REDUCING THE COGNITIVE LOAD ON ANALYSTS THROUGH HAMMING DISTANCE BASED ALERT AGGREGATION

Peter Mell[1], Richard Harang[2]

[1] National Institute of Standards and Technology, Gaithersburg, MD, USA

[2] U.S. Army Research Laboratory, Baltimore, Maryland, USA

ABSTRACT

Previous work introduced the idea of grouping alerts at a Hamming distance of 1 to achieve lossless alert aggregation; such aggregated meta-alerts were shown to increase alert interpretability. However, a mean of 84023 daily Snort alerts were reduced to a still formidable 14099 meta-alerts. In this work, we address this limitation by investigating several approaches that all contribute towards reducing the burden on the analyst and providing timely analysis. We explore minimizing the number of both alerts and data elements by aggregating at Hamming distances greater than 1. We show how increasing bin sizes can improve aggregation rates. And we provide a new aggregation algorithm that operates up to an order of magnitude faster at Hamming distance 1. Lastly, we demonstrate the broad applicability of this approach through empirical analysis of Windows security alerts, Snort alerts, netflow records, and DNS logs. The result is a reduction in the cognitive load on analysts by minimizing the overall number of alerts and the number of data elements that need to be reviewed in order for an analyst to evaluate the set of original alerts.

KEYWORDS

Alert aggregation, Cognitive load, Hamming Distance, Hypergraphs, Security logs

1. INTRODUCTION

Human review of security logs is a difficult and labor-intensive process. This is especially true in the area of intrusion detection systems (IDSs) which often suffer from extremely high false positive rates (see, e.g. [1] [2] [3] [4], among others). This problem is exacerbated in signature-based systems such as Snort[1] [5], where broadly-written rules may trigger repeatedly on innocuous packets. This large number of false positive results creates a significant workload for IDS analysts, who must sort through them in order to locate the relatively few true positives. It is thus desirable to automate abstraction and correlation of the logs so as to enable analysts to make efficient decisions.

The work of [6] mitigated this problem by providing an algorithm to aggregate Snort intrusion detection alerts with discretely-valued fields by combining those that are at most Hamming distance 1 apart. This approach was shown effective in both reducing the number of resulting meta-alerts that need to be reviewed by analysts and in increasing their interpretability (see [6] for ex-

[1] Any mention of commercial products or reference to commercial organizations is for information only; it does not imply recommendation or endorsement by the U.S. government nor does it imply that the products mentioned are necessarily the best available for the purpose.

DOI : 10.5121/ijnsa.2014.6503

ample meta-alerts). Related alerts were merged, reducing human analysis time and enabling more rapid identification of significant threats. Furthermore, the reduction of alerts to meta-alerts was lossless in that the original alerts could be reconstructed from the meta-alerts. This was important because it provided analysts all alert information when making relevancy decisions. The method achieved an average alert reduction of 83.2 % applying their method to 30 days of Snort alerts grouped by hour, with an execution complexity of $O(n^2)$ where n represents the number of alerts.

While the reduction on Snort alerts using a Hamming distance of 1 was significant, the remaining number of meta-alerts was still considerable (although it should be understood that their data was taken from a large enterprise-scale production network). For example, a mean 24 hour time slice of 84023 alerts was reduced to a mean of 14099 meta-alerts by aggregating on hourly batches. Reviewing such a number of meta-alerts on a daily basis still represents a challenge despite the improvement.

In this work, we address this limitation by investigating several approaches that all contribute towards reducing the cognitive load on the analyst, by both reducing the overal number of alerts and reducing the number of data elements that need to be reviewed in order for an analyst to evaluate the set of original alerts. We also investigate approaches that providing more timely analysis. In performing this exploration, we uncover several new aggregation capabilities not available in the original work.

1) We explore how to decrease the number of meta-alerts alerts down to some user defined maximum by varying the Hamming distance used for aggregation (the original work operated only at a Hamming distance of 1). We find that increasing the Hamming distance monotonically decreases the number of meta-alerts, while at the same time increasing the level of abstraction of those alerts. Also, we empirically discover that there exists an operating point that presents a minimum number of overall data elements, consistently at low Hamming distances (universally between 1 and 4 in all of our data sets). This is important because the number of data elements reflects the amount of work that must be done by the analyst.

2) We explore how increasing bin sizes reduces the number of meta-alerts (something not tested in [6]). For example, for Hamming distance 1 aggregation using the original algorithm, the aggregation rate for Snort alerts rises from a mean of 83.2 % for hourly bins to 96.1 % for daily bins. We can thus reduce the mean daily time slice of 14099 meta-alerts from [6] down to a mean of 3276 meta-alerts. This is a much more manageable workload for human review by a large enterprise.

3) Lastly, we present a new $O(n\log n)$ hypergraph based aggregation algorithm that can run up to an order of magnitude faster at small Hamming distances, thus facilitating on-demand analysis of larger time slices. Furthermore, its construction allows for a streaming mode where alerts are analyzed upon arrival and aggregated meta-alerts dumped on demand, a capability that cannot be replicated using the intrinsically batch-oriented approach in [6]. The new algorithm provides slightly better aggregation (a 4.9 % improvement for daily bins of Snort alerts at a Hamming distance of 1).

The original research also focused solely on a single Snort IDS data set. In this work, we demonstrate more general applicability of the Hamming distance aggregation approach by applying it to three new data types: Windows security alerts, Cisco version 5 Netflow, and Domain Name Service (DNS) request logs. We also reevaluate the Snort data set from [6] for comparative purposes. Lastly, we improve upon the original work by providing a Public domain Python 2.7.3 implementation for our new aggregation algorithm (available at [7]).

In summary, the primary contributions of this paper are:

- An analysis of how increasing Hamming distance decreases the number of meta-alerts. Also we empirically discover that there exist operating points that that yield a minimum number of overall data elements at consistently low Hamming distances (typically on the order of $\sqrt{m}$ where m is the number of columns in the data).
- The discovery that increased bin sizes reduces the number of meta-alerts.
- Provision of a new O($n$log$n$) hypergraph based aggregation algorithm that has streaming mode capabilities, better aggregation, and up to an order of magnitude improvement in runtime at small Hamming distances.
- Evidence of the applicability of Hamming distance aggregation to four data sets: Snort, Windows security events, Netflow, and DNS.
- Provision of public domain Python 2.7.3 code for our new aggregation algorithm.

The impact of these results is as follows. The results that minimize the number of alerts to review reduce the context switching that an analyst must perform between reading distinct alerts. The results that minimize the number of data elements to review reduce the overall amount of data that an analyst must ingest to understand the set of original alerts (and a certain view of their relationships). The results that increase the speed of analysis (and offer a streaming mode operation) reduce the latency in which an analyst receives the data to review. The provision of working code enables immediate testing and use by operational groups.

The development of the work is as follows. In section 2, we discuss related work. In section 3, we provide a definition and an example of Hamming distance aggregation. In section 4 we present a new Hamming distance aggregation algorithm and section 5 discusses our expansion to the previously published algorithm. In section 6 we describe the input data sets and our experiment; section 7 provides the results. Although covered in [6], section 8 reviews the intuition behind the Hamming Distance aggregation approach. We also provide our experience with its use and feedback from operational analysts. Section 9 summarizes our conclusions.

2. RELATED WORK

The problem of alert aggregation has been addressed from a variety of perspectives. A more general overview is presented in [6], however in brief, alert aggregation approaches fall into two broad categories.

The first category involves expert knowledge used to construct a system for classifying, correlating, and ranking alerts based on external knowedge of existing attacks [3] [8] [9] [10] [11] [12], network vulnerabilities [3] [9], or both [10]. Rules may also be aggregated using user-defined similarity metrics based on expert knowledge [3], and some alerts may be completely ignored outright if prior knowledge suggests that they are irrelevant [13].

The second category encompasses probabilistic and data mining approaches, which are used to group and aggregate alerts without requiring the construction and maintenance of external data stores or schema. IDS alerts are often discrete in nature, however, which often poses a challenge to such methods [14]. Nevertheless, a variety of approaches [1] [15] [16] have been developed in this area that have been empirically demonstrated to be effective, even when they rely on assumptions (such as a Gaussian distribution of continuous features) that while functional in practice are clearly not accurate in a theoretical sense.

Recent work somewhat related is found in [17]. They identify frequent patterns in alert logs by use of the Frequent Pattern Outlier Factor (as developed by [18]), classifying variable-length alerts by identical subsequences of values and ranking them by the frequency of the subsequences

in the data. The top arbitrary quantile of alerts with the least common patterns are presented as 'candidate true alerts' and the rest discarded. While our work also classifies on identical subsequences, our goal is aggregation of similar alerts into a reduced set of meta-alerts without discarding any data.

3. OVERVIEW OF HAMMING DISTANCE AGGREGATION

Hamming distance aggregation at an aggregation distance of *d* may merge a group of alerts into a meta-alert if the alerts have at most *d* fields that do not match. An alert may be merged only with itself to create a meta-alert that covers only that single alert. We define an optimal Hamming distance aggregation as the smallest possible set of meta-alerts that cover all original alerts at some maximum Hamming distance *d*.

As an example, consider the alerts in Table 1. Alerts 4 and 5 may be aggregated at Hamming distance 1 into meta alert (B,F,*), where * is equal to the set (J,K), because they disagree only on one column, 3. Alert 1 cannot be aggregated because it has two columns, 2 and 3, whose values do not match any other alerts. In this way, alerts 2 and 3 also may not be aggregated with any other alerts because each disagrees with all other alerts on 2 columns.

Table 1. Example set of alerts to be aggregated

Alert Number	Column 1	Column 2	Column 3
1	A	C	G
2	A	D	H
3	A	E	I
4	B	F	J
5	B	F	K

4. HYPERGRAPH-BASED ALGORITHM

This section provides a new hypergraph based Hamming distance aggregation algorithm that operates in O(*n*log*n*) through leveraging a set cover approximation algorithm.

4.1 Leveraging Set Cover

In the set cover problem as described in [19] we are provided "a finite set X and a family F of subsets of X, such that every element of X belongs to at least one subset in F." The problem is to find the minimal subset, S, of F such that S includes all elements of X. Thus, the union of the subsets in S must be equal to X. More formally, given X and F as described above where $\forall x \in X, \exists f \in F: x \in f$, we wish to find a minimal $S \subseteq F: \cup_{s \in S} s = X$. The set cover problem is known to be NP-complete [20] and is thus not currently tractable in polynomial time.

It can be proven that Hamming distance aggregation can be reduced to an instance of the set cover problem (proof omitted for space). Conceptually, each alert is encoded as an element of X. Each possible grouping of alerts separated by a Hamming distance of at most *d* is encoded as an element of F. Identification of all such possible groups can be done in O(*n*) time where *n* is the number of alerts, as described later in this paper. Through this, any Hamming distance aggregation problem can be reduced to a set cover problem in polynomial time. It therefore follows that any algorithm that can solve the set cover problem may be adapted to solve the Hamming distance problem. There exists a widely cited greedy approximation algorithm for set cover [21], and

in this paper we propose to use this approach to approximate optimal Hamming distance aggregation.

The greedy approximation algorithm for set cover is as follows [19]:

Set Cover Approximation (X,F)

1. **Remaining**=X
2. **Soln_set**= None
3. While **Remaining** is not None
    a. Select an element, **S**, of **F** that maximally covers the elements in **Remaining**
    b. Remove from **Remaining** the elements in **S**
    c. Add to **Soln_set** the subset **S**
4. Return **Soln_set**

As analyzed in [19] there exists an implementation that has time complexity $O(\sum_{S\in F}|S|)$; with Hamming distance aggregation the size of S may be equal to *n*, the number of alerts. Furthermore, the number of elements in F is always greater than or equal to *n* since in the worst case each alert may be merged only with itself to form a meta-alert. Using this analysis, the greedy set cover approximation algorithm has a time complexity of $O(n^2)$ when applied to Hamming distance aggregation. A major contribution of our work is to use unique characteristics of Hamming distance aggregation to reduce this time complexity to $O(n\log n)$.

4.2 Algorithm Description

In this section, we describe a high level approach using hypergraphs to aggregate alerts at varying Hamming distances. We then prove that the algorithm always extracts a meta-alert representing the largest available grouping of unaggregated alerts during each iteration, thus implementing the greedy approximation to the set cover problem. In the next section, we provide an efficient implementation and evaluate the algorithmic complexity.

In order to reliably extract all meta-alerts using a Hamming distance of *d*, we construct a hypergraph over the space of alerts, where each alert represents a node in the hypergraph, and each edge connects all nodes that are at most Hamming distance *d* from each other. If our alert set to be aggregated has *n* alerts with *m*, columns, we thus construct a hypergraph with *n* nodes, each node having mCd edges connecting it with other nodes in the graph (where mCd is the binomial coefficient). Each hyperedge thus represents a potential meta-alert and is labeled with the value of the meta-alert. Note that the field(s) that differ between the alerts covered by a hyperedge is denoted in a meta-alert by the wildcard '*' character. The hyperedge label then represents all alerts covered by it.

Using the example of the previously discussed Table 1 for illustrative purposes with a Hamming distance of 1, alert 1 would be entered as a node labeled with the tuple (A,C,G), and be covered by three hyperedges: (*,C,G), (A,*,G), and (A,C,*). Similarly, alert 5 in Table 1 would be entered as a node labeled (B,F,K) and be covered by hyperedges (*,F,K), (B,*,K), and (B,F,*). Note that alert 4 in Table 1 with label (B,F,J) would also be covered by hyperedge (B,F,*), indicating that both nodes covered by that hyperedge could be aggregated into a single meta-alert. Hyperedge (B,F,*) is the only hyperedge covering more than one node. The complete Hamming distance 1 aggregation hypergraph for the alerts is provided in

Figure **1**.

The power of this construction is that the largest meta-alert at some Hamming distance d can be easily determined by finding the largest hyperedge and examining the covered nodes.

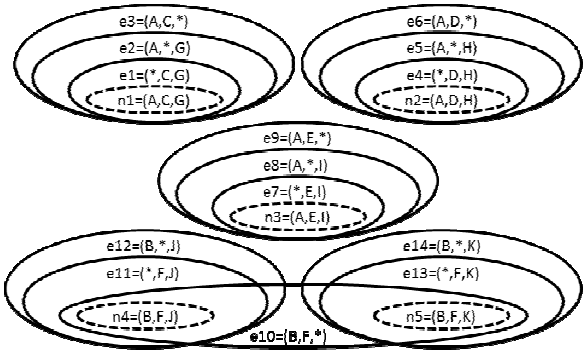

Figure 1. Hypergraph for Alert Aggregation Example

We now present our high level algorithm for Hamming distance 1 aggregation that implements the greedy set cover approximation algorithm:

1. In the initial collection step, we assemble the alerts into a hypergraph.
2. To extract the meta-alert covering the most available alerts, we identify the hyperedge with the largest incident node count; this edge and all nodes incident upon this edge are removed from the graph and processed into a meta-alert. In case of a tie, we arbitrarily choose a candidate meta-alert. This handling of ties is consistent with published implementations of the greedy set cover approximation algorithm [19].
3. The statistics of the hypergraph are updated to reflect the removed nodes. In particular, the incident node count for each hyperedge must be updated to reflect the removal of the nodes associated with the meta-alerts.
4. Proceed from step 2 in the updated hypergraph.

This algorithm is straightforward and will always produce the single largest meta-alert possible in the remaining data for each iteration. However, its performance is significantly impacted by the search for the next hyperedge to convert to a meta-alert as well as the process of updating the hypergraph after the removal of the largest meta-alert. The next section discusses optimizations to our algorithm to address these computational challenges and presents time complexity results.

4.3 Implementation Details

Assume that there are n alerts with m data fields per alert being aggregated at a Hamming distance d. Our implementation of the high level algorithm is given below.

Step 1: Hypergraph Construction

The hypergraph is constructed by iterating through each alert (e.g., (A,C,G)) and generating every possible hyperedge label (e.g. for d=1, (*,C,G), (A,*,G), and (A,C,*)) in $O(n\times(m\mathrm{C}d))$ time. Each hyperedge label is used as a key for inserting the alert into a hash table referred to as 'hdict'. The hdict keys represent the hyperedges and the values are themselves smaller hash tables representing the nodes covered by the respective hyperedge. Thus, the total complexity of hypergraph construction is $O(n\times(m\mathrm{C}d))$.

Step 2: Singleton Removal

This step extracts as meta-alerts all alerts (or groups of identical alerts) that cannot be aggregated with any other alerts because no hyperedge connects them to another distinct alert. This can be done in $O(n\times(m\mathrm{C}d))$ time.

Step 3: Sorted Hyperedge Tree

Next we create a red-black tree to sort the hyperedges by size. For all keys in the hdict, we look up the number of alerts covered by each hyperedge in $O(n\times(m\mathrm{C}d))$. This creates all key-value pairs of the form (hyperedge label, number of covered alerts). Each key-value pair is inserted into the tree sorted by the number of covered alerts in $O(n\times(m\mathrm{C}d)\times\log(n\times(m\mathrm{C}d)))$.

Step 4: Extract Meta-Alerts

We now iteratively identify the largest hyperedge, create an associated meta-alert, and call step 5 to update the data structures to reflect removal of the hyperedge and all covered alerts. This can be done in $O(n\times\log(n\times(m\mathrm{C}d)))$. In each meta-alert we store the hyperedge label and the values corresponding to the '*' fields (we don't store each alert separately), unless all alerts in the hyperedge are identical in which case we simply store a single copy of the alert along with the number of times it is repeated.

Step 5: Data Structure Update

For each meta-alert extracted in step 4, this step must update the data structures to make them ready for identification and removal of the next largest hyperedge. Doing this involves three parts: 1) removing the chosen hyperedge from the tree, 2) removing the chosen hyperedge from hdict, and 3) removing the covered alerts from all other hyperedge entries in hdict and then updating the hyperedge sizes in the tree. An amortized analysis yields $O(n\times(m\mathrm{C}d)\times\log(n\times(m\mathrm{C}d)))$.

All five steps together then produce a time complexity for the entire algorithm of $O(n\times(m\mathrm{C}d)\times\log(n\times(m\mathrm{C}d)))$. We treat m as a constant since for a particular data set, the number of fields will be fixed. For d=1 then, we get $O(n\log n)$ which empirically gives us up to an order of magnitude improvement over the $O(n^2)$ Hamming distance 1 aggregation in [6].

Note that, in contrast to the aggregation process described in [6], which evaluates a single column at a time across all alerts, the present method can operate in an incremental method. At any point in the collection process, the construction of the hypergraph in step 1 may be briefly suspended and an arbitrary number of meta-alerts may be extracted, at each stage updating the hypergraph to account for the removal of the alerts covered by the meta-alert (a single iteration of step 3, followed by alternating 4 and 5 for as many meta-alerts as desired). The updated hypergraph may then immediately begin accepting additional alerts.

To limit the $m\mathrm{C}d$ growth from overwhelming the execution time, we have coded in a threshold value. If $m\mathrm{C}d$ exceeds the threshold then we permanently wildcard the field with the greatest entropy and then try to run the algorithm again with $(m\text{-}1)\mathrm{C}(d\text{-}1)$. If this value still exceeds the threshold then we repeat the process iteratively, walking down the $m\mathrm{C}d$ curve until we fall below the threshold. This thresholding mechanism is crucial for enabling fast operation at mid-range Hamming distances.

5. COLUMN-BASED ALGORITHM

The prior approach presented in [6] was designed to work only at a Hamming distance of 1. Like our algorithm, it iteratively attempts to extract the largest available meta-alert until no alerts are left. It identifies the largest available meta-alert through iteratively identifying columns having maximal sets of alerts with the same value in order to identify meta-alerts. More specifically, it finds the alert column with the maximum number of identical values and creates a subset of the alert database based on alerts with that value in that column. This procedure is done repeatedly and the set of alerts to be aggregated into a particular meta-alert narrows with each iteration. When one final column is left, unique values have been identified for all other columns. The algorithm has now identified a meta-alert with an alert Hamming distance of 1.

For our work, we expand this algorithm to aggregate over variable Hamming distances. We do this by stopping the column identification and subset operation once *m-d* columns have been chosen and the values "fixed" for those fields to create the next meta-alert. Unfortunately, this algorithm will not always extract the largest grouping from the set of available alerts as proven in Theorem 1 below. Thus, it does not implement its greedy set cover approach perfectly and the size of the successively extracted meta-alerts is may not be monotonically decreasing. In our empirical studies, the size was never monotonically decreasing and thus non-optimal with respect to following the set cover approximation algorithm.

Theorem 1: *The column-based algorithm may not choose the largest available alert grouping.*

Proof by counterexample: Assume that the algorithm will always select the largest available alert grouping at Hamming distance 1 and consider the data shown previously in Table 1. It will first pick column 1 because it has three "A" values and it will subset alerts 1, 2, and 3 because they have that value. It will then look at columns 2 and 3 for those alerts and will be unable to identify any alerts to aggregate because they are all at Hamming distance 2. The resulting meta-alert for this iteration will contain only a single alert (a random choice of alert 1, 2, or 3) in order to meet the Hamming distance 1 constraint. However, this is a contradiction to our assumption that the greedy algorithm always extracts the largest set of alerts first because the actual largest grouping with a Hamming distance of 1 is a grouping with alerts 4 and 5. Q.E.D

6. EXPERIMENTAL DESIGN

We apply variable Hamming aggregation to four distinct sets of data. To show general applicability of the approach, we evaluate DNS request logs, Cisco v5 Netflow data, and Microsoft Windows security logs. For comparative purposes, we evaluate the Snort IDS dataset from [6].

The Windows logs were Event Viewer security logs (.EVTX) monitoring the file system accesses of 11 workstations over a period of 6 months in 2013. We performed extensive tests on aggregating other Windows security event types but omit the results due to space limitations (the results are similar except for variance in the optimal Hamming distance required to minimize the number of data elements). We chose file system access events for this work as an example Windows alert type with both a large number of instances in the data set as well as a large number of data fields.

Table 2. Experiment Data Sources

Data source	Number of fields	Number of records
Snort alerts	11	2300000
DNS request logs	7	6310856
Cisco Netflow records*	13	14066423
Windows file system events	24	78485

*Typical Cisco v5 Netflow records contain 22 fields; 9 of which (2 mask fields, 2 padding fields, 2 ASN-related fields, 2 interface-related fields, and the nexthop field) could not be populated due to hardware limitations.

Our experiment compares our hypergraph aggregation algorithm against the column-based algorithm in [6] (that, as discussed previously, we modified to enable aggregation across variable Hamming distances). We focus on varying the Hamming distance and batch sizes in order to see the effect on the number of meta-alerts and data fields presented to the analysts while recording execution times.

All experiments were performed on commodity computers using 3GHz quad-core Intel processors and 8GB of RAM running Python version 2.7.2.

7. EXPERIMENTAL RESULTS

The hypergraph approach of the present work, even when threshold values are used, reliably produces aggregation results that are as good as or better than the work of [6]. This is in respect to both the number of meta-alerts created, as well as the total reduction in data elements. We also find that increasing batch sizes results in a significant reduction in the number of meta-alerts and a lesser reduction in the number of data elements provided to the analysts (for both algorithms). Universally across our 4 data sets, the optimal Hamming distance for reduction of data elements was small, varying from 1 to 4. With respect to execution time, our hypergraph based algorithm can be orders of magnitude faster at lower Hamming distances. It is these Hamming distances that should be most useful as the alerts are abstracted the least and the number of data fields is minimal. In some low Hamming distance cases, the column based approach was incomputable, making use of the hypergraph algorithm necessary. Additionally, the preference towards larger batch sizes further supports use of our $O(n\log n)$ hypergraph approach over the $O(n^2)$ column based algorithm. We now discuss the empirical results supporting these findings.

7.1 Reduction in Number of Meta Alerts Through Increasing Hamming Distance

As the Hamming distance of aggregation increases, the total number of meta-alerts drops in a non-increasing fashion, to the trivial minimum of 1 when the Hamming distance is equal to the number of columns (since in a data set with m columns, the maximum possible Hamming distance between two items is m, and hence all items may be collected within a single meta-alert). Results for Snort, DNS, and Flow data are provided in Figure 2, below. Note that in each case, there is a Hamming distance at which the number of meta-alerts drops dramatically; in all cases that we examined, this distance corresponds to the optimal Hamming distance with respect to total number of data elements presented (see the following section).

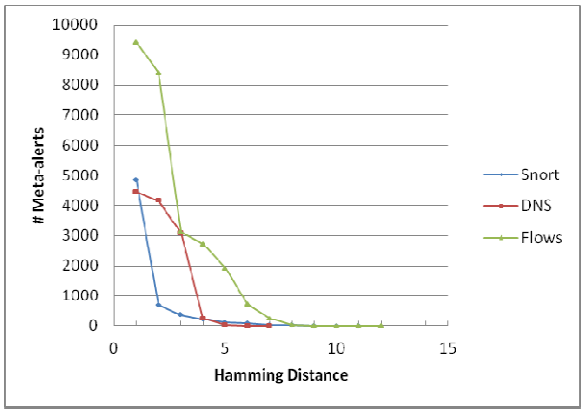

Figure 2. Reduction in number of meta-alerts as Hamming distance increases.

7.2 Reduction in Number of Data Elements Through Adjusting Hamming Distance

We empirically discovered that, for each data type, there exists a Hamming distance that minimizes the number of data elements presented to the user. Furthermore, these global optimal Hamming distances are small (between 1 and 4) for all data types as shown in Table 3, even though the number of fields varies between 7 and 24. This is important because these Hamming distances are those for where the meta-alerts are abstracted the least and are most likely to be operationally useful. Note the correlation in Table 3 between the optimal Hamming distance and the number of fields.

Table 3. Optimal Hamming Distance for Hypergraph Algorithm Data Element Reduction

Data source	Number of fields	Optimal Hamming distance
Snort alerts	11	2
DNS request logs	7	1
Flow records	13	3
EVTX records	24	4

Discussed further in later sections, the column based approach of [6] has a significant disadvantage with respect to execution time at these lower Hamming distances.

We now look at data element reduction curves and compare the two algorithms.

Figure **3**, Figure 4, and Figure 5 show the proportion of displayed data elements in the aggregated results as compared to the original Snort alerts, DNS requests, and flow files, respectively. In all cases with the exception of flow data, note that when the values are distinguishable, the hypergraph-based algorithm displays fewer elements than the column-based approach, even though the hypergraph approach is thresholding the total number of permitted hyperedges. In the case of flow data, the column-based algorithm outperforms the hypergraph approach for several values of Hamming distance; this is attributable to the thresholding, and the advantage to the column-based algorithm vanishes if the threshold is expanded to 500 (not shown). It is also worth noting that for the flow data, the total aggregation time required at Hamming distance 4 (the optimal distance for the column-based algorithm) for the column-based algorithm is 123.6s, versus 60.7s for the hypergraph approach, using a batch size of 10000. Due to the poor scaling of the column-based approach (discussed in more detail below), this further limits its usefulness in this case.

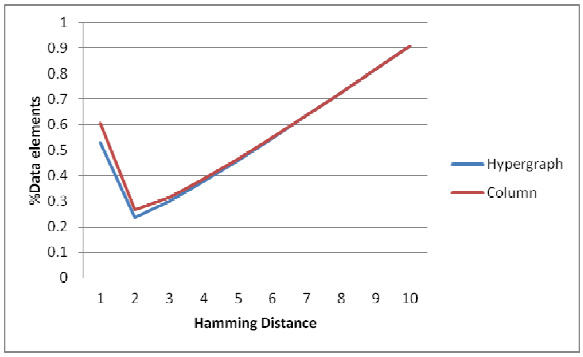

Figure 3. Data element reduction for Snort alerts using a batch size of 10000. Hypergraph aggregation thresholded at 250 hyperedges per node.

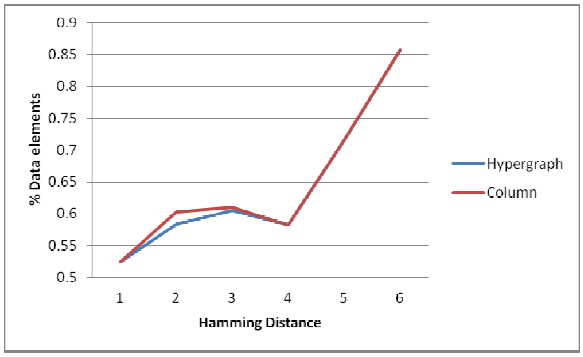

Figure 4. Data element reduction for DNS requests using a batch size of 10000. Hypergraph aggregation thresholded at 250 hyperedges per node.

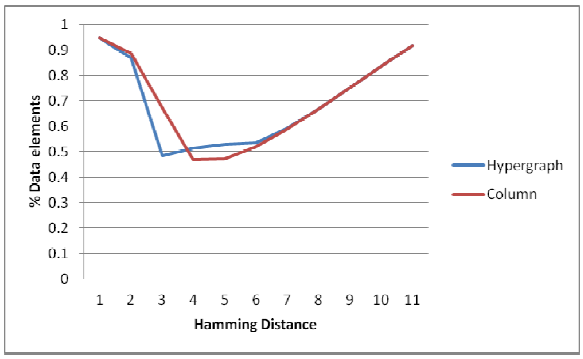

Figure 5. Data element reduction for flow data using a batch size of 10000. Hypergraph aggregation thresholded at 250 hyperedges per node.

7.3 Reduction in Number of Meta Alerts Through Increasing Batch Sizes

The reduction in number of meta-alerts can also be influenced by the number of alerts available for aggregation. As additional alerts are added to the set available for aggregation, the likelihood that two alerts will fall within the minimum required Hamming distance of each other increases, and so the average number of meta-alerts may be expected to approach some limit determined by the inherent redundancy of the data. Figure 6 displays this effect for the DNS, Snort alert, and Netflow logs, each aggregated under their respective optimal Hamming distances, using a threshold of 250; comparisons with the column-based algorithm are not provided due to prohibitive running times. Note that at a batch size of 1000 alerts the algorithm is still capable of substantial

reduction of alerts; however as more alerts are added to the collection the number of meta-alerts remaining relative to the original number of alerts decreases sharply for both the Snort alert and the Netflow data sets, and a slight but measurable amount for the DNS request record data set.

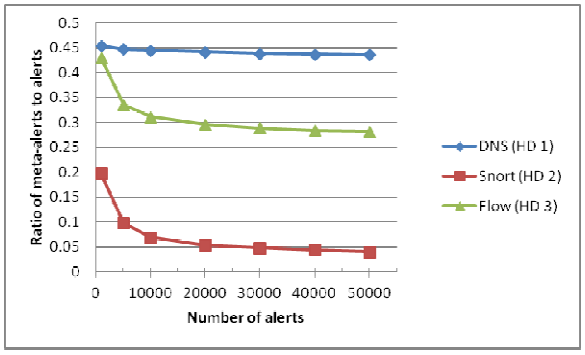

Figure 6. Ratio of meta-alerts to alerts for DNS, Snort, and Flow data types at optimal Hamming distances.

Thus, following the selection of an optimal Hamming distance, the overall aggregation performance of the algorithm may be further improved by increasing the total number of alerts provided to it. While there is little reason to suspect that the column-based algorithm would not see similar improvements with respect to aggregation effectiveness (and limited experimentation – not shown – confirms that at the batch sizes tested the same monotonic effect appears to hold), the high time complexity of that algorithm generally rapidly reduces the utility of this approach. The improved time complexity of the hypergraph algorithm allows much greater flexibility in the selection of the batch size to aggregate, thus enabling further gains in aggregation.

7.4 Reduction in Number of Data Elements Through Increasing Batch Sizes

While selecting the correct hamming distance can allow for optimization of the number of data elements presented to an analyst, the total number of alerts that are presented to the algorithm for aggregation can also have a significant impact. Figure 7 shows the impact of batch size on the number of data elements for comparatively small batch sizes across three data types, while Figure 8 shows the impact for a larger range of batch sizes for Snort alerts.

In minimizing the number of data elements presented to an analyst, our aggregation system should use as large a batch size as is feasible given time and memory constraints at the optimal Hamming distance for the data type in question. The optimal Hamming distance does not appear empirically to change with variations in batch size, and so may be rapidly selected via a test batch.

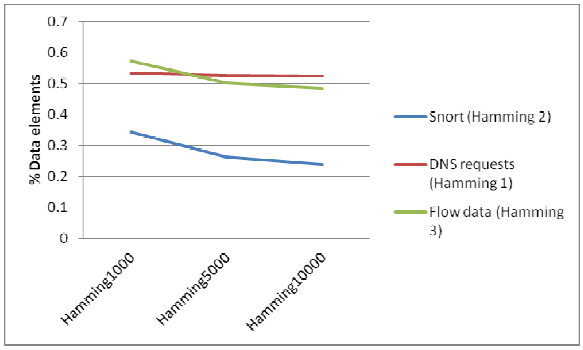

Figure 7. Data Element Reduction as Batch Size Increases

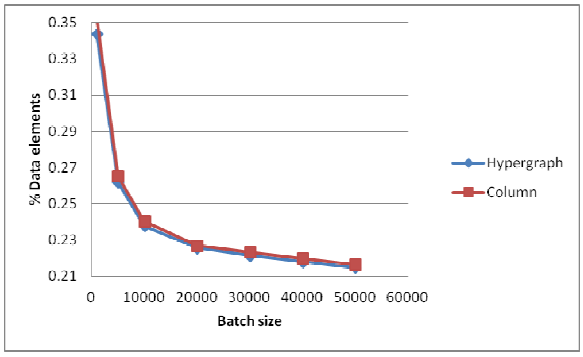

Figure 8. Data Element Reduction for Snort alerts (Hamming distance 2) with Larger Batch Sizes; Hypergraph Algorithm Thresholded at 250)

7.5 Execution Time Comparison

We compared the execution time of the column-based approach of [6] against our hypergraph algorithm using all four data sets. We focused on both varying the number of alerts processed as well as the Hamming distance used. In

Figure **9**, we show the execution time of both algorithms on an increasing number of Snort alerts. We chose a Hamming distance of 2 because that was empirically discovered to optimize the number of data fields presented to analysis (discussed earlier). The column-based approach takes increasingly more time, as was expected given its $O(n^2)$ complexity, compared to the $O(n\log n)$ hypergraph approach.

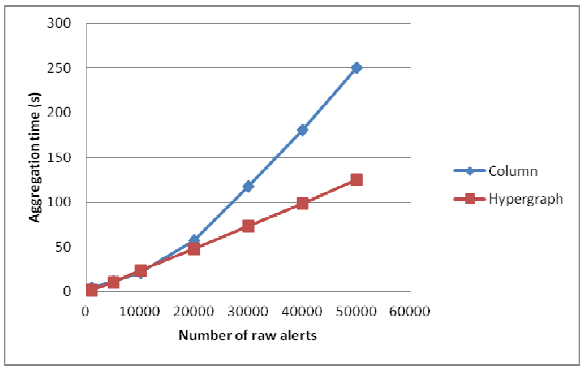

Figure 9. Aggregation of Snort Alerts at Hamming distance 2

The difference between the algorithms is even more pronounced at Hamming distance 1. Revisiting the analysis of [6], at Hamming distance of 1 (not shown) the column-based algorithm is over an order of magnitude slower for daily alert bins (containing a mean of 84023 alerts).

In Figure 10 we explore this variance in execution time relative to the Hamming distance using the EVTX data. The column-based approach performs extremely well at higher Hamming distances but the execution time increases dramatically at low Hamming distances. It fails even to complete at Hamming distances less than 3 due to the EVTX data having 24 columns, the largest among our data sets. The column-based approach does complete at all Hamming distances for the other data sets, but can be over an order of magnitude slower. The hypergraph algorithm performs very fast at low Hamming distances but slows down in the mid-ranges, preventing its completion for some data sets due to the combinatorial term in the complexity analysis unless thresholding is used. Using thresholding, the hypergraph algorithm can execute fast for all data sets and Ham-

ming distances. Note how thresholding causes spikes in the execution time as the combinatorial term is decreased to values below the threshold. At the minimum threshold value of 1, the execution time becomes so small as to overlap the x-axis in Figure 10.

The lower Hamming distances are ideal candidates for operational use. At these distances, the alerts are abstracted the least and thus directly provide details to aid human analysis. Given the inability of the column approach to process certain scenarios (especially those most likely to be operationally useful), we claim that the hypergraph algorithm is necessary for variable Hamming distance alert aggregation and not just an incremental improvement over our expanded version of the column based approach.

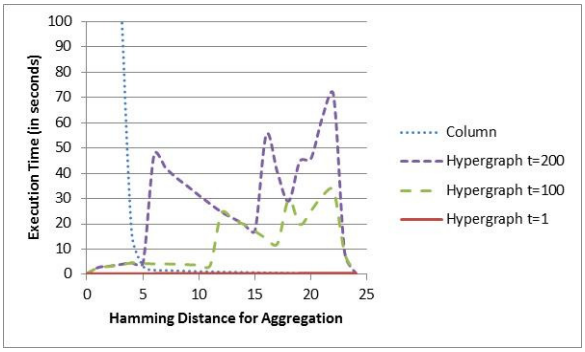

Figure 10. Mean Execution Time to Aggregate EVTX Alerts

8. INTUITIVE UNDERSTANDING AND OPERATIONAL USE

While explained in [6], we review here the intuition behind the usefulness of the Hamming Distance aggregation approach in reducing the cognitive burden to the analysts. We also discuss our experience with the approach and qualitative results from operational use.

In enterprises with well-funded security teams, there may exist a requirement to review every alert produced from a certain set of sensors. The example set of operational alerts from [6] measured 2.3 million Snort alerts a month. For each alert, an analyst would have to review the corresponding set of values in the data fields. To evaluate the next alert, the analyst would have to repeat this procedure. The alerts are thus evaluated independently and there is a mental context switch between alerts. Related alerts are often scattered throughout the data set, requiring explicit searching by the analyst to find them.

Hamming distance aggregation groups together related alerts into meta-alerts (it doesn't delete any alerts or data). A grouping of alerts share many data fields in common. An analyst only has to read and absorb each common data field once for the entire group. In analyzing each individual alert, only the unique field values need to be perused. In some cases, the analyst can discount an entire group based on the common values and avoid even read the unique values. This can happen, for example, in a horizontal attack scenario where the unique values are the individual addresses attacked. Based on the common values, the analysts can evaluate the group of alerts. Without Hamming Distance aggregation, such related alerts would be scattered and hidden among many other alerts.

Analysts often find that Hamming Distance aggregation creates large groupings out of the many 'trash' alerts that can be discarded after an evaluation of the common fields. Related attacks may end up in small groups. Unusual attacks tend not to be grouped and stand out to analysts because of the small group size. This said, Hamming Distance aggregation does not guarantee to put

'trash' in large groups and interesting alerts in small groups. A more significant benefit is in bringing together duplicate but scattered data fields values so that each value can be read only once by the analyst (as opposed to being re-read every time an instance of the duplicated value is encountered). Such benefits enhance the ability of large security teams to review, for example, the 2.3 million alerts per month cited in [6].

To validate our results empirically (though qualitatively), our Hamming Distance aggregation code was provided to operational analysts who used it on actual security logs taken from a large enterprise network. Their feedback supports the usefulness of the approach through informal evaluation. Qualitative feedback indicates that Hamming distance based alert reduction at low Hamming distances both reduces analysis time and enhances interpretation of the alerts. Quantification of these observations require a formal human study, were outside the scope of our experiments, and so must be addressed in future work.

9. CONCLUSIONS

Variable Hamming Distance alert aggregation can reduce the cognitive load on the analysts with respect to minimizing the number of alerts and data elements. Our aggregation algorithms enable efficient human review of large sets of original alerts without removing or abstracting away any data. While the algorithm in [6] successfully reduced the number of data elements, it only considered a Hamming distance of 1, limiting its ability to effectively aggregate some data without manual adjustments. In addition, its limited scaling with respect to the number of data points as well as the dimensionality of that data restrict its applicability to large data sets, which we demonstrate further reduces its ability to effectively aggregate them. We present an algorithm with improved worst-case time complexity capable of handling arbitrary Hamming distance aggregation, as well as an approximate implementation that can significantly reduce the constant terms in the time and memory complexity to no greater than a specified maximum value. We demonstrate that even the approximate version of the algorithm has performance with respect to aggregation at least equal to that of the original algorithm in [6] while improving time requirements for optimal hamming distances, particularly when using large batch sizes. This improvement in time complexity allows for aggregation of larger batches of data, thus further improving the effectiveness of aggregation in practice. Our algorithm has further benefits in its ability to handle on-line or streaming data, which we propose to examine further in future work. Future work will also examine the relationship between processing time, assessment accuracy, and perceived effort on the part of analysts under different levels of alert aggregation and data element reduction.

10. ACKNOWLEDGMENTS

This research was sponsored by the U.S. National Institute of Standards and Technology and the Army Research Labs, and was partially accomplished under Army Contract Number W911QX-07-F-0023. The views and conclusions contained in this document are those of the authors, and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes, notwithstanding any copyright notation hereon.

11. REFERENCES

[1] H. Farhadi, M. AmirHaeri and a. M. Khansari, "Alert Correlation and Prediction Using Data Mining andHMM," The ISC International Journal of Information Security, pp. 1-25, 2011.

[2] H. Debar and A. Wespi, "Aggregation and Correlation of Intrusion-Detection Alerts," in Recent Advances in Intrusion Detection, Springer, 2001, pp. 85--103.

[3] F. Cuppens, "Managing alerts in a multi-intrusion detection environment," in Proceedings of the 17th Annual Computer Security Applications Conference, 2001.

[4] J. J. Treinen and R. Thurimella, "A framework for the application of association rule mining in large intrusion detection," in Recent Advances in Intrusion Detection, Berlin Heidelber, Springer-Verlag, 2006, pp. 1-18.
[5] M. Roesch, "Snort -- lightweight intrusion detection for networks," Proceedings of the 13th USENIX conference on System administration, pp. 229--238, 1999.
[6] R. E. Harang and P. Guarino, "Clustering of Snort Alerts to Identify Patterns and Reduce Analyst Workload," MILCOM proceedings, 2012.
[7] P. Mell, "Hyperagg: A Python Program for Efficient Alert Aggregation," 2013. [Online]. Available: http://csrc.nist.gov/researchcode/hyperagg-mell-20131227.zip
[8] J. Zhou, M. Heckman and B. C. A. B. M. Reynolds, "Modeling network intrusion detection alerts for correlation," ACM Transactions on Information and System Security, vol. 10, no. 1, 2007.
[9] A. Siraj and R. B. Vaughn, "Alert Correlation with Abstract Incident Modeling in a Multi-Sensor Environment," International Journal of Computer Science and Network Security, pp. 8-19, 2007.
[10] A. Siraj and R. Vaughn, "A cognitive model for alert correlation in a distributed environment," Intelligence and Security Informatics, pp. 1017--1028, 2005.
[11] B. Morin, L. Me, H. Debar and M. Ducasse, "M2D2: A formal data model for IDS alert correlation," Proceedings of the 5th international conference on Recent advances in intrusion detection, pp. 115--137, 2002.
[12] P. Ning, D. Xu, C. G. Healey and R. S. Amant, "Building Attack Scenarios through Integration of Complementary Alert," Proceedings of the 11th Annual Network and Distributed System Security Symposium, 2004.
[13] T. Chyssler, S. Burschka, M. Semling, T. Lingvall and K. Burbeck, "Alarm Reduction and Correlation in Intrusion Detection Systems," in DIMVA, Dortmund, Germany, 2004.
[14] B. Morel, "Anomaly Based Intrusion Detection and Artificial Intelligence," in Intrusion Detection Systems.
[15] A. Hofmann and B. Sick, "Online Intrusion Alert Aggregation with Generative Data Stream Modeling," IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, pp. 282-294, 2011.
[16] L. Wang, A. Liu and S. Jajodia, "An Efficient and Unified Approach to Correlating,Hypothesizing, and Predicting Intrusion Alerts," Computer Security--ESORICS 2005, pp. 247-266, 2005.
[17] H. Gabra, A. Bahaa-Eldin and H. Mohamed, "Data Mining Based Technique for IDS Alerts Classification," arXiv preprint arXiv:1211.1158, 2012.
[18] Z. a. X. X. a. H. Z. a. D. S. He, "Fp-outlier: frequent pattern based outlier detection," Computer Science and Information Systems/ComSIS, vol. 2, no. 1, pp. 103--118, 2005.
[19] T. Cormen, C. Leiserson and R. Rivest, in Introduction to Algorithms, The MIT Press, McGraw-Hill Book Company, 1994.
[20] V. Chvatal, "A Greedy Heuristic for the Set-Covering Problem," Mathematics of Operations Research, vol. 4, no. 3, pp. 233-235, 1979.
[21] P. Slavik, "A tight analysis of the greedy algorithm for set cover," in STOC '96 Proceedings of the twenty-eighth annual ACM symposium on Theory of computing, 1996.