CHALMERS

Analysis and Presentation of Combinatorics in Product Configuration

Master of Science Thesis

HU ZIYANG

Department of Product and Production Development
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden, 2010

Master of Science Thesis

Analysis and Presentation of Combinatorics in Product Configuration

HU ZIYANG

Examined by:

Professor Johan Malmqvist

Department of Product and Production Development

CHALMERS UNIVERSITY OF TECHNOLOGY

Göteborg, Sweden, 2010

Analysis and Presentation of Combinatorics in Product Configuration

HU ZIYANG

Examiner: Professor Johan Malmqvist

Department of Product and Production Development

CHALMERS UNIVERSITY OF TECHNOLOGY

SE-412 96 Göteborg

Sweden

Telephone +46 (0) 31-772 1000

Chalmers Reproservice

Göteborg 2010

Abstract

In automotive industry, engineers are working hard to check whether existing constraints are correct and valid combinations are expected among the allowed combinations. However, this is not a trivial task due that the number of variables and the constraints may be very large. The goal of this thesis is to examine feasibility of analyzing and exploring such large data sets to support engineers' work.

For this purpose, a two-step strategy has been taken to solve this task. Workload is divided to firstly investigate the feasibility to present big number of string-formatted allowed combinations. The second step is to find techniques that are of value to navigate and query the data. Several terminologies, Cartesian Product, Treemap, etc, are introduced to fulfill the two tasks respectively.

The decision on rejection or acceptance of the alternatives are made with the help of graphs and tables. The outcome of the study covers a set of workflows and user cases that describe the principles and results of the proposed solutions.

Key words: Constraints, Allowed Combinations, Presentation, Visualization

Acknowledgments

This thesis is the last part of my study in the Master of Science program in Computer Science at Chalmers University of Technology.

I would like to express my gratitude to all those who have helped me during the writing of this thesis. I gratefully acknowledge the help of my supervisor Professor Johan Malmqvist. I do appreciate his patience, encouragement and instructions during this specific period.

My gratitude also extends to my friends and my family who have been assisting, supporting and caring for me all of my life.

Table of Contents

1. Introduction: The Nature of The Problem

1.1 Background of the problem

In the automotive industry, companies design product ranges instead of individual products to offer customers different options. For example, a vehicle could be equipped with engines of different sizes, different suspensions, different electronic systems, etc. Normally, a vehicle is equipped with around 500 features and there are at least 2 alternatives for each feature. Consequently, for a single vehicle, the theoretical product range will contain more than 2^{500} products, a massively huge number to present. However, a specific engine may not be offered in all types of vehicles due to the constraints among the features. These constraints may be due to engineering considerations, legal issues, or marketing strategy and they are expressed as configuration rules. The configuration rules are stored in the PDM (Product Data Management) system along with other critical documentation. Though these constraints greatly remove a considerable part of theoretical options, the number of remaining configurable vehicles is hundreds of thousands, still on a high level.

A PDM system is the framework used to describe technical product offerings and corresponding design solutions [1]. Currently, when searching after allowed partial product configurations the PDM system requires a limited/specific input of component features in order to be able to display the results. The aim of the research is to find alternative presentation methods for large results.

1.2 Problem Statement

There are around 500 features to consider building a single vehicle and when the engineers just review a partial configuration, there is a risk that selected features do not get along with other unselected ones to build a complete vehicle. Such conflicts are hard to detect and need to be solved by the experts. We may advise the engineers to select the features in a bigger scope to cover all possible conflicts. However, this leads to another problem. Since there are 500 features in a complete vehicle and there are at least two alternatives in a single feature, all the theoretical combinations will be at least 2 to the power of 500. Though not all of them are offered to customers due to technical or market considerations, the remaining allowed combinations will still be on a high level and there is not a good method to display and analyze this huge data.

The classical exploration of big datasets usually follows a three step process: overview first, zoom and filter, and then details-on demand, which has been called the Information Seeking Mantra [2]. Based on these facts, the analysis of large data in this research reveals two tasks. The first one is the question, how presentations for this massive data sets can be constructed without losing important information. The second task is to find techniques to efficiently navigate and query such large data sets.

1.3 Relevance of the Problem

In large scale product development applications, constraints are changing constantly, rules are frequently added, deleted and modified. Thus, the configuration engineer must be able to check out whether new rules are valid and whether invalid (obsolete) combinations have been correctly removed. Normally, the more information provided to the engineers, the easier to detect the faults. The capability to present a larger scope of allowed configurations supports the configuration engineers to detect the incorrect configurations easier, letting them work more efficiently.

1.4 Research Objectives

In general, the aim of this research is to analyze how to present and navigate the huge data in a user-friendly and comprehensive way. Accordingly, four main objectives have been set up to ensure the purpose of the project will be fully covered. The objects are as follows:

- Understand the problem
- Literature study of presenting and navigating methods for huge data
- Propose solutions of presenting and navigating huge data
- Test and verify key components of the suggested solutions

1.5 Project Scope and Challenges

The feasibility of presenting and navigating massive allowed combinations in an user-friendly and comprehensive way has been studied and analyzed in this project. To better achieve the above research objectives, some in-scope and out-of-scope issues need to be clarified and taken into account in this study.

1.5.1 In Scope

Investigating alternatives and constraints for presenting and navigating huge data and developing a feasible solution based on the findings formed the majority of research activities in this research. Feasible methods are selected and have been tested that they are proper to present and navigate huge data. Several real examples have also been given to test their validities and stabilities.

1.5.2 Out of Scope

The author will only present two principles to accomplish the two tasks respectively, alternatives under these two principles will be introduced and evaluated. It is beyond the scope of this paper to assess all possible principles.
On the other hand, the author will not study the mechanism that generates the source data. All focus is given to present and navigate the huge data in a good way.

1.6 Project Resources

Several persons have assisted and different sources have been used in this research. Professor Johan Malmqvist has examined the project at Chalmers University of Technology. In order to check the validation of the solutions, several real examples have been provided and tested.

1.7 Thesis Outline

Chapter 2 briefly introduces several important terminologies that help to better understand the research. Chapter 3 suggests two methods to present the huge data. Chapter 4 enumerates several ways to visualize the compressed combinations. Chapter 5 sums up major conclusions in this research and proposes some ideas of the future work. Appendix A illustrates the complete flow of using identifiers to compress the fuel tank example while Appendix B displays a partial VB script that is used to mark differences of two block data in an excel file.

2. Terminologies and Concepts in Automotive Industry

This chapter presents the terminologies that support this research. It is generally aimed to derive a good understanding about variants, variant families, configuration rules and allowed combinations. A small example is given to better illustrate these concepts.

2.1 Terminology

Two terminologies, *Variant Family* and *Configuration Rule* are introduced as they are critical to understand the research problem.

Variant Family:	In order to describe different products, different features are used. These features are called "variant families" and the variations within the family are called "variants". In automotive industry, "Fuel Tank Material" may be used as a *Variant Family* to describe this tank property. Variants of "Fuel Tank Material" could be "steel", "aluminum", and "plastic". In this paper, VF is the acronym for Variant Family.
Configuration Rule:	There are multiple types of configuration rules, inclusion and exclusion rules are two types of the rules reflecting the constraints among variants. Inclusion rule is on the format that "If Variant X is selected, then Variant Y must be selected" while exclusion rule is on the format "if Variant X is selected, then Variant Y can not be selected". Take exclusion rule for instance, in automotive industry, the fuel tank is not made of plastic with high volume fuel content due to the strength of the material. To reflect the constraint that "plastic is not selected for a 40L volume tank", an exclusion rule "If tank volume is 40L, then plastic will not be selected as material" will be added.

2.2 Concept: Allowed Combinations

We describe a vehicle component from different angles to state its distinctive feature, different variant families will be selected to describe a single object. A fuel tank, for instance, could be partially described by its volume, material and color. To completely describe a fuel tank, all these factors will be combined together. A red plastic fuel tank of 20L volume will be presented by a combination as "20L-plastic-red".

A fuel tank example is given (*Table 1*) to further illustrate the concept of *allowed* *combinations.* Three Variant Families, “Fuel tank volume”, “Fuel tank material”, and “Fuel tank color” have been selected to describe a fuel tank. Variants of the three families are as follows:

Table 1 A Fuel Tank example

Tank Volume	Tank Material	Tank Color
10L	Aluminum	Red
20L	Steel	Black
30L	Plastic	
40L		

Given the above variants, totally, there are 24 theoretical combinations, each represents a fuel tank that may be possible to produce. They are listed in *Table 2*

Table 2 Theoretical combination in the small example:

1	10L	Aluminum	Red
2	10L	Aluminum	Black
3	10L	Steel	Red
4	10L	Steel	Black
5	10L	Plastic	Red
6	10L	Plastic	Black
7	20L	Aluminum	Red
8	20L	Aluminum	Black
9	20L	Steel	Red
10	20L	Steel	Black
11	20L	Plastic	Red
12	20L	Plastic	Black
13	30L	Aluminum	Red
14	30L	Aluminum	Black
15	30L	Steel	Red
16	30L	Steel	Black
17	30L	Plastic	Red
18	30L	Plastic	Black
19	40L	Aluminum	Red
20	40L	Aluminum	Black
21	40L	Steel	Red
22	40L	Steel	Black
23	40L	Plastic	Red
24	40L	Plastic	Black

However, not all theoretical combinations are valid. Constraints from different aspects need to be taken into account and they are presented as configuration rules which reduce a large portion of them. The remaining combinations are offered as *allowed combinations* since they could be produced in reality. Still take the previous small example for instance, adding two configuration rules as below:

Table 3 Variant families with constraints

Tank Volume	Tank Material	Tank Color	Configuration Rule
10L	Aluminum	Red	If 20L,then material must be plastic
20L	Steel	Black	If aluminum, then color can not be red
30L	Plastic		
40L			

Given the variant information, there will be 17 allowed combinations as follows:

Table 4 Allowed combination in the small example

1	10L	Aluminum	Black
2	10L	Steel	Red
3	10L	Steel	Black
4	10L	Plastic	Red
5	10L	Plastic	Black
6	20L	Plastic	Black
7	20L	Plastic	Red
8	30L	Aluminum	Black
9	30L	Steel	Red
10	30L	Steel	Black
11	30L	Plastic	Red
12	30L	Plastic	Black
13	40L	Aluminum	Black
14	40L	Steel	Red
15	40L	Steel	Black
16	40L	Plastic	Red
17	40L	Plastic	Black

Allowed combinations is a critical concept in this research as they are the very data the author tries to present in a comprehensive way. However, presenting the allowed combinations is not an easy task, two factors contribute to the complexity of presenting such data in our research: discerning the structure of a hierarchy and presenting the huge number of allowed combinations. Even with the increasing power of computer systems, displaying millions of data in the screen is time consuming. Alternatives to present this huge data and discussions are shown in the next chapter.

3. Compress Combinations in a Logical Way

Normally, engineers are dealing with tens of thousands allowed combinations at one time. Improper variant may be selected in a specific combination due to engineers' knowledge limitation. In order to have an understanding of where the erroneous configurations may be, it is good to have an overview of the complete data that supports a plausible hypothesis. In addition, engineers need to zoom in a specific area, finding the exact combinations to prove whether the previous hypothesis is correct. While filters support to dig in a specific area of information, the tricky issue is how to present the complete combinations on a high level. How presentations for this massive data can be constructed without losing important information?

In mathematics, permutation methodology takes a set of limited values to represent a big number of arrangements of those values into particular order. That is to say, this big set of arrangements could also be represented by a small set of limited values. Inspired from this idea, the author proposes that one approach is to increase scalability by generating an initial compact presentation of the whole data. That is to compress the allowed combinations in a logical way without damaging the relations among variant families. In this research, two methods, Cartesian Product and Identifier Representation are selected for compression. Both two methods are analyzed and compared respectively in the paper.

3.1 Cartesian Product Definition

The Cartesian Product is a mathematical terminology and the definition of Cartesian product A $\times$ B (read "A cross B") of two sets A and B is defined as the set of all ordered pairs (a, b) where a is a member of A and b is a member of B [3]. There are two important notes to be stated in Cartesian Product:

(i) A x B $\neq$ B x A unless A = B

(ii) n(A x B) = n(A) x n(B), where n is a rational number

A small example:

Question:

If A = {3, 5}, B = {2, 4, 6}, write the Cartesian product (i) A x B (ii) B x A

Answer:

A x B = set_AB={(3, 2), (3, 4), (3, 6), (5, 2), (5, 4), (5, 6)}

B x A = set_BA={(2, 3), (2, 5), (4, 3), (4, 5), (6, 3), (6, 5)}

As shown in the example, both set_AB and set_BA could be represented by A x B and B x A respectively. Instead of enumerating every element in set_AB, Cartesian Product presents the group of enumerated elements with A cross B. In comparison, this representation is more compact.

Following this idea, the list of allowed combinations could also be presented by Cartesian Product with a set of variant families. The previous fuel tank example supports to interpret how Cartesian Product presents the allowed combinations. The transformation from allowed combinations to Cartesian Product Representation is also explained.

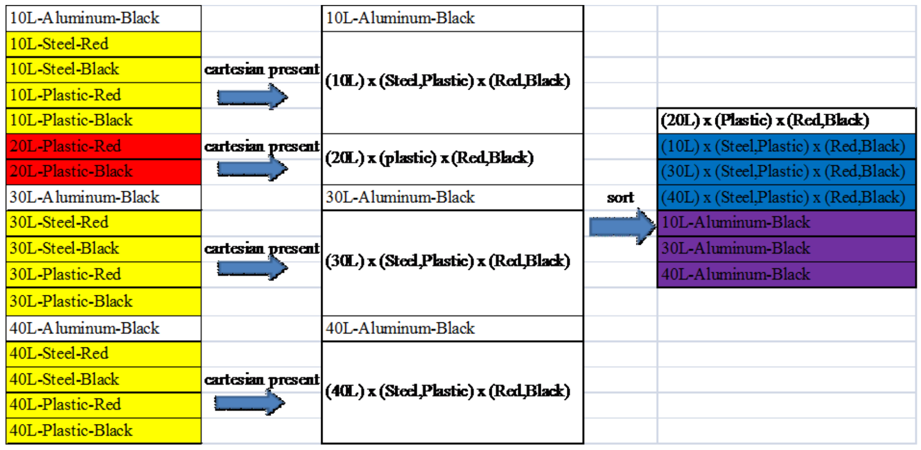
Table 5 Allowed combination in the small example

1	10L	Aluminum	Black
2	10L	Steel	Red
3	10L	Steel	Black
4	10L	Plastic	Red
5	10L	Plastic	Black
6	20L	Plastic	Black
7	20L	Plastic	Red
8	30L	Aluminum	Black
9	30L	Steel	Red
10	30L	Steel	Black
11	30L	Plastic	Red
12	30L	Plastic	Black
13	40L	Aluminum	Black
14	40L	Steel	Red
15	40L	Steel	Black
16	40L	Plastic	Red
17	40L	Plastic	Black

The analysis starts from the second row as the first row is single and currently can not be grouped with others. The four records start from the second to the fifth are Cartesian product of three sets (10L), (Steel, Plastic) and (Black, Red). Likewise, the sixth and the seventh rows are Cartesian Product of three sets, (20L), (Plastic) and (Black, Red). Following this, the transform from the 17 string-based combinations to Cartesian Product Representation would be as follows:

Step 1: fix variant in VF *volume* and use Cartesian Product to group VF *material* and *color*

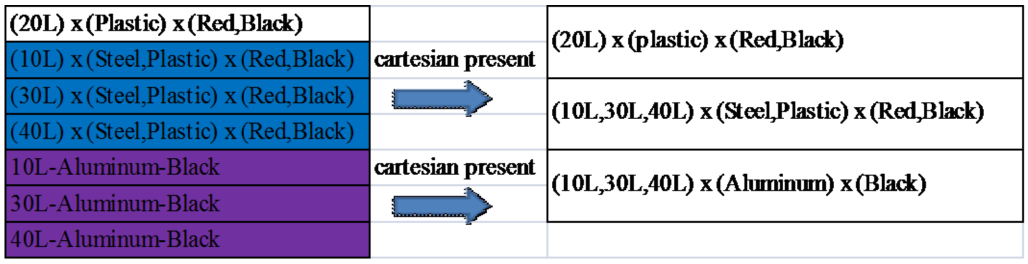
Table 6 Transform from allowed combination to Cartesian Product (1)



10L-Aluminum-Black		10L-Aluminum-Black		
10L-Steel-Red				
10L-Steel-Black	cartesian present	(10L) x (Steel,Plastic) x (Red,Black)		
10L-Plastic-Red				
10L-Plastic-Black				(20L) x (Plastic) x (Red,Black)
20L-Plastic-Red	cartesian present	(20L) x (plastic) x (Red,Black)		(10L) x (Steel,Plastic) x (Red,Black)
20L-Plastic-Black				(30L) x (Steel,Plastic) x (Red,Black)
30L-Aluminum-Black		30L-Aluminum-Black	sort	(40L) x (Steel,Plastic) x (Red,Black)
30L-Steel-Red				10L-Aluminum-Black
30L-Steel-Black	cartesian present	(30L) x (Steel,Plastic) x (Red,Black)		30L-Aluminum-Black
30L-Plastic-Red				40L-Aluminum-Black
30L-Plastic-Black				
40L-Aluminum-Black		40L-Aluminum-Black		
40L-Steel-Red				
40L-Steel-Black	cartesian present	(40L) x (Steel,Plastic) x (Red,Black)		
40L-Plastic-Red				
40L-Plastic-Black				

Step 2: fix variant in volume *material* and *color*, use Cartesian Product to group VF *volume*

Table 7 Transform from allowed combination to Cartesian Product (2)



(20L) x (Plastic) x (Red,Black)		**(20L) x (plastic) x (Red,Black)**
(10L) x (Steel,Plastic) x (Red,Black)	**cartesian present**	
(30L) x (Steel,Plastic) x (Red,Black)	➡	**(10L,30L,40L) x (Steel,Plastic) x (Red,Black)**
(40L) x (Steel,Plastic) x (Red,Black)		
10L-Aluminum-Black	**cartesian present**	**(10L,30L,40L) x (Aluminum) x (Black)**
30L-Aluminum-Black	➡	
40L-Aluminum-Black		

In this stage, a new question comes to our mind, when should the compression be terminated. The answer is that the compression does not stop until all combinations differ from each other in more than one column. Take the above example for instance, “x” symbol separates compressed combinations into three columns (*Table 8*)

Table 8 Three compressed allowed combination in the small example

Combination Number	Volume (Column_1)	Material (Column_2)	Color (Column_3)
1	20L	Plastic	(Red, Black)
2	(10L,30L, 40L)	(Steel, Plastic)	(Red, Black)
3	(10L,30L, 40L)	Aluminum	Black

In the table, Column_1 and Column_2 of the first row differ from the ones of the second combination while the first row differs from the third in all three columns. On the other hand, the second row differs from the third in both Column_2 and Column_3. Consequently, all three compressed combinations differ with each other in more than one column, Cartesian Product compression stops then.

With *Cartesian Product Representation*, the 17 allowed combinations have been compressed to 3 entries, providing a compact overview to the engineers. More combinations could be manipulated and presented with this method.

However, implementation of *Cartesian Product Representation* in this research is complex. Normally, to determine whether a list of combinations could be presented by Cartesian Product, the first step is to divide the list into several smaller portions. Secondly, it is to construct proper Cartesian Product sets that present each portion and then finally combine all sets to present the complete data. The tricky part is to properly construct the Cartesian Product sets. One approach is to firstly construct Cartesian Products sets with a few variants that present some of the combinations and then gradually fill in more variants in the previous sets that present more combinations until the sets cover all combinations. Yet, the criterion of picking variants to construct the Cartesian Product set to present certain combinations is not obvious. It will be more complex and time-consuming when it is dealing with a large number of combinations consisting of multiple variant families.

3.2 Identifier Representation

A new approach, *Identifier Representation* is introduced in this section. Based on this self-created idea, two different but similar algorithms are developed and compared to compress the allowed combinations. Before introducing the two algorithms, three terms *Identifier, Partial Combination* and *Pattern* should be firstly studied:

Identifier:	a set of variants in the same variant family. It could either contain all the variants in the variant family or contain a subset.
Partial Combination:	a combination that removes some variant families. It is a portion of the original combination.
Pattern:	a set of partial combinations.

The formats of an *Identifier* and a *Pattern* are as follows:
Identifier: {VF_Name}_{Identifier_Sequence}_{Number of Variant}.
Pattern: {Variant_Name}_{Pattern_Sequence}_{Number of Partial Combination}.

Though the formats of the two terms are quite similar, they represent different content. An identifier represents a group of variants while a pattern represents a group of partial combinations.

Take the combinations in *Table 8* for example, the set (Red, Black) in Column_3 could be presented as Identifier *Color_1_2*. "1" shows this is the first identifier in Variant Family "color" while "2" shows there are two variants, *red* and *black* in it. Likewise, Identifier *Material_1_2* could present the set (Steel, Plastic) while Identifier *Volume_1_3* presents set (10L,20L,30L). *Table 8* would then be changed to

Table 9 Compressed combination by Identifier

Combination Number	Volume (Column_1)	Material (Column_2)	Color (Column_3)
1	20L	Plastic	Color_1_2
2	Volume_1_3	Material_1_2	Color_1_2
3	Volume_1_3	Aluminum	Black

If we remove Volume (Column_1), the new table will then be ;

Table 10 Compressed combinations without *Volume* variant family

Combination Number	Material (Column_2)	Color (Column_3)	
1	Plastic	Color_1_2	
2	Material_1_2	Color_1_2	pattern, {Volume_1_3}_1_2
3	Aluminum	Black	

The first row *Plastic-{color_1_2}* is a partial combination of *20L-Plastic-{color_1_2}*. On the other hand, the second and the third rows are represented by a pattern, *{Volume_1_3}_1_2*. In the pattern, “1” shows it is the first pattern of variant “Volume_1_3” while “2” shows there are two partial combinations in it.

If there are two combinations that differ in only one column, then two records can be compressed to a new combination with an identifier. The compression could either be based on one variant family or multiple variant families.

3.2.2 Compression Based on One Variant Family

Method Description:

1. Determine Variant Family order. Sort the combinations and divide them into m blocks. From block i_1 to i_{m-1}, the combinations differ from others of the same block in only one column. The remaining combinations that differ all the others in more than one column would be sorted and stored in block i_m.
2. From block i_1 to i_{m-1}, use identifiers to compress the allowed combinations.
3. Combine the compressed combinations with the records in block i_m. If possible, generate new identifiers to recompress them. The compression does not stop until all combinations differ from others in more than one column.

For the first five rows in *Table 5*, the first row is excluded as it is single and can not be grouped with the rest four. Combinations from the second to the fifth are sorted and divided into two blocks as follows:

Table 11 Four combinations in two blocks

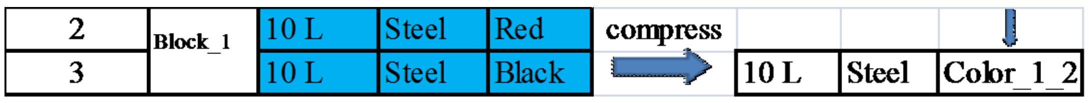
2	Block_1	10L	Steel	Red
3		10L	Steel	Black
4	Block_2	10L	Plastic	Red
5		10L	Plastic	Black

The compression starts from right to left:

Step 1: Determine Variant Family Order and divide combinations into blocks. In this example, variant families are ordered as *volume*, *material* and *color*. The combinations are divided into two blocks shown in *Table 11*

Step 2.1: use Identifier *Color_1_2* which represents a set of two variants (Red, Black) to compress the combinations in Block 1

Table 12 compression in Block_1

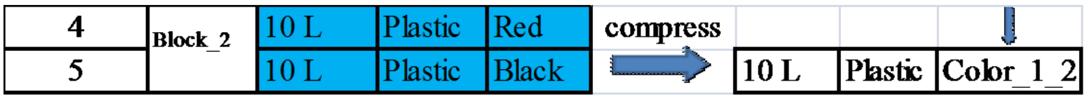


2	Block_1	10 L	Steel	Red	compress
3		10 L	Steel	Black	

10 L	Steel	Color_1_2

Step 2.2: still use Identifier *Color_1_2* to compress the combinations in Block 2

Table 13 compression in Block_2



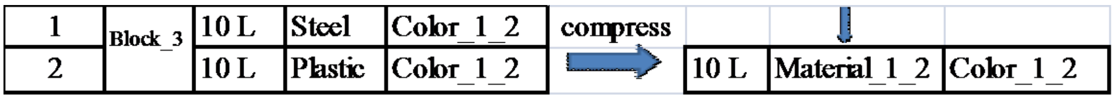
4	Block_2	10 L	Plastic	Red
5		10 L	Plastic	Black

compress

10 L	Plastic	Color_1_2

Step 3: combine the compressed records and recompress with identifier *Material-1-2* which represents a set of two variants (Steel, Plastic)

Table 14 compression in Block_3



1	Block_3	10 L	Steel	Color_1_2
2		10 L	Plastic	Color_1_2

compress

10 L	Material_1_2	Color_1_2

As shown in the above, the arrow moves from the third column to the second, right to left, during the compression. It firstly compresses the records in Block_1 whose combinations differ in the third column. Likewise, the same procedure happens in Block_2. After combining the compressed combinations, the compression occurs in Block_3 whose records differ in the second column. The compression stops when all combinations differ in more than one column. In this example, there is only one single combination left, the compression stops.

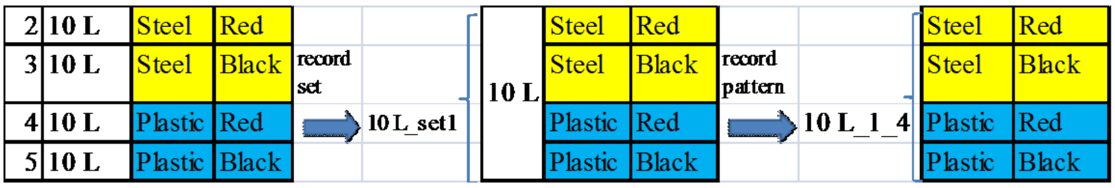
3.2.3 Compression Based on Multiple Variant Families

Method Description:

1. Starting from the first variant V_1 in the first Variant Family VF$_1$, record all combinations begin with V_1 as a set, $V_1\_set_1$. In $V_1\_set_1$, the combinations that have removed variant V_1 are grouped as a pattern, $V_1\_pattern_1$. The process applies to all the variants in VF_1. The last variant V_m of VF$_1$ have a set $V_m\_set_m$ and a pattern $V_m\_pattern_m$.
2. Use identifier to compress the variants in VF$_1$ that have identical pattern.
3. Starting from the first set $V_1\_set_1$, follow the procedures from step1 to step 2 to compress it. The compression applies to all the sets from $V_1\_set_1$ to $V_m\_set_m$.
4. Combine the combinations that represent the compressed sets. If possible, generate new identifiers to recompress them. The compression does not stop until all combinations differ in more than one column.

Take the combinations in *Table 11* for example, the compression starts from left to right:

Step 1: from variant "10L", record set *10L_set1* and pattern *10L_1_4*



2	10 L	Steel	Red
3	10 L	Steel	Black
4	10 L	Plastic	Red
5	10 L	Plastic	Black

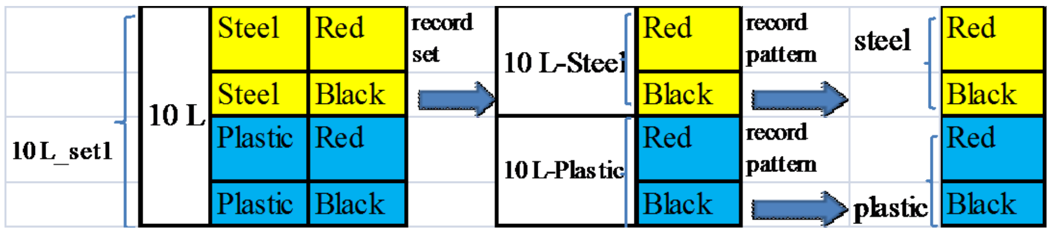
record set → 10 L_set1

10 L	Steel	Red
	Steel	Black
	Plastic	Red
	Plastic	Black

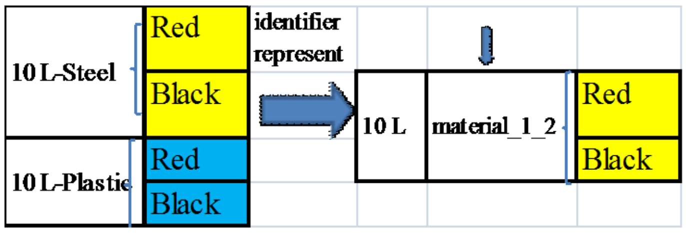
record pattern → 10 L_1_4

Steel	Red
Steel	Black
Plastic	Red
Plastic	Black

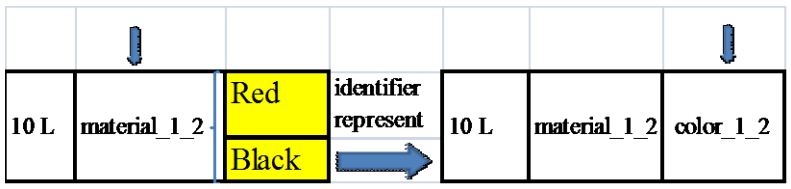
Step 2: divide set *10L_set1* into another two sets, *10L-Steel_set1* and *10L-Plastic_set1*. Record corresponding patterns *steel_1_2* and *plastic_1_2* for these two sets.   ,



Step 3: As two sets, *10L-Steel_set1* and *10L-Plastic_set1* have identical pattern, two sets can be compressed by identifier *material_1_2*



Step 4: compress the pattern *steel_1_2*(same as pattern *plastic_1_2)* with identifier *color_1_2*



As shown, the arrow moves from the second column to the third, left to right, during the compression. It firstly compresses the variants, steel and plastic, in sets *10L-steel_set1* and *10L-plastic_set1* as the two sets have identical patterns. Then an identifier *color_1_2* is selected to compress the two combinations as they only differ in the third column. The compression stops when all combinations differ with each other in more than one column. In this example, there is only one combination left and the compression stops.

3.2.4 Identifier Compression Method Selection

In general, two methods are similar. Both methods compress two combinations when they differ in only one column. The first method checks the condition based on one column, while the latter checks it based on a pattern, multiple columns. Assume there are three combinations with three variant families and all combinations differ in the second column. Compressions of each method are as follows:

Table 15 Compression based on one VF. Identifier *VF2_1_2* presents (V21,V22), Identifier VF2_2_2 presents (VF2_1_2,V23)

Number	VF1	VF2	VF3
1	V1	V21	V3
2	V1	V22	V3
3	V1	V23	V3

compress $\Rightarrow$

Number	VF1	VF2	VF3
1	V1	VF2_1_2	V3
2	V1	V23	V3

compress $\Rightarrow$

Number	VF1	VF2	VF3
1	V1	VF2_2_2	V3

Table 16 Compression based on multiple VF. Identifier VF2_1_3 presents (V21, V22, V23)

Number	VF1	VF2	VF3
1	V1	V21	V3
2	V1	V22	V3
3	V1	V23	V3

compress $\Rightarrow$

Number	VF1	VF2	VF3
1	V1	VF2_1_3	V3

Though the final results in both methods are actually the same, compression based on multiple VF is more efficient. As shown, the first method takes two times with two identifiers to compress three combinations while the second method only takes one time with one identifier. The effects will be more impressive when it deals with a larger data set. Thereby, we will use Identifier to compress allowed combinations based on multiple Variant Families in this research.

Additionally, we can observe another interesting fact in this example. There is a balance between compression ratio and the number of identifiers. By removing the limitation that one identifier cannot contain another identifier of the same VF, it leads to the maximal compression ratio by generating more identifiers.

3.2.5 Variant Family Order Selection

If an object is represented by a combination consisting of N variant families, totally there will be N! ways to represent this object due to multiple variant family orders. This section is to discuss how different variant family orders impact the compression.

Let us assume that there is a VF_A, and a set of variants from VF_A, named A. Consequently, there is at least a pair of two subsets of A, A_1 and A_2, where A_1 and A_2 do not have duplicate variants, and A is the union set of the two. Suppose there is another pair of two subsets, A_3 and A_4, where A_3 and A_4 do not have duplicates variants either and A is also the union set of A_3 and A_4. Hence, set A could be either presented by ($A_1 \cup A_2$) or ($A_3 \cup A_4$). In either form, set A will be the most compact representation that covers all the variants from VF_A. Thereby, with the Identifier representation, certain list of combinations will be compressed to the same result no matter what transient identifiers are used. Different variant family orders do not impact the final compression ratio.

Yet, different variant family orders impact the compression time. The easier to find out whether different sets have identical patterns, the less time compression would take. In this research, the Variant Family that has the most number of variants will be displayed in the leftmost columns.

3.2.6 Case Study

This section is to present four cases which have been successfully compressed using identifiers. In real work, there could be either a list of combinations with a few variant families or with many variant families. Case one covers the first scenario while the second case covers the second scenario. Case three and case four are two big cases that have successfully been compressed. They are provided to test the scalability of the identifier compression method. The complete final results of the compressed results are provided in both case one and case two while excepts of case three and case four are given due to the space limitation. Identifiers used in these four cases are not provided in all four cases.

Case 1

Table 17 Compress 2854 combinations of 7 VF to 8 entries

YDX-1-51	Z9X-4-7	YAX-1-3	R	X5X-1-2	F30	RA
RL1345	Z9X-1-4	YAX-2-2	R	X5X-1-2	F30	RA
YDX-1-51	Z9X-2-3	YAX-2-2	R	X5X-1-2	F20	RA
RL1345	Z9X-2-3	YAX-2-2	R	X5X-1-2	YBX-1-2	RA
RL2365	FIL-EEEB	YAX-2-2	T	X5X-1-2	F30	YLX-1-2
RL2175	FIL-EEEB	YAX-2-2	T	UFRACLOS	F30	YLX-1-2
RL2365	Z9X-3-2	YAX-2-2	T	X5X-1-2	YBX-1-2	YLX-1-2
RL2175	Z9X-3-2	YAX-2-2	T	UFRACLOS	YBX-1-2	YLX-1-2

Case 2

Table 18 Compress 3932 combinations of 15 VF to 22 entries

PDC-OFF	FAA11	KEX-1-2	BBOX-L	USUP	KSX-1-3	TNK-SING	UADCHAS	UTFUEL	UFRF	FAX-1-2	MTNK-R	KFX-1-2	DDX-1-2	F30
PDC-OFF	FAA10	KEX-1-2	BBOX-L	YRX-2-2	KSX-1-3	KDX-1-2	UADCHAS	UTFUEL	UFRF	FAX-1-2	7VB-1-2	KFX-1-2	DDX-1-2	YBX-1-2
PDC-OFF	FAA10	FTANK-PL	BBOX-L	YRX-2-2	KSX-1-3	TNK-SING	UADCHAS	TFUEL100	UFRF	FAX-1-2	MTNK-R	FCAP-UL	DDX-1-2	YBX-1-2
PDC-OFM	FAA11	KEX-1-2	BBOX-L	USUP	KSX-1-3	TNK-SING	UADCHAS	UTFUEL	FRFS-BS	FAX-1-2	MTNK-R	KFX-1-2	R	F30
PDC-OFR	FAA20	KEX-1-2	BBOX-L	YRX-1-3	KSX-1-3	TNK-SING	UADCHAS	UTFUEL	Q9A-1-2	FAX-1-2	MTNK-R	KFX-1-2	R	F30
PDC-OFR	FAA21	KEX-1-2	BBOX-L	USUP	KSX-1-3	TNK-SING	UADCHAS	UTFUEL	Q9A-1-2	FAX-1-2	MTNK-R	KFX-1-2	R	F30
PDC-OFM	FAA10	KEX-1-2	BBOX-L	YRX-2-2	KSX-1-3	KDX-1-2	UADCHAS	UTFUEL	FRFS-BS	FAX-1-2	7VB-1-2	KFX-1-2	R	YBX-1-2
PDC-OFM	FAA10	FTANK-PL	BBOX-L	YRX-2-2	KSX-1-3	TNK-SING	UADCHAS	TFUEL100	FRFS-BS	FAX-1-2	MTNK-R	FCAP-UL	R	YBX-1-2
PDC-OFF	FAA10	KEX-1-2	BBOX-AC	YRX-1-3	KSX-1-3	KDX-1-2	UADCHAS	UTFUEL	UFRF	FAX-1-2	7VB-1-2	KFX-1-2	T	F30
PDC-OFF	FAA10	FTANK-AL	BBOX-EF	YRX-3-2	KSX-1-3	KDX-1-2	UADCHAS	UTFUEL	UFRF	FAX-1-2	MTNK-L	KFX-1-2	T	F30
PDC-OFF	FAA10	FTANK-PL	BBOX-AC	YRX-1-3	KSX-1-3	TNK-SING	UADCHAS	TFUEL100	UFRF	STWPOS-L	7VB-1-2	FCAP-UL	T	F30
PDC-OFF	FAA10	FTANK-PL	BBOX-EF	YRX-3-2	KSX-1-3	TNK-SING	UADCHAS	TFUEL100	UFRF	FAX-1-2	MTNK-R	FCAP-UL	T	F30
PDC-OFR	FAA20	FTANK-AL	BBOX-AC	USUP	EAS-SCR	TNK-DUAL	UADCHAS	UTFUEL	UFRF	STWPOS-R	7VB-1-2	FCAP-L	T	F30
PDC-IF	FAA10	FTANK-AL	BBOX-EF	YRX-3-2	KSX-1-3	KDX-1-2	UADCHAS	UTFUEL	UFRF	FAX-1-2	MTNK-L	KFX-1-2	T	YBX-1-2
PDC-IF	FAA10	FTANK-AL	BBOX-EF	SUP-BAS	KSX-1-3	KDX-1-2	4TX-1-2	UTFUEL	UFRF	FAX-1-2	MTNK-L	KFX-1-2	T	YBX-1-2
PDC-IF	FAA10	FTANK-AL	L4X-1-2	YRX-3-2	KSX-1-3	KDX-1-2	UADCHAS	UTFUEL	UFRF	FAX-1-2	7VB-1-2	KFX-1-2	T	YBX-1-2
PDC-IF	FAA10	FTANK-AL	L4X-1-2	SUP-BAS	KSX-1-3	KDX-1-2	4TX-1-2	UTFUEL	UFRF	FAX-1-2	7VB-1-2	KFX-1-2	T	YBX-1-2
PDC-IF	FAA10	FTANK-PL	L4X-2-3	YRX-3-2	KSX-1-3	TNK-SING	UADCHAS	TFUEL100	UFRF	FAX-1-2	MTNK-R	FCAP-UL	T	YBX-1-2
PDC-IF	FAA10	FTANK-PL	L4X-2-3	SUP-BAS	KSX-1-3	TNK-SING	4TX-1-2	TFUEL100	UFRF	FAX-1-2	MTNK-R	FCAP-UL	T	YBX-1-2
PDC-IF	FAA10	FTANK-ST	L4X-1-2	YRX-3-2	KSX-1-3	KDX-1-2	UADCHAS	UTFUEL	UFRF	FAX-1-2	7VB-1-2	KFX-1-2	T	YBX-1-2
PDC-IF	FAA10	FTANK-ST	L4X-1-2	SUP-BAS	KSX-1-3	KDX-1-2	4TX-1-2	UTFUEL	UFRF	FAX-1-2	7VB-1-2	KFX-1-2	T	YBX-1-2
PDC-OFF	FAA10	KEX-1-2	BBOX-L	SUP-LOW	KSX-1-3	KDX-1-2	UADCHAS	UTFUEL	UFRF	FAX-1-2	7VB-1-2	KFX-1-2	T	YBX-1-2
PDC-OFF	FAA10	FTANK-PL	BBOX-L	SUP-LOW	KSX-1-3	TNK-SING	UADCHAS	TFUEL100	UFRF	FAX-1-2	MTNK-R	FCAP-UL	T	YBX-1-2

Case 3

Table 19 Compress 39650 combinations of 7 VF to 108 entries – except of final result

YDX-25-8	RAD-GR	Z9X-9-2	YAX-2-3	RFEC-L	F30	UFRACLOS
RL825	RAD-G2	Z9X-3-3	YAX-2-3	X6X-1-2	F30	UFRACLOS
RL825	RAD-GR	Z9X-5-4	YAX-2-3	X6X-1-2	F30	UFRACLOS
RL825	RAD-G2	Z9X-4-4	YAX-2-3	X6X-2-3	F30	UFRACLOS
YDX-10-30	RADD-GR	UFIL	FST8080	RFEC-L	YBX-1-2	UFRACLOS
YDX-31-2	RADD-GR	UFIL	FST8080	X6X-2-3	YBX-1-2	UFRACLOS
YDX-10-30	RADD-GR	FIL-TXEB	YAX-1-2	RFEC-L	YBX-1-2	UFRACLOS
YDX-25-8	RAD-GR	Z9X-8-2	YAX-1-2	RFEC-L	YBX-1-2	UFRACLOS
RL825	RAD-GR	Z9X-7-2	YAX-1-2	X6X-1-2	YBX-1-2	UFRACLOS
YDX-31-2	RADD-GR	FIL-TXEB	YAX-1-2	X6X-2-3	YBX-1-2	UFRACLOS
RL825	RAD-GR	UFIL	YAX-1-2	X6X-3-3	YBX-1-2	UFRACLOS
YDX-15-23	RAD-GR	FIL-TXEF	FST6060	X6X-1-2	F30	X5X-1-2
YDX-21-19	RAD-GR	Z9X-6-3	FST6060	X6X-1-2	F30	X5X-1-2
YDX-14-23	RAD-GR	Z9X-8-2	FST6060	X6X-2-3	F30	X5X-1-2

Case 4

Table 20 Compress 64955 combinations of 5 VF to 398 entries – except of final result

TAX-69-3	HHX-1-7	DKX-2-2	VT2214B	T-FLAT
RAT3.61	HHX-1-7	DKX-27-2	VT2214B	QCX-1-3
RAT3.61	HHX-1-7	GCW32.0	VT2214B	QCX-3-4
TAX-63-5	HHX-1-7	DKX-2-2	VT2214B	QCX-1-3
RAT5.41	HHX-1-7	DKX-1-4	VT2214B	QCX-3-4
RAT5.41	HHX-1-7	DKX-5-3	VT2214B	QCX-1-3
RAT7.21	ETOR2180	DKX-2-2	VT2214B	T-FLAT
RAT7.21	HHX-11-2	DKX-2-2	VT2214B	T-FLAT
RAT7.21	HHX-2-3	DKX-1-4	VT2214B	QCX-1-3
RAT7.21	HHX-2-3	GCW50.0	VT2214B	QCX-2-2
RAT7.21	HHX-15-7	DKX-1-4	VT2214B	QCX-3-4
TAX-71-5	HHX-2-3	GCW40.0	VT2214B	QCX-1-3
TAX-4-2	ETOR2400	DKX-5-3	VT2214B	QCX-1-3
TAX-4-2	HHX-1-7	DKX-21-2	VT2214B	QCX-1-3
TAX-70-4	HHX-2-3	DKX-10-3	VT2214B	QCX-1-3
TAX-68-3	HHX-1-7	DKX-10-3	VT2214B	QCX-3-4
TAX-67-4	HHX-1-7	GCW56.0	VT2214B	QCX-1-3

It is impressive that all the four cases have been compressed with a good ratio. The detailed information is listed in *Table 21*

Table 21 Compressed Case Analysis

Combination_Number	VF_Number	Identifier_Number	Compressed_Number	Compression_Ratio
2854	7	10	8	0.0028
3932	15	15	22	0.0056
39650	7	57	108	0.0027
64955	5	78	398	0.0061

Compressed by identifiers, the number of manageable combinations have increased from thousands to tens of thousands. As there are at least two variants in one identifier, the analysis and conclusion for a single variant holds for other variants of the same identifier. Data analysis turns to be flexible and accurate accordingly.

In addition, with compressed data, users could detect the hidden information among the Variant Family easily. Take *Table 17* for instance, the Identifier *YLX-1-2* contains two variants *RA* and *RL*. It is interesting to notice that all combinations begin with variant *R* are combined with variant *RA* while the combinations begin with variant *T* are combined with both *RA* and *RL*. Some users may be interested about this hidden information. Similar findings could be discovered in rest of the three cases that prove compression supports the users to get more hidden information among allowed combinations.

3.3 Discussion and Conclusion

The first task in this research is to present the huge data in an user-friendly and comprehensive way. To be more specific, three more factors should be taken into account to fully evaluate the two methods:

1. the comprehensiveness of the compressed combinations
2. the complexity to implement the method in practice
3. the capacity to deal with huge data in practice

Accordingly, three criteria are set up to evaluate above factors:

1. the data format of the compressed combinations
2. the complexity of the key component to implement the method
3. the scalability of the method

Evaluations for the criteria in two methods are listed in *Table 22*

Table 22 Evaluations of Identifier Product and Cartesian Product method

Criteria	Cartesian Compression	Identifier Compression	Winner Method
Data Format	(20L) x (plastic) x (red, black)	20L–plastic-{color_1_2}	Cartesian Compression
Key component /complexity	Formulate Cartesian Product sets present certain combinations / Hard	Find combinations that differ only in one field / Medium	Identifier Compression
Scalability	Medium	Good	Identifier Compression

The Cartesian Product compression fills the variants in the sets to present the combinations while the Identifier method could not provide variant information to the users directly. Users have to learn the variants behind the identifiers by heart. The content in the Cartesian method is more comprehensive. However, from the point of implementation, Identifier Compression is superior to Cartesian Compression. In

Identifier Compression, the process of checking compression condition can be well structured and consequently, it is easier to implement. Moreover, since Identifier is easier to implement, it guarantees its capability to deal with bigger data in practice, a better scalability.

Due to the high importance to present the huge data in practice, we select Identifier Compression to present the compact overview of data in our research, and the compression is based on multiple variant families. Identifier Compression’s weakness in comprehensiveness will be made up with a technique that works well at navigating and querying the compressed combinations. Details are given in next chapter.

4. Visualize the Compressed Combinations

In the last decade, data visualization techniques have proven to be valuable in huge data analysis. It successfully combines human recognition capabilities with the ever increasing power of computer systems to detect the patterns and trends in the data [4]. These hidden knowledge are used to identify bottlenecks, errors or any other interesting information among the data. In addition, data visualization adds aesthetic value to originally lifeless data, making information communication clear through graphical representations.

As stated, Identifier compression has been selected to present the allowed combinations. The second task in this research is then changed to find techniques to efficiently navigate and query the compressed combinations. Several popular visualization techniques are studied and evaluated to accomplish the goal in this chapter.

4.1 Treemap

Treemap is a space-filling approach based in dividing a display into nested rectangles, each with an area that corresponds to a weight associated with the node [5]. It is a hot visualization technique for displaying hierarchically structured data. Directory structures, internet news are some of the common applications of treemaps. *Figure 1* shows one example that helps users to navigate hard disk content.

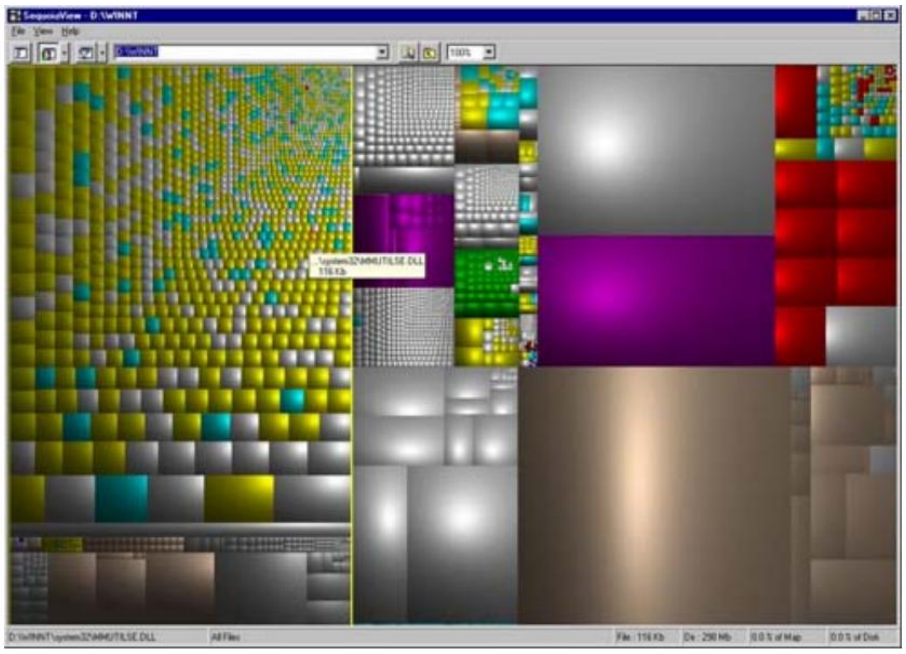


Figure 1 SequoiaView uses treemaps to show the content of hard drive. Area indicates file size and color shows file type [6]

An allowed combination is a string format data consist of variants from corresponding variant families. In visualization, the variant of the first variant family could be viewed as "above" the variant of the second variant family while the variants from the same variant family are considered as "at the same level". Hence, a list of compressed combinations can be constructed to a "hierarchically" structured dataset, and treemap can be used to drill down in the combinations and visualize the corresponding area. A possible application may be *Figure 2*.

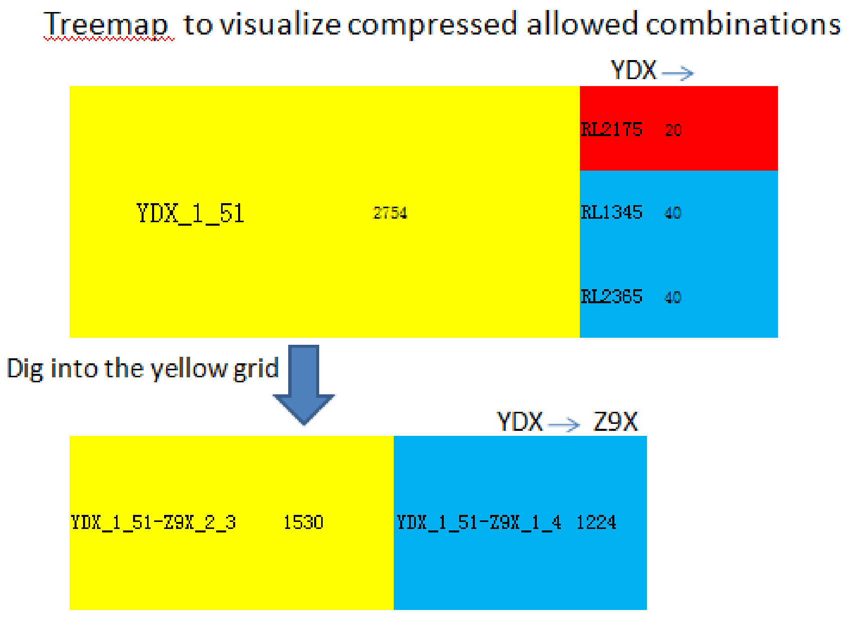


Figure 2 Use Treemap to dig the compressed combinations

In each grid, the figure follows every variant represents the number of combinations that contain this specific variant. When the cursor points at the specific grid, users could get a summary of the combinations in it. Users could zoom in and digs deeper inside the grid by clicking at the grid. On the top of each grid, there is a pointer floating and indicating the user what level of variant family he is currently in. This pointer also helps the user to navigate the data. Users could go back to an upper level or drill down to a lower level by clicking the pointer.

4.2 Tree Structure

A **tree structure** is a way of representing the hierarchical nature of a structure in a graphical form. The "root" resides at the top while the leaves reside at the bottom.

As stated, the combinations could be viewed as "hierarchically" structured data in visualization, a tree structured data in this case. Variants of the same family are at the same height in the tree structure and the variant of the posterior family is the "child" of its preceding variant. The identifier that contains all variants of the family will be presented as a circle instead of a rectangular. A tree structured example that visualizes the compressed combinations is given as below

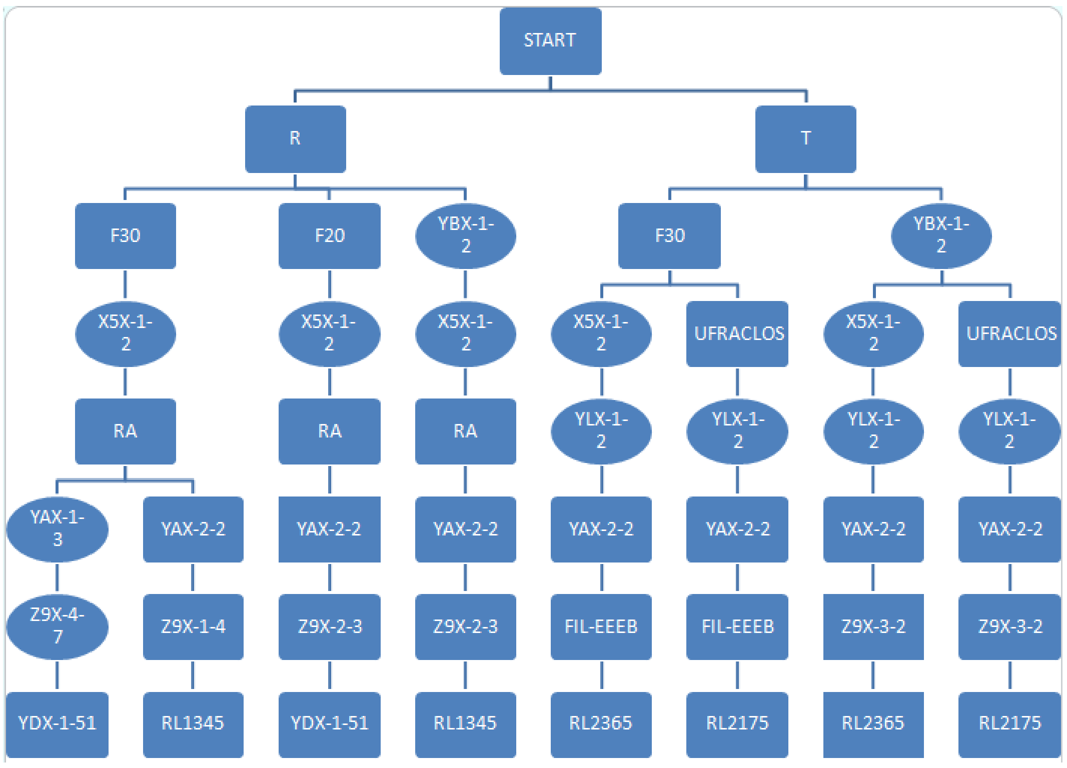


Figure 3 Tree structure to present the compressed combination

Elaborations such as pruning of sub-structures of the tree are also available. A compact variation is that same nodes at the same level could sometimes be grouped. The variation is presented in *Figure 4*.

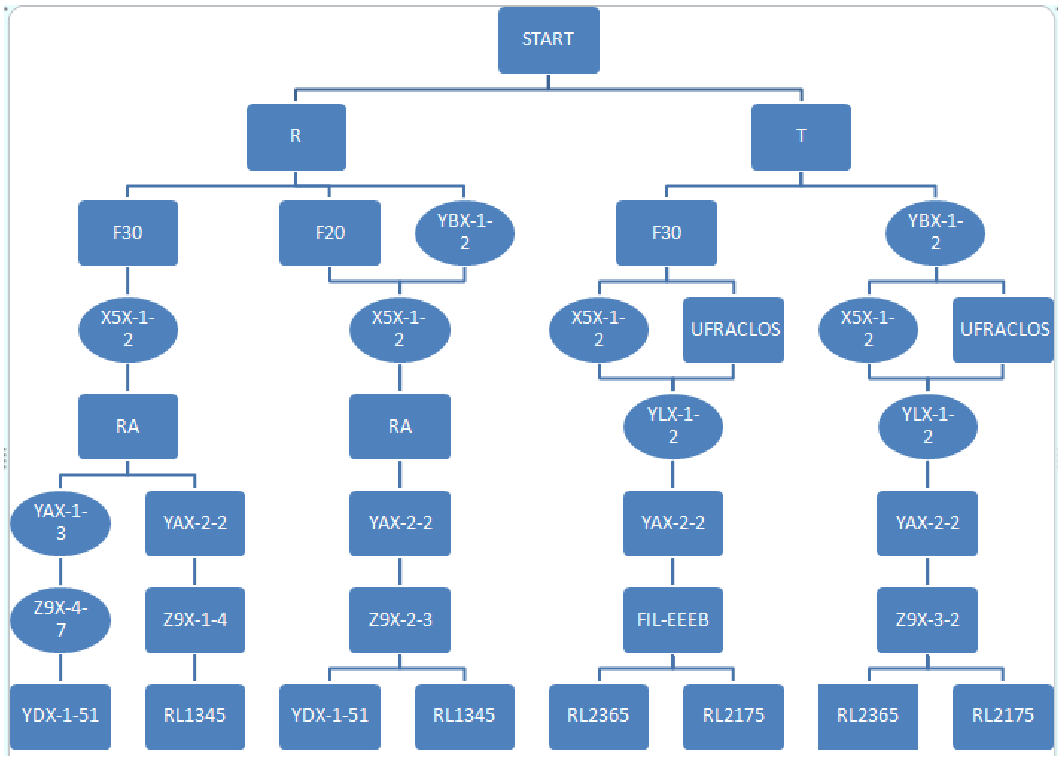


Figure 4 A compact tree structure to present the compressed combination

4.3 Elastic List to Randomly Filter Specific Combination

Elastic lists allow users to navigate large, multi-dimensional information with just a few clicks. By selecting a value and filter the results, users navigate and explore desired information through an iterative refining process. They enhance traditional UI approaches for facet browsers by visualizing weight proportions, animated transitions, emphasis of characteristic values and sparkline visualizations [7]. The process is illustrated in *Figure 5*

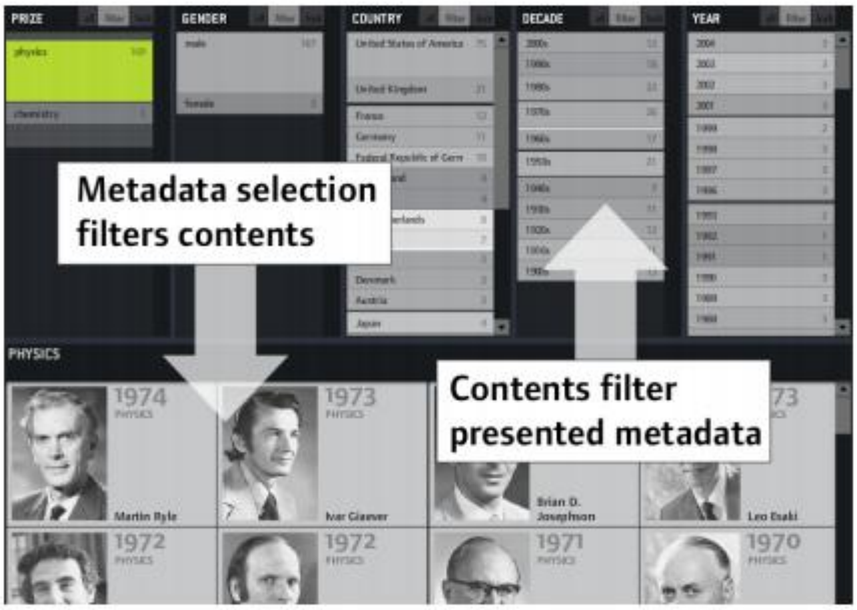


Figure 5 Facet browsing principle [8]

Inspired from this, *Figure 6* illustrates how elastic list can help to visualize the compressed combinations.

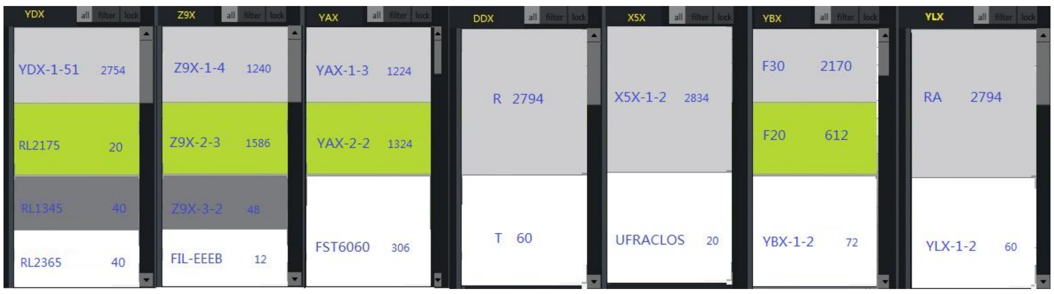


Figure 6 Elastic List to randomly filter specific combination

Each Variant Family is presented as a list, separated by its variants. The number of the combinations that contain this specific variant is listed after the variant name. Let us assume in the initial navigation, the variant *R* is selected in variant family *DDX*. This restricts the display of contents to those combinations matching this value. Accordingly, all metadata attribute are restricted only to variants occurring together with *R*. By subsequent filtering, users gradually narrow down to the target information, making it impossible to construct queries with an empty result. This is commonly considered as one of the best benefits of this method. In general, elastic list is easy to manipulate and it provides a good style for users to navigate and query big data.

4.4 Combination Shot

Interactive History Timeline [9] divides the history of the United Kingdom, in this example, into several interactive blocks. In this system, each colorful block reflects a historical period and the white spots represent the major events in that period. When users zoom in each block, images of the events in this period will appear in the background. Additionally, information about this event will be given when clicking at the white points. Following this, a similar visualization is given in *Figure 7* to navigate and query the compressed combinations.

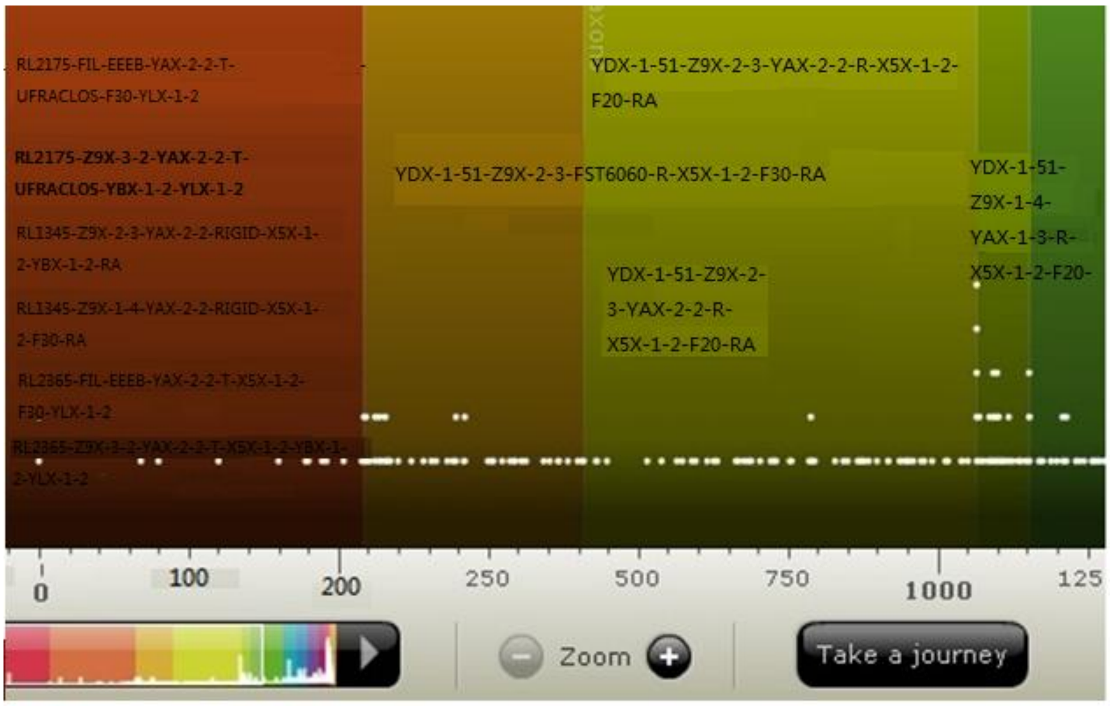

RL2175-FIL-EEEB-YAX-2-2-T-UFRACLOS-F30-YLX-1-2
RL2175-Z9X-3-2-YAX-2-2-T-UFRACLOS-YBX-1-2-YLX-1-2
RL1345-Z9X-2-3-YAX-2-2-RIGID-X5X-1-2-YBX-1-2-RA
RL1345-Z9X-1-4-YAX-2-2-RIGID-X5X-1-2-F30-RA
RL2365-FIL-EEEB-YAX-2-2-T-X5X-1-2-F30-YLX-1-2
RL2365-Z9X-3-2-YAX-2-2-T-X5X-1-2-YBX-1-2-YLX-1-2
YDX-1-51-Z9X-2-3-YAX-2-2-R-X5X-1-2-F20-RA
YDX-1-51-Z9X-2-3-FST6060-R-X5X-1-2-F30-RA
YDX-1-51-Z9X-1-4-YAX-1-3-R-X5X-1-2-F20-
YDX-1-51-Z9X-2-3-YAX-2-2-R-X5X-1-2-F20-RA
0
100
200
250
500
750
1000
125
Zoom
Take a journey

Figure 7 Combination Shot to navigate compressed combination

Compressed combinations are presented as nodes and they are distributed over the plane. The location and the size of the nodes are determined based on how many combinations have been presented by this single node. The darker color of area nodes reside in, the more combinations have been presented by this single entry. When clicking at the certain node, text-based information will be given to explain what have been presented by it. Moreover, user could navigate the color bar to filter the nodes that represent specific number of combinations.

4.5 Discussion and Conclusion

The second task in this research is to navigate and query the compressed combinations. As all four techniques guarantee users' capacities to navigate the data, more criteria need to set up before making final decision.

In daily work, engineers are frequently adjusting the filters to sort out needed information from allowed combinations. This progressive adjustment is mostly based on their expertise knowledge and the relations among variant families they have so far discovered. Thereby, it is critical to get this information at the first glance. Besides, as many visualization methods do not scale efficiently even for the moderate size of data [10], the scalability of each method should be evaluated. Finally, the complexity to implement the method is also an impacting factor. Accordingly, four criteria are set up to evaluate above factors:

1. possibility to detect variant family relation
2. possibility to adjust variant family combination
3. scalability
4. implementation complexity

Evaluations for the criteria of four visualization methods are listed in *Table 23*

Table 23 Evaluations of four visualization methods

criteria	possibility to detect VF relation	possibility to adjust VF combination	Scalability	Implement Complexity
Tree Map	Medium	Difficult	Scalable	Medium
Tree Structure	Easy	Difficult	Scalable	Low
Elastic Menu	Difficult	Easy	Scalable	Medium
Combination Shot	Difficult	Difficult	Scalable	Medium

By fixing the variants in different variant families in sequence, tree map enables users to detect corresponding distributions when drilling down the data from top to bottom. Though restrained to the fact that it gets the variant family relation information on a low level instead of an overview, it is still an alternative to partly detect variant family relation. Elastic Menu performs well to adjust variant family combination. Users are flexible to refine proper variant family combinations, analyze the hidden information from different angles. Yet, Combination Shot is weak in all fields, it is excluded as a poor choice in this research.

Tree structure works well in all fields despite of its weakness to adjust variant combinations. Considering the high importance of detecting the variant family relations during the analysis, tree structure is the first choice to navigate and query the compressed combinations. By navigating the tree from top to the bottom, it is easy to identify the data structure and how the variants are the same height are related. In addition, users could either get an overview or drill down to particular information by expanding or collapsing a specific node.

Due that tree structure is not flexible to change variant family combination, it is necessary to combine it with other methods, elastic list for instance. Before visualization, users will be asked to filter combinations that are restricted to specific variants (or variant combination). The complete data will be visualized if users choose not to filter and the number of combinations to be visualized does not exceed a certain threshold (500, for instance).

Finally, visualization of combinations is sensitive to the variant family order. The family order in compression may not be in accordance with the one in visualization. Users may organize same families with different orders for various purposes. Hence, users are expected to determine family order before visualizing the data

5 Conclusion and Future Work

5.1 Conclusion

This research has been concentrating on configuration knowledge, its representation and visualization. The paper highlights how compression can be used to present huge data without losing important information and how visualizations support engineers in navigating and querying the massive data. The data used in this research is real-world industrial data and it is representative for similar product development organizations. Generally, we have following conclusions:

1. The compression approach is sensitive to the number and the size of the problem. Different family orders affect the calculation time.
2. There is a balance between compression ratio and the number of identifiers. Better compression ratio is together with more number of identifiers.
3. Discerning the structure of the combinations and detecting hidden variant relations are critical factors to support engineers' work. They are given higher priority when comparing different visualization techniques.
4. Visualization is sensitive to the family order. Users are expected to determine the family orders before visualizing the combinations. Moreover, the filtering feature needs to be built-in in any selected visualizing approach to fast locate the interesting information and limit the data to be visualized.
5. In this research, we use identifier which is based on multiple variant families to compress the allowed combinations and tree structure to visualize the compressed data

5.2 Future Work

As the author just describes one feasible approach that could be used for compression, more ideas could be taken into account in future research. Moreover, when describing *Identifier* technique to compress the allowed combinations, the paper orders the variant family by *the most comes first* rule, that is the VF that has the most number of variants will be put in the leftmost columns. More family order choices should be studied and compared to come to the best solution.

Exploration in improving comprehensiveness of identifiers' name format is also helpful. Identifiers that contain all variants of the variant family need to be presented in a unique way and the variants that are never shown should also be marked for users' information.

Moreover, it may be a good idea to have compression as a built-in function in PDM system. Thereby, PDM could either provide original allowed combinations or the compressed ones to the end-users.

References

[1] Miller, E. (1997) What's PDM? Computer-Aided Engineering Magazine, September 1997.

[2] B. Shneiderman. The eye have it: A task by data type taxonomy for information visualizations. In Proc. 1996 IEEE Conference on Visual Languages, 336-343

[3] Gibbard A (1974) A Pareto-Consistent Libertarian Claim. J Econ Theory7:388-410

[4] Daniel A. Keim, Jörn Schneidewind (2005) Scalable Visual Data Exploration of Large Date Sets via MultiResolution, page 1

[5] Johnson, B. and Shneiderman, B. (1991). Tree-Maps: A space-filling approach to the Visualization of Hierarchical Information Structures, Proc. 2nd International Visualization Conference, IEEE, 284-291

[6] Ben Shneiderman, University of Maryland < http://www.cs.umd.edu/hcil/treemap-history/>

[7] Mortitz Stefaner, Information Aesthetics <http://moritz.stefaner.eu/projects/elastic-lists/>

[8] Mortitz Stefaner, Information Aesthetics <http://well-formed-data.net/experiments/elastic_lists/>

[9] Interactive History Timeline, BBC UK < http://www.bbc.co.uk/history/interactive/timelines/british/index.shtml

[10] Philip Kegelmeyer, Robert Calderbank, Terence Critchlow, (2008) Mathematics for Analysis of Petascale Data, 13-19

Bibliography

[1] Ben Shneiderman. (2006) Using a hierarchical structure, treemmaps provide Meaningfully organized displays of high-volume information, 57-64

[2] Card, S., Mackinlay, J., and Shneiderman, B., (1999) Readings in Information Visualization: Using Vision to Think, Morgan Kaufmann Publ., San Francisco，CA

[3] Daniel A. Keim. (2000) Designing pixel-oriented visualization techniques: Theory and applications. IEEE Transactions on Visualization and Computer Graphics

[4] Gouthami Chintalapani, Catherine Plaisant, and Ben Shneiderman.(2004) Extending the Utility of Treemaps with Flexible Hierarchy

[5] Ketan Babaria. (2001) Using Treemaps to Visualize Gene Ontologies, 58-77

[9] William W. Hargrove, Forrest M. Hoffman (1999) Using Multivariate Clustering to Characterize Ecoregion Borders, 619-626

[10] An introduction of permutation methodology, Wikipedia <http://en.wikipedia.org/wiki/Permutation>

[11] Michael Brestrich, Eindhoven University of Technology <http://w3.win.tue.nl/nl/onderzoek/onderzoek_informatica/visualization/sequoiaview//>

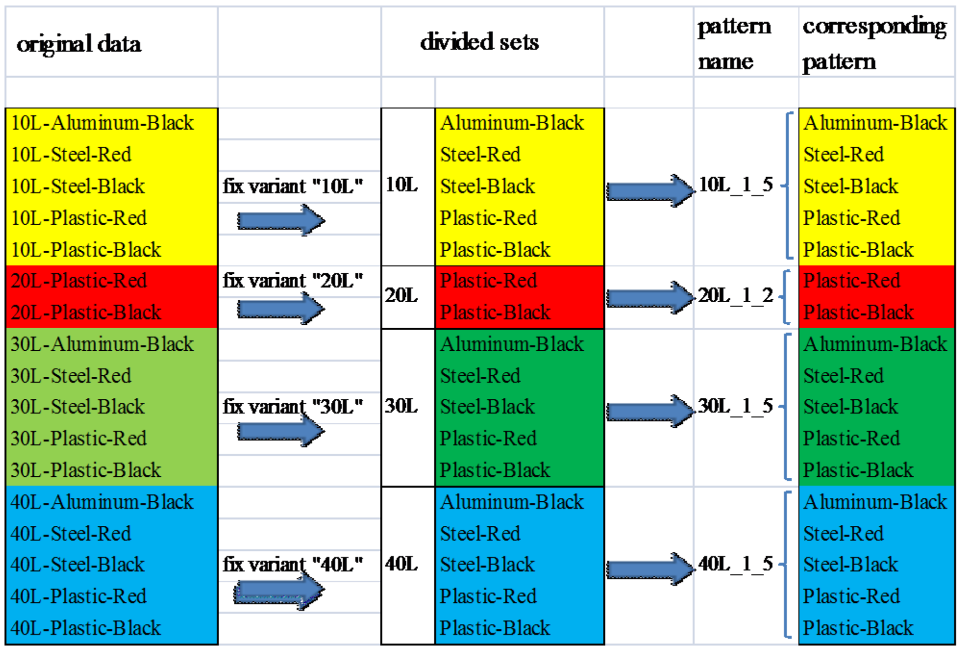
[12] Map of the market, Dow Jones & Company, Inc <http://www.smartmoney.com/map-of-the-market/>

[13] Product data management, Wikipedia <http://en.wikipedia.org/wiki/Product_data_management>

Appendix A: Use Identifier to Compress the Fuel Tank Example

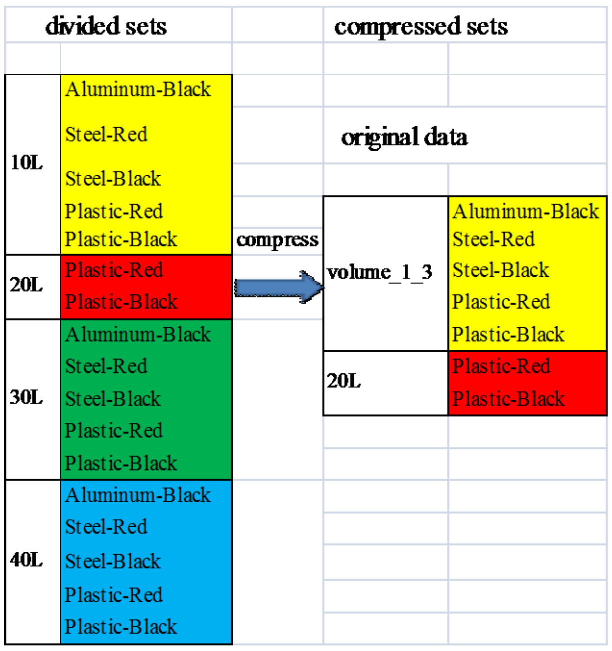
Take fuel tank example for instance, a set of flows are to illustrate the complete procedures of compressing allowed combinations based on multiple variant families. In the example, identifier *volume_1_3* presents set (10L,30L, 40L), *material_1_2* presents set (steel, plastic) and *color_1_2* presents set (black, red).

Step 1: fix variants in VF *Volume* and divide allowed combinations into corresponding sets and patterns



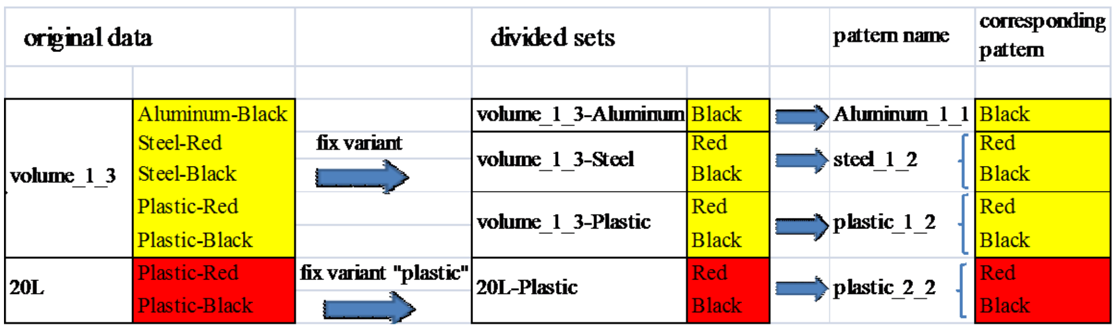
original data		divided sets			pattern name	corresponding pattern
10L-Aluminum-Black		10L	Aluminum-Black		10L_1_5	Aluminum-Black
10L-Steel-Red			Steel-Red			Steel-Red
10L-Steel-Black	fix variant "10L"		Steel-Black			Steel-Black
10L-Plastic-Red			Plastic-Red			Plastic-Red
10L-Plastic-Black			Plastic-Black			Plastic-Black
20L-Plastic-Red	fix variant "20L"	20L	Plastic-Red		20L_1_2	Plastic-Red
20L-Plastic-Black			Plastic-Black			Plastic-Black
30L-Aluminum-Black		30L	Aluminum-Black		30L_1_5	Aluminum-Black
30L-Steel-Red			Steel-Red			Steel-Red
30L-Steel-Black	fix variant "30L"		Steel-Black			Steel-Black
30L-Plastic-Red			Plastic-Red			Plastic-Red
30L-Plastic-Black			Plastic-Black			Plastic-Black
40L-Aluminum-Black		40L	Aluminum-Black		40L_1_5	Aluminum-Black
40L-Steel-Red			Steel-Red			Steel-Red
40L-Steel-Black	fix variant "40L"		Steel-Black			Steel-Black
40L-Plastic-Red			Plastic-Red			Plastic-Red
40L-Plastic-Black			Plastic-Black			Plastic-Black

Step 2: compress sets that have identical patterns



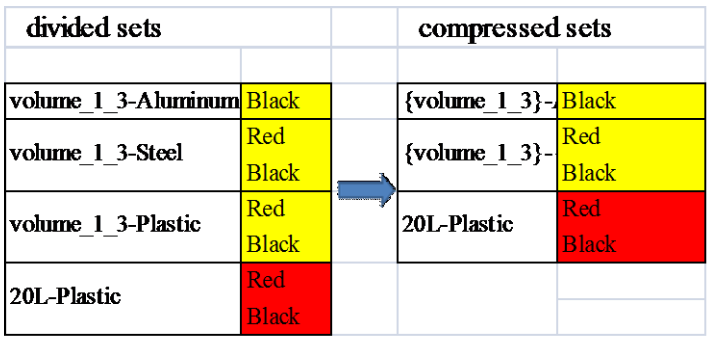
divided sets			compressed sets	
			original data	
10L	Aluminum-Black			
	Steel-Red			
	Steel-Black			
	Plastic-Red		volume_1_3	Aluminum-Black
	Plastic-Black	compress		Steel-Red
20L	Plastic-Red			Steel-Black
	Plastic-Black			Plastic-Red
30L	Aluminum-Black			Plastic-Black
	Steel-Red		20L	Plastic-Red
	Steel-Black			Plastic-Black
	Plastic-Red			
	Plastic-Black			
40L	Aluminum-Black			
	Steel-Red			
	Steel-Black			
	Plastic-Red			
	Plastic-Black			

Step 3: divide set *volume_1_3-set1, 20L-set1* into corresponding small sets and patterns



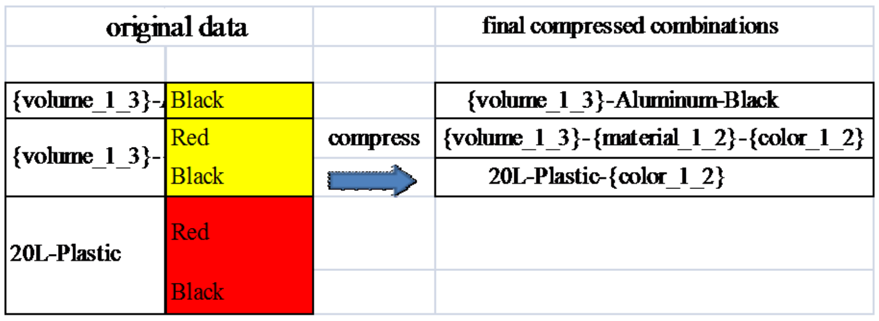
original data			divided sets			pattern name	corresponding pattern
volume_1_3	Aluminum-Black		volume_1_3-Aluminum	Black	→	Aluminum_1_1	Black
	Steel-Red	fix variant	volume_1_3-Steel	Red	→	steel_1_2	Red
	Steel-Black	→		Black			Black
	Plastic-Red		volume_1_3-Plastic	Red	→	plastic_1_2	Red
	Plastic-Black			Black			Black
20L	Plastic-Red	fix variant "plastic"	20L-Plastic	Red	→	plastic_2_2	Red
	Plastic-Black	→		Black			Black

Step 4: compress sets that have identical patterns



divided sets			compressed sets	
volume_1_3-Aluminum	Black		{volume_1_3}-	Black
volume_1_3-Steel	Red		{volume_1_3}-	Red
	Black	→		Black
volume_1_3-Plastic	Red		20L-Plastic	Red
	Black			Black
20L-Plastic	Red			
	Black			

Step 5: combine and recompress the newly-generated sets



original data			final compressed combinations
{volume_1_3}-	Black		{volume_1_3}-Aluminum-Black
{volume_1_3}-	Red	compress	{volume_1_3}-{material_1_2}-{color_1_2}
	Black	→	20L-Plastic-{color_1_2}
20L-Plastic	Red		
	Black		

Appendix B: A VB Script to Mark Different Cells in Two Regions in an Excel File

Usage: Mark differences and highlights the cells that are different in the two regions. The code loops through the first region's collection, looking for corresponding cells in the second collection. If it finds a corresponding cell, it compares the cells' values and highlights the first cell if they are different. If there is no corresponding cell in the second collection, the code highlights the unmatched first cell. The code repeats this step to compare the cells in the second collection to those in the first.

Script content:

```
Sub MarkDifferences()
Dim active_sheet As Worksheet
Dim name1 As String
Dim name2 As String
Dim range1 As Range
Dim range2 As Range
Dim cells1 As Collection
Dim cells2 As Collection
Dim cell1 As Range
Dim cell2 As Range
Dim key As String
Dim no_match As Boolean

    Set active_sheet = ActiveSheet
'       name1 = InputBox$("First Range Name:", "First Range",
' "")
name1 = "Range1"
    If Len(name1) = 0 Then Exit Sub
    Set range1 = active_sheet.Range(name1)

'       name2 = InputBox$("Second Range Name:", "Second
' Range", "")
name2 = "Range2"
    If Len(name2) = 0 Then Exit Sub
    Set range2 = active_sheet.Range(name2)

    ' Make normal collections holding the cells.
    Set cells1 = New Collection
    For Each cell1 In range1.Cells
        key = cell1.Row - range1.Row & "," & cell1.Column - _
        range1.Column
    cells1.Add cell1, key
Next cell1

Set cells2 = New Collection
For Each cell2 In range2.Cells
    key = cell2.Row - range2.Row & "," & cell2.Column - _
        range2.Column
    cells2.Add cell2, key
Next cell2

' Examine the cells in the first collection.
For Each cell1 In cells1
    On Error Resume Next
    Err.Clear
    key = cell1.Row - range1.Row & "," & cell1.Column - _
        range1.Column
    Set cell2 = cells2(key)
    If Err.Number <> 0 Then
        ' The second cell is missing.
        no_match = True
    ElseIf cell1.Text <> cell2.Text Then
        ' The cells don't match.
        no_match = True
    Else
        no_match = False
    End If

    ' If the cells don't match, color cell1.
    If no_match Then
        With cell1.Interior
            .ColorIndex = 35
            .Pattern = xlSolid
        End With
    Else
        With cell1.Interior
            .ColorIndex = xlNone
        End With
    End If
Next cell1

' Examine the cells in the second collection.
For Each cell2 In cells2
    On Error Resume Next
```

```
        Err.Clear
        key = cell2.Row - range2.Row & "," & cell2.Column - _
            range2.Column
        Set cell1 = cells1(key)
        If Err.Number <> 0 Then
            ' The second cell is missing.
            no_match = True
        ElseIf cell2.Text <> cell1.Text Then
            ' The cells don't match.
            no_match = True
        Else
            no_match = False
        End If

        ' If the cells don't match, color cell2.
        If no_match Then
            With cell2.Interior
                .ColorIndex = 35
                .Pattern = xlSolid
            End With
        Else
            With cell2.Interior
                .ColorIndex = xlNone
            End With
        End If
    Next cell2
End Sub


End Sub
```