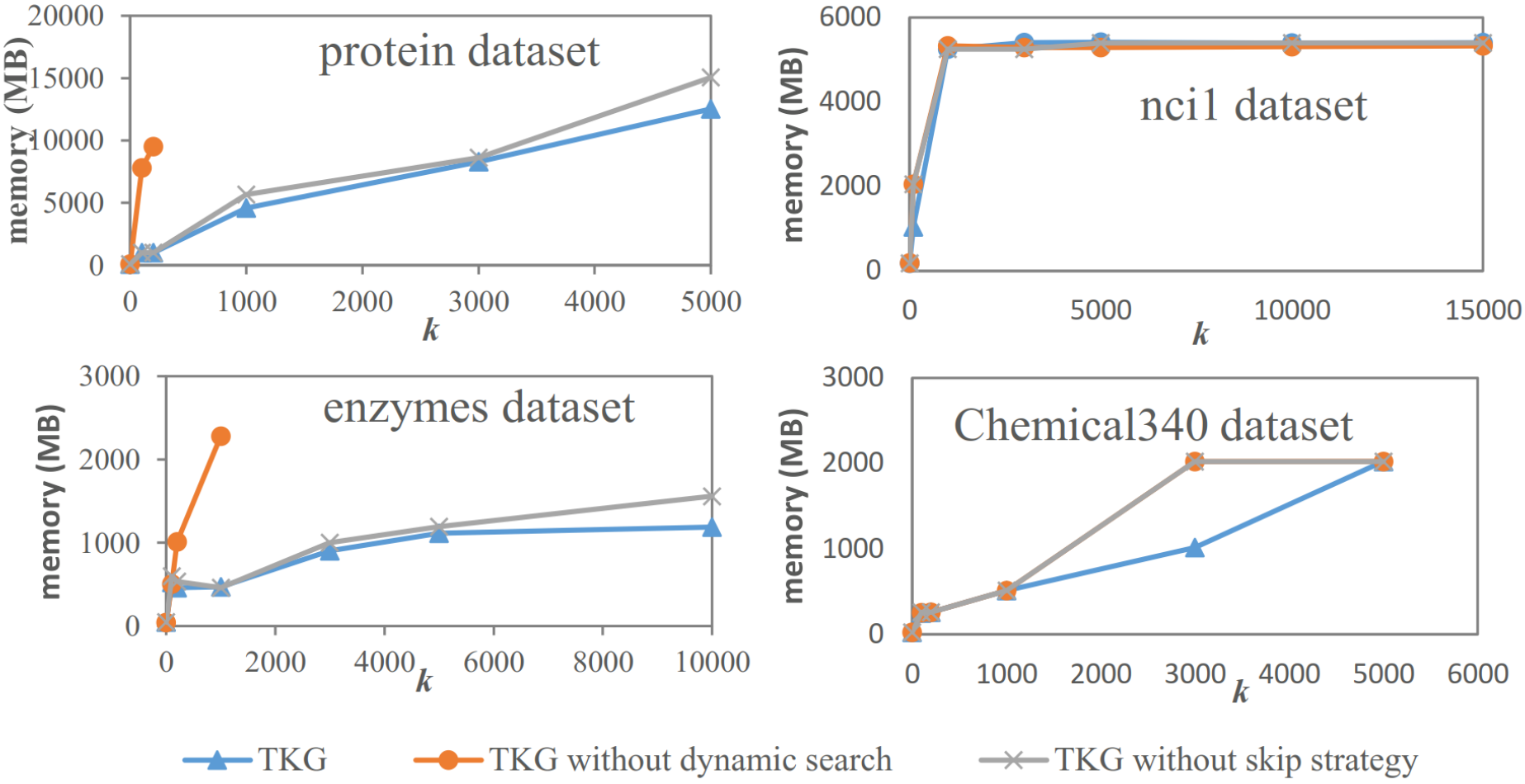
- TKG with or without optimizations
- gSpan with an optimum *minutil* value

Influence of *k* on runtime



- For **larger *k* values**, runtime increases because more patterns to consider
- The **dynamic search** reduces TKG's runtime by up to **100 times**
- The **Skip strategy** reduces TKG's runtimes

Influence of *k* on memory



- For **larger *k* values**, memory increases because more patterns to consider
- The **dynamic search** reduces TKG's memory by up to **8 times** on some datasets
- The **Skip strategy** slightly reduces TKG's memory on some datasets

Comparison with gSpan

Comparison of TKG and gSpan with optimal *minsup* threshold on the protein dataset

k	*minsup*	TKG runtime (s)	gSpan runtime (s)	TKG memory (MB)	gSpan memory (MB)
1	0.9227	1	1	85	85
100	0.6720	14	13	1020	507
200	0.6289	31	31	1019	976
1000	0.5247	321	275	4583	3503
3000	0.4618	1205	1198	4583	3503
5000	0.4367	2673	2650	8310	6182

Comparison of TKG and gSpan with optimal *minsup* threshold on the en-zymes dataset

k	*minsup*	TKG runtime (s)	gSpan runtime (s)	TKG memory (MB)	gSpan memory (MB)
1	0.9767	1	1	46	46
100	0.8067	12	8	527	276
200	0.7817	19	15	462	252
1000	0.6650	151	134	469	296
3000	0.600	625	612	1016	902
5000	0.5700	1280	1249	1113	1060

Runtimes are very close, despite that top-*k* pattern mining is more difficult.

Conclusion

- The **TKG** algorithm for **top-k frequent subgraph mining**
  - An exact algorithm
  - Three optimizations
  - Performance close to that of running gSpan with an optimal *minsup* threshold
- **Future work:** optimizations and extensions for other graph mining problems.
- Source code and datasets available as part of the **SPMF data mining library** (GPL 3).

An Open-Source Data Mining Library

Introduction

SPMF is an **open-source data mining mining library** written in **Java**, specialized in **pattern mining**.

It is distributed under the **GPL v3 license**.

It offers implementations of **120 data mining algorithms** for:

- **association rule mining,**
- **itemset mining,**
- **sequential pattern mining,**
- **sequential rule mining,**
- **sequence prediction,**
- **periodic pattern mining,**
- **high-utility pattern mining,**
- **clustering** and **classification**

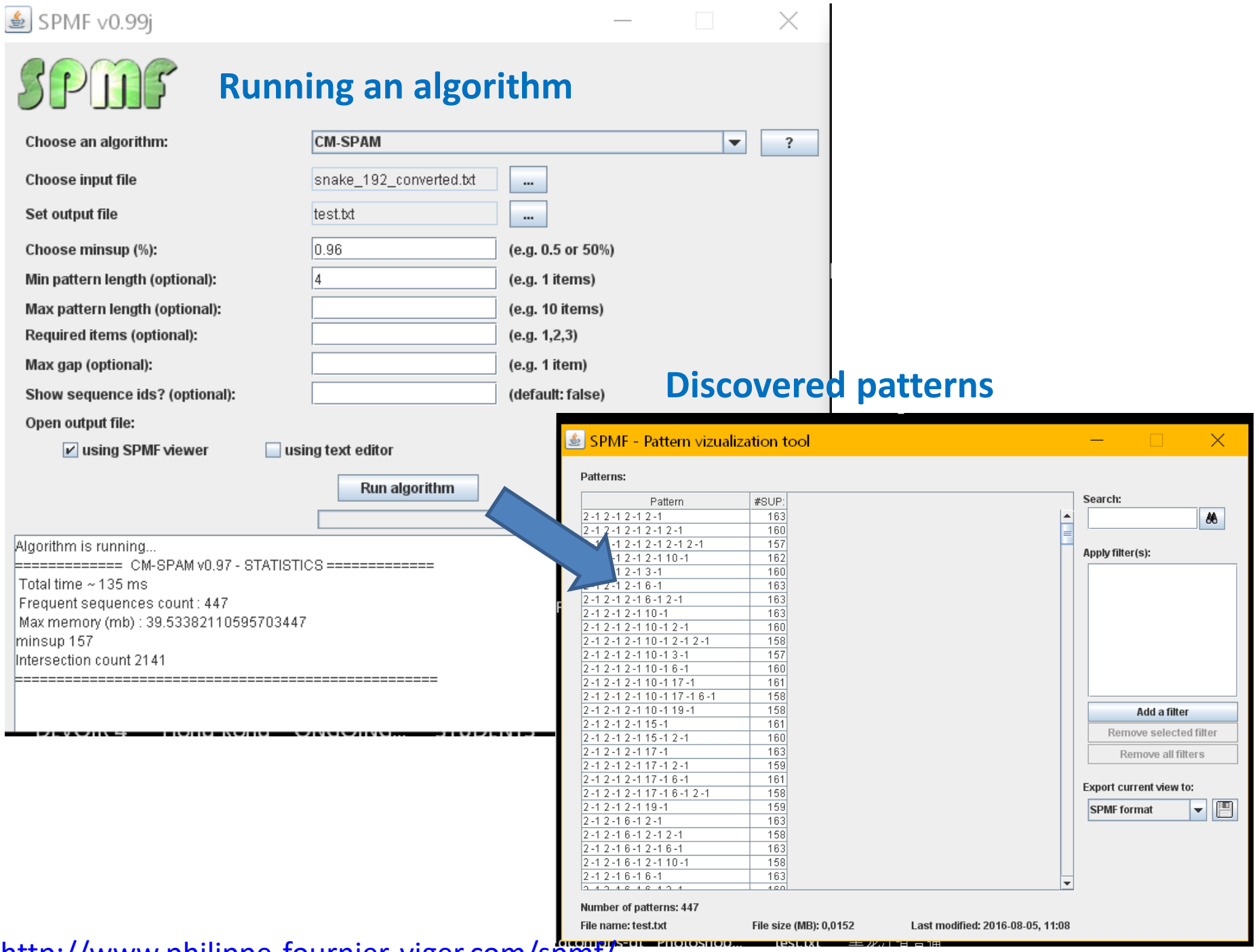
The **source code** of each algorithm can be easily integrated in other Java software.

Moreover, SPMF can be used as a **standalone program** with a simple user interface or from the **command line**.

SPMF is fast and lightweight (no dependencies to other libraries).

The current version is **v0.99j** and was released the **16th June 2016.**

http://www.philippe-fournier-viger.com/spmf/



http://www.philippe-fournier-viger.com/spmf/

Thank you. Questions?

SPMF Open source Java data mining software, 177 algorithms http://www.phillippe-fournier-viger.com/spmf/

References

1. Borgwardt, K.M., Ong, C.S., Schonauer, S., Vishwanathan, S.V.N., Smola, A.J., Kriegel, H.P.: Protein function prediction via graph kernels. Bioinformatics 21 Suppl 1, 47–56 (2005)
2. Cheng, Z., Flouvat, F., Selmaoui-Folcher, N.: Mining recurrent patterns in a dynamic attributed graph. In: Proc. of 21st Pacific-Asia Conf. on Knowledge Discovery and Data Mining. pp. 631–643. Springer (2017)
3. Duong, V.T.T., Khan, K.U., Jeong, B.S., Lee, Y.K.: Top-k frequent induced subgraph mining using sampling. In: Proc. 6th Intern. Conf. on Emerging Databases: Technologies, Applications, and Theory (2016)
4. Duong, V.T.T., Khan, K.U., Lee, Y.K.: Top-k frequent induced subgraph mining on a sliding window using sampling. In: Proc. 11th Intern. Conf. on Ubiquitous Information Management and Communication (2017)
5. Fournier-Viger, P., Lin, J.C.W., Gomariz, A., Gueniche, T., Soltani, A., Deng, Z., Lam, H.T.: The spmf open-source data mining library version 2. In: Proc. 20th European conference on machine learning and knowledge discovery in databases. pp. 36–40. Springer (2016)
6. Fournier-Viger, P., Lin, J.C.W., Kiran, U.R., Koh, Y.S.: A survey of sequential pattern mining. Data Science and Pattern Recognition 1(1), 54–77 (2017)
7. Fournier-Viger, P., Lin, J.C.W., Truong-Chi, T., Nkambou, R.: A survey of high utility itemset mining. In: High-Utility Pattern Mining, pp. 1–45. Springer (2019)
8. Fournier-Viger, P., Lin, J.C.W., Vo, B., Chi, T.T., Zhang, J., Le, B.: A survey of itemset mining. WIREs Data Mining and Knowledge Discovery (2017)
9. Inokuchi, A., Washio, T., Motoda, H.: An apriori-based algorithm for mining frequent substructures from graph data. In: Proc. 4th European Conference on Principles of Data Mining and Knowledge Discovery (2000)
10. Jiang, C., Coenen, F., Zito, M.: A survey of frequent subgraph mining algorithms. Knowledge Engineering Review 28, 75–105 (2013)
11. Kuramochi, M., Karypis, G.: Frequent subgraph discovery. In: Proc. 1st IEEE Intern. Conf. on Data Mining (2001)
12. Lee, G., Yun, U., Kim, D.: A weight-based approach: frequent graph pattern mining with length-decreasing support constraints using weighted smallest valid extension. Advanced Science Letters 22(9), 2480–2484 (2016)
13. Li, Y., Lin, Q., Li, R., Duan, D.: Tgp: Mining top-k frequent closed graph pattern without minimum support. In: Proc. 6th Intern. Conf. on Advanced Data Mining and Applications (2010)
14. Mrzic, A., Meysman, P., Bittremieux, W., Moris, P., Cule, B., Goethals, B., Laukens, K.: Grasping frequent subgraph mining for bioinformatics applications. In: BioData Mining (2018)
15. Nguyen, D., Luo, W., Nguyen, T.D., Venkatesh, S., Phung, D.Q.: Learning graph representation via frequent subgraphs. In: Proc. 2018 SIAM International Conference on Data Mining. pp. 306–314 (2018)

References (cont'd)

16. Nijssen, S., Kok, J.N.: The gaston tool for frequent subgraph mining. Electronic Notes in Theoretical Computer Science 127, 77–87 (2005)
17. Saha, T.K., Hasan, M.A.: Fs3: A sampling based method for top-k frequent subgraph mining. Proc. 2014 IEEE Intern. Conf. on Big Data pp. 72–79 (2014)
18. Sankar, A., Ranu, S., Raman, K.: Predicting novel metabolic pathways through subgraph mining. Bioinformatics 33 24, 3955–3963 (2017)
19. Wale, N., Watson, I.A., Karypis, G.: Comparison of descriptor spaces for chemical compound retrieval and classification. Proc. 6th International Conference on Data Mining pp. 678–689 (2006)
20. Yan, X., Han, J.: gSpan: Graph-based substructure pattern mining. In: Proc. 2nd IEEE Intern. Conf. on Data Mining (2002)
21. Yan, X., Han, J.: Closegraph: mining closed frequent graph patterns. In: Proc. of the 9th ACM SIGKDD Intern. Conf. on Knowledge Discovery and Data Mining (2003)
22. Yan, X., Yu, P.S., Han, J.: Graph indexing: A frequent structure-based approach. In: Proc. of the 2004 SIGMOD Conference (2004)
23. Yun, U., Lee, G., Kim, C.H.: The smallest valid extension-based efficient, rare graph pattern mining, considering length-decreasing support constraints and symmetry characteristics of graphs. Symmetry 8(5), 32 (2016)
24. Zhu, F., Yan, X., Han, J., Yu, P.S.: gprune: A constraint pushing framework for graph pattern mining. In: Proc. of the 11st Pacific-Asia Conf. on Knowledge Discovery and Data Mining (2007)