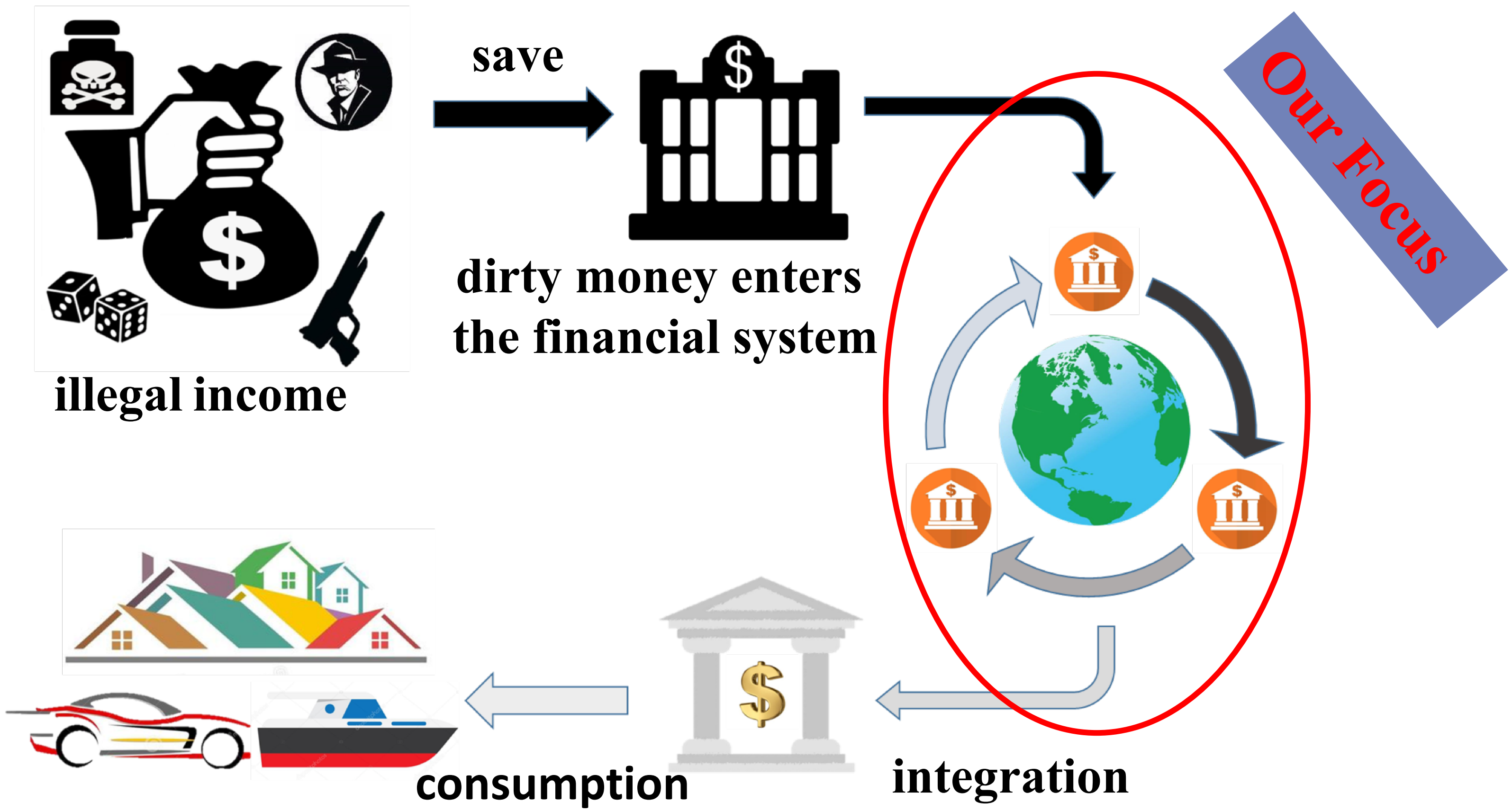
FlowScope: Spotting Money Laundering Based on Graphs

Xiangfeng Li[1], **Shenghua Liu[2]**, Zifeng Li[3], Xiaotian Han[4], **Chuan Shi[1]**,**Bryan Hooi[5]**, He Huang[6], **Xueqi Cheng**[2]

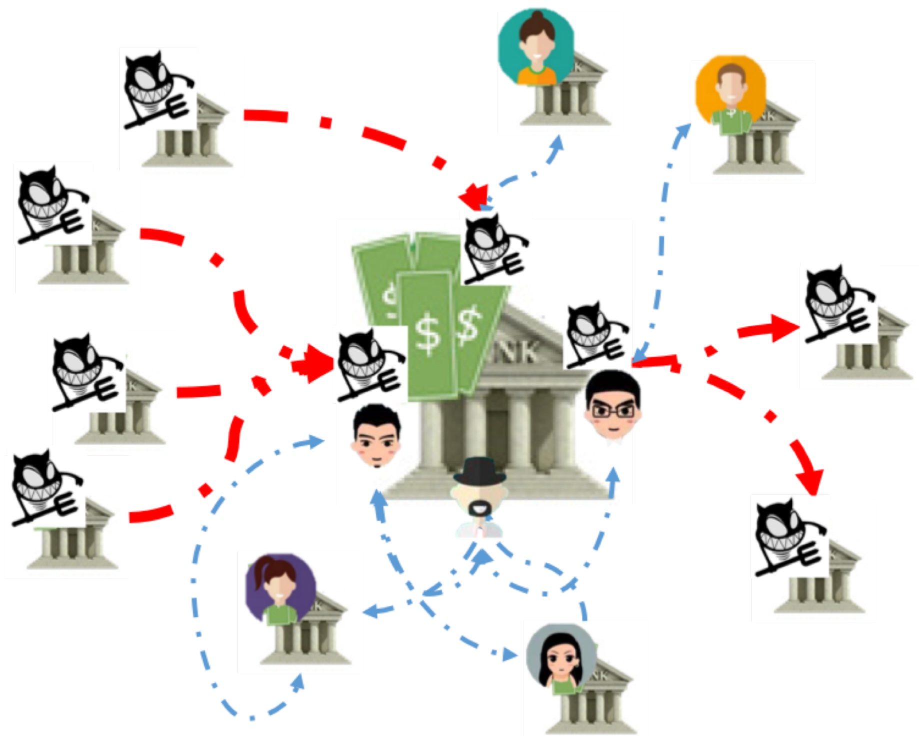
[1]Beijing University of Post and Telecommunication [2]Institute of Computing Technology, Chinese Academy of Sciences
[3]University of Surrey [4]Texas A&M University
[5]School of Computer Science, National University of Singapore [6]China Citic Bank

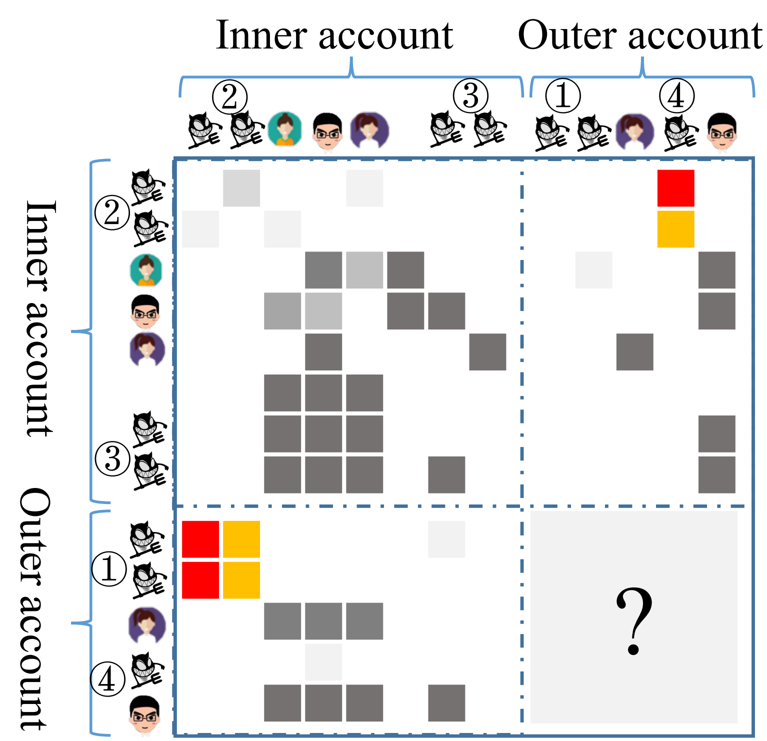
Motivation

- Typical method of money laundering (ML)



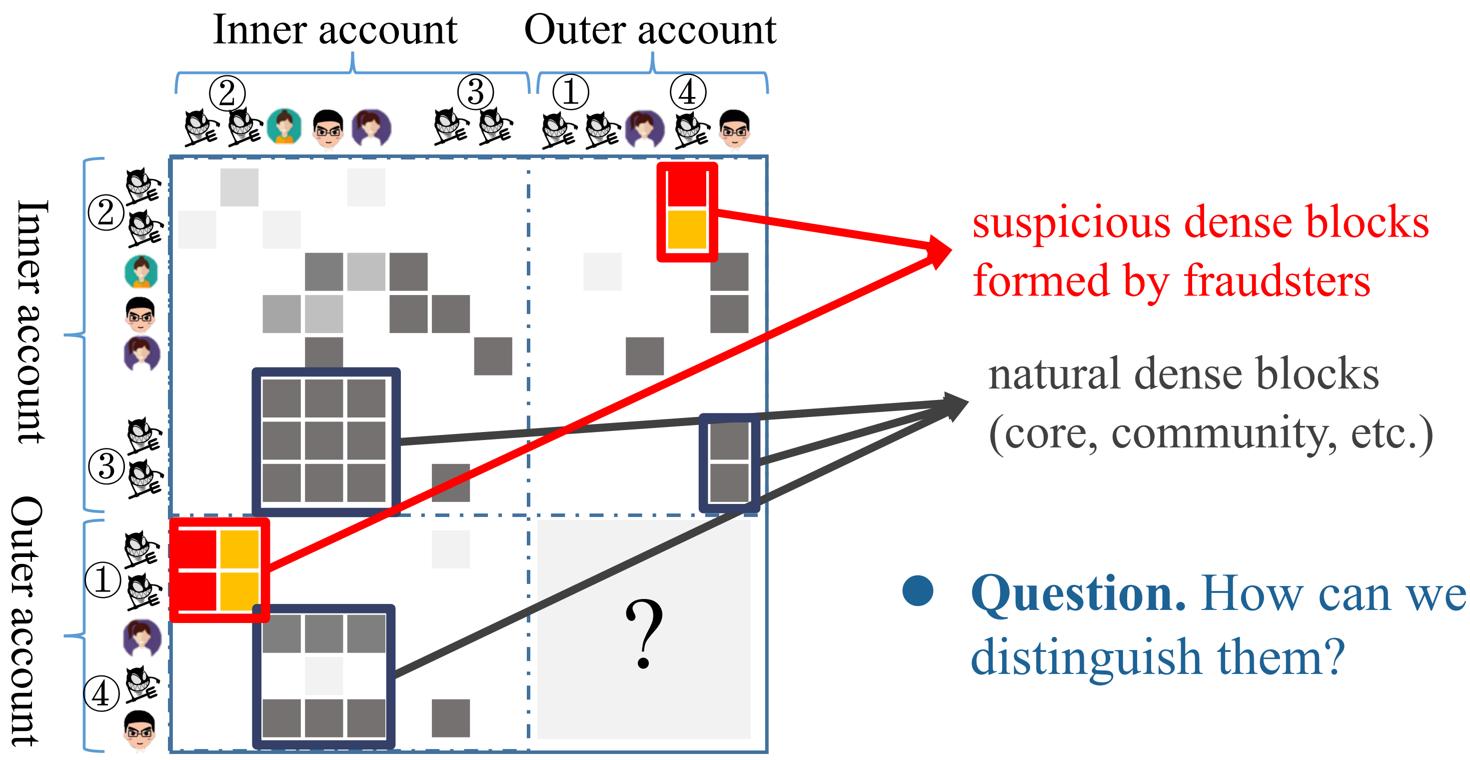
ML Forms a Multipartite Dense Subgraph





Adjacency Matrix

Problem: Natural Dense Subgraph

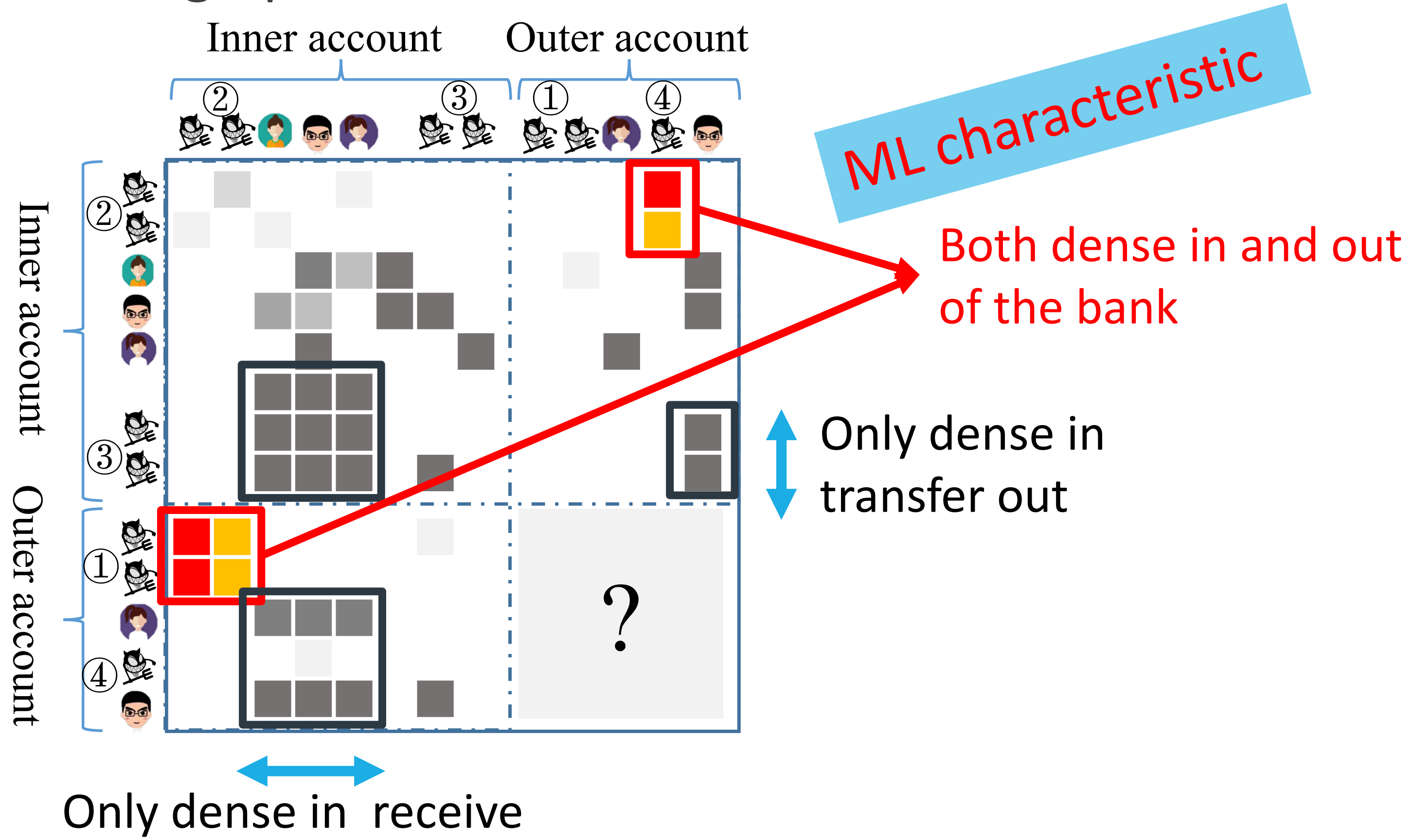


Adjacency Matrix

- **Question.** How can we distinguish them?

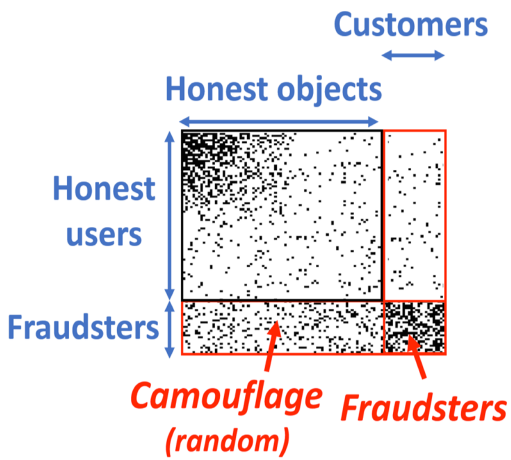
Solution: Multipartite Dense Subgraph

- Natural dense transfer not always form a multipartite dense subgraph

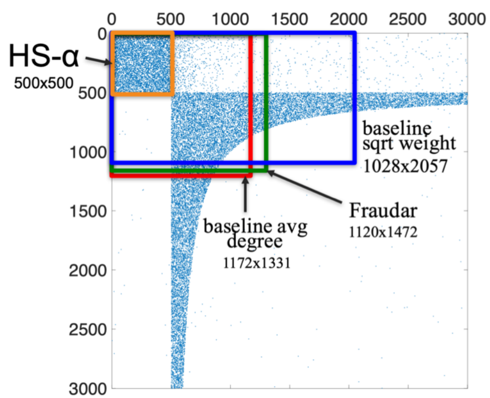


Solution: Multipartite Dense Subgraph (cont.)

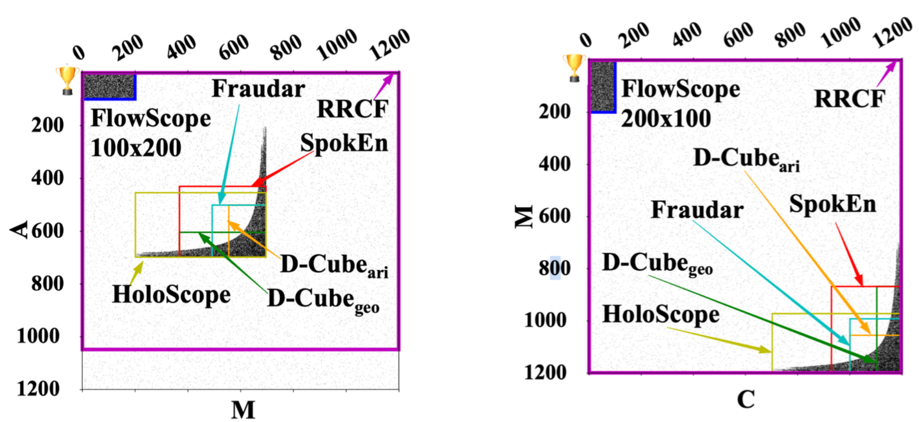
- Our FlowScope catches exactly multipartite dense subgraph



Fraudar



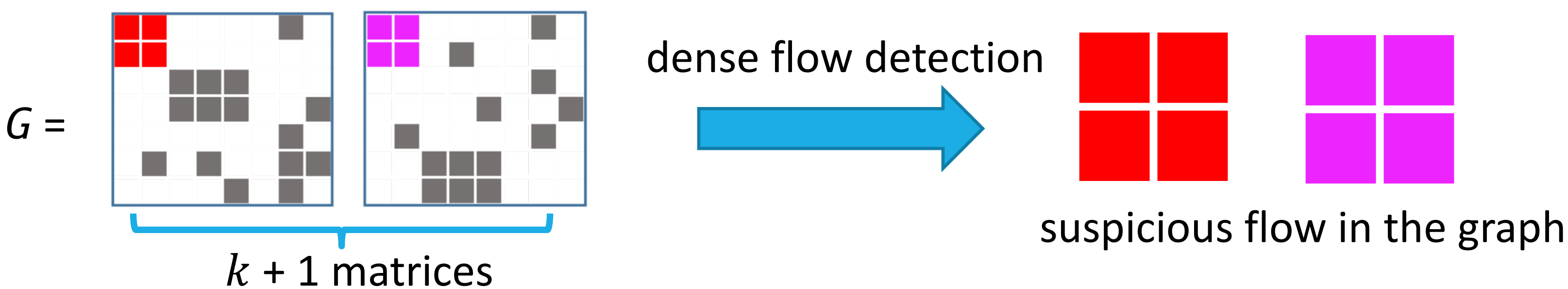
HoloScope-α



Our FlowScope

Problem formulation

- **Given**
  - $G = (V, E)$: a graph of money transfers
  - V: accounts as nodes
  - E: money amount as edges weight
  - k: number of middle layers
- **Find**
  - a dense flow of money transfers (i.e. a subgraph of G),
- **Such that**
  - 1) the flow involves high-volume money transfers into the bank, and out of the bank to the destinations;
  - 2) it maximizes density as defined in our ML metric.



Requirements

- Our goal is to design an algorithm which is

☐ ***Fast***: runs in near-linear time
☐ ***Accurate***: provides an accuracy guarantee
☐ ***Effective***: produces meaningful results in practice

FlowScope, our proposed method, satisfies all the requirements

Model

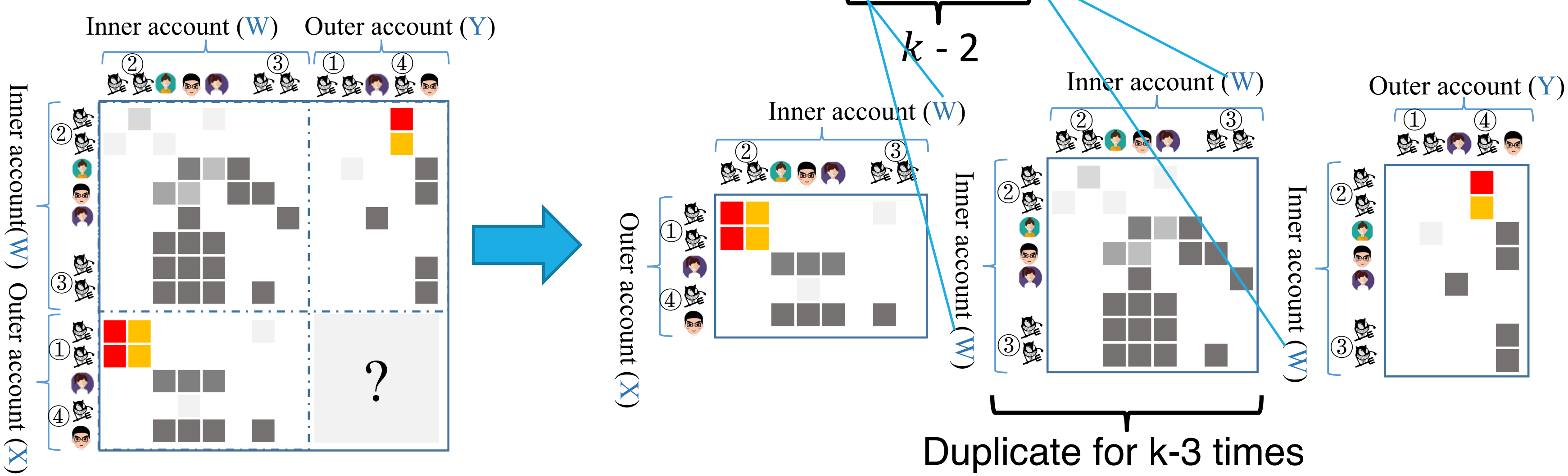
- Graph

$$G = (V, E),\quad V = X \cup W \cup Y$$

  - W is the inner accounts of the bank, and X and Y are sets of outer accounts

- Generate multipartite graph

$$G^k = (V^k, E^k),\qquad V^k = X \cup \underbrace{W \cup \ldots \cup W}_{k-2} \cup Y$$



Model (cont.)

- Out/in degree of each middle-layer node

$$d_i^+(S) = \sum_{v_j \in M_{l+1} \wedge (i,j) \in E} e_{ij}$$
$$d_i^-(S) = \sum_{v_k \in M_{l-1} \wedge (k,i) \in E} e_{ki}$$

- Definition of min and max flow

$$f_i(S) = \min\{\, d_i^+(S), d_i^-(S) \,\}, \forall\, v_i \in M_l$$
$$q_i(S) = \max\{\, d_i^+(S), d_i^-(S) \,\}, \forall\, v_i \in M_l$$

- Suspicious metric

balance parameter

~ flow

$$g^k(S) = \frac{1}{|S|} \sum_{l=1}^{k-2} \sum_{v_i \in M_l} f_i(S) - \lambda(q_i(S) - f_i(S))$$
$$= \frac{1}{|S|} \sum_{l=1}^{k-2} \sum_{v_i \in M_l} (\lambda + 1) f_i(S) - \lambda q_i(S), k \geq 3$$

retention/deficit

Algorithm

- **Input:** Graph $G = (V, E)$
- **Output:** Node set of dense multipartite flow: S
- **Key idea:** priority tree and greedy deletion

Algorithm 1: FlowScope

Input: Graph $G = (\mathcal{V}, \mathcal{E})$
Output: Node set of dense flow: $\hat{\mathcal{S}}$

Step 1. initialize

```
A ← X, M_1 ← W, ··· , M_{k−2} ← W, C ← Y
    // generate k-partite node
    subsets from G
S ← A ∪ M_1 ∪ ··· ∪ M_{k−2} ∪ C
w_i ← calculate node weight as Eq. (5)
T ← build priority tree for S with w_i(S)
```

Step 2. greedy deletion

```
while A, M_1, ··· , M_{k−2} and C is not empty do
    v ← find the minimum weighted node in T
    S ← S \ {v}
    update priorities in T for all neighbors of v
    g(S) ← calculate as Eq. (3)
end
```

Step 3. get the result

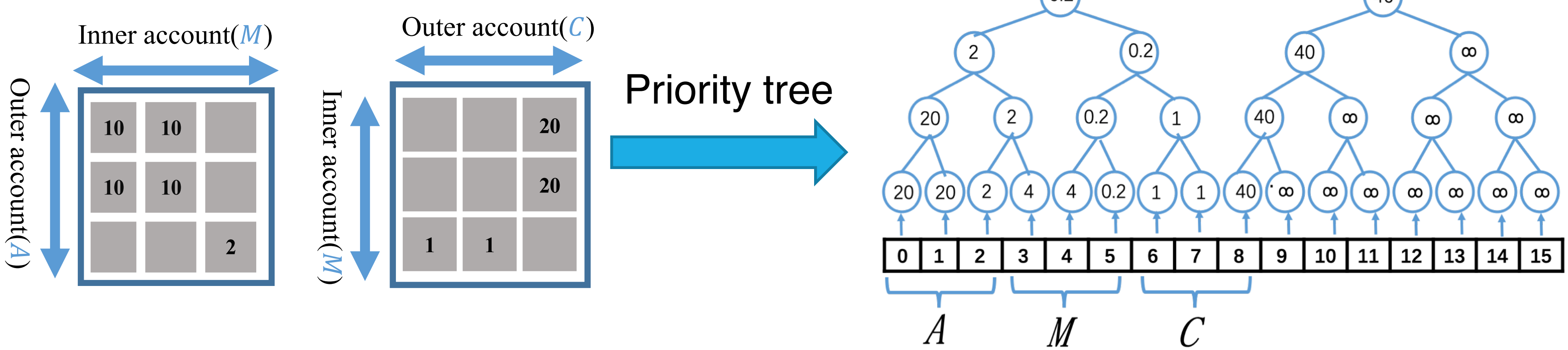
```
return Ŝ that maximizes g(S) seen during the loop.
```

Algorithm (cont.)

- Step 1. initialize
  - 1. generate the k-partite graph, $A \leftarrow X, M_1 \leftarrow W, \dots, M_{k-2} \leftarrow W, C \leftarrow Y$
  - 2. initialize subset $S \leftarrow A \cup M_1 \cup \dots \cup \mathrm{M}_{k-2} \cup C$
  - 3. calculate the priority of node

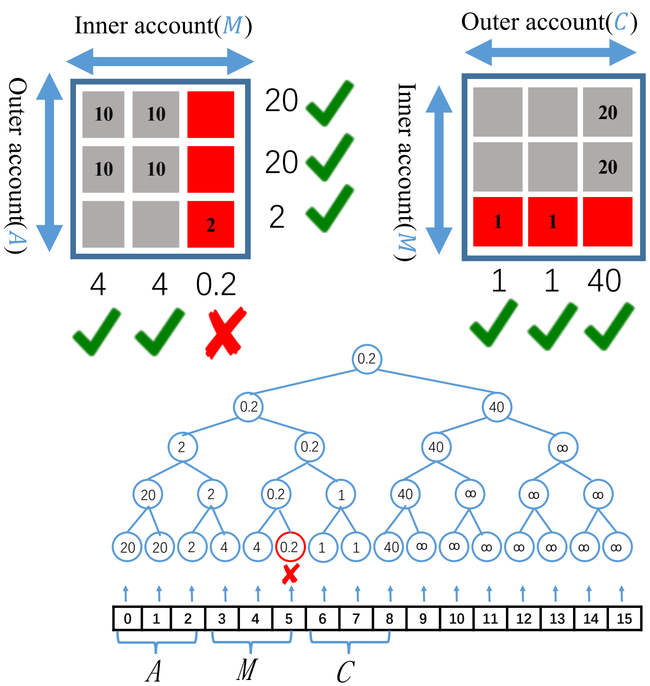
$$w_i(\mathrm{S}) = \begin{cases} f_i(S) - \dfrac{\lambda}{\lambda+1} q_i(S), & \text{if } v_i \in M_l, l \in \{1, 2, \dots, k-2\} \\ q_i(S) = d_i(S), & \text{if } v_i \in A \cup C \end{cases}$$

  - 4. build priority tree for S with $w_i(S)$



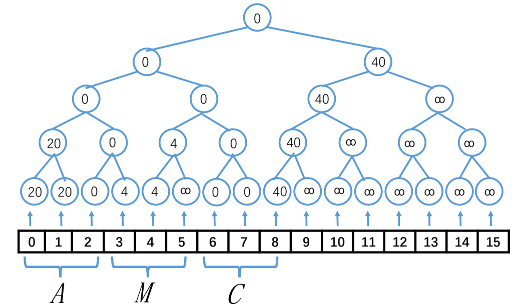
Algorithm (cont.)

- Step 2. greedy deletion
  - 1. get the node v with minimum weight
  - 2. delete the selected node, update the value of $g(S)$ and update node's weight that corelated with v
  - 3. repeat 1 and 2 until one of $A, M_1, \ldots, M_{k-2}, C$ is empty

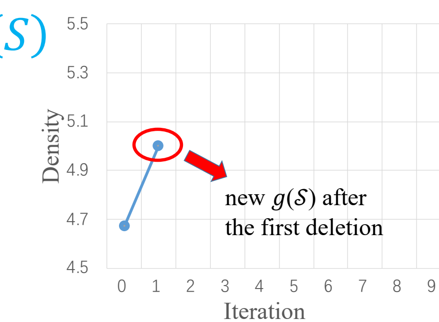


Delete minimum weighted node

Update priority of node

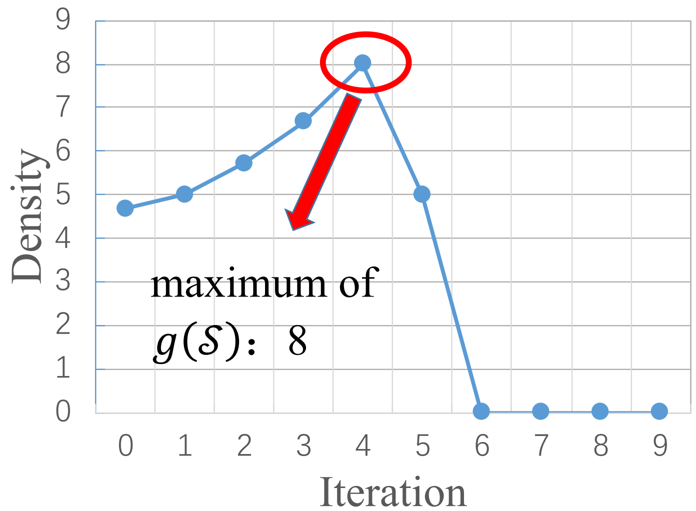


Update $g(S)$

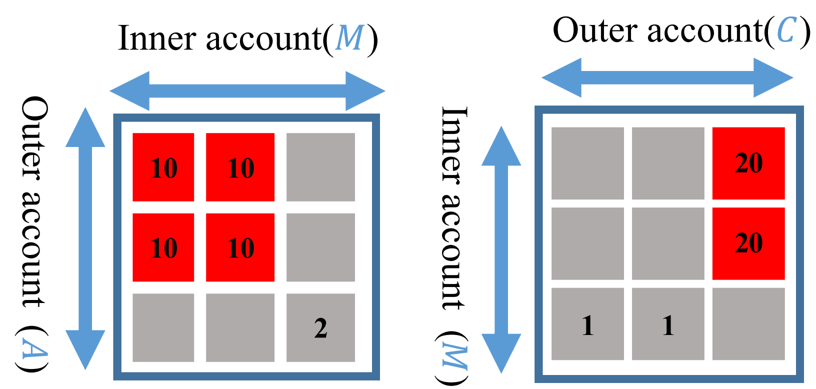


Algorithm (cont.)

- Step 3. get the result
  - 1. find the maximum value of $g(S)$
  - 2. recover correspond node set $\hat{S}$ corresponding to maximum $g(S)$



Recover $\hat{S}$



Result $\hat{S}$:

$\hat{A}$: {0,1}, $\widehat{M}$: {0,1}, $\hat{C}$: {2}

Algorithm (cont.)

- Theorem [Approximation Guarantee]
  - in 3-step ML (tripartite)

middle counts in S'

$$g(\hat{S}) \geq \frac{|M'|}{|S'|}\left(g(S^*) - \lambda\varepsilon \right)$$

FlowScope

node set just before the first optimal node removed

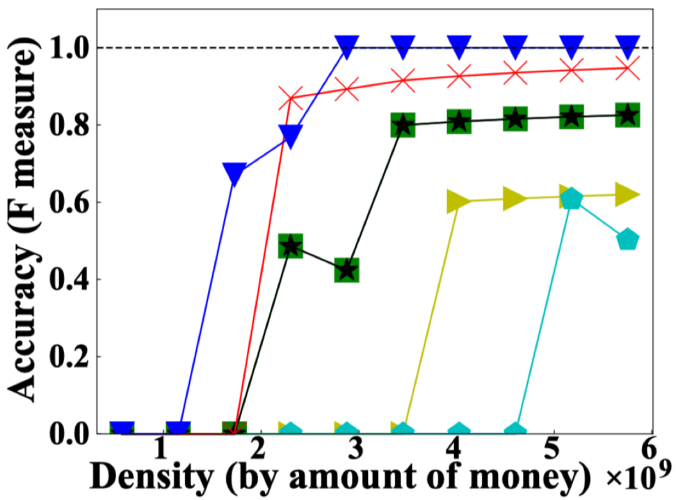
optimal

amount of camouflage transfers

Properties of **FlowScope:**

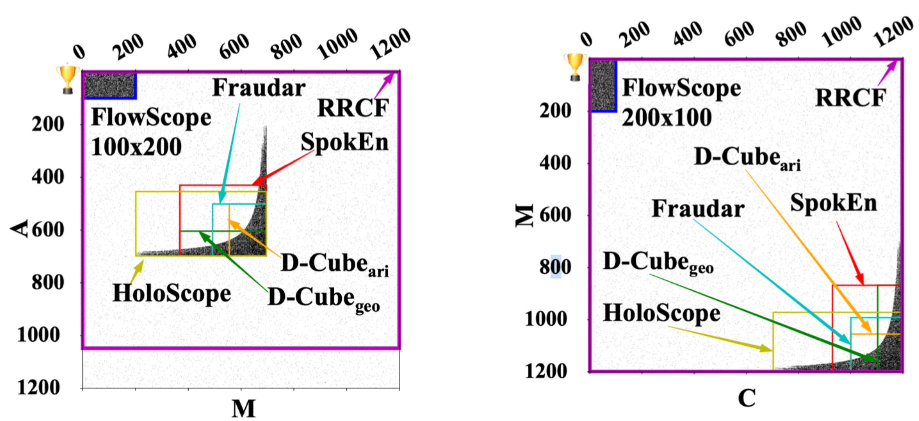
- ☐ ***Fast***: runs in near-linear time
- ☑ ***Accurate***: provides an accuracy guarantee
- ☐ ***Effective***: produces meaningful results in practice

Real-world performance



With ground-truth labelled

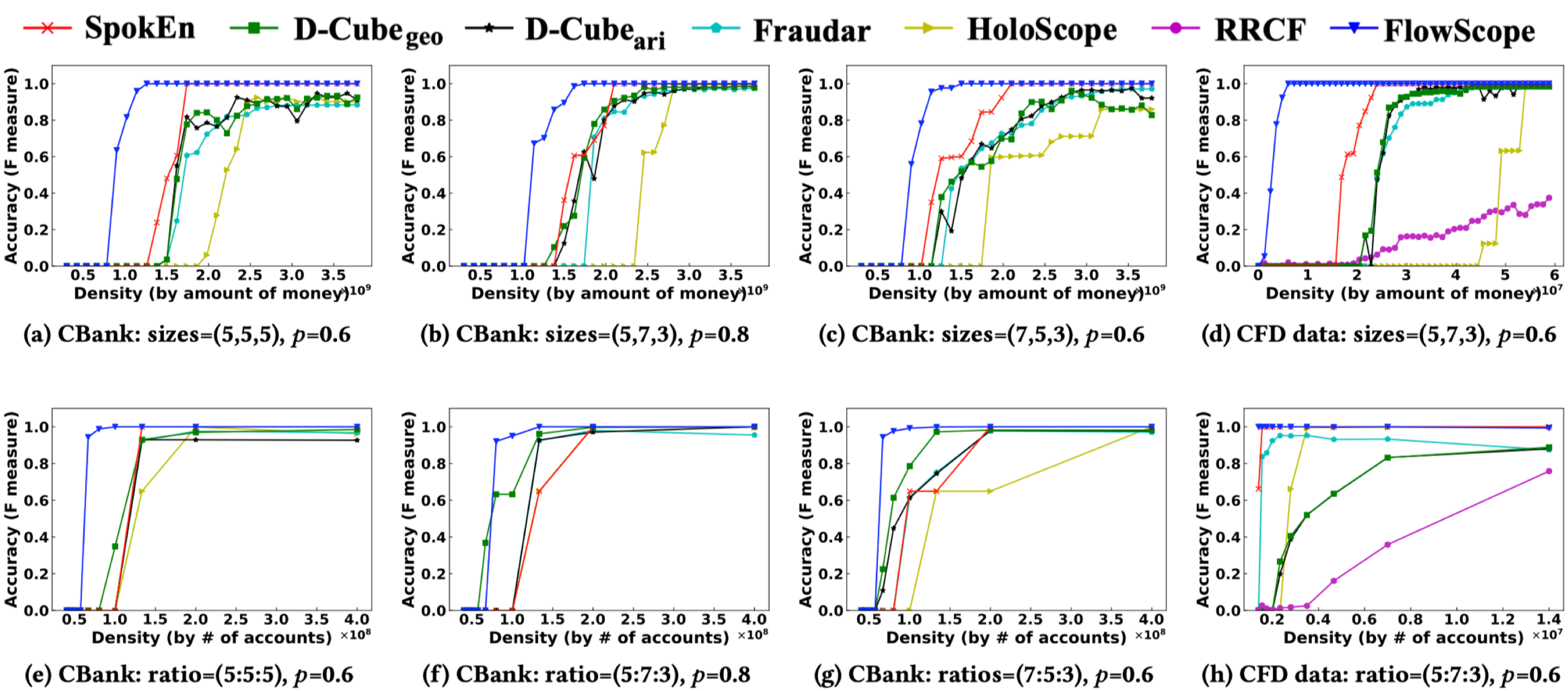
FlowScope Wins



Performance on synthetic data

Effectiveness: one middle layer

Good performance under variety of topologies



(a) CBank: sizes=(5,5,5), p=0.6
(b) CBank: sizes=(5,7,3), p=0.8
(c) CBank: sizes=(7,5,3), p=0.6
(d) CFD data: sizes=(5,7,3), p=0.6
(e) CBank: ratio=(5:5:5), p=0.6
(f) CBank: ratio=(5:7:3), p=0.8
(g) CBank: ratios=(7:5:3), p=0.6
(h) CFD data: ratio=(5:7:3), p=0.6

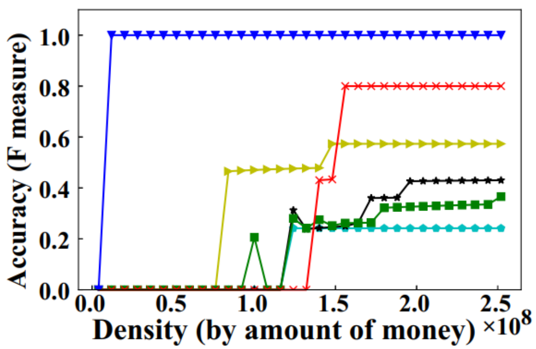
Effectiveness: one middle layer (cont.)

Summary in table

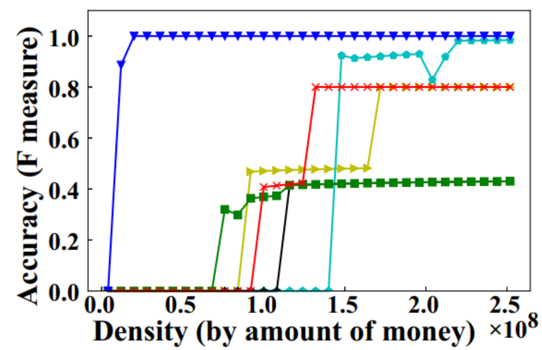
Dataset	metrics*	A:M:C	D-Cube$_{ari}$	D-Cube$_{geo}$	Fraudar	HoloScope	SpokEn	RRCF	FlowScope
CBank	FAUC	5:9:1	0.417 / 0.600	0.591 / 0.810	0.347 / 0.634	0.276 / 0.466	0.610 / 0.753	- / -	**0.633 / 0.800**
		5:5:5	0.502 / 0.658	0.501 / 0.709	0.467 / 0.683	0.379 / 0.655	0.598 / 0.708	- / -	**0.757 / 0.843**
		7:5:3	0.533 / 0.727	0.522 / 0.779	0.529 / 0.704	[illegible] / 0.547	0.633 / 0.708	- / -	**0.761 / 0.843**
	$F1 \geq 0.9$ (million \$ / node size)	5:9:1	190 / 30	- / 45	- / 30	[illegible]	154 / 30	- / -	**132 / 75**
		5:5:5	150 / 45	- / 45	[illegible]	[illegible]	116 / 45	- / -	**84.0 / 90**
		7:5:3	175 / 30	166 / 54	[illegible]	[illegible] / 15	122 / 30	- / -	**76.0 / 90**
CFD	FAUC	5:9:1	0.498 / 0.577	[illegible]	[illegible]	0.125 / 0.773	0.716 / **0.894**	0.253 / 0.538	**0.939** / 0.877
		5:5:5	0.565 / 0.633	[illegible]	[illegible] / 0.867	0.143 / 0.810	0.716 / 0.897	0.236 / 0.364	**0.962 / 0.900**
		7:5:3	[illegible]	[illegible]	0.593 / 0.826	0.0356 / 0.818	0.728 / 0.898	0.213 / 0.434	**0.970 / 0.900**
	$F1 \geq 0.9$ (million \$ / node size)	5:9:1	[illegible]	[illegible] / 15	- / 60	3.52 / 60	1.71 / 120	- / 15	**0.400 / 150**
		5:5:5	[illegible]	2.05 / 30	- / 150	- / 75	1.23 / **150**	- / 15	**0.240 / 150**
		7:5:3	[illegible]	- / 30	- / 120	- / 60	1.46 / 135	- / 15	**0.240 / 150**

FlowScope Wins

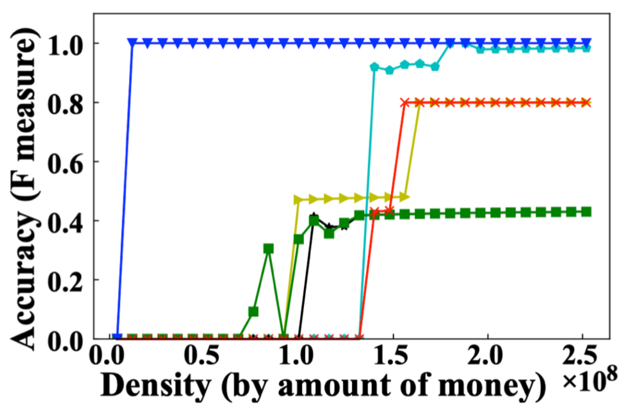
Robustness against longer transfer chains



(a) CBank: Transfer chains of (5,5,5,5)



(b) CBank: Transfer chains of (2,8,8,2)



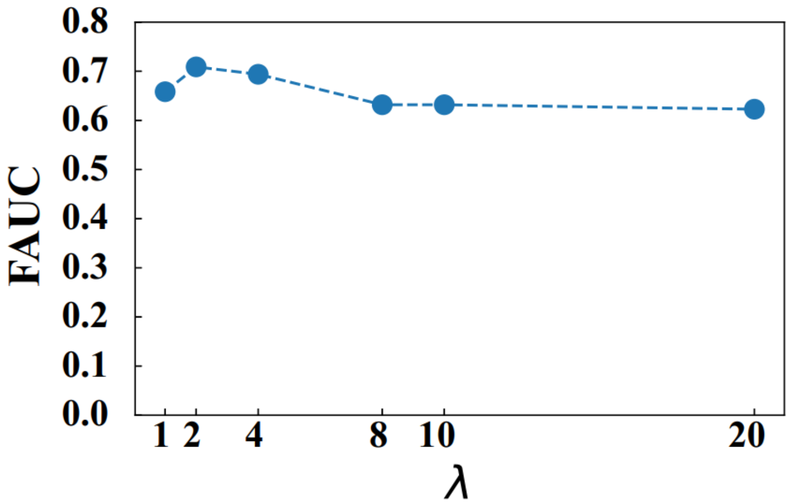
CBank: Transfer chains of (3,7,7,3)

Effectiveness: varies topologies and labelled data

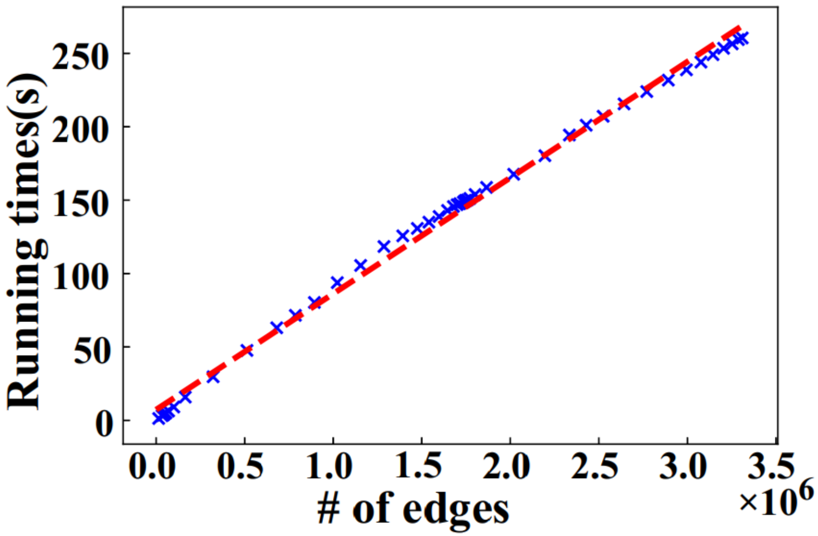
Properties of **FlowScope:**

- ☐ ***Fast***: runs in near-linear time
- ☑ ***Accurate***: provides an accuracy guarantee
- ☑ ***Effective***: produces meaningful results in practice

Sensitivity and Scalability



FlowScope is robustness to parameter

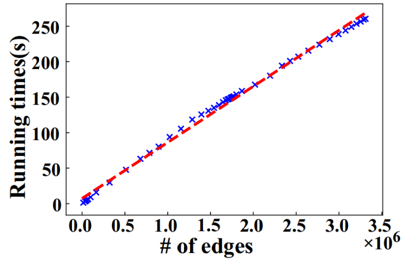


FlowScope runs in near-linear time with the # of edges

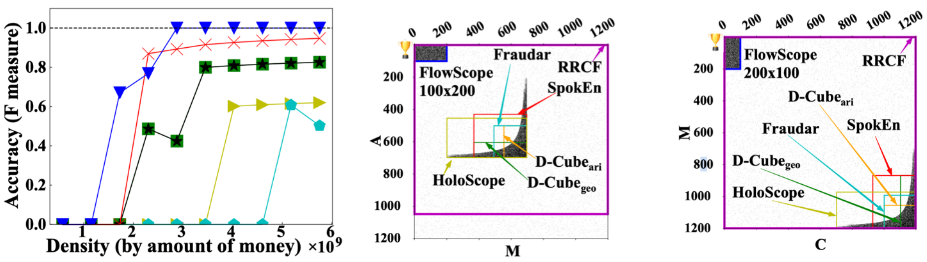
Conclusion

- FlowScope detects money laundering fast and effectively

☑ ***Fast***



☑ ***Effective***



☑ ***Accurate***

$$g(\hat{S}) = \frac{|M'|}{|S'|}\left(g(S^*) - \lambda\varepsilon \right)$$

☑ ***Reproducible***

GitHub

https://github.com/aplaceof/FlowScope

Reference

- [Charikar M, 2000] Charikar, Moses. "Greedy approximation algorithms for finding dense components in a graph." International Workshop on Approximation Algorithms for Combinatorial Optimization, 2000.
- [Asahiro et al, SWAT'96] Asahiro, Yuichi, et al. "Greedily finding a dense subgraph." Algorithm TheorySWAT'96 (1996): 136-148.
- [B Hooi et al, KDD'16] Bryan Hooi, Hyun Ah Song, Alex Beutel, Neil Shah, Kijung Shin, and Christos Faloutsos. 2016. Fraudar: bounding graph fraud in the face of camouflage. KDD 2016
- [M-Zoom] Kijung Shin, Bryan Hooi, and Christos Faloutsos. M-Zoom: Fast Dense- Block Detection in Tensors with ality Guarantees. ECML-PKDD. 2016, 264–280.
- [D-Cube] Kijung Shin, Bryan Hooi, Jisu Kim, and Christos Faloutsos. 2017. D-Cube: Dense-Block Detection in Terabyte-Scale Tensors. WSDM '17. 2017.
- [SpokEn] B Aditya Prakash, Mukund Seshadri, Ashwin Sridharan, Sridhar Machiraju, and Christos Faloutsos. Eigenspokes: Surprising patterns and scalable community chipping in large graphs. PAKDD 2010, 290–295.
- [Holoscope] Liu, S.; Hooi, B.; and Faloutsos, C. 2017. Holoscope: Topology-and-spike aware fraud detection. In Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, 1539–1548. ACM.
- [RRCF] Guha, S.; Mishra, N.; Roy, G.; and Schrijvers, O. 2016. Robust random cut forest based anomaly detection on streams. In International conference on machine learning, 2712–2721.

Thank you

Questions and Answers