Trivial Graph Features and Classical Learning are Enough to Detect Random Anomalies

– Online Appendix –

I. Decreasing sorted counters (DSC)

A counter is a named object associated to a non-negative integer value. A counter may be created (with value 0) or destroyed (when its value is 0), and its value may be increased or decreased by 1.

The DSC data structure maintains a sorted array of such counters in decreasing order of their value. Then, the first counter in the array has the maximal value, and the middle one has median value. In addition, DSC maintains an array giving the number of counters having each value. It also trivially maintains the sum of all counter values as well as the number of counters (hence the average value).

We propose here a DSC version ensuring that each operation works in $\mathcal{O}(1)$ time and space. It relies on several structures that it updates each time a counter is created, deleted, increased, or decreased:

values is the array of counter values sorted in decreasing order: it has one cell per counter, `values[0]` is the largest counter value, `values[1]` the second largest (that may be equal to the first one), and so on.

c2pos is the dictionary giving the index of counters in values: `c2pos[c]` gives the index of the value of counter named `c`, therefore the value of counter `c` is `values[c2pos[c]]`.

pos2c is the array that gives the name of the counter corresponding to index `i` in `values`; for any `i`, `c2pos[pos2c[i]]` is nothing but `i` and for any `c`, `pos2c[c2pos[c]]` is nothing but `c`.

val2pos is the dictionary giving the smallest index `val2pos[v]` of a counter with value `v` in `values`.

distrib is the dictionary giving the number of counters of value `v`, for any positive `v`.

Algorithm 1: Create a counter named c (with value 0)

if $distrib[0]$ *does not exist* **then** set it to 0;
if $val2pos[0]$ *does not exist* **then** set it to the length of $values$;
increase $distrib[0]$;
set $c2pos[c]$ to the length of $values$;
append c to $pos2c$ and append 0 to $values$;

Algorithm 2: Delete counter named c (with value 0)

decrease $distrib[0]$;
if $distrib[0] = 0$ **then**
  remove entry 0 in $distrib$ and $val2pos$
swap c and the last counter in $values$;
remove entry c in $c2pos$ and the last cell of $pos2c$;
remove the last cell of $values$;

Algorithm 3: Increase counter named c

let v be the value of c;
swap c and the first counter in $values$ with value v;
if $distrib[v+1]$ *does not exist* **then** set it to 0;
if $val2pos[v+1]$ *does not exist* **then** set it to $val2pos[v]$;
increase $distrib[v+1]$ and decrease $distrib[v]$;
increase $val2pos[v]$;
if $distrib[v] = 0$ **then**
  remove entry v in $distrib$ and $val2pos$
increase $values[c2pos[c]]$;

Algorithm 4: Decrease counter named c

let v be the value of c;
swap c and the last counter in $values$ with value v;
if $distrib[v-1]$ *does not exist* **then** set it to 0;
if $val2pos[v-1]$ *does not exist* **then** set it to $val2pos[v] + distrib[v]$;
increase $distrib[v-1]$ and decrease $distrib[v]$;
increase $val2pos[v]$ and decrease $val2pos[v-1]$;
if $distrib[v] = 0$ **then**
  remove entry v in $distrib$ and $val2pos$
decrease $values[c2pos[c]]$;

All these structures are initially empty (there is no counter), and they are updated by Algorithms 1 to 4.

With dictionaries implemented as hash tables, the expected time cost of dictionary operations is $\mathcal{O}(1)$. If counters are never deleted and if their values are bounded, then one may use arrays instead of hash tables, leading to a $\mathcal{O}(1)$ worst case cost for dictionary operations.

With dynamic arrays, all array operations are in $\mathcal{O}(1)$ amortized time. If the total number of counters (or a bound) is known in advance, then the worst case complexity is $\mathcal{O}(1)$.

Space complexity is linear with the number of counters, as the arrays and dictionaries contain one entry per counter, or

one entry per counter value at most.

Finally, we obtain a DSC implementation in which all operations have a cost in $\mathcal{O}(1)$ time and space. We will use it extensively to compute our trivial graph features in the following.

II. The case of bipartite graphs

A bipartite graph is a triplet $B = (\top, \bot, E)$ with $\top \cap \bot = \varnothing$ and $E \subseteq \top \times \bot$. Unlike unipartite graphs, bipartite graphs have two distinct node sets: $\top$ and $\bot$. Elements of $\top$ are called top nodes, the ones of $\bot$ are bottom nodes. Most importantly, links exist only between a top node and a bottom one.

Bipartite graphs are prevalent in anomaly detection problems. For instance, payment datasets correspond to links between credit-cards and stores, many log files correspond to links between users and the services they accessed, and social media data typically correspond to links between posts and their readers. We therefore present below a set of graph features dedicated to the bipartite case and obtained results, which are consistent with results on unipartite graphs presented in the paper.

This illustrates the flexibility of our approach: it is easily extensible to many types of graphs like bipartite, directed, multilayer, multiplex, or signed graphs, and their combinations.

1) Bipartite graph features: Our goal here is to characterize an occurrence of a link (u, v) in a bipartite history graph $B = (\top, \bot, E)$. We use the same notations as in the unipartite case.

Some unipartite features remain unchanged in the bipartite case. For instance, the number of links m in $B = (\top, \bot, E)$ is the same as in the unipartite graph $G = (\top \cup \bot, E)$. Likewise, the degree of a node, its weighted degree, the weight of a link, the number of links of a given weight, as well as the total, maximum, and median link weights are all the same in B and G. Other unipartite features lead to two twin bipartite features, a top and a bottom version. We give the list for top nodes; the features for bottom nodes are the same (just replace $\top$ by $\bot$ and u by v):

- the number of top nodes $n_\top = |\top|$,
- the number of degree 1 top nodes $|\{x \in \top,\ d(x) = 1\}|$, the number of degree 2 top nodes $|\{x \in \top,\ d(x) = 2\}|$, the maximal top degree $\max_{x \in \top}(d(x))$, the median top degree $\text{med}_{x \in \top}(d(x))$,
- the number of weighted degree 1 top nodes $|\{x \in \top,\ \delta(x) = 1\}|$, the number of weighted degree 2 top nodes $|\{x \in \top,\ \delta(x) = 2\}|$, the maximal top weighted degree $\max_{x \in \top}(\delta(x))$, the median top weighted degree $\text{med}_{x \in \top}(\delta(x))$,
- the numbers of top nodes having degree $d(u)$, having weighted degree $\delta(u)$, having a degree larger than $d(u)$ and having a weighted degree larger than $\delta(u)$.

Like in the unipartite case, computing these features has a $\mathcal{O}(1)$ time and space cost thanks to DSC structures. They also have intuitive meanings. For instance, if the considered data consists in credit-card payments in stores, the degree of a credit-card is the number of stores in which it was used, and its weighted degree is the number of times it was used; the degree of a store is its number of clients, and its weighted degree is its number of sales; and the weight of a link is the number of times the corresponding credit-card was used in the corresponding store.

2) Datasets: Like with unipartite graphs, we use first widely used and publicly available reference bipartite data described in [1]. They are named UCI Forum and MovieLens.

TABLE I: Basic characteristics of bipartite datasets used in this paper, ordered with respect to their total number of timestamped links ℓ (last column). The other columns indicate the number of nodes n, and the number of distinct links m.

Dataset	n	m	ℓ
UCI Forum	1,421	7,089	33,720
MovieLens	2,625	100,000	100,000
Peru	1,423,185	18,572,824	40,509,547

We add the following large scale dataset. **Peru** is a payment dataset: nodes correspond to bank accounts and sellers, and links correspond to payments, with timestamps in seconds. It was collected in Peru in 2016–2017 and it is studied in [2].

The basic characteristics of these bipartite datasets are presented in Table I.

3) Comparison to state-of-the-art:

TABLE II: AUC performances of state-of-the-art methods and TGF (with H-type history graphs of size 1000), applied to *bipartite* public reference datasets with 5% anomaly injection. The best results are in bold.

	UCI Forum 5%	MovieLens 5%
NetWalk	0.523	0.496
DAGMM	0.706	0.829
GraphSAGE	0.607	0.466
BEA	0.852	0.909
TGF	**0.908**	**0.988**

4) Using diverse resolutions:

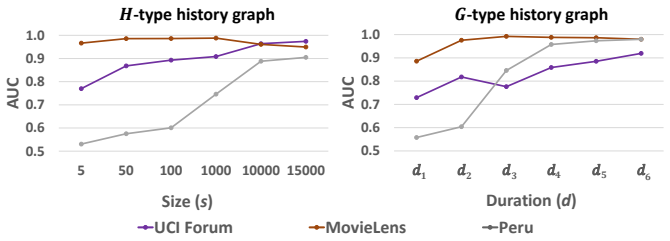

Fig. 1: The impact of size s and duration d (horizontal axis) of the H-type (left) and G-type (right) history graphs on AUC scores. We consider here 5% anomaly injection and learning with $r = 0.7$.

III. More on computational costs

The paper presents TGF global computation costs and average computation costs per link. However, it says nothing on the ability to process links on-the-fly in reasonable time: even if the average computation time is very low, the worst one may be very high. In order to explore this question, we present in Figure 2 the distribution of computation time per link, in the representative large-scale case of the Peru dataset.

The computation time distributions are well centered on their average. In extreme cases, computing features for H-type history graphs takes up to a few microseconds. Compared to an average below one nanosecond, this remains very fast. With similar average running times, G-type history graphs exhibit more variability, up to a few dozens of milliseconds. This is not surprising, as H-type history graphs do have a bounded size, but G-type ones do not. Finally, these distributions show that TGF is well suited for real-time processing of transactions, since observed worst case computational costs are very low.

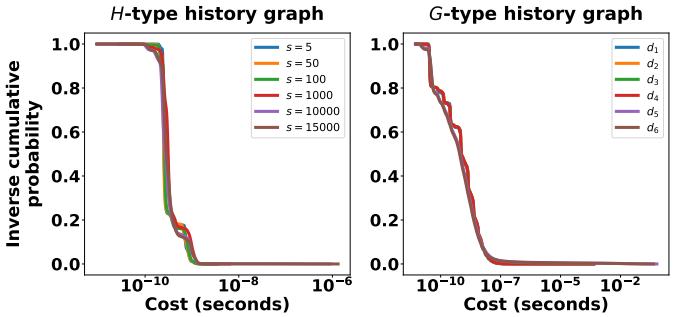

Fig. 2: Inverse cumulative distribution of the computation time (in seconds) per link in log-lin scales in the Peru dataset for H-type history graphs (left) and G-type history graphs (right), with 5% anomaly injection.

References

[1] J. Kunegis, "Konect: the koblenz network collection," in *Proc. of WWW*, 2013.

[2] H. Alatrista-Salas, V. Gauthier, M. N. del Prado, and M. Becker, "Impact of natural disasters on consumer behavior: Case of the 2017 el niño phenomenon in peru," *PLoS ONE*, 2021.