November 20, 2018

STABLE COCLUSTERING BY ITERATIVE DENSITY-BASED SPATIAL CLUSTERING OF APPLICATIONS WITH NOISE

Debasish Das

Antonio Nucci

Jaykishan Pandya

STABLE COCLUSTERING BY ITERATIVE DENSITY-BASED SPATIAL CLUSTERING OF APPLICATIONS WITH NOISE

AUTHORS:
Debasish Das
Antonio Nucci
Jaykishan Pandya

ABSTRACT

Techniques are described herein for providing a coclustering algorithm that iteratively applies Density-Based Spatial Clustering of Applications with Noise (DBSCAN) clustering with Jaccard distance to discover clusters of entities along with corresponding clusters of features. The algorithm provides a stable alternative to the existing coclustering algorithms that can discover distinct coclusters of different compactness beyond a threshold that can be controlled by the user. The algorithm may be used to discover patterns of syslog messages predictive of certain network device failures and simultaneously cluster the devices that encounter each of these patterns.

DETAILED DESCRIPTION

There exists a problem of clustering syslog messages relevant to predicting a specific network failure event encountered by a number of network devices (e.g., routers) while simultaneously clustering the devices. Since different devices can encounter the failure event for a number of different underlying conditions (e.g., device hardware, software release version, configuration, etc.), and many of these conditions can manifest themselves through a different sequence of syslogs, a stable and robust coclustering algorithm is needed to identify each of these distinct sequences along with the similar sets of devices that encounter those. However, the algorithm described herein is very generic and can be applied to a variety of coclustering problems.

Given a set of entities represented by a set of feature vectors and a distance metric defined in the space of the feature vectors, a clustering algorithm groups the entities in such a way that distance between within group entities are minimized and distance between out of group entities are maximized. On the other hand, in co-clustering, not only the entities are clustered, but features are clustered simultaneously with entities and corresponding to each group of entities so as to generate a group of features. For the algorithm presented

Published by Technical Disclosure Commons, 2018

herein, only binary features are considered where each feature represents presence or absence of an element in the feature vector. For example, in one motivating problem, a device represents an entity, and the presence or absence of a syslog message template represents a binary feature. Thus, when the devices are co-clustered along with the message templates, for each cluster of devices, a corresponding cluster of message templates is obtained. The device clusters are mutually exclusive while corresponding template clusters have no such restrictions.

An algorithm referred to herein as Stable Coclustering by Iterative Density-Based Spatial Clustering of Applications with Noise (DBSCAN), or SCID, is developed by iteratively applying DBSCAN clustering with Jaccard distance while in each successive iteration, devices clustered in the previous iteration are removed and compactness criteria for clustering is relaxed. The iterative process is stopped either when all devices are clustered or when compactness criteria can no longer be relaxed. A cluster of message templates corresponding to each device cluster is defined as the common templates present among the devices in the device cluster. If there are no common templates, the corresponding device cluster is ignored in that iteration.

The iterative application of clustering for coclustering with gradual relaxation of compactness criteria guarantees discovery of both highly compact and less compact but useful clusters. Most other co-clustering algorithms tend to merge highly compact clusters with less compact clusters. Merging of nearby clusters may be appropriate for some applications, but for applications where strong predictive patterns are sought, the algorithm may separate clusters with varying compactness. Use of DBSCAN as the core clustering algorithms also helps, as being a density-based clustering algorithm, it can be used to detect clusters of varying density simply by controlling one of its parameters (*eps*).

The algorithm described herein does not depend on the initialization and therefore is very stable. Unlike some of the freely available libraries for co-clustering, this algorithm always produces the same result if the algorithm is run with same dataset and parameters multiple times.

The use of Jaccard distance is especially useful to detect longer patterns that are less likely to be spurious. Overall, the method described herein is more suitable for applications where the purpose of co-clustering is to identify patterns of behavior. In these

applications, longer patterns tend to make more sense as valid indicators of some underlying behavior. One other class of problems where this approach may be applicable is user behavior analysis where users are co-clustered with features that define their behavior. Some examples of features include websites browsed, items browsed for e-commerce sites, applications browsed in an operating system, applications used in a cellphone, etc.

This algorithm may be adopted to other applications that require distance metrics other than Jaccard distance since the distance metric is a pluggable component to the algorithm.

There are several key concepts and algorithms which are used in the co-clustering algorithm described herein. A distance metric or distance function on a set *X* is defined as a mapping (or function) $d: X \times X \rightarrow [0, \propto)$ where $[0, \propto)$ is the set of non-negative real numbers and for all $x, y, z \in X$, the following conditions are satisfied:

1. $d(x, y) \geq 0$ (non-negativity)
2. $d(x, y) = 0 \Leftrightarrow x = y$
3. $d(x, y) = d(y, x)$ (Symmetry)
4. $d(x, y) + d(y, z) \geq d(z, x)$ (Triangle inequality)

Jaccard distance is a special distance or dissimilarity metric between two sets that is defined as the complement of their Jaccard index. Given two sets A and B, the Jaccard index is defined as

$$J_C(A, B) = \frac{|A \cap B|}{|A \cup B|}$$ if at least one of the sets is non-empty.

If both sets are empty, then the Jaccard index is defined as $J_C(A, B) = 1$. The Jaccard index is always a number between 0 and 1, and therefore Jaccard distance $J_D(A, B)$, which is defined as $J_D(A, B) = 1 - J_C(A, B)$, is also a number between 0 and 1. The Jaccard distance focuses on finding the number of matching components between two sets and therefore defines the dissimilarity or distance between two sets as a complement of similarity between the sets as opposed to most other distance metrics that focus exclusively on the differences between two sets (or feature vectors) to compute their distance.

DBSCAN is a density-based clustering algorithm that put the data-points (sets or feature vectors) with many nearby neighbors into the same cluster and tags data-points lying in the low-density regions as outliers. The algorithm takes two inputs, a distance

threshold (*eps*) and a minimum number of points to be considered a cluster (*minPts*). Two data-points are considered neighbors if their distance is less than *eps*. DBSCAN is useful as the base method for the co-clustering algorithm described herein due to the following properties.

First, unlike most other clustering algorithms, DBSCAN does not need number of clusters as an input. Both its inputs *eps* and *minPts* are directly tied to the domain of the problems and in many cases, users will have a fairly good estimate of these parameters. For example, if the Jaccard distance is used as the distance metric and data-points are considered sets of features, an *eps* of 0.1 means two data-points should have a Jaccard index greater than 0.9 to be considered neighbors, which in turn indicates 90% of the members between two data-points have to match compared to the union of members of both data-points. For a user with a good understanding of the data, this is an easier parameter to provide compared to the number of clusters.

Second, since *eps* can be used to control the density of the clusters, this input may be gradually increased for successive DBSCAN runs to directly relax the compactness criteria of the clustering which is a requirement for the co-clustering algorithm described herein. Third, unlike a number of other clustering algorithms, DBSCAN can handle clusters of varying sizes and shapes. Fourth, DBSCAN is very stable algorithm and produces the same clusters every time it is run on the same dataset with the same inputs.

Assume that $E = \{E_1, E_2, \dots, E_N\}$ is a set of entities where entity E_i is represented by a set of features $S_i = \{F_k;\ F_k \in \Phi\}$ where Φ is the universe of all features. Moreover, $|S_i| = n_i$, which may vary by entity. Since S_i is a set, the order of features is not important. The SCID algorithm have at least two objectives. The first objective is to split E into p mutually exclusive clusters $C_1, C_2, \dots C_p$ where $C_i = \{E_j; E_j \in E, E_j \notin C_k \forall k \neq i\}$. Here, p must be determined by the algorithm. p can be 1 (i.e., only one cluster) or even 0 (no meaningful clusters). The second objective is to, for each cluster of entities C_i, find the corresponding cluster of features $\mathbb{C}_i = \{F_j; F_j \in \Phi\}$ from the union of feature sets corresponding to each member entity of C_i. If no such cluster $\mathbb{C}_i$ is found, C_i should be ignored.

The first objective may be achieved by clustering the entities based on their feature sets using an iterative DBSCAN algorithm where the *eps* is increased gradually at each iteration. Increasing the *eps* implies relaxing the compactness of the clusters, i.e., increasing the distance between entities grouped together in the same cluster. As mentioned, the distance metric used with DBSCAN is the Jaccard distance. The reason behind adopting an iterative approach instead of a single pass clustering is that a single-pass clustering with a fixed *eps* may fail to discover a smaller but more compacted clusters of feature sets which may be assimilated by larger, more dispersed clusters if the chosen *eps* is too high. Conversely, if the value of *eps* is set too low, this may force the discovery of a larger set of small cardinality clusters losing the associations among distinct clusters which may still somehow be associated at a broader scale. The iterative approach of SCID starts with a low value of *eps*, which groups entities with very strong similarity of feature sets, and gradually increase *eps*, which step-by-step discovers other entities which have a looser similarity between them.

One reasons for using Jaccard distance is that Jaccard distance is defined on sets, not just on ordered vectors. Another reason is that Jaccard distance is defined as a complement to the Jaccard index, which measures the similarity of the sets. Therefore, Jaccard distance first measures the similarity between two entities and then finds the dissimilarity or distance as a complement. However, most other distance measures focus exclusively on dissimilarities to estimate the distance. Consider the following two pairs of sets:

Pair 1: $S_1 = \{F_1, F_2, F_3, F_4\};\ \ S_2 = \{F_1, F_2, F_3, F_4, F_5\}$.

Pair 2: $S_3 = \{F_1\};\ \ S_2 = \{F_1, F_2\}$.

In this example, the first pair should be closer to each other as they exhibit a longer matching pattern (and therefore less probable to be spurious) than the second pair. the Jaccard distance supports that intuition as $J_D(S_1, S_2) = 1/5$ and $J_D(S_3, S_4) = 1/2$. However, with the Manhattan distance metric, the distance between both the pairs is 1. Their distance will be same for most other dissimilarity-based distance metrics which is counter-intuitive for these purposes.

Whenever a new cluster C_i is discovered in each iteration of the algorithm, the second objective of finding the corresponding feature cluster $\mathbb{C}_i$ is achieved by identifying the common features in the features sets corresponding to the entities in C_i, i.e. $\mathbb{C}_i = \{F_k; F_k \in S_j \forall E_j \in C_i\}$.

The iterative process stops either when all entities are clustered or when the *eps* has reached maximum allowable value. Figure 1 below illustrates the complete iterative algorithm.

Input: Entities $E = \{E_1, E_2, \dots, E_N\}$ and corresponding feature sets $S_i = \{F_k;\ F_k \in \Phi\}$
Initialize: *eps* = 0.1; *clusteredEntities* = {}; *clusters* = {}; *iteration* = 1; minPts = 2

While (*eps* < 0.9) and (|*clusteredEntities*| < |*E*|) do
    remainingEntities := *E* - *clusteredEntities*
    $clusters_{iter}$ = DBSCAN(*remainingEntities, eps, minPts, JaccardDistance)*
    for clust in $clusters_{iter}$
        $S^{clust} = \{S_k; E_k \in clust\}$
        $F^{clust} = commonFeatures(S^{clust})$
        if $|F^{clust}| > 0$
            clusters := *clusters* + $\left\{\{E_k; E_k \in clust\}, F^{clust}\right\}$
            clusteredEntities := *clusteredEntities* + $\{E_k; E_k \in clust\}$

function $commonFeatures(S)$
    return $\{F_k; F_k \in S_j\ \forall S_j \in S\}$

Figure 1: Pseudocode for SCID algorithm

A simple example is provided to explain the difference between co-clustering and clustering. Assume seven entities in the set $E = \{a, b, c, d, e, f, g\}$ and their corresponding feature sets are given by

$S = \{\{1,4,7,8,9\}, \{1,2,4,7,8,9\}, \{5,1\}, \{6,9,10,3\}, \{4,6,9,10,3,7\}, \{4,6,9,3\}, \{2,10\}\}$, where the universe of all features is given by $\Phi = \{1,2,3,4,5,6,7,8,9,10\}$. To apply a regular clustering algorithm like K-means, the feature sets must be converted into binary vectors, which will look like following:

S' = [[1,0,0,1,0,0,1,1,1,0],
[1,1,0,1,0,0,1,1,1,0],
[1,0,0,0,1,0,0,0,0,0],

[0,0,1,0,0,1,0,0,1,1],

[0,0,1,1,0,1,1,0,1,1],

[0,0,1,1,0,1,0,0,1,0],

[0,1,0,0,0,0,0,0,0,1]],

where the columns indicate features and the rows indicate the corresponding feature set of any given entity. So, the presence of a "1" at location (3,5) in the matrix means feature 5 is present in the feature set corresponding to the third entity. Similarly, a "0" at location (1,3) means feature 3 is not present in the feature set corresponding to first entity. Two feature sets (for entities *c* and *g*) are totally different from the remaining feature sets. This distinguishes the performance of regular clustering from the co-clustering algorithm described herein. Also, two groups of feature sets – (a,b) and (d,e,f) have significantly similar feature set.

When K-means is applied on S' with two clusters (for K-means, number of clusters needs to be specified), the following clusters are obtained: {*a, b, g*} and (*d, e, f*). Clearly, K-means clusters all entities in one of the cluster even though some their feature sets are completely different from each other.

On the other hand, the co-clustering algorithm described herein generates the following coclusters: {(*a, b*), (*1, 8, 9, 4, 7*)} and {(*d, e, f*), (*9, 3, 7, 6*)}. The first group in each cluster is the group of entities in that cluster and the second group is the corresponding group of features. The step-by-step iterative discovery of the clusters is shown below.

For *eps* = 0.1 => No co-clusters found.

For *eps* = 0.2 => One co-cluster found {(*a, b*), (*1, 8, 9, 4, 7*)}, i.e., the Jaccard distance between feature sets for *a* and *b* is less than or equal to 0.2. Entities *a* and *b* are removed from the entity list for subsequent iterations.

For *eps* = 0.3 => No co-clusters found.

For *eps* = 0.4 => One co-cluster found {(*d, e, f*), (*9, 3, 7, 6*)}, i.e., the pairwise Jaccard distance between feature sets for *d, e* and *f* is less or equal to 0.4. Entities *d, e* and *f* are removed from the entity list for subsequent iterations.

For *eps* = 0.5 => No co-clusters found.

For *eps* = 0.6 => No co-clusters found.

For *eps* = 0.7 => No co-clusters found.

For *eps* = 0.8 => No co-clusters found.

For *eps* = 0.9 => No co-clusters found.

Therefore, the method described herein not only correctly identifies the clusters, but also generates co-clusters of features corresponding to each cluster of entities.

This algorithm is motivated by the need to simultaneously group network devices that encounter a specific type of failure and message template that are considered precursor to the actual failure. It is possible that the same failure may occur for different underlying conditions for different network equipment and therefore a different set of syslog message templates may indicate each of these different underlying conditions that leads to the same failure. Therein lies the need for co-clustering the devices and message templates. Each of the resulting clusters may indicate a different underlying cause of the same failure.

In one example of an application identifying predictive syslog templates that precede certain types of network equipment outage, the co-clustering algorithm is a component of a larger system of identifying patterns of syslog templates indicative of a network equipment failure. Figure 2 below illustrates the components relevant to co-clustering blocks.

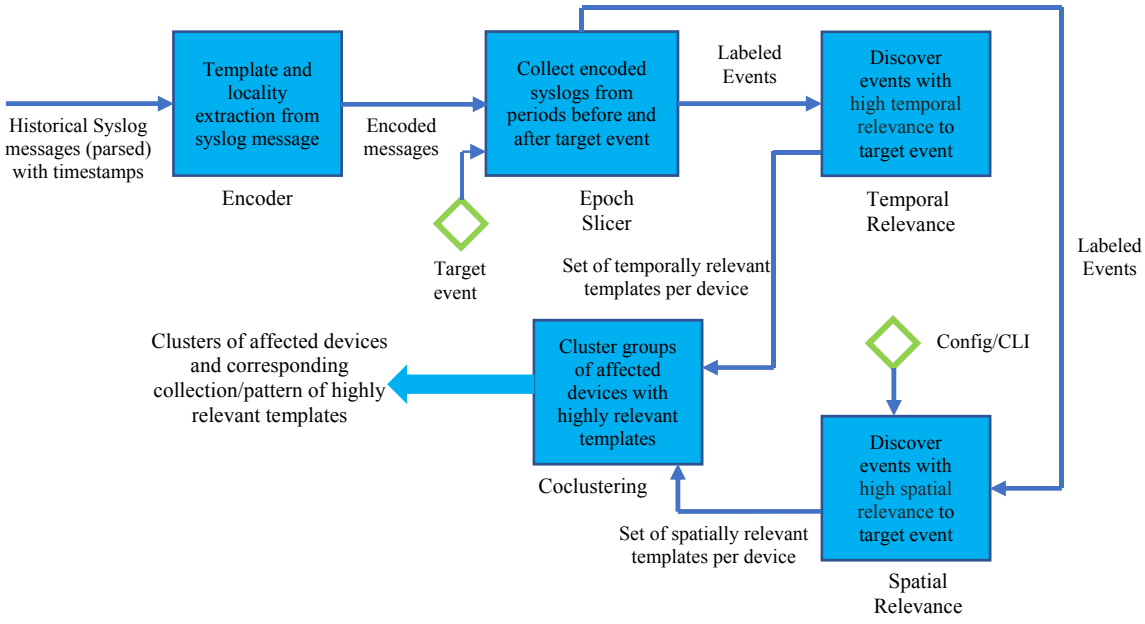

Figure 2: System components relevant to the coclustering block.

The encoder module collects the historical syslogs for all available network devices along with their timestamps and splits the messages in invariant and variant parts. The invariant part is called message template. The variant part in many cases carries useful information about the location of the syslog message. An example of a syslog message along with the template and variant is listed below:

Syslog message: OIR-SP-3-PWRCYCLE: Card in module 5, is being power-cycled 'Slot disabled'.

Template: OIR-SP-3-PWRCYCLE: Card in module %d, is being power-cycled 'Slot disabled'

Variant: 5

The variant information indicates the message relates to module 5.

The epoch slicer module takes a target event – a message template that indicates a failure – as an input, finds the devices that encountered the target event (affected devices) and collects syslogs from a period (observation period) immediately before the device enters the failure phase and also from a period (stability period) when the device is out of the failure phase.

The temporal and spatial relevance modules collect templates from the observation period that are temporally and spatially the most relevant to the target event. So, the output of this stage is a list of affected devices $D = \{D_1, D_2, \dots, D_N\}$ and a set of relevant templates per device $S = \{S_1, S_2, \dots, S_N\}$ where $S_i = \{T_k; T_k \in \mathrm{T}\}$ with T being the set of all possible templates.

The co-clustering module uses this algorithm to co-cluster the sets D and S to finally generate groups of affected devices and corresponding highly relevant groups of templates or patterns. The following is an actual example of devices (identified by their Internet Protocol (IP) addresses) and the relevant templates identified by the temporal and spatial relevance module for each device. The result of the co-clustering on these sets is also shown. In the following example, the actual templates are replaced by their corresponding machine generated Identifiers (IDs) for the sake of space.

Target Event: OS-DUMPER-4-SIGSEGV

Set of Affected Devices: {ip1, ip2, ip3, ip4, ip5, ip6, ip7, ip8, ip9} (Actual IP addresses were suppressed for privacy reasons.)

Set of relevant templates per device:

ip1 – {11161100, 13817100, 12553101, 10073101, 11742100, 13358100, 13278101, 13278100, 13278103, 12179100, 11715105, 11715109, 15192105, 12125100, 10818100, 14178100, 10215100, 14631101, 10535101, 15543103, 14631104, 15116100, 11164100, 14476102, 14748106, 11025100, 11286100, 15414101, 11446102, 11600101, 12080102, 11600102, 12080104, 13226100, 13226101, 13226102, 13226103, 15578101, 13226107, 11023100, 15517105, 14563100, 14563101}

ip3 – {15460100, 15572100, 15192104, 15543114, 15485100, 11161100, 13817100, 10482100, 11838101, 11838102, 11742100, 12718105, 14631100, 15543101, 10218116, 12937100, 13358100, 13278101, 14563100, 14563101, 12125100, 15231100, 15517105, 14242100, 14567100, 15116100, 13596100, 15414101, 11606105, 10432101, 12080102, 13226101, 13226102, 13226107, 11023100, 15231102, 15231103}

ip4 – {10003100, 14916100, 10480100, 12080102, 14968100, 12080104, 15192105, 14968101, 10903102, 11528100, 10903112, 11838100, 11838101, 11130100, 15463100, 15723102, 15231104, 10821122, 15407106, 10218114, 13588101, 10564103, 10218120, 15407113, 10218123, 13625100, 15407123, 15470100, 10766100, 12718102, 12553113, 14563100, 14563101, 11503100, 10821151, 15407136, 10104100, 13624100, 11503102, 14482104, 14631100, 14631101, 12599100, 13815106, 13815108, 13815110, 10481100, 12145100, 10481102, 10481103, 10481101, 10481105, 10481107, 10358100, 13590100, 14630100, 10481112, 10481113, 10481114, 10481116, 10481118, 10481119, 13035109, 10481127, 10481128, 10481130, 12981101, 10481134, 10481135, 10821109, 10218105, 13231100, 15231102, 13759103}

ip6 – {10564100, 13588101, 13588102, 10564102, 10397100, 12130101, 15407169, 10641101, 10481105, 10358100, 14630100, 13590100, 10481123, 12981101, 10481137, 10481140, 11226104, 10821117, 13625100}

ip7 – {15192104, 11742100, 10903108}

ip2 – {11715105, 12080102, 15192104, 11838101, 11023100, 10432101, 13596100, 12937100, 13224101, 11742100, 13226102, 13226103, 13226101, 15414101, 13226107,

14567100, 14563100, 15517105, 13226100, 13278101, 11606105, 15231100, 14563101, 15231102, 15231103}

ip8 – {11742100, 10903108}

ip5 – {10564100, 13588101, 13588102, 10564102, 13625100, 12654101, 11838101, 11838102, 10702100, 12451100, 11715101, 11715100, 10218145, 13624100, 10397100, 10818100, 15463100, 14631100, 15407169, 14631108, 10641101, 10481105, 10481107, 13062100, 10358100, 13590100, 14630100, 10481113, 14603100, 15195100, 11867102, 13659102, 10481123, 12080102, 10481126, 12080104, 12981101, 13659118, 10481136, 10481137, 10481140, 11226104, 10821117, 11503102}

ip9 – {10903108}

Result of coclustering for cluster 1:

Devices: {ip1, ip2, ip3}

Templates (pattern): {14563100, 12080102, 15517105, 11742100, 13278101, 15414101, 13226101, 13226102, 13226107, 11023100, 14563101}

Template Dictionary:

14563100 OS-SYSMGR-3-ERROR %AlphaNum.. (fail count %XXX will be respawned in %XXX seconds

12080102 L2-L2VPN_PW-3-UPDOWN %AlphaNum.. with address %IPV4 id %XXX state is Down

15517105 FORWARDING-IP_TUNNEL-4-EA_INIT %AlphaNum.. EA process failed to initialize platform DLL: "prm_server' detected the 'warning' condition 'Invalid data found."

11742100 IP-DHCPD-3-NOPACKET %AlphaNum.. setup or duplicate a DHCPD server socket packet

13278101 OS-RSI_AGENT-6-CARD_ROLE_CHANGE %AlphaNum.. on the card configuration/type the AFI %AlphaNum.. role of the card has changed from Invalid to Not Interested

15414101 IP-CE_TFTP-6-KERNEL_DUMP_MSG %AlphaNum.. writing to filename: %AlphaNum..
13226101 PLATFORM-SHELFMGR-6-NODE_STATE_CHANGE %AlphaNum.. %AlphaNum.. state:IOS XR FAILURE
13226102 PLATFORM-SHELFMGR-6-NODE_STATE_CHANGE %AlphaNum.. %AlphaNum.. state:ROMMON
13226107 PLATFORM-SHELFMGR-6-NODE_STATE_CHANGE %AlphaNum.. %AlphaNum.. state:MBI-BOOTING
11023100 OS-SYSMGR-5-NOTICE %AlphaNum.. is COLD started
14563101 OS-SYSMGR-3-ERROR %AlphaNum.. (jid %XXX exited will be respawned with a delay (slow-restart)

Result of coclustering for cluster 2:
Devices: {ip4, ip5, ip6}

Templates (pattern): {13588101, 13625100, 12981101, 10481105, 13590100, 10358100, 14630100}

Template Dictionary:
13588101 OS-DUMPER-7-DLL_INFO %AlphaNum.. %MEM_ADDRESS %MEM_ADDRESS %MEM_ADDRESS %MEM_ADDRESS %XXX
13625100 OS-DUMPER-7-DLL_INFO_HEAD%AlphaNum.. path Text addr. Text size Data addr. Data size Version
12981101 OS-DUMPER-7-PROC_PAGES %AlphaNum.. memory pages %XXX
10481105 OS-DUMPER-7-REGISTERS_INFO %AlphaNum.. %HEX %HEX %HEX %HEX
13590100 OS-DUMPER-6-FALLBACK_CHOICE %AlphaNum.. back choice: %AlphaNum.. in use
10358100 OS-DUMPER-7-TRACE_BACK %AlphaNum.. %MEM_ADDRESS
14630100 OS-DUMPER-7-INSTALL_PKG_SHOW_FAILED %AlphaNum.. installed packages show failed: Information not available

Result of coclustering for cluster 3:

Devices: {ip7, ip8, ip9}

Templates (pattern): {10903108}

Template Dictionary: 10903108 MGBL-CONFIG-6-DB_COMMIT %AlphaNum.. committed by user 'netengapalanke'. Use 'show configuration commit changes %AlphaNum.. to view the changes.

Each of the clusters discovered by the co-clustering algorithm corresponds to a different underlying cause of failure.

Step by step discovery of the clusters by this iterative method is shown below:

For eps = 0.1 => No clusters found.

For eps = 0.2 => No clusters found.

For eps = 0.3 => No clusters found.

For eps = 0.4 => No clusters found.

For eps = 0.5 => One cluster found.

Clustered devices = {ip7, ip8, ip9}

Co-clustered templates = {10903108}

For eps = 0.6 => No clusters found

For eps = 0.7 => No clusters found

For eps = 0.8 => One cluster found.

Clustered devices = {ip1, ip2, ip3}

Co-clustered templates = {12080102, 13278101, 11742100, 15517105, 11023100, 13226107, 14563100, 14563101, 13226102, 13226101, 15414101}

For eps = 0.9 => One cluster found.

Clustered devices = {ip4, ip5, ip6}

Co-clustered templates = {10481105, 12981101, 13625100, 13590100, 14630100, 13588101, 10358100}

Multiple runs of this algorithm with the same data and parameters resulted the exact same results every time. However, when an existing coclustering algorithm (CoclustMod) from coclust library (which is based on the direct maximization of modularity of the adjacency graph of the device-template matrix) is used, the results were highly unstable and produced completely different results in each run, even after running with the exact same set of parameters. The CoclustMod algorithm found the following device clusters in three different runs.

Run1: {ip2, ip3, ip6}, {ip4, ip5}, {ip1, ip7, ip8, ip9}
Run2: {ip2, ip3, ip7, ip8, ip9}, {ip5, ip6}, {ip1, ip4}
Run3: {ip1, ip2, ip3, ip7, ip8, ip9}, {ip4}, {ip5, ip6}

This instability is the result of the random initialization component of the algorithm which is not needed for the method described herein. Furthermore, the number of coclusters must be specified, which is very difficult to guess.

In summary, techniques are described herein for providing a coclustering algorithm that iteratively applies DBSCAN clustering with Jaccard distance to discover clusters of entities along with corresponding clusters of features. The algorithm provides a stable alternative to the existing coclustering algorithms that can discover distinct coclusters of different compactness beyond a threshold that can be controlled by the user. The algorithm may be used to discover patterns of syslog messages predictive of certain network device failures and simultaneously cluster the devices that encounter each of these patterns.