A Geometric Approach for Efficient Licenses Validation in DRM

Amit Sachan[1], Sabu Emmanuel[1], and Mohan S. Kankanhalli[2]

[1] School of Computer Engineering, Nanyang Technological University, Singapore
[2] School of Computing, National University of Singapore, Singapore
amit0009@ntu.edu.sg, asemmanuel@ntu.edu.sg, mohan@comp.nus.edu.sg

Abstract. In DRM systems contents are distributed from the owner to consumers, often through multiple middle level distributors. The owner issues redistribution licenses to its distributors. The distributors using their received redistribution licenses can generate and issue new redistribution licenses to their sub-distributors and new usage licenses to consumers. For the rights violation detection, all the newly generated licenses must be validated. The validation process becomes complex when there exist multiple redistribution licenses for a content with the distributors. In such cases, it requires the validation using an exponential number of validation equations, which makes the validation process much computation-intensive. Thus to do the validation efficiently, in this paper we propose a method to geometrically derive the relationship between different validation equations to identify the redundant validation equations. These redundant validation equations are then removed using graph theory concepts. Experimental results show that the validation time can be significantly reduced using our proposed approach.

1 Introduction

Prolific growth of the Internet technology over the last decade has made the Internet a convenient mode of digital contents distribution. However, it has also increased the fear of illegal contents redistribution and usage. Digital Rights Management(DRM)systems[5][9][6][1][2] emerge as one of the possible solutions to prevent illegal redistribution and usage. DRM systems often involve multiple parties such as owner, multiple distributors and consumers[5][9]. The owner gives the rights for redistribution of contents to distributors by issuing redistribution licenses. The distributors in turn can use their received redistribution license to generate and issue new different types of redistribution licenses to their sub-distributors and new usage licenses to the consumers. A redistribution license allows a distributor to redistribute the content to its sub-distributors and consumers as per the permissions and constraints [11] specified in the redistribution license that it has received. Thus, as part of the rights violation detection, it is necessary to validate these newly generated licenses against the redistribution licenses with the distributors.

The format of both redistribution(L_D) and usage licenses (L_U) for a content K is defined as: (K; P; $I_1, I_2, ..., I_M$; A), where P represents a permission(e.g.

play, copy, rip, etc.[4][9]), I_i represents the i^{th} ($1 \leq i \leq M$) instance based constraint and A represents aggregate constraint. Instance based constraints in the redistribution licenses are in the form of range of allowed values for distribution, such as region allowed for distribution, period of distribution, etc. Instance based constraints in usage licenses, such as expiry date of license, region allowed for play etc. may be in the form of a single value or range. The range/value of an instance based constraint in further generated licenses using a redistribution license must be within the respective instance based constraint range in it[9]. The aggregate constraint decides the number of permission P *counts* that can be distributed or consumed using a redistribution license or usage license respectively. For aggregate constraints, the sum of the aggregate constraint counts in all the licenses generated using a redistribution license must not exceed the aggregate constraint's value in it. For an issued license to be valid, it must satisfy both instance based and aggregate constraints in the redistribution license which is used to generate it. A validation authority does the validation of newly generated licenses based on instance based and aggregate constraints.

For business flexibility reasons, distributors may need to acquire multiple redistribution licenses for the same content. In case of multiple redistribution licenses, the validation becomes complex[10]. This is because a newly generated license can satisfy all the instance based constraints in more than one redistribution license(say a set of redistribution licenses S). So, for the aggregate constraints validation, the validation authority needs to select a redistribution license from set S. However, selecting a redistribution license randomly for the validation from S may cause potential loss to distributors as we discuss in section 2. To avoid a random selection, a better aggregate validation approach using validation equations is proposed in [10](equations are described in section 2). Validation equations use the whole set of redistribution license S instead of randomly selecting a redistribution license from it.

Doing the validation using the validation equations, the loss to distributors can be reduced but the approach requires the validation using an exponential number (to the number of redistribution licenses present for media)of validation equations. Also each equation may contain up to an exponential number of summation terms. This makes the validation process computationally intensive and necessitates to do the validation efficiently. In [10] an efficient offline aggregate validation method using the *validation tree* was proposed. The *validation tree* uses a prefix tree[7][8] based structure to calculate summation terms in each validation equation efficiently. But, still it requires the validation using exponential number of validation equations. Thus, in this paper we propose a method to geometrically derive the relationship between different validation equations to identify the redundant validation equations. These redundant validation equations are then removed by modification in original validation tree using graph theory concepts. Both theoretical analysis and experimental results show that our proposed method can reduce the validation time significantly.

Rest of this paper is organized as follows. In section 2, we discuss preliminaries required for this work. In section 3, a geometric approach for efficient validation is

discussed. In section 4, we modify the original validation tree to use the algorithm proposed in section 3. The performance of our proposed method is analyzed in section 5. Finally, the paper is concluded in section 6.

2 Preliminaries

In this section we first discuss the problem of validation in case of multiple redistribution licenses. Then we give an overview of the *validation tree*[10].

2.1 Validation in Case of Multiple Licenses

In case of multiple received redistribution licenses at the distributors, a newly generated license can satisfy all the instance based constraints in more than one redistribution license(say a set of redistribution licenses S). For the validation purpose, the validation authority needs to select a redistribution license from S. Selecting one redistribution license randomly out of multiple redistribution licenses may cause potential loss to the distributors as illustrated using example 1.

Example 1. Consider five redistribution licenses acquired by a distributor to distribute the play permissions according to two instance based constraints(validity period T, and region allowed R) and aggregate constraint A.
$L_D^1 = (K; Play; I_D^1 : T = [10/03/09, 20/03/09], R = [Asia, Europe]; A_D^1 = 2000)$
$L_D^2 = (K; Play; I_D^2 : T = [15/03/09, 25/03/09], R = [Asia];\ A_D^2 = 1000)$
$L_D^3 = (K; Play; I_D^3 : T = [15/03/09, 30/03/09], R = [America];\ A_D^3 = 3000)$
$L_D^4 = (K; Play; I_D^4 : T = [15/03/09, 15/04/09], R = [Europe];\ A_D^4 = 4000)$
$L_D^5 = (K; Play; I_D^5 : T = [25/03/09, 10/04/09], R = [America];\ A_D^5 = 2000)$

Now, the distributor generates a usage license $L_U^1 = (K;\ Play; I_U^1 : T = [15/03/09, 19/03/09], R = [India]; A_U^1 = 800)$. L_U^1 satisfies all instance based constraints for L_D^1 and L_D^2. Let the validation authority randomly picks L_D^2 for validation then remaining *counts* in L_D^2 will be 200(i.e. 1000-800). Next, let the distributor generates $L_U^2 = (K; Play; I_U^2 : T = [21/03/09, 24/03/09], R = [Japan]; A_U^2 = 400)$. L_U^2 satisfies all the instance based constraints only for L_D^2. The validation authority will now consider L_U^2 as invalid as L_D^2 now cannot be used to generate more than remaining 200 counts. In this case, a better solution would be to validate L_U^1 using L_D^1, and L_U^2 using L_D^2. This will result in both L_U^1 and L_U^2 as valid licenses. Thus, the challenge is to do the validation such that the distributors can use their redistribution licenses in an intelligent way.

A Method for Validation: In this section, we present the symbols(see table 1) used and validation equations. Details about the derivation of validation equations can be found in [10].

An issued license is said to belong to a set S of redistribution licenses if it satisfies all the instance based constraints in all redistribution licenses in set S.

Table 1: List of Symbols

Symbol	Explanation
S^N	The set of all N redistribution licenses for a content i.e. $S^N = [L_D^1, L_D^2, ..., L_D^N]$.
$SB^r[S]$	The r^{th} subset of set S of redistribution licenses. If the set S contains k redistribution licenses then $r \leq 2^k - 1$, and $k \leq N$.
$C[S]$	Aggregate of the *permission counts* in all previously issued licenses which belong to the set S of redistribution licenses.
$A[S]$	Sum of aggregate *permission counts* in all redistribution licenses in the set S.
$C\langle S\rangle$	LHS of the validation equation for the set S.

For example, L_U^1 in example 1 belongs to the set $\{L_D^1, L_D^2\}$. $C[S]$ denotes the sum of permission counts in all previously issued licenses that belongs to the set S of redistribution licenses. Let the table 2 represents the licenses issued using redistribution licenses L_D^1, L_D^2,..., L_D^5 in example 1. After L_U^6 being issued, the value of $C[\{L_D^1,\ L_D^2\}]$, $C[\{L_D^2\}]$, $C[\{L_D^1,\ L_D^2,\ L_D^4\}]$, $C[\{L_D^3,\ L_D^5\}]$ and $C[\{L_D^5\}]$ will be 840, 400, 30, 800 and 20 respectively. $A[S]$ denotes the sum of aggregate permission counts in all the redistribution licenses in the set S. For example, $A[\{L_D^1,\ L_D^2,\ L_D^3\}]$ for the redistribution licenses in example 1 will be sum of aggregate permission count in L_D^1, L_D^2 and L_D^3 i.e. 2000+ 1000+ 3000= 6000.

Table 2: Table of log records

Issued Licenses	Set(S)	*Set Counts*(C)
L_U^1	$\{L_D^1,\ L_D^2\}$	800
L_U^2	$\{L_D^2\}$	400
L_U^3	$\{L_D^1,\ L_D^2\}$	40
L_U^4	$\{L_D^1,\ L_D^2,\ L_D^4\}$	30
L_U^5	$\{L_D^3,\ L_D^5\}$	800
L_U^6	$\{L_D^5\}$	20

If there exists N received redistribution licenses for a content then we need to do the validation using $2^N - 1$ validation equations [10], one for each subset of S^N(see table 1 for explanation). The validation equation for the r^{th} subset of S^N, $SB^r[S^N]$, is of the form:

$$\sum_{l=1}^{2^m-1} C[SB^l[\mathbf{SB^r}[\mathbf{S^N}]]] \leq A[\mathbf{SB^r}[\mathbf{S^N}]].$$
$$\text{where}, m = |\mathbf{SB^r}[\mathbf{S^N}]|,\ \text{and}\ 1 \leq m \leq N \quad (1)$$

The LHS of the equation 1 does the summation of counts for the sets that are subset of the set $SB^r[S^N]$. Since there can be $2^m - 1$ subsets excluding the null set of a set containing m redistribution licenses therefore the summation

has a limit from 1 to $2^m - 1$. The RHS is the summation of aggregate constraint counts in all redistribution licenses in the set $SB^r[S^N]$.

Example 2. consider five redistribution licenses in example 1. Since there are five redistribution licenses therefore N=5 in this case and total $2^5 - 1$=31 validation equations are required. The validation equation for an example set $\{L_D^2, L_D^3, L_D^4\}$ will be $C[\{L_D^2\}]+C[\{L_D^3\}]+C[\{L_D^4\}]+C[\{L_D^2, L_D^3\}]+ C[\{L_D^2, L_D^4\}]+C[\{L_D^3, L_D^4\}]+C[\{L_D^2, L_D^3, L_D^4\}] \leq$ A$[\{L_D^2, L_D^3, L_D^4\}]$.

From table 1, LHS of equation 1 can also be referred as $C\langle SB^r[S^N]\rangle$. *Therefore, further in this paper, in short, we will refer validation equation for a set S(by replacing* $SB^r[S^N]$ *with S in equation 1) as:* $\mathbf{C\langle S\rangle \leq A[S]}$.

Complexity of Validation: If N number of redistribution licenses are present with a distributor then $2^N - 1$ validation equations are possible. Let a newly generated license satisfies all instance based constraints in k redistribution licenses out of the total N redistribution licenses. The set formed by these k redistribution licenses will be a subset of $2^{(N-k)}$ sets thus it will be present in $2^{(N-k)}$ validation equations. So, we need to do the validation using $2^{(N-k)}$ validation equations. Validation using such a large number of validation equations every time a new license is issued is computationally intensive. Also, the violation of aggregate constraints is not a frequent event as the received redistribution licenses generally have sufficient *permission counts* to generate thousands of licenses. So, instead of doing validation every time a new license is issued, the aggregate validation is done offline by collecting the logs of the sets of redistribution licenses(to which the issued licenses satisfies all instance based constraints) and *permission counts* in issued licenses. The format of the log is as shown in table 2.

2.2 Overview of Validation Tree

In this section, we briefly discuss about construction of *validation tree* using the log records and validation using *validation tree*[10].

Validation Tree Construction: As shown in figure 1, each node in the *validation tree* stores the following fields: name of a redistribution license(L), a count(C) and links to its child nodes. The count value C determines the count associated with the set formed by the redistribution license in the node and all its prefix nodes(nodes in the path from root node to the current node) in the same branch. For example, count=840 for the node $root \rightarrow L_D^1 \rightarrow L_D^2$ implies that the *set count* associated with the set $\{L_D^1, L_D^2\}$ is 840. In the *validation tree* child nodes of a node are ordered in increasing order of their indexes. To generate the *validation tree* initially a *root* node is created. Validation tree is then expanded by inserting the log records using the insertion algorithm(algorithm 1). The insertion algorithm one by one reads the records in the log and then one by one reads the indexes of redistribution licenses in each record. In algorithm 1, let the record to be inserted currently be given by R=[r, R'], where r is the first

redistribution license and R' is the set of remaining redistribution licenses and let *count* denotes the set count associated with record R. Algorithm 1 inserts the record R in the validation tree with the *root* node initially denoted by T. The algorithm traverses the validation tree according to the redistribution licenses in the log record and inserts/adds the count in the last node traversed for the record. Figure 1 shows the validation tree generated for the records in table 2.

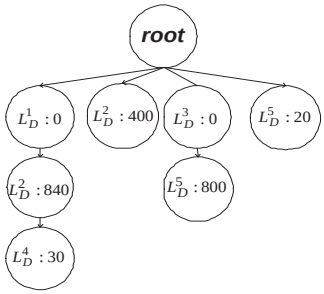


Fig. 1: The validation tree

Algorithm 1 *Insert*(T, R, *count*)

1. Sequentially traverse the child nodes of T until all child nodes are traversed or we find a child T' such that $T.L \geq r$.
2. If $T'.L = r$ then go to step 4.
3. Else add a node T' such that $T'.L = r$ and $T'.C = 0$ as the child node of T(in the order).
4. If R'=null set then $T'.C = T'.C + count$. Else, call Insert(T', R', *count*).

Validation Using Validation Tree: Algorithm 2 is used to do the validation for all possible validation equations. The parameter i takes care that validation is done for all possible sets. If N redistribution licenses are present, it takes value from i=1 to $i = 2^N - 1$. Each value of i corresponds to a particular set of redistribution licenses(and hence validation equation), which is determined by bits=1 in binary format of i, e.g. i=7 has first, second and third bits from LSB as 1 and rest as 0. So, it represents validation equation for set $\{L_D^1, L_D^2, L_D^3\}$.

The algorithm calculates and compares LHS(CV) and RHS(AV) of each validation equation(see equation 1) to determine whether an equation is valid or not. The RHS of each validation equation is calculated using an array A of size N containing aggregate constraint values in all N received redistribution licenses. The j^{th} element, $A(j)$, of A contains the aggregate constraint value in the j^{th} received redistribution license. Since the set of redistribution license is decided by the position of bits=1 in i in algorithm 2 so we only need to add

the aggregate constraint values for the redistribution licenses that correspond to bits=1. It is taken care by left shift and AND operation in algorithm 2. Parameter $licNumber$ in algorithm 2 is calculated to facilitate the traversal of validation tree. It is equal to the number of redistribution licenses corresponding to the current validation equation(or number of bits=1 in i). The LHS for the validation equation for a set S does the summation of set counts for set S and all its subsets. It is calculated by traversing the validation tree. Detailed about tree traversal algorithm are out of scope of this paper and can be found in [10].

Algorithm 2 Validation($root$, A, N)

```
Temporary Variables:AV=0, CV=0
for i=1 to 2^N − 1 do
    licNumber=0.
    for j=1 to N do
        if (1<< (j − 1) AND i) ≠ 0 then
            /* <<: left shift operator                    */
            AV=AV+ A(j).
            licNumber=licNumber+1;
    Call CV=VaLHS(root, i, licNumber).

    if CV ≤ AV then
        Declare(Valid Equation).
    else
        Declare(Invalid Equation)
```

3 Proposed Efficient Validation Approach

In this section, we present a geometric approach to identify the redundant validation equations and a method to remove the redundant validation equations.

3.1 Geometric Representation of Licenses

If there are M number of instance based constraints in the licenses then each redistribution /usage license can be represented as an M dimensional hyper-rectangle e.g. figure 2 shows five redistribution licenses and two usage licenses represented in a 2-dimensional space. The set of redistribution licenses to which an issued license can be instance validated is given by the set of redistribution licenses whose hyper-rectangles completely contain the hyper-rectangle formed by the issued license. For example, in figure 2, hyper-rectangle formed by L_U^1 is completely within L_D^4 only. Thus, the set of redistribution licenses to which L_U^1 can be instance based validated is given by $\{L_D^4\}$. However, L_U^2 is not completely within any of the 5 redistribution licenses so it cannot be instance based validated using any of the 5 redistribution licenses and it is treated as invalid.

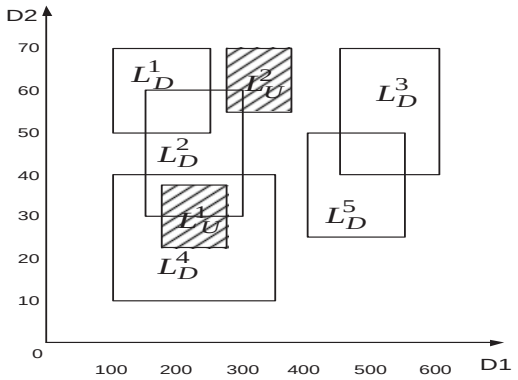


Fig. 2: Representation of 5 received redistribution licenses with two constraints

3.2 Redundant Validation Equations

In this section we first define overlapping redistribution licenses and non-overlapping sets of redistribution licenses. Then with the help of theorems 1 and 2, we present a method to identify the redundant validation equations.

Overlapping Redistribution Licenses. Two redistribution licenses, L_D^j and L_D^k, with instance based constraints $\{I_1^j, I_2^j, ..., I_M^j\}$ and $\{I_1^k, I_2^k, ..., I_M^k\}$, are overlapping if $I_m^j \cap I_m^k \neq \emptyset$, $\forall\ m \leq M$, where M is the number of total instance based constraints. Geometrically, two redistribution licenses L_D^j and L_D^k are overlapping if all their constraint dimensions have an overlap. Thus, in figure 2, licenses L_D^1 and L_D^2 are overlapping but L_D^1 and L_D^4 are non-overlapping.

Non Overlapping Sets. Two sets S_1 and S_2 of received redistribution licenses are said to be non overlapping if S_1 and S_2 contain different licenses(i.e. $S_1 \cap S_2 = \emptyset$) and any of the constraint ranges in the license in the set S_1 does not overlap with any license in the set S_2. For example, the sets $S_1 = \{L_D^1, L_D^2\}$ and $S_2 = \{L_D^5\}$ are non overlapping in figure 2 as neither L_D^1 nor L_D^2(in set S_1) overlaps with L_D^5(in set S_2).

Theorem 1. *If all the redistribution licenses in a set S do not have a common overlapping region then count($C[S]$) associated with that set will always be 0.*

Proof. If all the redistribution licenses in a set S do not have a common region then any valid issued license cannot be simultaneously within the hyper-rectangles formed by all redistribution licenses in the set. Thus, the count value for the set will always be 0. For example, redistribution licenses L_D^1, L_D^2, and L_D^3 in figure 2 do not form a common region and hence the hyper-rectangle formed by any issued license cannot be inside the hyper-rectangles of L_D^1, L_D^2, and L_D^3 simultaneously. So, $C[\{L_D^1, L_D^2, L_D^3\}]$ will always be 0.

Corollary 1.1. Set S of received redistribution licenses formed by taking at least one redistribution license from two or more non overlapping sets of received

redistribution licenses will always have *set count* equal to 0. This is because if we take at least 1 received redistribution licenses from two different non overlapping sets of received redistribution licenses then a common region cannot be formed by all the received redistribution licenses in S.

Theorem 2. *If a set S containing n received redistribution licenses can be represented with m number of non overlapping sets($S_i \cap S_j = \emptyset$, $1 \leq i < j \leq m \leq n$) of received redistribution licenses such that $S_1 \cup S_2 \cup ... \cup S_m = S$ then the validation equation for the set S can be expressed as the sum of validation equations for the sets S_1, S_2, ... and S_m. Validation equation for a set S_i is: $C\langle S_i \rangle \leq A[S_i]$.*

$$C\langle S \rangle = C\langle S_1 \rangle + C\langle S_2 \rangle + C\langle S_3 \rangle + ... + C\langle S_m \rangle \leq$$
$$A[S] = A[S_1] + A[S_2] + A[S_3] + ... + A[S_m] \quad (2)$$

Thus, if the validation equations for the sets S_1, S_2, ... and S_m are already validated then we do not need to validate the equation for the set S.

Proof. m number of non overlapping sets of received redistribution licenses are represented by S_1, S_2, ...S_m, where $2 \leq m \leq N$. Now consider a set formed by at least one received redistribution license each from $m^{'}$ $(2 \leq m^{'} \leq m)$sets out of total m sets. As $S_i \cap S_j = \emptyset$ therefore using corollary 1.1, any such set formed will always have the *set count* value equal to 0. Thus, all the sets which are created using taking at least one received redistribution license each from any $m^{'}$ sets out of m sets will always have count value equal to 0. So, $C\langle S \rangle$ will be only due to the sets formed due to redistribution licenses within each of the m sets individually. Thus, $C\langle S \rangle$ can be written as: $C\langle S \rangle = C\langle S_1 \rangle + C\langle S_2 \rangle + C\langle S_3 \rangle + ... + C\langle S_m \rangle$. Also, the sets S_1, S_2, ... and S_m are non overlapping and $S_1 \cup S_2 \cup ... \cup S_m = S$. Therefore, $A[S]$ can be written as: $A[S] = A[S_1] + A[S_2] + A[S_3] + ... + A[S_m]$.

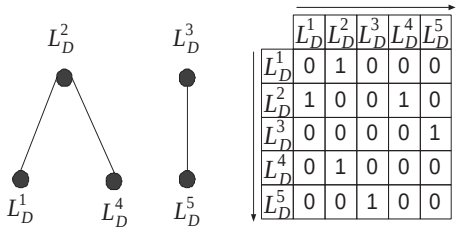
Thus, equation 2 is correct and it can be obtained by summation of the validation equations for the individual sets S_1, S_2, ... and S_m. Hence if the validation equations for the sets S_1, S_2, ... and S_m are evaluated then the validation equation for the set S can be removed.

For example, the redistribution licenses in figure 2 can be divided in two groups, with group 1 and group 2 containing the redistribution licenses (L_D^1, L_D^2, L_D^4) and (L_D^3, L_D^5) respectively. According to theorem 2, if validation equations for all subsets of the sets $\{L_D^1, L_D^2, L_D^4\}$ and $\{L_D^3, L_D^5\}$ are evaluated independently then we need not evaluate the validation equations for any set generated by taking at least one redistribution license from each group. So, equations for the sets $\{L_D^1, L_D^3\}$, $\{L_D^1, L_D^2, L_D^3\}$, etc. need not be evaluated.

3.3 Identification of Disconnected Groups

To use the basis discussed in the previous sub-section, we need to identify the disconnected groups of redistribution licenses. In this section, we present an algorithm to find the disconnected groups of redistribution licenses.

We use a graph[12][3] G=(V, E) to identify the disconnected groups of redistribution licenses, where V is the set of vertices and E is the set of edges in G. As shown in figure 3, each redistribution license is represented using a vertex in G. There is an edge between the vertices corresponding to the i^{th} and j^{th} redistribution licenses if they are overlapping. We represent the graph using an adjacency matrix Adj of size N X N. The entry in the i^{th} row and j^{th} column, $Adj_{i,j}$, is 1 if the i^{th} and j^{th} redistribution licenses are overlapping else it is 0. Figure 3 shows the graph and matrix Adj for redistribution licenses in figure 2.



	L_D^1	L_D^2	L_D^3	L_D^4	L_D^5
L_D^1	0	1	0	0	0
L_D^2	1	0	0	1	0
L_D^3	0	0	0	0	1
L_D^4	0	1	0	0	0
L_D^5	0	0	1	0	0

Fig. 3: Graph and adjacency matrix for redistribution licenses in figure 2

We perform depth first search(DFS)[12] in algorithm 3 on G to identify the number of groups of redistribution licenses and redistribution licenses in each group. The DFS starts from the node corresponding to the first redistribution license. It finds all the nodes that are directly or indirectly(through other nodes) connected to the node corresponding to the first redistribution license. All such nodes form one group of redistribution licenses. Next, the algorithm sequentially searches for a redistribution license, which is not present in any previously formed groups. Then, finds all the nodes that are directly or indirectly connected to the node corresponding to the searched redistribution license. The process continues until a group is allocated to every redistribution license. To perform DFS, two arrays viz. *Visited* and *Group* of size N and N X N, respectively, are initialized with each element equal to 0 in algorithm 3. Array *Visited* keeps information about all the nodes traversed in the graph during the DFS. Array *Group* is modified by the algorithm and after completion of algorithm it stores the information about the redistribution licenses in different groups of redistribution licenses. Each row in *Group* corresponds to a group of redistribution licenses and stores the information about the redistribution licenses present in that group. If j^{th} redistribution license is in the i^{th} group then $Group_{i,j}$=1 else it is 0. Please note that the number of rows in *Group* are N as maximum number of groups formed can be N(assuming no connected licenses). If there are g number of groups then only first g rows give information about groups and redistribution licenses in each group. For example, the first two rows in array *Group* for the graph in figure 3 are given by (1, 1, 0, 1, 0) and (0, 0, 1, 0, 1) as the redistribution licenses in the first and second group are (L_D^1, L_D^2, L_D^4) and (L_D^3, L_D^5) respectively. Rest of

the rows have all entries 0. Algorithm 3 also calculates array $GroupSize$, which determines the number of redistribution licenses in each group.

Algorithm 3 Group Formation

```
int g=0.
array Visited: Size N. All elements are initialized to 0.
array Group: Size NXN. All elements are initialized to 0.
array GroupSize: Size N. All elements are initialized to 0.
Adj: Adjacency matrix of the graph.
for i=1 to i ≤ N do
    if Visited[i]=0 then
        g=g+1.
        Call Depth_first(i, g).
Print(Number of groups=g).
Subroutine:Depth_first(i, k)
{
Group[k][i]=1, Visited[i]=1
GroupSize[k]=GroupSize[k]+1.
for j=i+1 to i ≤ N do
    if Adj[i][j]=1 and Visited[j]=0 then
        Call Depth_first(j, k).
}
```

4 Validation Algorithm

In this section, we modify the *validation tree* [10] to enable it to use the approach in section 3 for efficient validation. We do this by dividing the original *validation tree* in g parts and modifying the index of the nodes in each newly generated *validation tree*, where g is the number of groups of redistribution licenses determined by algorithm 3.

4.1 Division of Validation Tree

Using corollary 1.1, a set formed by taking at least one redistribution license from two or more groups out of the total g groups will have *set count* equal to 0. So, in the log records any such set cannot exist. This implies that in *validation tree* any branch will not contain the redistribution licenses from two or more groups out of the total g groups. For example, in figure 2, any issued license cannot belong to the sets $\{L_D^1, L_D^3\}$ or $\{L_D^1, L_D^3, L_D^5\}$. So these will not be present in the logs and there cannot be branches $root \rightarrow L_D^1 \rightarrow L_D^3$ or $root \rightarrow L_D^1 \rightarrow L_D^3 \rightarrow L_D^5$. Thus, the *validation tree* can be divided into g independent parts; each part contains the nodes corresponding to the redistribution licenses in a particular group. Algorithm 4 (by calling $Separation(root, g)$) is used to divide the original

validation tree into g new *validation trees*. The algorithm checks for the group to which a child node of the *root* node belongs. If a child node belongs to the j^{th} group then link the child nodes to the j^{th} newly generated validation tree with root node as $root_j$. Figure 4 shows two validation trees obtained after division of the validation tree in figure 1.

Algorithm 4 *Separation*(T, g)

Initialize m number of root nodes.
Let the root of the i^{th} validation tree be defined as $root_i$.
foreach *Child of* T **do**
  Let current child be T'
  if $T'.index \in Group[j]$ **then**
    Link T' as child node of $root_j$.

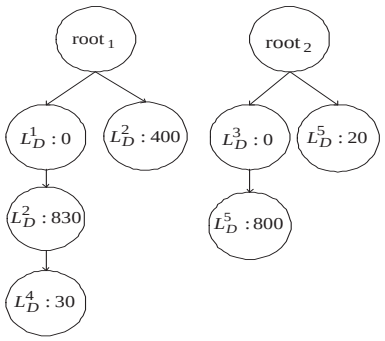


Fig. 4: Division of Validation tree

4.2 Modification of Indexes

In this step, the indexes of the nodes in each new validation tree are modified. This is to ensure that if N_k(=$GroupSize[k]$, calculated in algorithm 3) redistribution licenses are in the k^{th} group then the indexes of nodes in the k^{th} validation tree vary from 1 to N_k, which is required for the validation algorithm (algorithm 2). Algorithm 5 is used to modify the indexes for each validation tree. To modify the indexes of the nodes in the k^{th} validation tree (with root node $root_k$), algorithm *Modification*($root_k$, A_k) is called. The algorithm calculates and uses an array named $position_k$ to modify the indexes. If a redistribution license(say j^{th}) is in the k^{th} group then the value stored at $position_k[j]$ determines the new index of the j^{th} redistribution license. The original index is replaced by the new index in the last step in algorithm 5. For instance, consider array $position_2$ for the second validation tree in figure 4. According to algorithm 5, it will be (0, 0,

1, 0, 2). So, the indexes 3 and 5 in the nodes of the second *validation tree* in figure 4 are replaced by 1 and 2 respectively in figure 5. The *validation trees* in figure 4 after modification of indexes are shown in figure 5.

Algorithm 5 *Modification*($root_k$, A_k)

array *Group*: Same as calculated in algorithm 3.
array $position_k$: size N. All elements are initialized to 0.
p=1.
for *j=1 to N* **do**
    if *Group[k][j]=1* **then**
        $position_k[j]=p$.
        $A_k[p]=A[j]$
        $p=p+1$.
Traverse the k^{th} validation tree.
for *Each node traversed(except* $root_k$*)* **do**
    Let the index of the node be given by *index*.
    Replace *index* by $position_k[index]$.

For each group, we also need to calculate the array containing the aggregate constraint values in the redistribution licenses in the group. Let it be A_k for the k^{th} group (corresponding to k^{th} validation tree). The size of array A_k is given by N_k, where N_k is the number of redistribution licenses in the k^{th} group. A_k is derived using the initial array A(see section 2.2) containing aggregate constraint counts for all redistribution licenses. The procedure for derivation of A_k using A is mentioned in algorithm 5. After the modification of indexes and calculation of aggregate constraint array, we can directly use the validation algorithm(algorithm 2) in section 2.2 for each validation tree. So, we call Validation($root_k$, A_k, N_k) for each value of $k \leq g$, where g is the total number of groups of redistribution licenses.

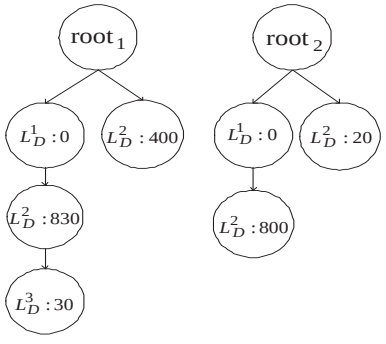


Fig. 5: Modification of Indexes

The validation time performance gain of our proposed approach depends on the number of groups formed and number of redistribution licenses in each group. If N_k number of redistribution licenses are present in the k^{th} group then $2^{N_k}-1$ validation equations are needed for the k^{th} group. Therefore, the total number of validation equations required to validate will be $\sum_{k=1}^{g}(2^{N_k}-1)$. Whereas, without using our approach the total number of validation are 2^N-1. Thus, the approximate performance gain(G) can be given as:

$$G \approx \frac{2^N-1}{\sum_{k=1}^{g}(2^{N_k}-1)} \tag{3}$$

The value of G varies from 1 to $(2^N-1)/N$ for m=1 to m=N. Thus, the performance gain always remains greater than or equal to 1.

As an illustration, consider the five redistribution licenses in example 1. These redistribution licenses can be divided in two groups (group1:(L_D^1, L_D^2, L_D^4) and group2:(L_D^3, L_D^5)). So, the approximate gain in this case would be $(2^5-1)/((2^3-1)+(2^2-1))$=3.1 times.

5 Performance Analysis

In this section, we analyze the theoretical and experimental performance of our proposed algorithm in terms of validation time complexity, storage space requirements, and validation tree division and modification time. All the experiments were performed on Intel(R) core(2) 2.40 GHZ CPU with 2 GB RAM. All the programs are written in Java. To perform the experiments, first we created a number of redistribution licenses and issued licenses. The set of redistribution licenses for which an issued license satisfies all instance based constraints along with the aggregate constraint counts is saved in the log records. For the experiments, each redistribution license is assumed to contain 4 instance based constraints and aggregate constraint count in between 5000 and 20000. Each issued license is assumed to contain permission counts in between 10 and 30. The number of log records in our experiments varies from about 600 for n=1 redistribution license to 22000 for n=35 redistribution licenses.

A. Validation Time Complexity: First, we show the variation of number of groups of redistribution licenses with the number of redistribution licenses(N) in figure 6. The number of groups in our experiments varies from 1 to 5 for different values of N. As in figure 6, the number of groups may remain same increase or decrease after addition of a new redistribution license. For example, let in figure 2 a new redistribution license L_D^6 is added. The number of groups will remain same if L_D^6 is connected to redistribution licenses in only one group out of two existing groups. The number of groups will increase(increased to 3) if L_D^6 is not connected to redistribution licenses in any group. The number of groups will decrease(decreased to 1) if L_D^6 is connected to redistribution licenses in both groups.

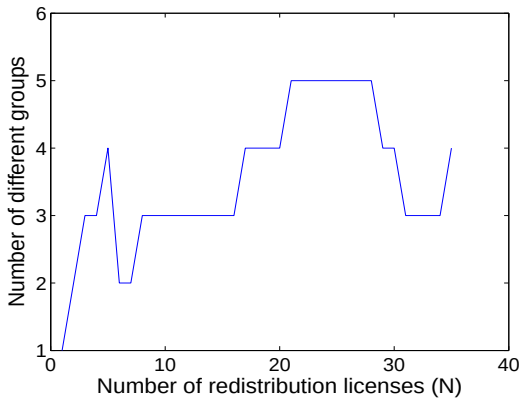


Fig. 6: Variation of number of groups

Figure 7 shows a comparative analysis of validation time(V_T) required using our proposed method with the approach in [10]. To show that the time required for division(D_T) of the original validation tree is small as compared to V_T, we also plot the curve for V_T+D_T. From the experiments, it can be observed that using our proposed method the validation time is significantly reduced. Also, the effect of D_T becomes very small as compared to V_T for N>2.

To show whether the result is in accordance with equation 3, we also compare the theoretical(using equation 3) and experimental gain in figure 8. We observe that the experimental gain is always greater or equal to the theoretical gain. This is because when we divide the *validation tree* in g parts, we only traverse one validation tree out of g *validation trees*. Due to this some redundant traversals required in the original validation tree are not required.

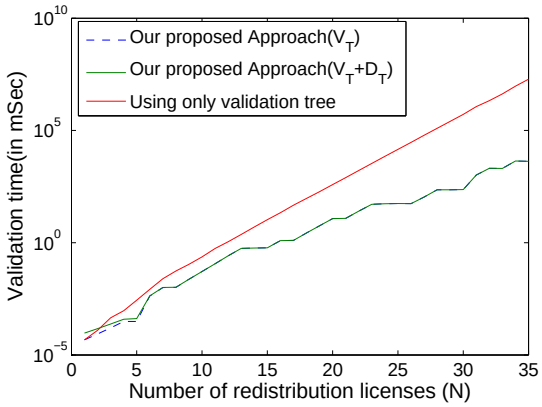


Fig. 7: Validation Time Complexity

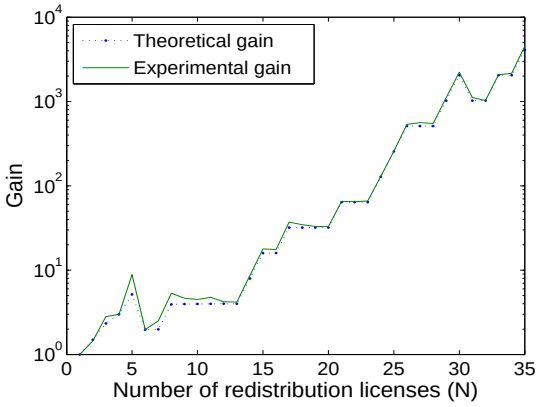


Fig. 8: Theoretical Vs. Experimental Gain

B. Construction Time of Data Structure: The construction time of the data structure is the sum of *validation tree* construction time(C_T) and time required for division(D_T) of the original *validation tree*. In comparison to [10], the additional time required is D_T. The time required to divide the *validation tree* includes the time required to identify the groups of redistribution licenses using graph and traverse the child nodes of the *root* node. Figure 9 shows the comparison between the time required to insert 1 log record and time required for division of *validation tree*. The time required for the division of *validation tree* is only 3-4 times as compared to time required for insertion of a single record. But, the insertion process generally requires thousands of log records insertion whereas division of *validation tree* is performed only once. So, the overhead due to the *validation tree* division is very small.

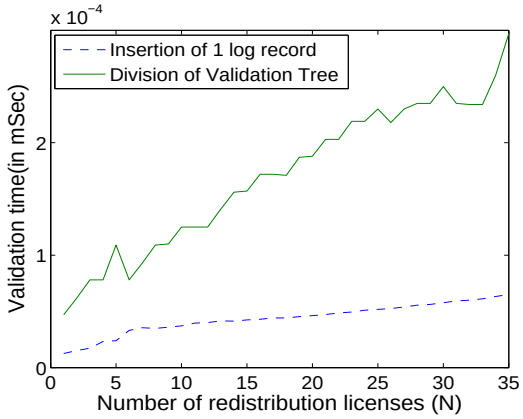


Fig. 9: Insertion time complexity

C. Storage Space Requirement: Storage Space requirement is equal to the space required to store the *validation tree(s)*. Figure 10 compares the storage space required for storing new *validation trees* after the division of original *validation tree* with the original *validation tree*. As no new nodes(except the *root* nodes) for the validation trees generated after the division of *validation tree* so the storage space requirement will also be almost same for both original *validation tree* and *validation trees* formed after division.

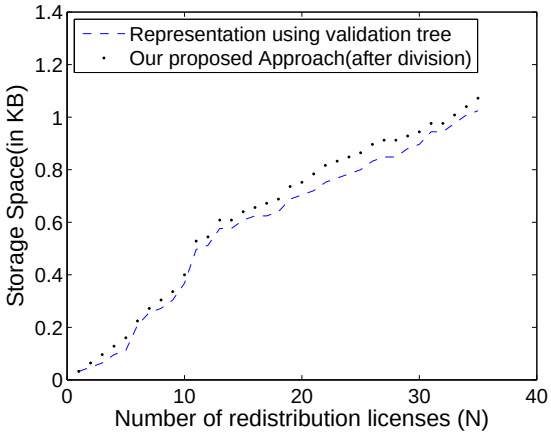


Fig. 10: Storage space complexity

6 Conclusion

In this paper, we presented an efficient method of doing license validation in DRM systems by removing the redundant validation equations. For this purpose, first we proposed a geometry based approach to identify the redundant validation equations. Then we removed the redundant validation equations by grouping of redistribution licenses and division of the *validation tree*. The theoretical analysis and experimental results show that the proposed method can do the validation much efficiently as compared to doing using the original *validation tree*. The experiments show that the time required for identification of redundant validation equations and division of the *validation tree* is very small. Also, the storage and insertion time complexities are almost same as that required for original *validation tree* based validation method.

Acknowledgment: Thanks to the Agency for Science, Technology and Research (A-STAR), Singapore for supporting this work under the project "Digital Rights Violation Detection for Digital Asset Management" (Project No: 0721010022).

References

1. C. Conrado, M. Petkovic, and W. Jonker. Privacy-preserving digital rights management. In *Secure Data Management(SDM)*, pages 83–99, 2004.
2. E. Diehl. A four-layer model for security of digital rights management. In *Proceedings of the 8th ACM workshop on Digital rights management*, pages 19–28, 2008.
3. J. Gross and J. Yellen. *Handbook of Graph Theory and Applications*. CRC press, 2003.
4. M. Hilty, A. Pretschner, D. Basin, C. Schaefer, and T. Walter. A policy language for distributed usage control. *Lecture notes in computer science*, 4734:531–546, 2007.
5. S. O. Hwang, K. S. Yoon, K. P. Jun, and K. H. Lee. Modeling and implementation of digital rights. *The Journal of Systems and Software*, 73(3):533–549, 2004.
6. R. Iannella. Digital rights management (DRM) architectures. *D-Lib Magazine*, 7(6), 2001.
7. D. E. Knuth. *The art of computer programming, volume 3: sorting and searching*. Addison Wesley Longman Publishing Co., Inc. Redwood City, CA, USA, 1998.
8. G. Liu, Hongjun Lu, W. Lou, Yabo Xu, and J. X. Yu. Efficient mining of frequent patterns using ascending frequency ordered prefix-tree. *Data Mining and Knowledge Discovery*, 9(3):249–274, 2004.
9. A. Sachan, S. Emmanuel, and M. S. Kankanhalli. Efficient license validation in MPML DRM architecture. In *9th ACM workshop on digital rights management(DRM'09)*, Chicago, pages 73–82, 2009.
10. A. Sachan, S. Emmanuel, and M. S. Kankanhalli. Efficient aggregate licenses validation in DRM. In *Database Systems for Advanced Application(DASFAA(2))*, Tsukuba, Japan, pages 313-319, 2010.
11. R. Safavi-Naini, N. P. Sheppard, and T. Uehara. Import/export in digital rights management. In *Proceedings of the 4th ACM workshop on Digital rights management*, pages 99–110. ACM New York, 2004.
12. D. West. *Introduction to graph theory*. Prentice Hall Upper Saddle River, NJ, 2001.