CASPER: Cross-platform Automated Space Planning Engine for Retailers

By

Ziad Tauk

A thesis submitted to the
Faculty of Natural and Applied Sciences (FNAS)
in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science

NOTRE DAME UNIVERSITY
FACULTY OF NATURAL AND APPLIED SCIENCES
DEPARMENT OF COMPUTER SCIENCE

December, 2009

ABSTRACT

Keywords: Retail space management, Self-Organizing Map, Knapsack problem, Process automation, Service-Oriented Architecture

Retail space planning is an accurate and complex process affecting the overall performance of a retail environment. Such task, confronting large retailers consisting of huge malls and hypermarkets, appears infinite in the absence of a complete automated process, starting from the store plan generation, passing by the optimal product assortment and finishing with the product-to-shelf allocation problem. Moreover, the ever-changing factors affecting the retail space planning process, such as merchandising rules, competitive strategies and consumer behavior require continuous follow-up and optimization of the overall process. Based on the previous requirements, we propose an automated engine which initiates by generating a store plan based on the results of the market basket analysis, selects the optimal item assortment for each item category and finally allocates the resulting items on their respective shelves. Moreover, a service-oriented architecture is proposed to ensure interoperability between the engine and the corresponding external modules.

TABLE OF CONTENTS:

LIST OF FIGURES:

LIST OF TABLES:

LIST OF ABBREVIATIONS:

AAE	Automatic Allocation Engine
AEA	Automatic Expansion Advisor
ANN	Artificial Neural Network
ASE	Automatic Segmentation Engine
ASPGE	Automatic Store Plan Generation Engine
BKP	Bounded Knapsack Problem
BMU	Best Marching Unit
BRR	Business Rule Repository
CASPER	Cross-platform Automated Space Planning Engine for Retailers
DP	Dynamic Programming
ERP	Enterprise Resource Planning
ESB	Enterprise Service Bus
IDE	Integrated Development Environment
KP	Knapsack Problem
KPI	Key Performance Indicator
MDM	Master Data Management
OSGi	Open Services Gateway initiative
PDF	Portable Document Format
SaaS	Software as a Service
SOA	Service-Oriented Architecture
SOM	Self-Organizing Map
SVG	Scalable Vector Graphics
XML	eXtensible Markup Language

1 Introduction

1.1 Introduction to the general problem

Product assortment, product display area selection, shelf space allocation, and inventory control are critical retailing operations having major impact on the financial performance of retail stores. Managing these three operations individually will obviously result in sub-optimal overall retail store's profit [1]. Such a process induces multiple benefits such as:

- Maximizing space usability in the store
- Improving visibility and accessibility for the customers
- Adapting an optimal distribution of categories, sub-categories and items
- Maximizing profit and revenue

1.2 Problem definition

The choice of which brands to stock and the allocation of scarce shelf space among stocked brands are important to the retailer because these decisions are key determinants of his revenue and costs [2]. Hence, an optimal space planning for large retailers consisting of huge malls and hypermarkets appears as an infinite problem, especially in the absence of a complete automated process, starting from the store plan generation, passing by the optimal product assortment and finishing with the product-to-shelf allocation problem. Moreover, the ever-changing factors affecting the retail space planning process, such as merchandising rules, competitive strategies and consumer behavior yield to a continuous follow-up and optimization of the overall process. Such an intensive and critical maintenance cannot be managed manually since it needs to be performed in a fast, yet accurate manner.

1.2.1 Need for automation

Suppose CVS Caremark, the largest pharmacy chain in the United States, with approximately 6900 stores (http://en.wikipedia.org/wiki/CVS/pharmacy), is planning to apply a space planning process. It is to note that CVS Caremark sells both medical and grocery products. For each of the 6900 stores, a proper store plan is to be generated, taking into consideration the geo-spatial and demographic factors of the store's location.

Per example, a store located in an ethnic region with an intense Asian population will obviously have a store plan different from another one located in New York's downtown. Moreover, for each store, different product assortment plans, hence different product-to-shelf allocation outputs are to be applied since the marketing and promotional strategies as well as the spatial constraints (number and distribution of fixtures / shelves) vary from store to store. For the ethnic cluster of stores, non-national products (per example, Chinese spices) might cover 5% of the overall assortment while it won't cross 0.5% in the other cluster, where there is a significantly lower ethnic concentration. As a result, the product selection as well as distribution will considerably differentiate. Such a huge variation cannot be managed manually store by store, especially in the presence of continuous change in affecting factors (i.e. promotions, holidays, seasonality …).

1.2.2 Need for a complete on-going process

Most of the automated solutions involved in the space planning process focus on the optimal assortment generation or the product-to-shelf allocation problem without taking into consideration the whole lifecycle of the process. In a matter fact, ignoring one or more sub-processes can yield to degradation in both performance and accuracy of the resulting output. Automating the product assortment without a proper store plan or automating the product-to-shelf allocation problem without an optimal underlying assortment can create serious gaps. Suppose a given retailer is using an automated allocation engine to maximize space utilization in his stores. As for the product assortment, the category manager is manually, based on previous experience, selecting the proper items. In such a real-life scenario, the following problems may occur:

- The items selected by the category manager do not reflect an optimal assortment; given the complexity of certain strategies and the possible high number of related dimensions or criteria, the category manager might pick the wrong, or "un-optimal" set (selecting items yielding to a considerable waste of space in the product-to-shelf allocation phase or products with low affinity to each other, ignoring items with unobvious profitability)
- The rules considered by the category manager for selecting the assortment and which are only present in his mind, and the merchandising rules applied by the auto-

allocation engine to distribute the items might diverge from the common global strategy. Per example, the selected assortment includes two items A and B with very low affinity to each other (must not be placed adjacent to each other). On the other hand, the auto-allocation engine, based on "hidden" criteria ignored by the category manager, placed A and B next to each other, yielding to a strategical conflict.

1.2.3 Need for dynamicity

As mentioned in 1.2.1, the space planning process is tightly bound to various categories of dynamic factors (i.e geo-spatial, marketing, managerial…). Hence, automated solutions need to take into consideration such factors, regardless of their number, respective types and business meaning; the engine should read a dynamic set of weighted rules and reflect their strategical impact on the resulting output. Such a dynamic scalability needs to be applied by both value and period. Suppose, for a given US retailer, the "pain relievers" category is seasonal; the sales activity is higher in cold seasons. Such a rule cannot be statically applied for the different store clusters; per example, stores located in Alaska differ from stores located in Texas in terms of seasonality. Moreover, the weight of such a rule will not be same in both clusters. It is also to note that some rules and factors may be present and effective for one store cluster and insignificant for others.

1.2.4 Need for accuracy

As explained before, completeness of the process is a crucial aspect of automated solutions of the retail space planning problem. Hence, defects in early stages of the process produce a "chain effect" with more severe impact in child sub-processes. Per example, while generating the store plan of a given store, if two categories with low affinity to each other (ex: child food and pet food) were placed adjacent to each other due to defects in the automated engine, the corresponding items will consequently be allocated on the wrong fixtures, which yield to inaccuracy in merchandising and waste of resources (time and labor). Moreover, a poor selection of the assortment of a certain category due to an inefficient clustering mechanism, yield to a deviation from the targeted strategy, a decrease in the performance of the plan in terms of expected profitability and an erroneous product-to-shelf allocation.

1.3 Research objectives

The main objective of this master thesis is to provide a reliable, accurate and complete method for automated retail space planning. Based on the previous, the research should answer the following aspects:

- Finding a complete automated flow which covers the whole retail space planning process.
- Exploring and choosing the most convenient algorithms in terms of performance, cost and optimization accuracy.
- Finding a methodology which takes into consideration the dynamicity of the factors affecting the process.

1.4 Approach and main results

To achieve the research objectives goal, multiple computer science algorithms are explored and analyzed to fit the addressed problem. We divided the problem into three major components:

- Automatic store plan generation: which distributes, based on the market basket analysis, the different categories on the different physical fixtures of a given store.
- Auto-segmentation engine: which optimally divides a given store cluster into a set of product categories, while taking into consideration constraints (i.e. affinity between categories, consumer behavior, etc…). On the micro-level, the same engine is used to divide a category's planogram into optimal segments, based on a given strategy. All these terms will be explained in detailed in Chapter 2. This problem has been mapped to a special type of neural networks, the self-organizing maps.
- Auto-allocation engine: which optimally allocate items over shelves and fixtures while applying a dynamic set of merchandising rules specified by the user as well as ensuring the profit / space maximization. This part has been solved by a custom method which includes the Bounded Knapsack problem.

1.5 Thesis organization

This thesis consists of 4 chapters and one appendix. Chapter 1 introduces the main problem, defines the research objectives lists the approach and main results. In Chapter 2, definitions of concepts related to the retail space planning process are presented. Moreover, previous algorithms and methodologies are overviewed as well as the main motivation behind this research. Chapter 3 contains the original work, including the proposed automated process, the adopted algorithms and the corresponding methodologies. Finally, Chapter 4 summarizes the findings, highlights the contributions of the thesis and lists the research limitations and future work.

2 Literature Review

2.1 Definitions of the basic concepts

Below is the definition of the basic business concepts used in "retail space planning":

- Item category / sub-category: the group / sub-group or family to which a given item belongs. Per example, Coca Cola below belongs to the category "Beverage" and sub-category "Soda".
- Item facing: vertical / horizontal copy of an item.

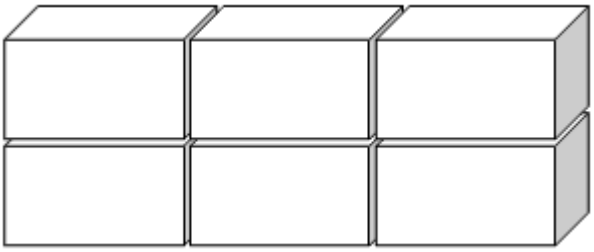


Figure 1 – Example of item facings: 3 horizontal and 2 vertical facings

- Shelf allocation: process of allocating items over shelves and fixtures in a retail environment. The process is usually performed by item category or sub-category.
- Segmentation: process of dividing a given store by item department/category (or category/sub-category). Below is an example of basic store segmentation into 5 item categories:

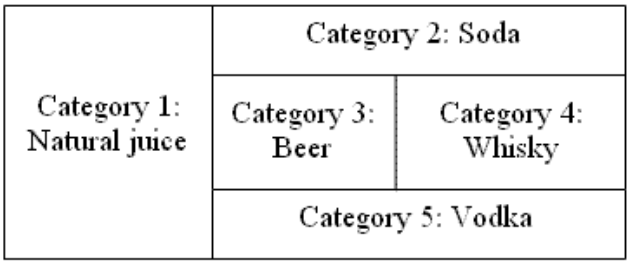


Figure 2 – Example of segmentation

Such a process defines the portion of each category in respect to the overall store, the geographical location as well as the neighborhood, which takes into consideration the affinity between different categories. Per example, there is a low affinity between baby food and pet food, hence it is not recommended to place the two categories close to each other.

- Merchandising rule: business rule used to strategically perform segmentation and shelf allocation. Such rules can affect vertical allocation, horizontal allocation or simply imply positioning properties. Below is a few examples of merchandising rules used in a retail environment:
    - The minimum horizontal space between items is 1.2 cm (property)
    - National brand items must be on left on non-national brand items (horizontal allocation)
    - The maximum number of vertical facings for stackable items is 5 (property)
    - Within a category, top selling items must be place on eye-level shelves (vertical allocation)
- Store plan: logical blueprint of the store resulting of the segmentation process. It shows the spatial distribution and location of different fixtures by category.

Figure 3 - Example of a store plan

- Planogram: The planogram is a visual diagram, or drawing, that provides in detail where every product in a retail store should be placed. These schematics not only present a flow chart for the particular merchandise departments within a store layout but also show on which aisle and on what shelf an item is located. Generating planograms is a challenging and time-consuming process because the simplest form of planogram problem (ignoring all marketing and retailing variables) is already a multi-knapsack problem, a well-known NP-hard problem which is very difficult to solve [19].

Figure 4 - Example of planogram: Beverage / Soda

The benefits of using planograms in a retail environment include:

- Enhancing consumer behavior by maximizing trade-up and impulse purchases
- Providing a detailed blueprint to facilitate the replenishment process
- Improving customer satisfaction and loyalty by providing a structured and accessible product organization

2.2 Space Planning Process

The space planning process covers both micro and macro-store planning. It includes managing standard and specific store plans, consolidating corporate, store group or store level view of regular and promotional performance and ensuring store compliance to corporate strategy and planogram definition.

The figure below illustrates the basic flow of the space planning process adopted by most retailers:

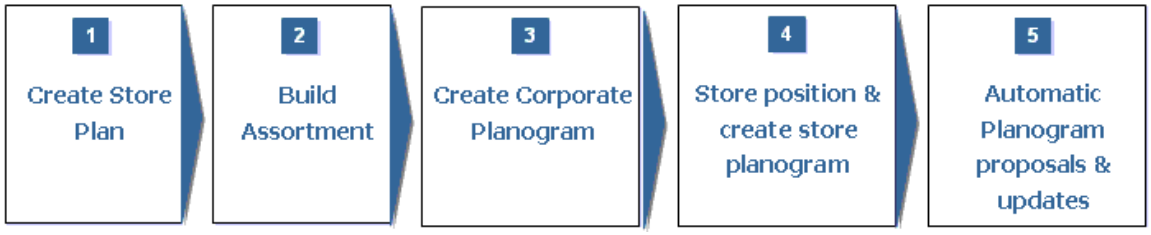


Figure 5 - Space planning process

The process initiates by creating store plans, then defining assortments. Next, a corporate planogram, which is the most global version including all items of a certain category, is created. Based on the corporate planogram, store planograms are generated. Given the complexity and time-consuming aspect of such a process on one hand and the critical need of accuracy on the other hand, automated solutions are needed to perform the space planning process. Based on this need, our proposed approach aims to automate the segmentation of the store (step 1) and the generation and filling of planograms (step 3 and 5). Moreover, the same approach adopted to implement the auto-allocation engine can be used to generate store planograms from the resulting corporate planogram. In this study, our main concern is automating the store plan and corporate planogram creation.

2.3 Previous work

2.3.1 Algorithms

A paper by Chen and Lin uses a popular data mining approach, association rule mining, instead of space elasticity to resolve the product assortment and allocation problems in retailing. In this paper, the multi-level association rule mining is applied to explore the relationships between products as well as between product categories [3]. The approach consists of three stages. First, the algorithm starts with the multi-level association rule mining between product items, product subcategories and product categories. Second,

product assortment is applied by estimating the frequent item set profits then resolving the corresponding product assortment mathematical model. Finally, the algorithm performs shelf space allocation for product categories, product subcategories and product items respectively. Another paper entitled "Heuristic Approach for Automated Shelf Space Allocation" presents a heuristic approach for solving the automatic shelf allocation problem. The heuristic method described in the paper consists of four phases: 1) the preparatory phase checks that enough shelf space is available for all products to be displayed, 2) the allocation phase constructs an initial arrangement, 3) the adjustment phase makes iterative changes to the arrangement in order to improve the over-all profit, 4) the termination phase computes the quality of the final arrangement and generates the corresponding planogram [4]. The approach was tested on two problems, a small one consisting of 135 products on 15 shelves and a large one consisting of 907 products on 114 shelves. The results show that the method is capable of generating much improved arrangements in terms of the overall profit. Another paper entitled "Metaheuristics with Local Search Techniques for Retail Shelf-Space Optimization" extends the shelf allocation problem to address other problems such as product groupings and non-linear profit functions. The metaheuristics approach starts by developing a network flow solution approach to the problem and then, using many-to-many neighborhood moves and finally employing a strategy of combining a strong local search with metaheuristics [5]. These techniques are then applied to more complex models that address product groupings and nonlinear profit functions. Another paper divided the problem into two sub-problems: the Product to Shelf Allocation Problem (P2SAP) and the Shelf-Space Allocation Problem (SSAP). The authors propose a genetic algorithm where the chromosomes are coded as vectors of length M where each component i stores the category allocated to module I [6].

2.3.2 Business Models

On the other hand, many business models have been proposed to solve the retail space allocation problem. In a paper entitled "A Dynamic Model for Strategically Allocating Retail Space" [7], the authors show how static models can be extended to incorporate dynamic market changes. The main issue is how to strategically allocate space among

product groups with widely different growth potentials. The proposed solution is a dynamic model which extends the classical concentration on retail sales and gross margins to include variables related to new growth markets. However, the model needs to be extended to include other competitive variables such as price, advertising and promotional strategies. Built on the work of Corstjens and Doyle, Bultez and Naret propose in their paper [8] a shelf space allocation model which focuses on the demand interdependencies prevailing across and within product-groups. The proposed model entitled SH.A.R.P (Shelf Allocation for Retailers' Profit) introduces several distinctive characteristics such as sales-share elastisticy among product categories. Even though the model yielded satisfactory experimental results in terms of assortment profitability, the authors suggested developing the model to integrate additional merchandising variables such as shelf heights and special types of display fixtures. On the other hand, Zufryden presents in his paper entitled "A Dynamic Programming Approach for Product Selection and Supermarket Shelf-Space Allocation" [9], a model intended to select optimally among a given set of products and allocate integer shelf-space units to the selected products in supermarkets. The approach takes into considerations strategical specifications related to space elasticity, product cost and demand-related marketing variables, and is bound to constraints such as supply availability, block product allocation and operational requirements.

2.3.3 Software packages

Many Enterprise Resource Planning (ERP) software providers developed advanced tools for managing and automating the retail space planning process. We will briefly overview the leading packages in the market.

2.3.3.1 SAS Integrated Merchandise Planning

The solution by SAS corporation includes multiple modules, from which we are interested in:

- SAS Merchandise Financial Planning: for setting merchandise financial goals based on analysis of historical data.
- SAS Merchandise Assortment Planning: used for managing assortments based on consumer behavior and financial strategies.

- SAS Space Planning: for visually planning assortments and building store plans and category-based planograms.
- SAS Space Optimizer: for automating the development of optimized store-specific planograms.
- SAS Merchandise Allocation: used to analyze store-specific needs and improve inventory management with allocation and replenishment of basic, fashion and promotional merchandise.

2.3.3.2 Demandtec Assortment & Space

DemandTec is a publicly traded company that provides pricing, promotion, and demand optimization solutions for retailers and consumer product (CP) manufacturers. They deliver their products as a Software as a Service (SaaS). DemandTec software services utilize a science-based software platform to model and understand consumer behavior. This science is based on a quantitative understanding of incrementality, which is an item's ability to increase overall category sales or profit, and transferable demand, which is the degree to which sales volume shifts to similar items in the category or leaves the store when an item is delisted. Based on DemandTec Assortment & Space™ white paper, retailers can create optimized assortments based on multiple criteria, including sales, profit, space productivity, and Gross Margin Return on Inventory Investment (GMROII). Merchants also have the flexibility to combine their optimization goals with additional controls to ensure optimized assortments align with company strategies such as protecting private label items and image items. They can also run multiple scenarios at once and compare results to identify the best strategy to meet their objectives.

2.3.3.3 JDA Space Planning

JDA Software Group, Inc. is a demand and supply chain partner to the world's leading retailers, manufacturers and suppliers and is located in Scottsdale, Arizona. Their space planning solution supports optimization and analyzing of planograms against any metric (i.e., balance space to sales and/or days of supply). Moreover, Space Planning enables multiple planograms to be managed simultaneously, improving consistency and accuracy as products are quickly and easily added, replaced or updated across the entire planogram set.

2.3.3.4 Galleria Space Planning

Founded in 1989, Galleria is a market leading provider of automated customer centric merchandizing solutions to retailers and manufacturers. Their space planning solution includes the following functionalities:

- Forecasting procedures for optimizing promotional display planning.
- Category management and logical clustering of merchandise.
- Management and automation of strategic planograms which ensure consistency of merchandising process across all plans.
- Store execution and scenario simulation.

2.3.3.5 Apollo Space & Assortment Optimization

Founded in 1986, Aldata is a supply chain management solutions provider with global retail, wholesale and logistics customers.The retail space planning solution provided by Aldata includes three modules:

- Apollo Desginer Workstation: which uses automated procedures to generate planograms by using merchandising rules, assortments, performance criteria and space constraints.
- Apollo Total Store: which is a database-driven desktop software product that allows users to analyze store layouts and plan the use of space in the store at the macro level. Data is analyzed from various perspectives and information is presented to best fit specific needs.
- Apollo Web Publisher: which is a web-based system used to publish planograms, merchandising reports and performance charts in multiple formats.

2.4 Reference books & Software tools

2.4.1 Books

2.4.1.1 Self-Organizing Maps (T. Kohonen, 3rd edition)

This book [10], written by the creator of the Self-Organizing Maps, presents a detailed study about such revolutionary structures, their structure, variations and usage scenarios.

The author starts by explaining mathematical preliminaries such as distance measures for patterns, statistical pattern analysis, subspace methods of classification and vector quantization. Then he presents the basics of neural modeling, the core philosophy behind his invention, as well as the relation between biological and artificial neural networks, the phases of development of neural networks and the different learning laws. After introducing the context of the problem, the detailed description of SOM is explained; the basic implementation, the physiological interpretation and the different variations. Finally, the author overviews the different real-life applications of SOM, as well as the different software / hardware packages used in this field. We will use this reference as a guideline for mapping the studied retail segmentation problem to a Self-Organizing Map, defining the optimal initialization parameters (dimensions of the map, neighborhood functions) as well as the training process parameters (number of learning steps, learning rate and neighborhood radius). Details about these notions and parameters are provided in Chapter 3.

2.4.1.2 Algorithms for Knapsack Problems (D. Pisinger)

The book [11], which is the result of a PhD thesis, provides a wide range of algorithms and methodologies to solve the different variations of the NP-complete Knapsack problems.

In the first chapters, the author overviews Knapsack problems, their applications in real life as well their distinctive properties. In the following chapters, innovative minimal algorithms are presented to solve the 0-1 Knapsack Problem, the Bounded Knapsack Problem and the Multiple-choice Knapsack Problem. Moreover, the author reviews dominance relations in Unbounded Knapsack Problems, subset-sum problems and finally presents an algorithm for large Multiple Knapsack Problems. The main referential usage of this book is overviewing and analyzing the minimal algorithm intended to solve the Multiple Knapsack Problem; usually, such problems are mapped to classical 0-1 Knapsack problems and solved using the corresponding approaches.

2.4.1.3 A Java Library of Graph Algorithms & Optimization (K. Rosen)

This book [12] includes a wide variety of algorithms and test cases implemented in Java language. Categories include graph-related problems (traveling salesman, network flow,

coloring…) as well packing problems (knapsack problems, set covering problems, assignment problems…). Every chapter is self-contained and largely independent. Each topic starts with a problem description and an outline of the solution procedure. Programmatic details about the implementation of the algorithms are supplied in the book's appendices. We will be using this reference for writing basic Java-based implementations of the various Knapsack problems, which will be modified and upgraded to fit the requirements of the studied optimization problem.

2.4.2 Software tools

The following open-source packages are used to implement test cases and visualize results throughout the research.

2.4.2.1 JavaSOM

JavaSOM package is an open-source implementation of self-organizing maps (SOM) written in Java language. The tool consists of two major components: JSOM, which is an implementation of the Self-Organizing Maps training algorithm and Clusoe, an independent graphical tool used to configure and manage the maps. The visualization of the trained map is displayed using Scalable Vector Graphics (SVG) or Portable Document Format (PDF) files. The parameters used for controlling JSOM during the learning process as well as the input data are passed in an XML file. The third party applications included in the JavaSOM package are Xerces, Xalan and FOP. Xerces is the XML parser used by JSOM for reading in input data and interpreting instructions. Xalan is the XSL transformation processor which is controlled by JSOM totransform the trained map information into different XML formats. Currently, it is used only to output generic XML and SVG formats of the map. FOP is the formatting object processor controlled by JSOM to generate PDF versions of the maps. Both Xalan and FOP use also Xerces for XML parsing. We will use JavaSOM to create test cases for the auto-segmentation engine and visualize the results in SVG format.

2.4.2.2 Eclipse IDE

Eclipse is an open-source IDE written in Java which includes an extensible plug-in system. Eclipse employs plug-ins in order to provide all of its functionality on top of (and

including) the runtime system, in contrast to some other applications where functionality is typically hard coded. The runtime system of Eclipse is based on Equinox, an OSGi standard compliant implementation. We will use Eclipse for developing Java-based algorithms and real-life scenarios, executing unit tests and measuring their performance.

2.5 Research motivation

Most of the related research papers presented solutions related to some, but not all, of the sub-processes of the retail space planning problem. One of the closest researches related to our solution is entitled "A joint optimisation model for inventory replenishment, product assortment, shelf space and display area allocation decisions", given its global approach to the problem even though it didn't tackle the process automation from an architectural perspective. However, the authors did not consider some other important factors, such as the strategic importance of an item, that can influence decisions on product assortment, shelf space and display area allocations [1]. Moreover, the solution was not suitable for large sized problems, lacked the automated aspect of the process and did not cover the whole lifecycle of retail space planning; the authors did not mention the methodology used to generate the store plan before selecting the optimal product assortment.

In an another related study entitled "A data mining approach to product assortment and shelf space allocation" [3], the authors focus on mining multi-level association rules from a set of store transactions to perform product assortment and shelf space allocation. The assortment model is mapped to a zero-one integer problem which is different from the approach that we adopted in this paper. Moreover, the item-to-shelf space allocation is bound to the shelf profit weight without taking into consideration the dynamicity of possible merchandising rules.

Based on the previous points, we will focus in this research on providing a "complete" automated solution which covers the different aspects of retail space planning, takes into consideration the relatively large size of the problem as well as the dynamicity of the involved factors.

3 CASPER: Cross-platform Automated Space Planning Engine for Retailers

3.1 Introduction

In this chapter, we will present our automated solution entitled CASPER (Cross-platform Automated Space Planning Engine for Retailers). First, we will overview the automation process flow used to cover the different aspects of retail space planning. For each of the embedded sub-processes, we will explain the arguments behind the selection of the different adopted techniques and methodologies. Then, we will explain, using a real-life test case the methodology used to automate the process, passing by the different phases (store plan generation, product assortment, product-to-shelf allocation & space maximization).

3.2 Global flow

The figure below illustrates the global process flow of our proposed engine. The process initiates by utilizing the results of the market basket analysis as well as the store activity data to generate an optimal store plan. The output of this phase is a map of the store specifying which categories of items correspond to which physical fixtures / shelves within the store. Given the fact that not all items within a category are allocated to their corresponding shelves, the next phase is an iterative sub-process which operates for each category and choose the optimal product assortment based on strategy-oriented metrics and rules. Once the assortment is selected for each of the categories, the final sub-process allocates the items to their relative shelves, applies the different merchandising rules and optimizes space utilization.

The main advantage of the proposed flow is the completely automated aspect, which doesn't require any external user interaction or supervision. Based on a data repository containing all information related to items / categories / store attributes and a business rule repository defining the strategy-related constraints and parameters, the proposed flow operates module by module throughout the complete space planning process.

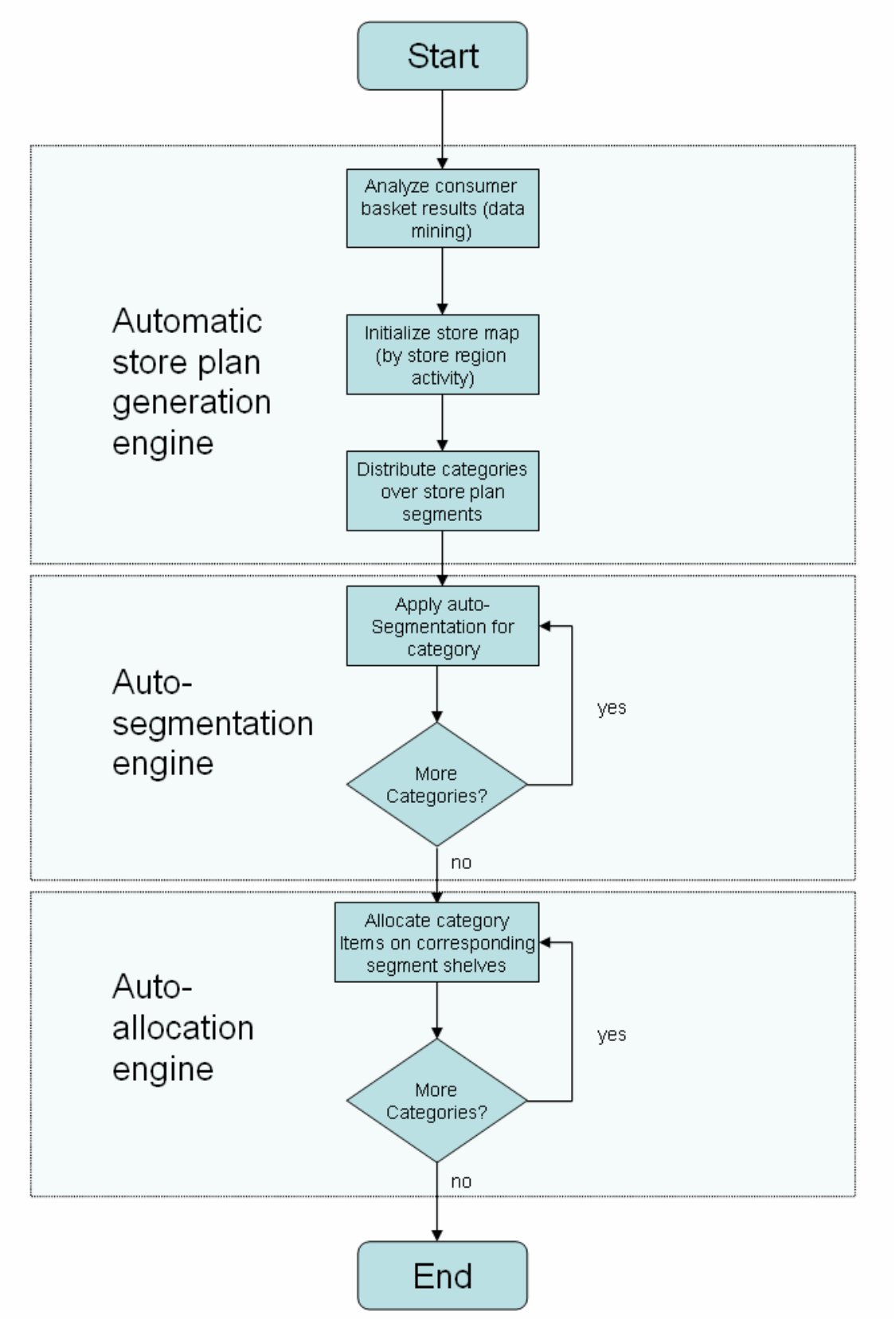


Figure 6 - CASPER global flow

3.3 Automatic Store Plan Generation Engine (ASPGE)

3.3.1 Introduction

Allocating retail space is an hierarchical problem where one first divides the space in a retail outlet among the major departments, then the subdepartments, and so forth right down to the single shelf. Although the actual allocation problem is identical at any level in the hierarchy, the rewards for getting the basic allocation right are higher, the higher one goes in the hierarchy [13]. Based on the above, the automatic store plan generation engine is the entry point for automating the retail space planning process. Its main purpose is to distribute the categories over the physical fixtures of the store, taking into consideration affinities and dependencies. Based on the portion allocated for each category by the category manager, the auto-segmentation engine will select the corresponding optimal assortment. On the other hand, the auto-allocation engine will distribute the items included in the optimal assortment on the shelves selected by the ASPGE.

3.3.2 Non-functional requirements

We assume that the following non-functional requirements are to be satisfied by the engine:

- Accuracy: as macro-level module, the ASPGE's result will affect the overall output of CASPER, hence it needs to accurately distribute the categories over the store, taking into consideration the different affecting factors.
- Scalability: ASPGE depends on the results of the market basket analysis and key performance indicators (KPIs) specified by the category manager. Given the fact that both sources are dynamically changed, ASPGE needs to be scalable enough to handle such an aspect.

3.3.3 Functional requirements

We assume that the ASPGE needs to perform the following functionalities:

- Reading the result of the market basket analysis and deriving input data related to affinity between the different categories present in the basket.

- Reading the KPIs specified by the category manager and reflecting their strategical meaning by allocating the corresponding categories on the corresponding physical fixtures.
- Taking into consideration the geo-spatial attributes and constraints of the store.
- Optimizing space utilization on the macro-level (fair distribution of categories over shelves).

3.3.4 Retail store structure

Depending on the retailer's operational strategy, a store can be divided into a number of levels. Unless the plan corresponds to a newly opened store, macro-levels (i.e. zones) are rarely changed and compose the static part of the store plan. Each macro-level includes multiple micro-levels (i.e. departments), which can be also divided into other sub-levels, and so on. In this study, we will adapt the structure illustrated in the example below, where a store is divided into zones, zones into departments and departments into categories. It is to note that the items within a category are not all used in the product-to-shelf allocation process; a product assortment is to be applied in a later phase, resulting in finding the corresponding set of items.

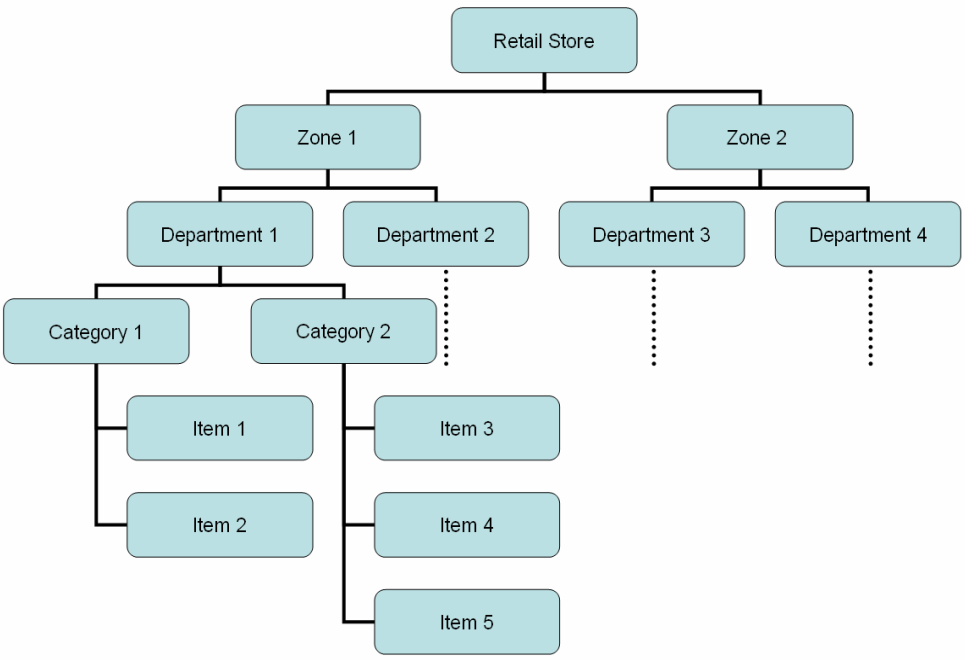


Figure 7 - Example of retail store structure

3.3.5 Global flow

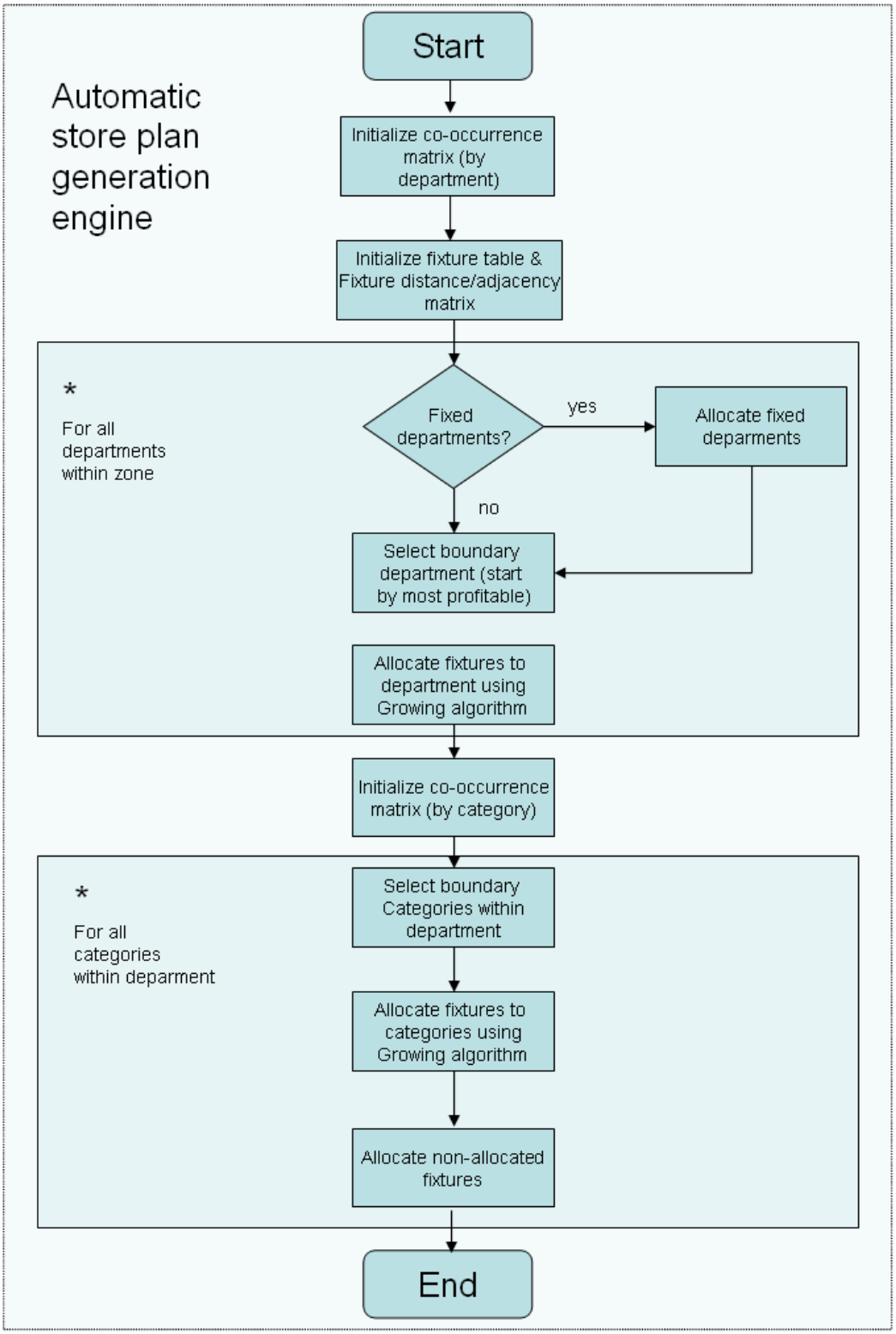


Figure 8 - Automatic store plan generation engine: global flow

3.3.6 Algorithm description

Our algorithm, based on the "divide-and-conquer" approach, initiates by extracting the results of the market basket analysis and building a co-occurrence matrix by product department. In other words, the engine scans the items found in different market baskets, checks their respective departments and computes the co-occurrence frequency of these departments, as follows:

$$(Di,Dj) = \begin{cases} +n & \text{if Di and Dj belong to the same store zone} \\ -n & \text{if Di and Dj belong to different store zones} \end{cases}$$

where n is the frequency of co-occurrence of departments Di and Dj in the market basket analysis.

In the example below, department D1 and department D2 are found together in 125 different baskets and belong to the same store zone (positive value) while department D1 and department D3 are found together 89 times but belong to different store zones (negative value).

	D1	D2	D3	.	.	.	Dn
D1	0	125	-89	.	.	.	59
D2	125	0	312	.	.	.	21
D3	-89	312	0	.	.	.	512
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
Dn	59	21	512	.	.	.	0

Figure 9 - Example of co-occurrence matrix (by department)

After, the fixture table and fixture adjacency/matrix are initialized. The fixture table contains properties and attributes related to each physical fixture in the store, as follows:

- Fixture ID: unique identifier of the fixture.

- Fixture-group (or planogram after filling it): which specifies the group of fixtures which contains the studied fixture. In Figure 3 - Example of a store plan, each group of assembled squares having the same color illustrates a different planogram.
- Allocated department: which specifies the department to which the fixture is allocated. This value is initially set to -1 (not allocated).
- Allocated category: which specifies the category within the department to which the fixture is allocated. This value is initially set to -1 (not allocated).
- Proportion: which denotes how much a given fixture constitutes from the overall volume of its zone (set of all fixtures). It is calculated as follows:

$$Proportion(FixtureA) = \frac{\sum_{i=0}^{p} w(i) * h(i) * d(i)}{\sum_{j=0}^{n} \sum_{k=0}^{m} w(k) * h(k) * d(k)}$$

where:

- p is the number of shelves in fixture A
- w(i), h(i) and d(i) are the width, height and depth of a given shelf i
- n is the total number of fixtures in the given zone

On the other hand, a fixture distance matrix is generated as follows:

$$(Fi,Fj) = \begin{cases} +n & \text{if Fi and Fj belong to the same fixture-group (planogram)} \\ -n & \text{if Fi and Fj belong to different fixture-groups} \\ 0 & \text{if Fi and Fj are adjacent} \end{cases}$$

where n is the distance between fixtures Fi and Fj.

Similarly, a fixture-group table and fixture-group distance matrix are initialized. The table contains the fixture-group identifier as well as the corresponding zone while the matrix is created as follows:

$$(FGi,FGj) = \begin{cases} +m & \text{if FGi and FGj belong to the same zone} \\ -m & \text{if FGi and FGj belong to different zones} \\ 0 & \text{if FGi and FGj are adjacent} \end{cases}$$

where m is the distance between fixture-groups FGi and FGj.

After completing the initialization of the needed structures, fixed departments, which preserve their locations regardless of the engine's output, are allocated to their respective fixtures. Per example, in a given grocery store, the "bread" department is statically allocated near the baking oven. Once all fixed departments are allocated, the engine selects the most profitable boundary departments, which are departments having a large co-occurrence frequency with departments in **other** zones and the highest KPI values specified by the category manager (i.e. profitability, net profit margin, …). In other words, if the entry (Di, Dj) in the matrix has a relatively small negative value and a large KPI (profitability) value, this means that department Di needs to be placed on the boundaries of its zone in such a way that it is shifted closer to the zone to which department Dj belongs. To choose the corresponding fixture-groups, one of the following two cases is confronted:

- Dj is already allocated to a fixture-group FGj: in this case, the following algorithm is applied:
    1. Read the fixture-group table and get all fixture-groups belonging to the zone which includes department Di.
    2. Get all entries (x, FGj) in the fixture-group distance matrix where x denotes the different fixture-groups found in previous step.
    3. Sort the **absolute value** of the entries found in previous step in ascending order (from closest fixture-group to furthest in respect to FGj).
    4. Read the sorted fixture-groups one-by-one and apply the following check:
        - If the total of proportions of all non-allocated fixtures within the fixture-group is greater than the proportion assigned to Di by the category manager, perform the following "growing algorithm" for department allocation:

A. Allocate fixture to department, starting from closest fixtures to FGj to the furthest ones.
B. Repeat until all the proportion assigned to Di is allocated on fixtures.
C. If the remaining non-allocated proportion of the fixture-group is greater than a predefined error margin (i.e. 20% of total proportion assigned for the department), separate allocated and non-allocated proportions of the fixture-group; a new fixture-group is created from the non-allocated proportion.

5. If no fixture-group fits department Di, allocate Di on the fixture-group having the largest free proportion.

- Dj is not yet allocated: in that case, the following algorithm is applied:
    1. Get all couples (FGi, FGj) from the fixture-group distance matrix such that FGi belongs to the zone containing Di and FGj belongs to the zone containing Dj.
    2. Sort the **absolute value** of the entries obtained in the previous step by ascending order (from smallest distance to largest).
    3. Read the sorted couples one-by-one and apply the following check:
        - If the total of proportions of all non-allocated fixtures within the fixture-group FGi is greater than the proportion assigned to Di, apply the "growing algorithm" explained above to Di.
    4. If no couple of fixture-groups fits department Di, allocate Di on the fixture-group couple having the greatest free proportion..

Once all departments are allocated to fixture-groups, a similar procedure is applied on the micro-level, to allocate categories to fixtures within fixture-groups. It is to note that the algorithm is applied department by department. Assuming we have a category Ci which is a boundary category in respect to another category Cj which belongs to a different department, the procedure can be summarized in one two possible cases:

- Cj is already allocated to a fixture Fj: in this case, the following algorithm is applied:

5. Read the fixture table and get all fixtures belonging to the department which includes category Ci.
6. Get all entries (x, Fj) in the fixture distance matrix where x denotes the different fixtures found in previous step.
7. Sort the **absolute value** of the entries found in previous step in ascending order (from closest fixture to furthest in respect to Fj).
8. Read the sorted fixtures one-by-one and apply the following check:
    - If the fixture is not yet allocated to another category, perform the following "growing algorithm" for category allocation:
        A. Allocate category to department, starting from closest fixtures to Cj to the furthest ones. When no additional **adjacent** fixtures are available, stop.
        B. Repeat until all the proportion assigned to Ci is allocated on fixtures.
        C. If there aren't enough non-allocated fixtures to fit Ci's proportion, allocate Ci on the set of **adjacent** fixtures having the greatest free proportion.

- Cj is not yet allocated: in that case, the following algorithm is applied:
    1. Get all couples (Fi, Fj) from the fixture distance matrix such that Fi belongs to the department containing Ci and Fj belongs to the department containing Cj.
    2. Sort the **absolute value** of the entries obtained in the previous step by ascending order (from smallest distance to largest).
    3. Read the sorted couples one-by-one and apply the following check:
        - If the total of proportions of all **adjacent** non-allocated fixtures within department Di is greater than the proportion assigned to Ci, apply the "growing algorithm" explained above to Ci.
    4. If no adjacent non-allocated fixtures on Di have enough free proportion to fit category Ci**,** allocate Ci on the set of non-allocated adjacent fixtures having the greatest free proportion on Di.

It is to note that in both cases when Dj (or Cj) is not yet allocated, Dj (or Cj) is used only as a reference point to its corresponding zone and should not be allocated using the "growing algorithm". To justify this statement, consider the following example:

In a given store, suppose we have 3 zones Z1, Z2 and Z3 and 5 departments D1, D2, D3, D4 and D5 such as D1 Є Z1, D2 Є Z2, D3 Є Z2, D4 Є Z3 and D5 Є Z3. Moreover, we assume that we have the following entries in the co-occurrence matrix (by department):

(D1, D2) = 125; (D1, D3) = 25; (D1, D4) = 512; (D1, D5) = 90;

(D2, D4) = 180; (D2, D5) = 700; (D3, D4) = 72; (D3, D5) = 2130;

The figure below illustrates the initial store plan:

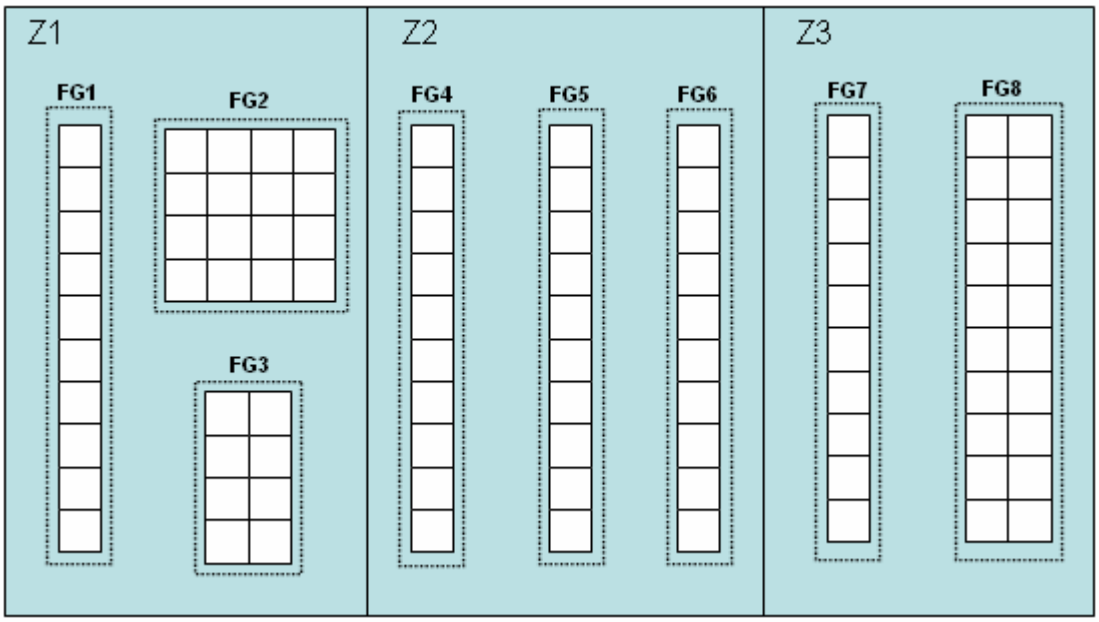


Figure 10 - Example: Initial store plan

The algorithm initiates with zone Z1 and allocates D1 to the closest fixture-group to Z3 since the largest co-occurrence value is (D1, D4) = 512 and D4 Є Z3. On the other hand D4 is not yet allocated. Suppose we also allocated D4 on the closest fixture-group to Z1 which is FG7. When proceeding to zone Z2, we find that the largest co-occurrence value corresponds to (D3, D5) = 2130 where D5 Є Z3. If we allocate D3 to FG6 and D5 to FG8 (since it the only available one), this will be considered as an unfair distribution since D5 should have been allocated on FG7 (the closest fixture-group to Z2) since (D3, D5) > (D1, D4). Hence, the previous statement is justified.

3.4 Auto-Segmentation Engine (ASE)

3.4.1 Introduction

The main purpose of the auto-segmentation engine is selecting the optimal product assortment for a given item category. By assortment, we mean choosing what items are to be included in the planogram and hence placed on the shelves. It is to note that not all items of a certain category are included in its planogram. To solve such a problem, we use a special type of Artificial Neural Networks, called Self Organizing Maps (or Kohonen Maps).

3.4.2 Non-functional requirements

We assume that the following non-functional requirements are to be satisfied by ASE:

- Scalability: the segmentation process is tightly bound to the strategy defined by the retailers, which usually varies based on the goals and circumstances. Hence, the engine needs to be able to process multiple dynamic dimensions or factors.
- Accuracy: An optimal assortment directly affects the performance of the store since it includes selecting what items to allocate on the physical fixtures. Hence, the auto-segmentation engine should be very accurate in reflecting the assortment strategy defined by the retailer.
- Interoperability: The segmentation process needs to take into consideration geo-spatial, demographic and analytical data. Hence, the engine needs to collect information from multiple heterogeneous systems and export results to visualization packages (i.e. Store plan rendering, Category segment graph…)

3.4.3 Functional requirements

The engine needs to analyze high-dimensional data corresponding for a certain category, generate the corresponding clusters and select the optimal ones based on the strategy defined in the business rule repository. This process needs to be repeated for all categories that need to be included in the studied store. Once the optimal assortment is selected, the engine communicates with the auto-allocation module to distribute the chosen items on the corresponding shelves.

3.4.4 Algorithm / approach selection

The main purpose of the auto-segmentation engine is the optimal assortment per category, which is "dividing" the items belonging to a given category (based on multiple dimensions reflecting a certain strategy) and selecting the optimal clusters, the ones which reflect the most the desired strategy. Given the dynamic automation objective in our research and the inability to specify the desired output, our selection scope can be narrowed down to unsupervised learning methodologies. One form is basic "clustering" algorithms, like K-means and vector quantization. Though these clustering algorithms are simple, they have several drawbacks. The radius of clusters or the number of clusters has to be predefined. Minor changes in these values will lead to change in the output which is not desirable [14]. In our case, we cannot pre-define the number of clusters for a given category, hence another approach is to be considered. The other alternative to unsupervised learning problems is adopting Artificial Neural Networks (ANN). Among ANNs, Self-Organizing Maps (SOM) and Adaptive Resonance Theory (ART) are the most commonly used approaches for unsupervised learning problems. The main advantage of ART is the ability to control the degree of similarity between the members of the same cluster via a pre-defined parameter called "vigilance parameter". On the other hand, SOM are used for multi-dimensional data and are able to preserve the topology through the neighborhood function; in a topology-preserving map, units located physically next to each other will respond to classes of input vectors that are likewise next to each other. Back to our application, given that the items of a certain category need to be segmented based on a set of dynamic criteria and that the optimal clusters are topologically adjacent, the main properties of SOMs respond more efficiently to the requirements of our auto-segmentation module.

3.4.5 Global flow

The flow initiates by initializing the different segmentation dimensions or criteria which reflect the strategy defined by the retailer. Next, the Self-Organizing Map is initialized and trained in terms of size (number of neurons), initial neuron values and learning parameters. Once the quality of the map is ensured, the input data is mapped to the trained map and the optimal obtained clusters are selected.

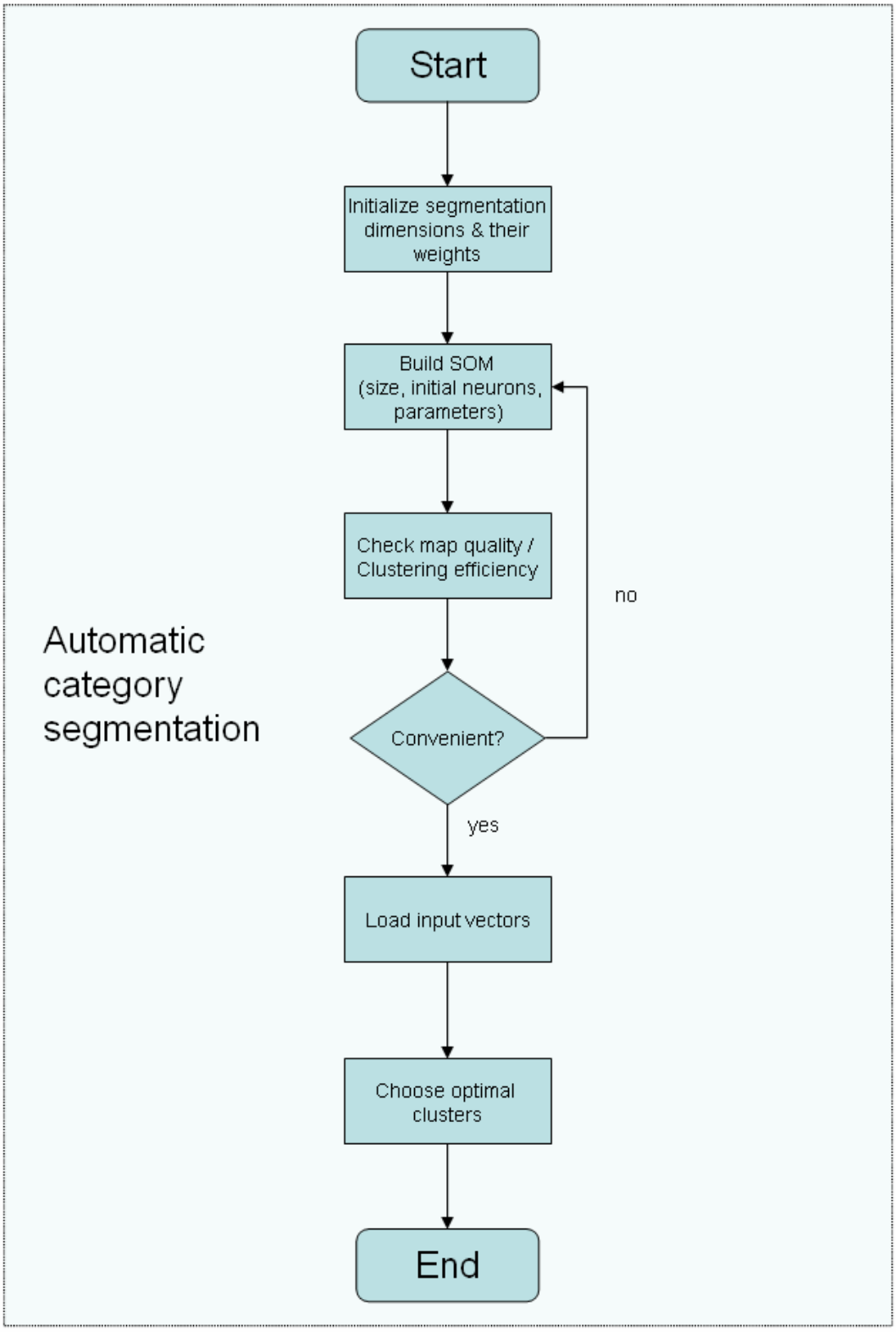


Figure 11 - Auto-segmentation engine: global flow

3.4.6 Self-Organizing Maps (SOM)

3.4.6.1 Artificial Neural Networks

An Artificial Neural Network (ANN) is a computational model, inspired from the structure of the brain, and used to transform a given input space into a desired output space.

The image below illustrates how ANN are related to the physiological neural system; different neurons cooperate based on certain functional similarities in order to produce the corresponding output. Per example, neurons in the occipital lobe of the brain are connected by their synapses in order to process the visual functions. The main difference between biological neural systems and ANN is their organization; unlike the heterogeneous organization of biological systems, the majority are of ANN are organized according to the same basic structure.

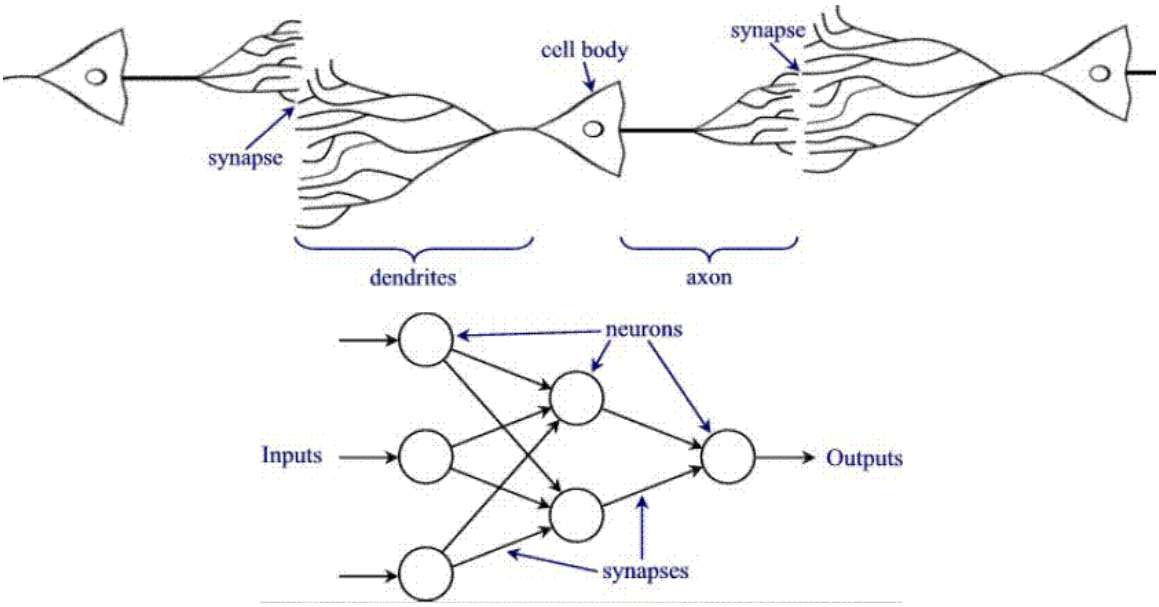


Figure 12 - ANN biological inspiration

An ANN can be used for the following computational applications:

- Classification: like pattern recognition, image comparison, behavior extraction and data clustering.

- Noise reduction: by recognizing noisy patterns in a certain input and producing a noiseless output.
- Prediction: forecasting based on historical data.

The main power of neural networks is their ability to "learn" their function based on the sample input and produce the corresponding generalized output without any pre-configuration. In order for an ANN to determine its functions, we need to assign weights for it. Such a process is applied through "training", by providing sample data to the network and adjusting the weights to fit the desired function. There are two types of learning:

- Supervised training: by supplying the network with the input as well as the desired output and modifying the weights to minimize the difference between the actual and desired output.
- Unsupervised training: by supplying the network with the input exclusively; the network identifies the similarities and differences in the input without any intervention or adjustment.

Depending on how the neurons are connected, ANN can be divided into three basic categories:

- Unidirectional networks: mainly one-layer networks, multi-layer networks and radial networks.
- Recursive networks: including Hopfield networks, Hamminga networks and Bam networks.
- Cellular networks.

3.4.6.2 Self-Organizing Maps

Self-Organizing Maps (SOM), also known as Kohonen Maps, are special types of Artificial Neural Networks similar to biological systems; Self-Organizing Maps are inspired from the topology of the human brain which is divided into regional clusters of informational representation (i.e. human sensory and motor maps). SOM are mainly based on Vector Quantization, a technique of representing multi-dimensional data in lower dimensional spaces, usually one or two dimensions. Moreover, the approach

generates a network which preserves the topology of the training data. Such a mapping can be achieved by the following aspects:

- Input layer and output layer (map) are completely connected.
- Output neurons, which are the nodes of the map, are interconnected through a neighborhood function.

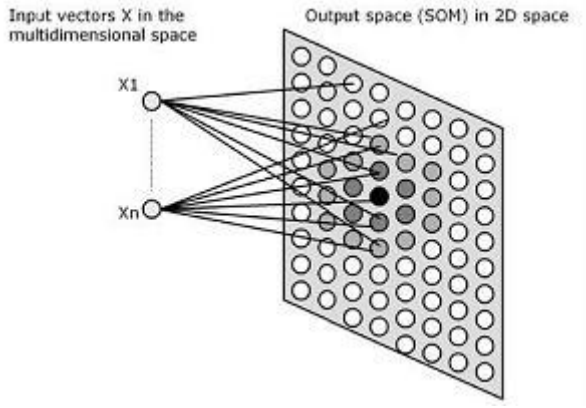


Figure 13 - Self-Organizing Maps (retrieved from http://www.ij-healthgeographics.com/content/download/figures/1476-072X-3-12-7.TIFF)

SOM have been intensively adopted in real-life applications, mainly for data mining, process analysis and control as well data analysis in economical and commercial fields. Examples of SOM include mapping of colors from their 3-dimensional representation (Red, Blue, Green) into a 2-dimensional grid representation and clustering of geographic maps based on certain criteria (poverty, population size, etc…). In a paper entitled "A cross-national market segmentation of online game industry using SOM" [15], the authors adopt a two-level SOM to develop clusters within each nation concerning the studied online game industries. The authors state that adopting SOM helped to effectively reduce the complexity of the reconstruction task and noise. Another paper entitled "Unsupervised segmentation using a self-organizing map and a noise model estimation in sonar imagery" [16] presented SOM as an effective approach to segment images provided by a high-resolution sonar; the learning of a Kohonen self-organizing map (SOM) is performed directly on the input image to approximate the discriminating functions, i.e. the contextual distribution function of the grey levels. These papers and many others

show how SOM can be used to efficiently cluster n-dimensional data; such an approach can be interesting to segment a retail store to different categories based on multiple criteria or dimensions (i.e. affinity, performance, consumer behavior, etc…).

3.4.6.3 Unsupervised Learning

One of the main characteristics of Self-Organizing Maps is unsupervised learning, which means the ability to classify data without needing to specify a target vector. In contrast, supervised training techniques such as back-propagation requires comparing the output vector to the target vector; if there's a difference, the weights of the nodes are altered to reduce the error in the output. This process is repeated several times until the network reflects the desired output.

Given an n-dimensional input space and m output neurons, unsupervised learning operates as follows:

1. Randomly generate a weight vector W for the m neurons.
2. Choose a random input x from the training data.
3. Iteratively examine all the nodes of the map and compare their weights to the input vector's weight; the closest matching node is selected and referred to as the Best Matching Unit (BMU).
4. Calculate the radius of the neighborhood of the BMU; this value, initially large and set to the radius of the map, diminishes after each iteration. Then, update the weight vectors of all neurons i in the neighborhood of the examined neuron k, as follows:

   $wi := wi + \eta \cdot \varphi(i, k) \cdot (x - wi)$, where φ is the neighborhood function and η the learning parameter (both explained below). Any nodes found within this radius are deemed to be inside the BMU's neighborhood.
5. Narrow neighborhood function φ and learning parameter η and repeat step 2.

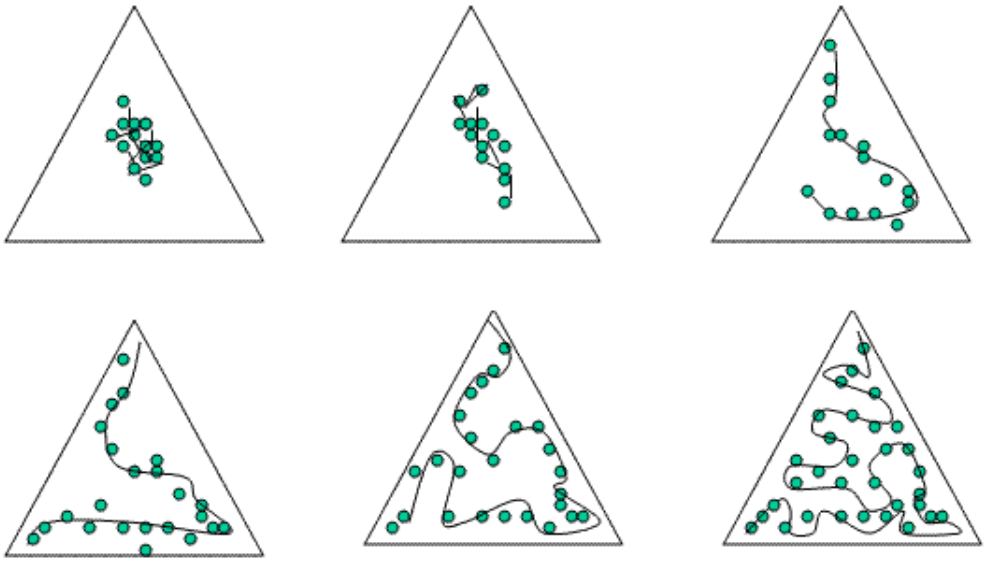


Figure 14 - Learning of a triangular input space

3.4.6.3.1 Calculating the BMU & Neighborhood

The BMU is selected by calculating the Euclidian Distance, as follows:

$$Dist = \sqrt{\sum_{i=0}^{i=n}(V_i - W_i)^2}$$

where (W1, W2, … , Wn) are the node's weights, and (V1, V2, … , Vn) are the input vector's values.

As for the node's neighborhood, it is expressed as an exponential decay function which decreases from iteration to iteration until reaching the value of the BMU. It can be formulated as follows:

$$\sigma(t) = \sigma_0 \; exp\left(-\frac{t}{\lambda}\right)$$

where σ (0) is the width of the radius at time t_0, λ is a time constant and t is the current iteration.

On the other hand, the effect of location within the neighborhood is defined by a

Gaussian curve as follows:

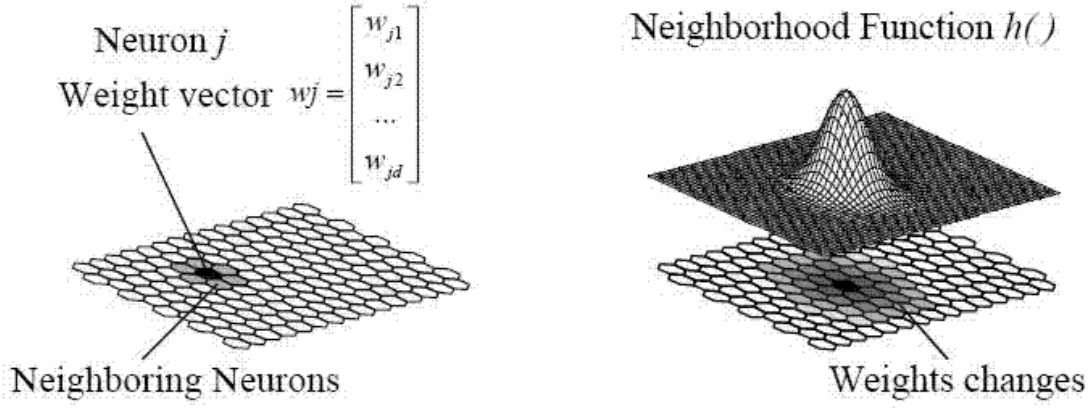


Figure 15 - Effect of location within the neighborhood

Finally, to ensure the convergence of the map, the learning rate L is defined as follows:

$$L(t) = L_0 \; exp\left(-\frac{t}{\lambda}\right)$$

3.4.6.4 Map Initialization

It is often reported in the technique literature that the success of the self-organizing feature map formation is critically dependent on the initial weights and the selection of main parameters (*i.e.* the learning-rate parameter and the neighborhood set) of the algorithm [17]. Before selecting the approach that we will adopt to initialize the map, we will present the different general methodologies used for this purpose.

3.4.6.4.1 Random Initialization

This is the most straight-forward approach consisting of initializing the map nodes with random values. Even though it's the easiest methodology, it's poorly efficient since it requires an additional number of training cycles.

3.4.6.4.2 Initialization using Random Training Data

This approach consists of selecting a random set of input vectors from the training data and using it to initialize the map. This method is certainly more efficient than the random initialization since it will produce a basic map state reflecting the input data, and hence

decrease the needed training cycles and computational time. However, such a state will not be very accurate: given that the map nodes are much less than the training data vectors and that the chosen set is picked randomly, the initialized map will end up reflecting a partial representation of the overall training data.

3.4.6.4.3 Initialization using Selected Training Data

In contrast to the previous approach, this method consists of selecting a more meaningful set of input vectors in order to reproduce a more accurate map state. To achieve this, we need to cluster our data before choosing the initialization set, which seems not that easy given the multi-dimensional aspect of the training data. Hence, Principle Component Analysis (PCA) can be used to reduce the dimensionality, while preserving the variability of the data .PCA involves a mathematical procedure that transforms a number of possibly correlated variables into a smaller number of uncorrelated variables called principal components. The first principal component accounts for as much of the variability in the data as possible, and each succeeding component accounts for as much of the remaining variability as possible.

3.4.7 Parameter selection

The selection of the map's initialization parameters depends on the purpose and outcome of the scenario, but there are a number of recommendations suggested by Kohonen to optimize the results. Below are the main "best practices":

The map dimensions affect its visualization; a small map is recommended for cluster identification purposes.

The length of the reference vector P(x) must be 1.3 times the length of the reference vector P(y).

The learning factor must be large in the initial training phase and relatively small in the final phase.

The initial network radius should be rather large, preferably larger than half the network diameter [10].

3.4.8 Implementation

The scenario for the auto-segmentation engine is divided into two parts: generation of the store plan and segmentation of a given planogram.

3.4.8.1 Planogram segmentation

The studied scenario will be applied to the category "pain relievers", consisting of 68 items, from which an optimal item assortment is to be selected. Based on the objective specified by the retailer, we will generate a self-organizing map, which clusters the multi-dimensional data and helps to decide which segments to choose for the assortment. The following variables (dimensions) will be taken into consideration:

- Last year total sales volume
- The Net Profit Margin: calculated as follows:

$$\text{Net profit margin} = \frac{\text{Net profit (after taxes)}}{\text{Revenue}} \times 100\%$$

- Indispensability of the item: if this value is true, this means that it is mandatory to select the corresponding item.
- Profitability: which is a value computed by the forecast engine specifying how much an item is profitable in respect to its spatial dimensions.

Using the parameter initialization tips mentioned in the previous section, we will apply our scenario on two map variations: small map (4*3 with diameter = 6) and large map (12*8 with diameter = 40). The learning factor has a value of 0.5 in the initial phase and 0.05 in the final phase.

3.4.8.1.1 Using a small map

The following results are obtained (colors of the segments will be illustrated later in Figure 16 - Planogram segmentation scenario 1: Self-Organizing Map):

Segment	Color	Frequency (%)	LY Sales	Net Profit Margin	Is Indispensable	Profitability
S1		22.06	438	23.87	0	0.5047

S2		5.88	393	76.25	1	0.3925
S3		25	182	37.18	0	0.8324
S4		16.18	196	57.82	0	0.2291
S5		14.71	220	24.3	0	0.237
S6		16.18	639	71.55	0	0.6918

Table 1 - Planogram segmentation scenario 1: Obtained segments

The table above shows the 6 segments obtained in the resulting map. By analyzing the attributes, it is clear that the segment S2 includes all indispensable items (Is Indispensable = 1), hence it will be automatically selected.

The items included in S2 are shown in the table below:

Item #	LY Sales	Net Profit Margin	Is Indispensable	Profitability
19	497	70	1	0.85
38	415	81	1	0.08
56	299	79	1	0.68
60	362	75	1	0.29

Table 2 - Planogram segmentation scenario 1: Indispensable items

Suppose that the retailer needs to choose around 25% of the items of each category and assign them to the corresponding planogram. After selecting the indispensable items, one additional cluster can be chosen (all remaining segments have a frequency between 14.71% and 25%). By reviewing the attributes of each cluster and given that the strategy tends to maximize sales / net profit margin / profitability, the segment S6 must be obviously selected (LY Sales = 639, Net Profit Margin = 71.55 and Profitability = 0.6918). Below are the items corresponding to the segment S6:

Item #	LY Sales	Net Profit Margin	Is Indispensable	Profitability
1	528	72	0	0.82

10	831	64	0	0.78
27	711	84	0	0.43
33	243	91	0	0.72
37	352	84	0	0.55
42	702	62	0	0.59
47	523	83	0	0.79
50	690	61	0	0.62
51	522	65	0	0.72
63	902	59	0	0.81
68	1025	62	0	0.78

Table 3 - Planogram segmentation scenario 1: Optimal cluster(s)

Below is the graphical presentation of the self-organizing map's clusters:

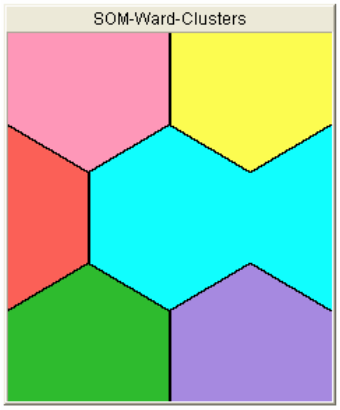


Figure 16 - Planogram segmentation scenario 1: Self-Organizing Map

The optimal selected segment corresponds to the upper-left pink segment in the above figure. To justify the accuracy of the map, we will inspect the items belonging to the bottom-right purple segment; such items must be the least compatible with the specified strategy. Below is the list of these items:

The table below shows the total efficiency for each cluster (assuming that the sales volume, net profit margin and profitability have the same priority). The efficiency is the sum of the ratio of the attribute over its maximum value for all indicators (except item indispensability):

Segment	LY Sales / maximum LY Sales	Net Profit Margin / maximum Net Profit Margin	Profitability / maximum Profitability	Efficiency
S1	0.685	0.313	0.606	1.604
S2	0.615	1	0.472	2.087
S3	0.285	0.488	1	1.773
S4	0.307	0.758	0.276	1.341
S5	0.344	0.319	0.285	0.948
S6	1	0.938	0.831	2.769

Table 4 - Planogram segmentation scenario 1: Calculating the segments efficiency

The results above show that segments S4 and S5 (green and purple segments respectively) have the lowest efficiency value, while the optimal segment is S6 (pink segment). This can be justified visually by the distance between the optimal segment and the least efficient segments (which are the furthest of the optimal segment).

Furthermore, the picture below illustrates the map clusters for each of the four dimensions. It is clear that the indispensability dimension clustered the map into 2 main regions: all 4 indispensable items (red region) on one side, and all the other non-indispensable items on the other side. In terms of overall sales (first part of the figure), it is clear that the selected optimal cluster outperforms the others significantly. As for the net profit margin and the profitability dimensions, the optimal cluster is among the top two in both cases. The relatively acceptable performance of S3 (top-right segment), third-best in terms of overall efficiency, can be explained by its leading score in terms of profitability.

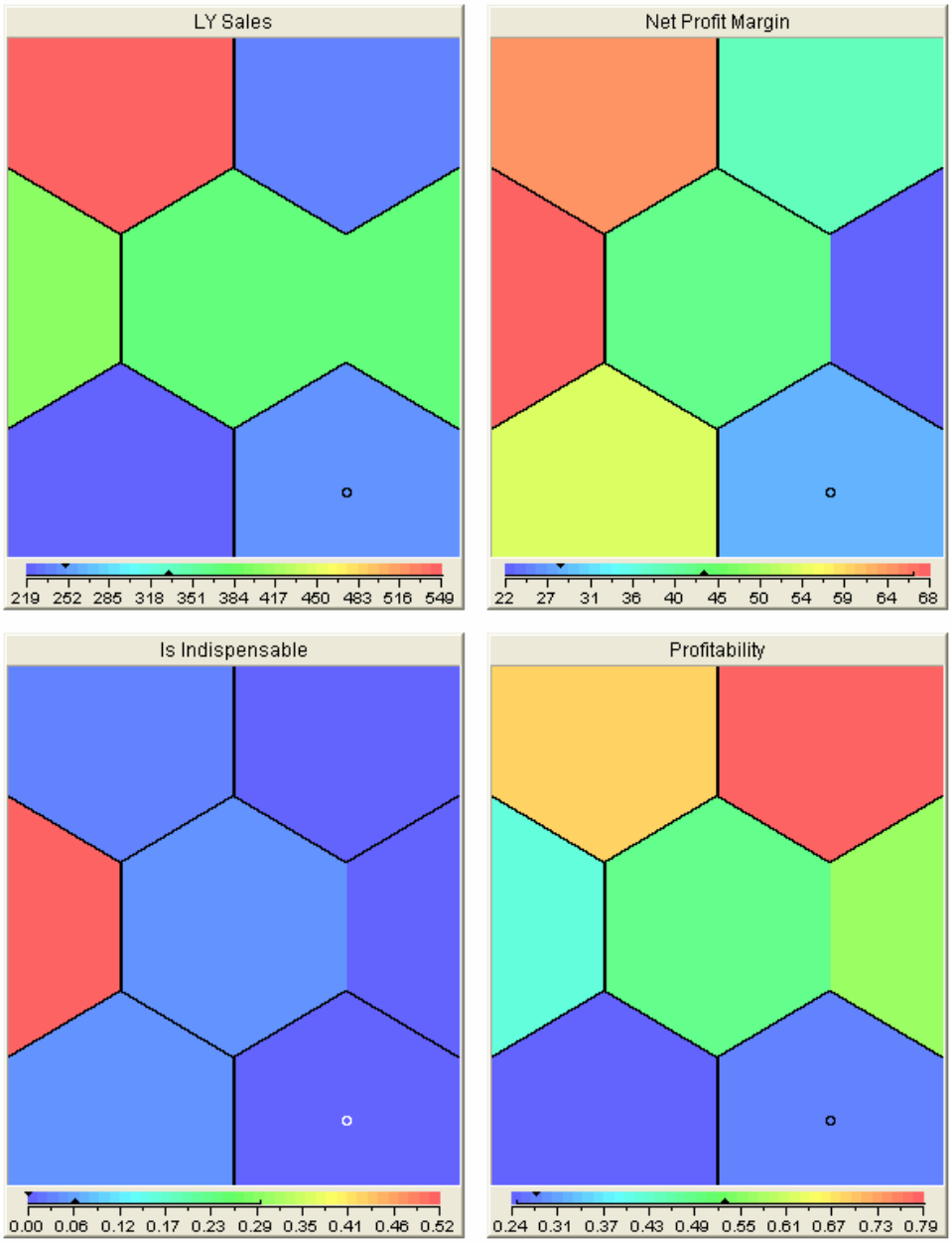


Figure 17 – Planogram segmentation scenario 1: Clusters for each variable

3.4.8.1.2 Using a large map

Using a large map of size (12*8), the following segments are obtained:

Segment	Color	Frequency (%)	LY Sales	Net Profit Margin	Is Indispensable	Profitability

S1		51.47	301	35.17	0	0.3309
S2		26.47	194	35.44	0	0.835
S3		16.18	639	71.55	0	0.6918
S4		5.88	393	76.25	1	0.3925

Table 5 - Planogram segmentation scenario 2: Obtained segments

Below is the graphical representation of the self-organizing map's clusters:

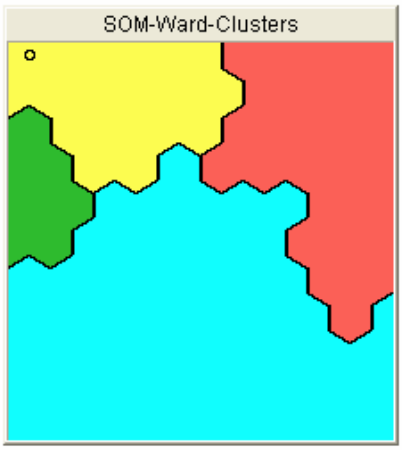


Figure 18 - Planogram segmentation scenario 2: Self-Organizing Map

Similarly to the previous scenario, segment S4 is picked since it includes all indispensable items. The additional segment will be selected based on its efficiency.

Segment	LY Sales / maximum LY Sales	Net Profit Margin / maximum Net Profit Margin	Profitability / maximum Profitability	Efficiency
S1	0.471	0.461	0.396	1.328
S2	0.304	0.465	1	1.769
S3	1	0.938	0.829	2.767
S4	0.62	1	0.47	2.09

Table 6 - Planogram segmentation scenario 2: Calculating the segments efficiency

Based on the results above, segment S3 will be selected. To validate if it includes the same items selected in scenario 1, we will extract the records corresponding to it:

Item #	LY Sales	Net Profit Margin	Is Indispensable	Profitability
1	528	72	0	0.82
10	831	64	0	0.78
27	711	84	0	0.43
33	243	91	0	0.72
37	352	84	0	0.55
42	702	62	0	0.59
47	523	83	0	0.79
50	690	61	0	0.62
51	522	65	0	0.72
63	902	59	0	0.81
68	1025	62	0	0.78

Table 7 - Planogram segmentation scenario 2: Optimal cluster(s)

It is clear that the same optimal clusters are obtained in both scenarios (small and large map respectively). The only difference is their number; using a small map, we obtained a larger number of clusters. Given the fact that the segmentation process is a clustering approach in its nature, we recommend using relatively small maps for the ASE.

3.5 Auto-Allocation Engine (AAE)

3.5.1 Introduction

The purpose of the auto-allocation engine is to optimally distribute items belonging to a certain category (optimal assortment generated by the auto-segmentation engine for the related category) over the corresponding fixtures. Such a process must take into consideration a set of dynamic rules specified by the user in the business rule repository and must ensure optimal profitability while maximizing space utilization. We will show in the coming sections how the dynamicity of the strategy is maintained and how the problem is mapped to Bounded Knapsack Problem. Moreover, the different implementation aspects of the engines are explained.

3.5.2 Non-functional Requirements

We assume that the following non-functional requirements are to be satisfied by the auto-allocation engine:

- Performance: Given the huge amount of data that needs to be processed by the engine as well as the broad number of categories within a store, the engine needs to have an optimal performance (i.e. less than one second for an average size segment, consisting of less than a hundred items and a dozen of fixtures).
- Scalability: The architecture of the engine needs to be flexible enough to allow interaction with other modules of an enterprise application (i.e. fetching data from heterogeneous data sources, requesting statistical data from analytical engines and exporting results to presentation interfaces, including 3D rendering). The service-oriented aspect, which will be discussed in the architecture-related section, ensures the satisfaction of this crucial requirement.
- Accuracy: The main purpose of the engine is to maximize the profit derived from a certain strategy defined by the end user. Hence, this should be reflected accurately in the output; per example, if space maximization is the highest priority rule, then the engine's output should intelligently consider minimizing space wasting.

- Dynamicity: Given the fact that the merchandising rules change frequently from period to period (i.e. seasonality, promotion, competitor strategy, etc…), the engine should be able to process dynamic sets of rules and reflect the underlying strategy.

3.5.3 Functional Requirements

We assume that the following functionalities and capabilities are to be provided by the auto-allocation engine:

- Rule parsing & execution: the engine needs to parse the merchandising rules defined in the business rule repository and apply the priority weights.
- Item ranking: by classifying the different items based on the different attributes and provided criteria.
- Applying mandatory rules and properties: defined by the user, such as horizontal / vertical spacing between items, minimum facings, etc…
- Item allocation: Based on the results of the above steps, the engine can locate the horizontal / vertical coordinates of the item on the corresponding shelf / fixture.
- Space maximization: by optimally ensuring that the space / profit constraints are satisfied. This problem is mapped to the bounded Knapsack problem, explained in coming sections.

3.5.4 Engine Phases

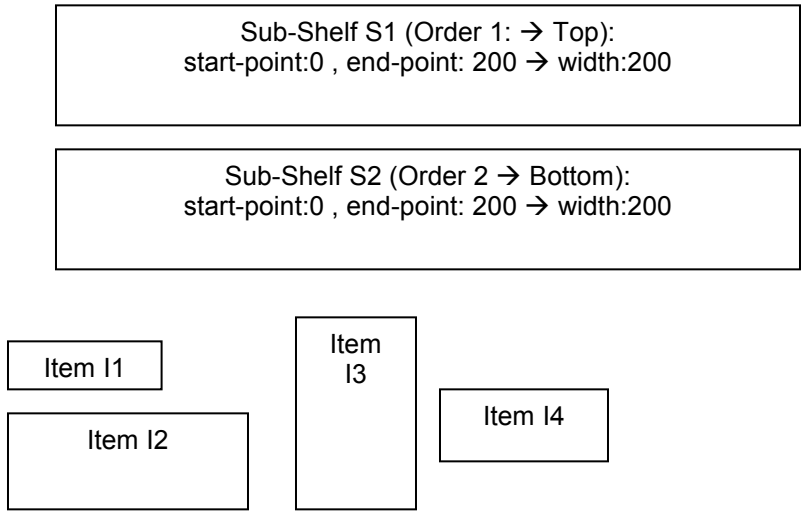
Suppose a certain planogram (set of fixtures, each consisting of several shelves) is segmented by the user. As explained in the previous sections, on the micro level, segmentation means that the user specifies that a certain percentage of the overall planogram corresponds to a specific category of items. On the macro level, the store is divided into regions where categories are allocated to corresponding sets of physical fixtures and shelves.

Per example: (Note that results are not accurate and intended for illustration purposes)

<table>
<tr><td rowspan="2">A
(50%)</td><td colspan="2">B
(25%)</td></tr>
<tr><td>C
(12.5%)</td><td>D
(12.5%)</td></tr>
</table>

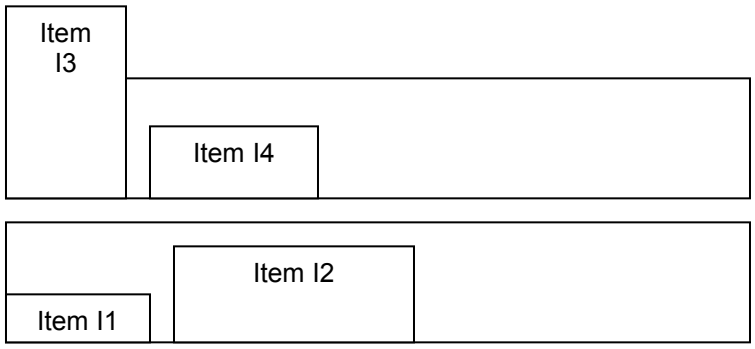
Based on the above, the application supplies the auto-allocation engine with information about each segment (from A to D), consisting of:

- Set of items corresponding to the related category
- Set of sub-shelves

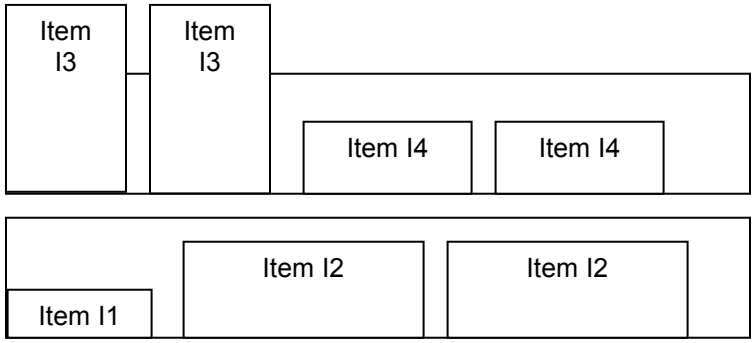


Next, the vertical merchandising rules prioritized by the user are applied. Per example, new items are to be placed from top to bottom, from left to right. By iteratively applying

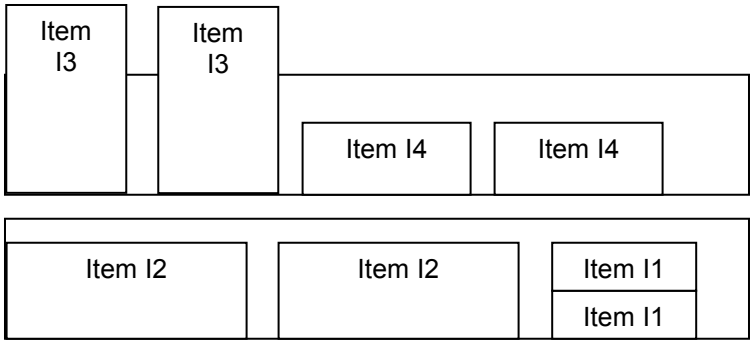
these rules (from lowest priority to highest priority), we will end up knowing which items are to be placed on which sub-shelves.



The next step is to fill up the wasted space on each shelf by adding "facings". To solve this problem, we will assume that we have a 0-N (bounded) knapsack problem (items having a width W and a profitability P). By applying the algorithm, we will be able to maximize space utilization.



The final step is to apply horizontal merchandising rules which will arrange items on the same sub-shelf (local brand items are to be placed on the left of national brand items), as well as apply "stacking" rules (items stacked vertically).



3.5.5 Global flow

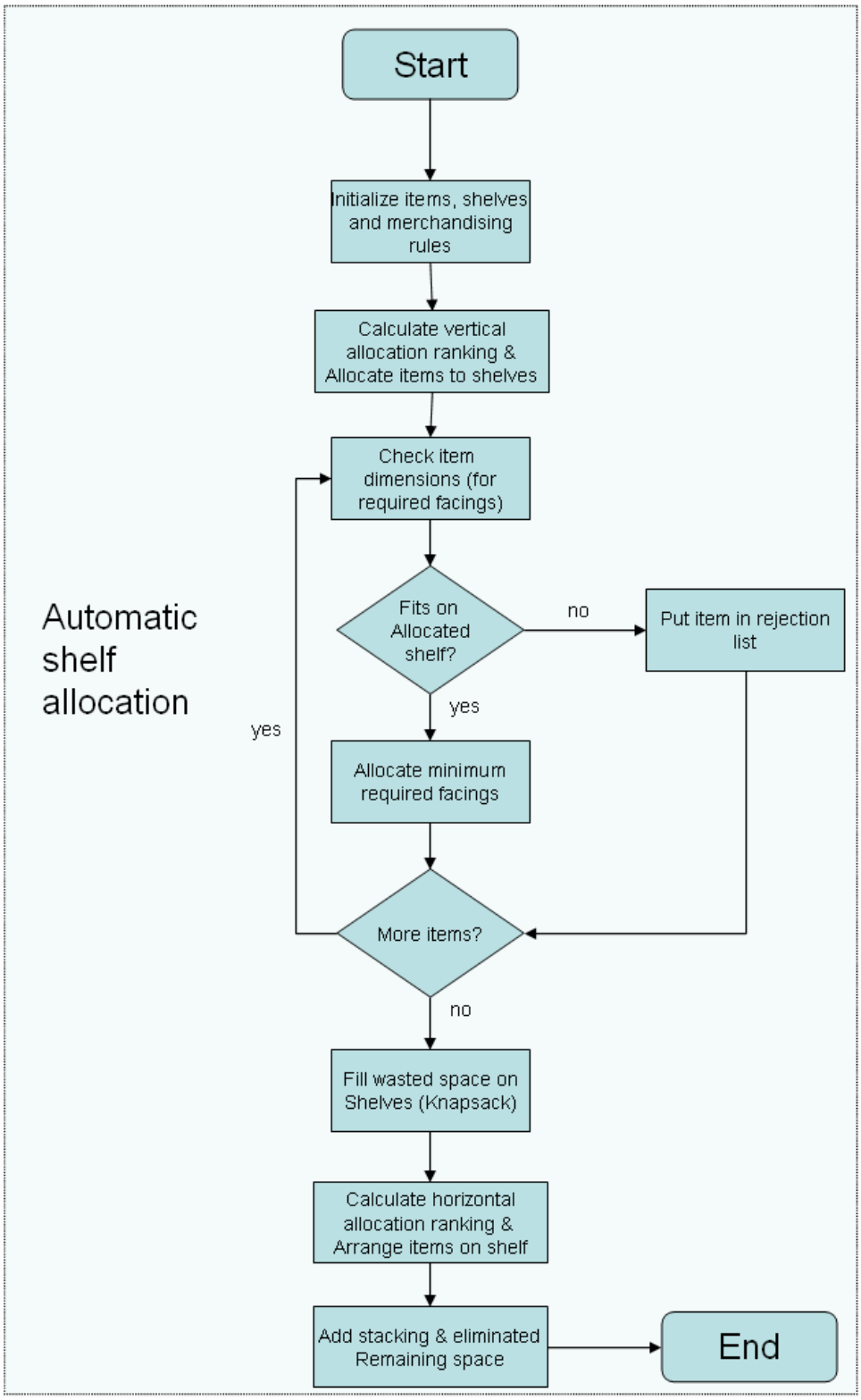


Figure 19 - Auto-allocation engine: global flow

3.5.6 Knapsack Problem

3.5.6.1 Introduction

The knapsack problem or rucksack problem is a problem in combinatorial optimization: Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than a given limit and the total value is as large as possible. It derives its name from the problem faced by someone who is constrained by a fixed-size knapsack and must fill it with the most useful items.
The problem usually arises in financial problems as well combinatorics, complexity theory, cryptography and applied mathematics. We will introduce this problem to solve the problem of retail space planning (i.e. shelf allocation) with the presence of dynamic sets of prioritized merchandising rules.

3.5.6.2 Variations

There are three main variations of the N-P complete Knapsack problem:

- 0-1 Knapsack problem: the most common variation of the problem, where 0 or 1 copy of each item can be included in the knapsack.
- Bounded (0-N) Knapsack problem: where 0 to N copies of each item can be included in the knapsack. N is the maximum number of copies for each item.
- Unbounded Knapsack problem: where an unlimited number of copies of each item can be included in the Knapsack.
- Multiple-choice Knapsack problem: where we have a set of classes, each consisting of several items and one item is to be picked from each class.
- Subset sum problem: if for each item, the profit and weight are equal.

The third variation is the most complex. It is to note that the multiple Knapsack problem (i.e. more than one knapsack is introduced) can be formulated as the “Bin Packing” problem. Our approach can be formulated as a bounded Multi-knapsack problem with the presence of multiple dynamic constraints.

3.5.6.3 Solving the Knapsack Problem

All Knapsack problems belong to the family of NP-hard problems, meaning that it is very

unlikely that we can ever devise polynomial algorithms for these problems. But despite the exponential worst-case solution times of all Knapsack algorithms, several large scaled instances may be solved to optimality in fractions of a second [12]. To do so, multiple computer science algorithms and approaches can be adopted [11], notably:

- Dynamic Programming: which is a metatechnique, not an algorithm, similar to "divide & conquer". It is used when the examined problem can be divided into recurring sub-problems; the resulting solution of each sub-problem is stored in memory for reuse. Additional details about dynamic programming will be provided in the next section.
- Branch-and-Bound: is a general algorithm for finding optimal solutions of various optimization problems, especially in discrete and combinatorial optimization. It consists of a systematic enumeration of all candidate solutions, where large subsets of fruitless candidates are discarded *en masse*, by using upper and lower estimated bounds of the quantity being optimized.
- Greedy approximation algorithm: which is frequently used to obtain sparse solutions to learning problems. In a paper entitled "Approximation algorithms for the multiple knapsack problem with assignment restrictions" [18], the authors show that simple greedy approaches yield 1/3-approximation algorithms for the objective of maximizing assigned weight of the studied problem. Two different 1/2-approximation algorithms are proposed: the first one solves single knapsack problems successively and the second one is based on rounding the LP relaxation solution.
- State Space Relaxation: which is a concept related to dynamic programming, where the optimization problem can be formulated as a task to find the smallest-cost path of transitions between initial and final states by exhaustively exploring the corresponding state-space.

Those are some, and not all, of the main approaches used to solve the Knapsack problem. Additional approaches include preprocessing, genetic algorithms, backtracking and metaheuristics.

3.5.7 Dynamic Programming

3.5.7.1 Why not Brute Force?

The most basic solution to the Knapsack problem would be adopting Brute Force. Suppose we have a set of n items, each having a profit P and size S, and we need to fill a shelf of size W. The mentioned approach initiates by generating all 2^n subsets, eliminating all subsets whose sum of sizes exceed W and select the maximum total profit of the remaining subsets.

Example: 0-1 Knapsack problem: only one copy of each item is allowed

We have 3 items (item, size, profit):

(item A, 9, 12.5), (item B, 6, 10) and (item C, 14, 4) and a Knapsack of size W=20.

The Brute Force approach initiates 2^3=8 possible subsets:

S1: empty set

S2: item A with total profit of 12.5 and total size of 9

S3: item B with total profit of 10 and total size of 6

S4: item C with total profit of 4 and total size of 14

S5: item A + item B with total profit of 22.5 and total size of 15

S6: item A + item C with total profit of 16.5 and total size of 23

S7: item B + item C with total profit of 14 and total size of 20

S8: item A + item B + item C with total profit of 26.5 and total size of 29

The subsets S6, S7 and S8 are eliminated since their total size exceeds W. The subset S5 is selected since it has the maximum total profit between the remaining sets. Even for the simplest variation of the Knapsack problem (0-1) and for a small problem (3 items and a single knapsack), such a process has a runtime of $O(2^n)$, which is obviously not suitable for large-scale applications such as retail shelf allocation; even for a simple planogram consisting of one shelf and 50 items, brute force can take up to 2^{50} operations to derive the optimal allocation, which is clearly inconvenient.

3.5.7.2 Greedy Approach: Performance vs. Optimality

The greedy approach obtains the optimal solution by passing through a series of choices. At each pass, the algorithm makes the best local solution without referring to results from

previous sub-problems; this is the main aspect that differentiates it from dynamic programming. However, the greedy approach shares some similarities with dynamic programming such as optimal substructure (An optimal solution to the entire problem contains within it optimal solutions to sub-problems) and recursive solutions.
Theoretically, the greedy approach is more efficient than dynamic programming in terms of simplicity and performance but it cannot be useful to in our case, as shown in the example below.

Counter-example: 0-1 Knapsack problem:

We have 3 items (item, size, profit):

(item A, 25, 10), (item B, 10, 9), (item C, 10, 9) and Knapsack of size W=20.

Suppose the greedy strategy consists of picking the items with highest profit first → The strategy will pick item A, yielding a total profit of 10 while the **optimal** solution consists of picking B and C, yielding to a total profit of 18.

Hence, given the crucial need of optimality in retail shelf allocation, we can discard the greedy approach. Nevertheless, such a methodology can be useful in solving the Fractional Knapsack Problem, where fractions of an item can be included in the Knapsack; this approach can be suitable for solving the Cutting Stock problem (i.e. papers rolls in an industrial mill).

3.5.7.3 Dynamic Programming for the Knapsack Problem

The shelf auto-allocation problem can be solved using dynamic programming since it satisfies its main two characteristics:

- Principle of Optimality (or Optimal substructure): since the optimal solution to our problem contains within it optimal solutions to sub-problems
- Overlapping sub-problems: since in certain cases, the same sub-problem is solved more than once.

Using dynamic programming, the Knapsack Problem can be expressed by the following recursive formula:

$$B[k,w] = \begin{cases} B[k-1,w] & \text{if } w_k > w \\ \max\{B[k-1,w], B[k-1,w-w_k]+b_k\} & \text{else} \end{cases}$$

which means that the subset cannot contain an item k if the total weight becomes larger than the Knaspack's weight w; otherwise, we choose between two subsets based on the largest total profit:

- Total profit of original subset (without item k)
- Total profit of new subset (with item k added)

Below is the pseudo-code for the 0-1 Knapsack Problem using Dynamic Programming:

```
for w = 0 to W
   B[0,w] = 0
for i = 1 to n
   B[i,0] = 0
for i = 1 to n
   for w = 0 to W
      if w_i <= w // item i can be part of the solution
            if b_i + B[i-1,w-w_i] > B[i-1,w]
                  B[i,w] = b_i + B[i-1,w- w_i]
            else
                  B[i,w] = B[i-1,w]
      else B[i,w] = B[i-1,w]  // w_i > w
```

Hence, the algorithm repeats O(W) n times resulting in an overall time of O(n*W), where n is the number of items and W is the weight of the Knapsack. The pseudo-polynomial time is divided as follows: O(nw) times to fill the array, which has (n +1)*(w +1) entries, each requiring O(1) time to compute and O(n) time to trace the solution, because the tracing process starts in entry n of the array and moves up 1 row at each step.

3.5.7.4 Memoization vs. Dynamic Programming

Memoization is a top-down variation of dynamic programming based on the concept of storing solutions to sub-problems as solved in the recursion algorithm. The solutions are stored in a table-like structure, indexed by the arguments of the corresponding function. Such an approach can outperform dynamic programming when not all solutions to sub-problems are needed. The following summarizes the difference between the two approaches:

- Memoization has the overhead of recursion, but computes fewer entries in the table than dynamic programming.

- Dynamic programming avoids the overhead of recursion, but computes more entries than necessary

To determine if it's suitable to adopt it for the retail auto-allocation engine, let us analyze a real-life scenario for a given simple planogram.

Example: Consider the following 10 items (profit, width in cm) that needs to be allocated on a small shelf of width W=50 cm (Assume that 0-1 copy of each item is allowed → 0-1 Knapsack Problem):

Item1 (5,28) Item2 (9,26) Item3 (6,22) Item4 (1,34) Item5 (7,28)

Item6 (8,20) Item7 (7,17) Item8 (4,14) Item9 (5,18) Item10 (3,16)

The following graph illustrates the progress of the dynamic programming algorithm:

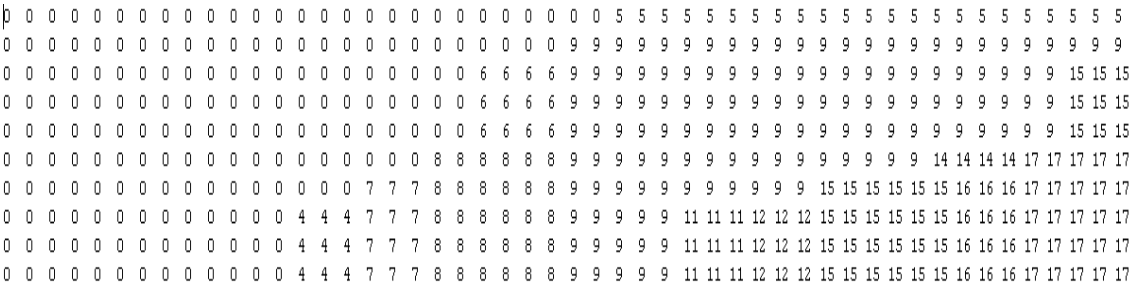


Figure 20 – Scenario 1: Memoization vs Dynamic Programming

The main difference is that dynamic programming tries to fill the knapsack for all W=50 different sizes, while memoization only fills sizes that occur in the recursive call (only 12 in this case). **But**, it is to note that the memorization approach implied 338 recursions. Moreover, in retail applications, the shelf (knapsack) size is at least 10 times larger than the average item size. To translate such a real-life constraint, let us consider the same scenario as above but with a realistic shelf size of 200 cm; like the previous scenario, memorization outperforms dynamic programming in terms of memory saving (up to 70%) but yielded to 1988 recursions, which induces very large overhead.

Hence, the selection will be based on the following fact: memory saving vs. less recursion overhead. Given that real auto-allocation scenarios in the retail industry are very similar to the previous scenario and that memoization is mainly efficient for the unbounded Knapsack problem, dynamic programming will still be the most efficient candidate for the engine.

3.5.7.5 Tailoring dynamic programming

Let us compute the histogram of the different sub-problems occurring in the second scenario of the previous section (real-life scenario: realistic values of retail items and shelves):

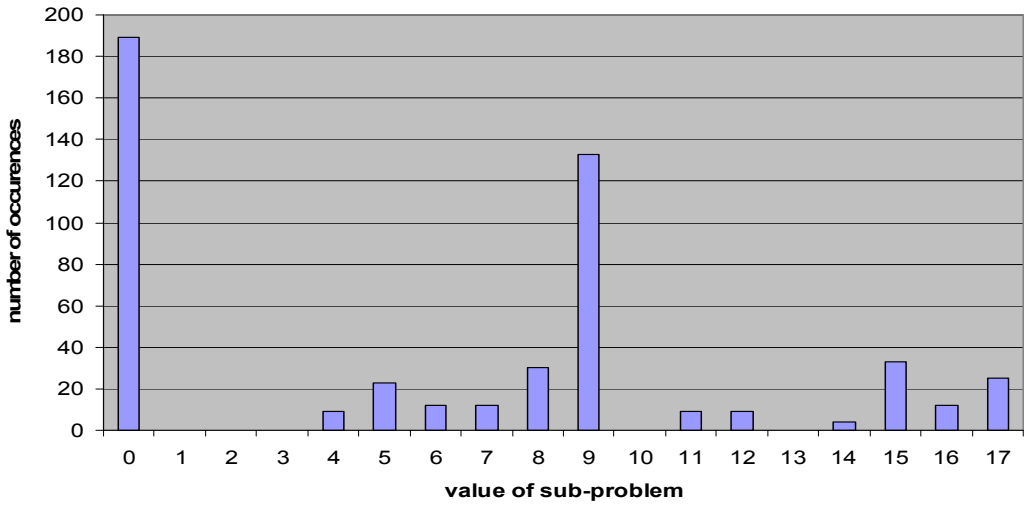


Figure 21 - Histogram of sub-problems - Scenario 2

By analyzing the histogram above, we can clearly observe that some sub-problems are re-computed considerably by the traditional dynamic programming approach, resulting in loss of performance. In figure 6, we can also observe that a set of trailing values in each line in the output is repeated; this is can be explained by the following:

- For each item, dynamic programming is computing the profit for ALL weights ($0 \leq w \leq W$), regardless if the sub-problem will be used or not.

As a result, the first enhancement would be specifying an initial bound for each iteration (each item), which is the weight of that corresponding item. Implementation-wise, the change will affect the inner loop in the pseudo-code as follows:

```
for i = 1 to n
   for w =w_i to W
        if w_i <= w // item i can be part of the solution
                if b_i + B[i-1,w-w_i] > B[i-1,w]
                        B[i,w] = b_i + B[i-1,w- w_i]
                else
                        B[i,w] = B[i-1,w]
        else B[i,w] = B[i-1,w]  // w_i > w
```

3.5.8 Implementation

3.5.8.1 Dynamic Rules

The auto-allocation engine's behavior is directly linked to the defined strategy, hence to the combination of merchandising rules. It is to note that rules with high priority affects the most the outcome of the engine. To ensure both dynamicity and strategy-driven computation, the following methodology is adopted:

- Dividing merchandising rules as follows:
    - Vertical rules: affecting vertical allocation on different shelves. These rules can be affect top-down or bottom-up distribution.
    - Horizontal rules: affecting horizontal allocation / positioning within the same shelf (given that the item is already allocated to a given shelf)
    - Hybrid rules: when an item needs to be allocated to a certain region within the planogram
- Applying vertical rules then horizontal rules: this will initialize the components of the Knapsack problem; a set of items and a wasted space to be filled in an optimal manner.

Assume that we have a set of vertical rules, a set of horizontal rules and a set of items that need to be allocated on 3 shelves S1, S2 and S3.
The items will hold attributes for each of the rules (if we have n different rules then we have n different attributes):
Item 1: r_1:5 r_2:10; r_3:2 ……., r_n: 8

Item 2: r_1:2 r_2:* r_3:7, ……., r_n: 3 (* indicated that no value is defined)

…….

Item j: r_1:7 r_2:* r_3:*, ……., r_n: 11

Each of the rules will have a priority weight defined by the user:

a_1:2 a_2:1; a_3:4 ……., a_n: 10

First of all, we need to decide which items belong to which shelf, hence to apply the vertical rules.

For each vertical rule, we rank the items based on the corresponding rule attribute; if the item has no value, we assign to it the average of all other non-null item attributes. The final rank of each item can be expressed as:

$$\sum_{i=1}^{n} a_i x_i$$

where: n is the total number of vertical rules

a_i is the priority weight of the rule

x_i is the rank of the item for the corresponding rule

After calculating the ranks for all the items, we generate a sorted vector as follows (the sorting order, ascending or descending, depends on the type of the vertical allocation – whether it's top-down or bottom-up):

Figure 22 - Vertical allocation vector

The figure above illustrates how items (black squares) are sorted by calculated rank and distributed proportionally over shelves (colored squares). The proportion depends on the size, as well as the profitability of the shelf; per example, the user can specify that eye-level shelves can hold more items than top or bottom shelves. As a result, each item is assigned to specific shelf, reflecting the specified vertical rules. After filling each shelf iteratively (explained in the coming sections), the same procedure will be applied for horizontal allocation (positioning of the item on the shelf).

3.5.8.2 Mapping to Knapsack Problem

We assume that the space maximization process of the auto-allocation engine can be mapped to a bounded (0-N) Knapsack problem. After allocating items on corresponding shelves with the minimum required number of horizontal facings, we need to optimally fill the remaining space on each shelf. The selection of the "bounded" aspect is due to the business rule stating that 0 to N copies of a given item can be allocated horizontally on the remaining space, where N is the maximum allowed number of horizontal facings. Mathematically the bounded knapsack problem can be formulated as:

$$\text{maximize} \quad \sum_{j=1}^{n} p_j x_j$$

$$\text{subject to} \quad \sum_{j=1}^{n} w_j x_j \leq W, \qquad x_j \in \{0, 1, \ldots, b_j\}$$

where:

- n is the number of distinct items on the shelf
- p_j, w_j and x_j are respectively the profit (calculated by an independent forecasting engine), weight (width in our case) and number of copies of a given item j
- W is the remaining total space on the shelf

3.5.9 Real-life Scenario

Based on the output of the auto-segmentation engine obtained in the previous scenario, we need to allocate the items within the obtained assortment (for the category "pain relievers") on the corresponding fixture (composed of 4 shelves, each of 200 cm width). The minimum horizontal space between 2 items is 1 cm.

Below is the list of items with the following attributes:

Item name	Item image	National / non-national	Item attributes	
Advil 100 mg		National brand	Depth: 12	Width: 18
			Height: 10	Min Stacks: 4
			Min Facing: 2	Max Facing: 5
			LY Sales: 528	Profit: 60

Advil 200 mg		National brand	Depth: 10	Width: 16
			Height: 9	Min Stacks: 4
			Min Facing: 3	Max Facing: 5
			LY Sales: 831	Profit: 78
Advil PM 500 mg		National brand	Depth: 14	Width: 17
			Height: 9	Min Stacks: 4
			Min Facing: 2	Max Facing: 4
			LY Sales: 497	Profit: 55
Advil Liqui-gels 500 mg		National brand	Depth: 13	Width: 17
			Height: 11	Min Stacks: 3
			Min Facing: 3	Max Facing: 6
			LY Sales: 711	Profit: 82
Herbal Nights		Non-national brand	Depth: 8	Width: 20
			Height: 15	Min Stacks: 3
			Min Facing: 1	Max Facing: 3
			LY Sales: 243	Profit: 38
Lanes Quiet Life		Non-national brand	Depth: 11	Width: 12
			Height: 18	Min Stacks: 2
			Min Facing: 2	Max Facing: 4
			LY Sales: 352	Profit: 47
Natrol Melatonin		National brand	Depth: 8	Width: 11
			Height: 20	Min Stacks: 1
			Min Facing: 2	Max Facing: 4
			LY Sales: 415	Profit: 44
Vivarin Blu-emu		Non-national brand	Depth: 9	Width: 13
			Height: 16	Min Stacks: 3
			Min Facing: 3	Max Facing: 5
			LY Sales: 702	Profit: 68
Snore relief		Non-national brand	Depth: 8	Width: 8
			Height: 16	Min Stacks: 1
			Min Facing: 2	Max Facing: 4
			LY Sales: 323	Profit: 49
Xiboprofen Activon		National brand	Depth: 11	Width: 11
			Height: 18	Min Stacks: 2
			Min Facing: 3	Max Facing: 5

			LY Sales: 690	Profit: 61
Xiboprofen Activon Forte		National brand	Depth: 11	Width: 11
			Height: 18	Min Stacks: 2
			Min Facing: 2	Max Facing: 5
			LY Sales: 522	Profit: 52
Cengent Bayer		Non-national brand	Depth: 9	Width: 19
			Height: 10	Min Stacks: 4
			Min Facing: 2	Max Facing: 4
			LY Sales: 299	Profit: 45
Cengent Bayer Children		Non-national brand	Depth: 9	Width: 18
			Height: 11	Min Stacks: 4
			Min Facing: 2	Max Facing: 5
			LY Sales: 362	Profit: 52
Aleve		National brand	Depth: 11	Width: 19
			Height: 12	Min Stacks: 3
			Min Facing: 3	Max Facing: 6
			LY Sales: 902	Profit: 83
Rapid Sleep PM		National brand	Depth: 12	Width: 13
			Height: 20	Min Stacks: 2
			Min Facing: 3	Max Facing: 6
			LY Sales: 664	Profit: 59
Tylenol Extra Package		National brand	Depth: 10	Width: 20
			Height: 20	Min Stacks: 2
			Min Facing: 2	Max Facing: 7
			LY Sales: 1025	Profit: 90

Table 8 - Scenario: Items with attributes

The following rules are to be applied to the scenario:

- Items with high profitability are to be placed on upper shelves (vertical rule with priority=1). The profitability is computed by an external forecast engine which takes into consideration factors such as the item's space elasticity, item replenishment cost and demand function.

- Items with high profitability are to be placed on upper shelves (vertical rule with priority=2). Profitability is also computed by the forecast engine and is dependent on the specified strategy (i.e. maximizing space utilization, maximizing profit …)
- National-brand items must be on the left of non-national brand items (horizontal rule with priority=1).

The first phase is to apply the vertical rules which will assign the items to their corresponding shelves.

For each rule, we rank the items and multiply their position by the rule's priority coefficient. We obtain the following table (sorted in descending order):

Item	$\sum_{i=1}^{n} a_i x_i$	Allocated Shelf	Min. Facings
2274727	45	1	2
2274726	42	1	3
2274714	38	1	3
2274716	37	2	3
2274720	33	2	3
2274722	30	2	3
2274713	27	3	2
2274715	22	3	2
2274723	22	3	2
2274725	17	4	3
2274719	14	4	2
2274718	12	4	2
2274721	11	4	2
2274724	7	4	2
2274717	3	4	1

Table 9 - Scenario: Vertical allocation ranking

It is to note that shelves are counted from top to down (i.e. shelf 1 is the top-most shelf) and priorities are counted in ascending order (priority=1 is the highest priority).

We can deduce from the table that top items have a larger number of minimum horizontal facings than bottom items; this can be explained by the fact that the forecast engine,

which computes this value, assign more minimum facings to profitable items (the vertical rules specifies that best-selling items and items with high profitability are to be allocated from top to down). By proportionally assigning items to shelves (in a random order) and adding the minimum required number of horizontal facings, we obtain the following view of the resulting planogram:

Figure 23 - Scenario: Planogram after vertical allocation

The figure above shows the distribution of the items over the shelves, along with the minimum number of horizontal facings. The next step is to optimally fill the remaining

space (red arrow). To solve this problem, we map it to a Bounded Knapsack Problem (BKP) where the set of items is nothing but the items already allocated on the shelf (having a width w and a profit P), the knapsack of size W is the remaining space on the shelf (red arrow in the figure) and N copies of each item can be added to the Knapsack, where N = maximum number of horizontal facings – allocated number of horizontal facings for each of the items.

Figure 24 - Scenario: Planogram after space maximization (BKP)

The last step in the auto-allocation process is to apply the horizontal rules which will re-arrange the items on their relative shelf and to add the stacks (which must be less then the shelf's height). By applying the same ranking methodology adopted for the vertical rules, we obtain the following final planogram:

Figure 25 - Scenario: Final planogram

3.5.10 Wasted space

In the real-life scenario above, we used the auto-allocation engine to fill a fixture consisting of 4 shelves, each of 200 cm width. Below is a table showing the percentage of unallocated space in each of the shelves:

Shelf number	Total Width (in cm)	Unallocated space (in cm)	Percentage of unallocated space
1	200	5	2.5%
2	200	0	0%
3	200	3	1.5%
4	200	5	2.5%

Table 10 -Scenario: Wasted space

Hence, the average percentage of unallocated space in the scenario is 6.5/4 = 1.625%, which is acceptable. Such wasted space can be filled by dividing it by the number of facings on the shelves and adding the resulting value to the minimum horizontal spacing between items. By applying this procedure, we obtain the following:

Shelf number	Total number of facings	Unallocated space (in cm)	Total horizontal space between items (in cm)
1	10	5	1.5
2	14	0	0
3	13	3	1.23
4	12	5	1.42

Table 11 - Scenario: Adjustment of horizontal spacing

4 Architecture

4.1 Global environment

The figure below illustrates the global environment of the Cross-platform Automated Space Planning Engine for Retailers where the different external elements are shown as well as their interaction with the three modules which compose CASPER (AAE, ASE and ASPGE):

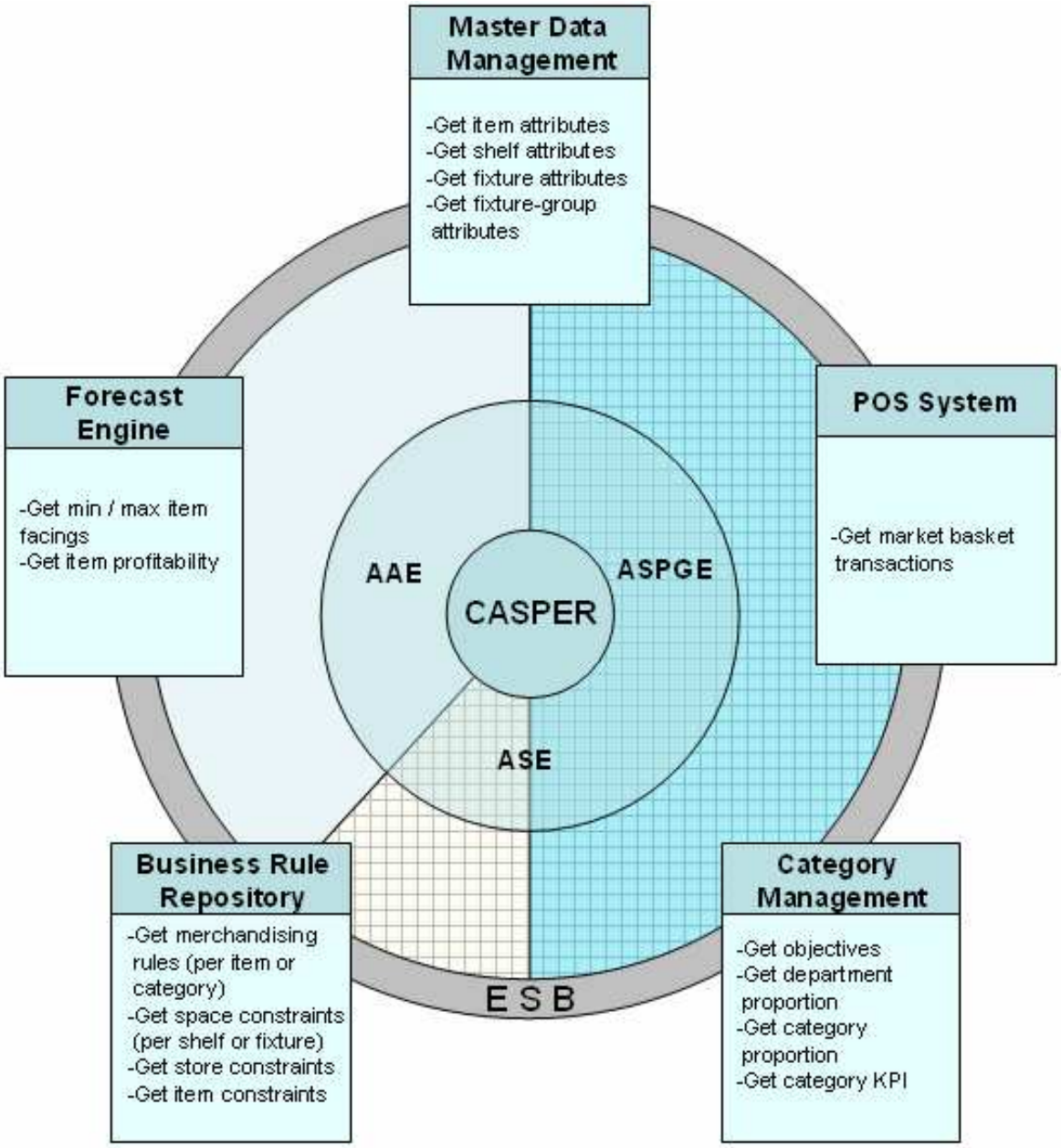


Figure 26 - CASPER: global environment

The Automatic Store Plan Generation Engine (ASPGE), the initial module of CASPER, communicates with the Master Data Management application via corresponding web services to gather information related to fixture-groups, fixtures and shelves within the store. Web services, used for implementing the Service-Oriented Architecture (SOA) aspect of the our engine, allow interoperability between heterogeneous systems via cross-platform messaging (i.e. XML). The information gathered by the web services is used by ASPGE to initialize the structures needed for automating the process. Moreover, ASPGE communicates with the POS system via another web service to read market basket transactions which will be used to compute the co-occurrence matrix by item department and item category. Finally, ASPGE calls web services exposed on the Category Management application in order to get the proportions assigned by the category manager for the item departments and categories. On the other hand, the Automatic Segmentation Engine (ASE) interacts with the Category Management application to read the objectives per category needed for the assortment process, with the Business Rule Repository to get item constraints and rules (i.e. indispensable items, complementary items…), with the POS system to gather item activity information (i.e. sales, net profit margin) and with the Master Data Management application to get the general item attributes as well as the item/category relationships. Finally, the Auto-Allocation Engine (AAE) initiates by using the results of the ASE, collecting corresponding item and shelf attributes from the Master Data Management application, reading computed item profitability from the Forecast Engine and allocating items on shelves based on the merchandising rules extracted from the Business Rule Repository. Moreover, communication between the external modules is ensured via the corresponding web services, mainly between the forecast engine and POS system to read transactional data and predict the different indicators (i.e. profitability, space elasticity, forecasted sales volume, and forecasted net profit margin).

It is to note that the orchestration between the different services is managed via an event-driven and standards-based messaging-engine, the Enterprise Service Bus (ESB) shown in the image below. ESB allows standardizing the service-oriented communication between the modules by eliminating the coupling between the invoked service and the transport medium.

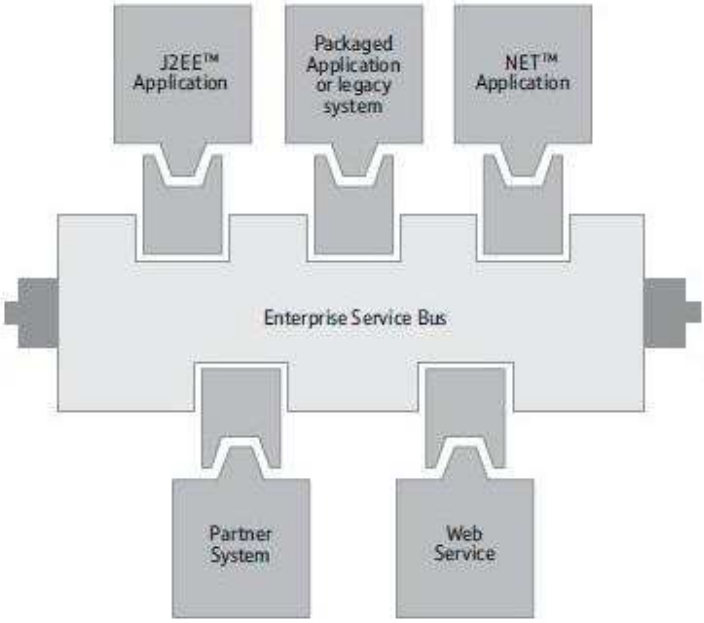


Figure 27 - Enterprise Service Bus

4.2 System context

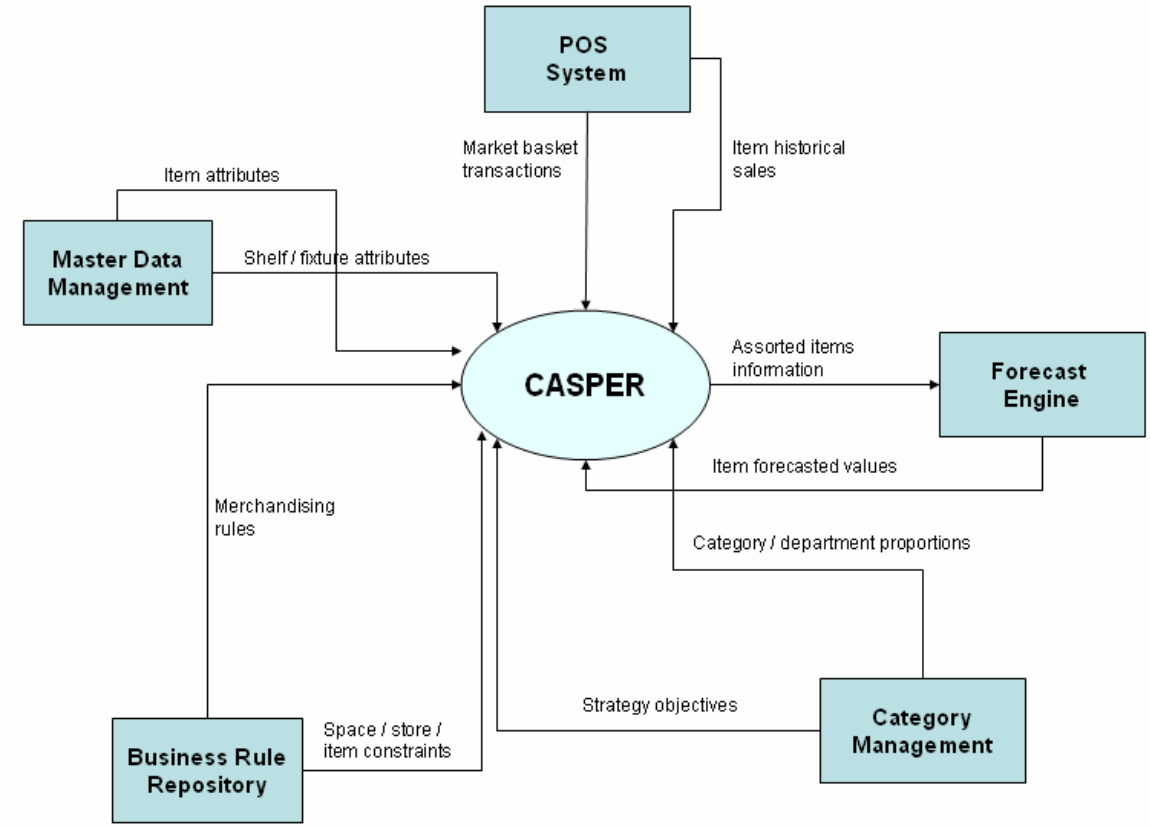


Figure 28 - CASPER: System context

4.3 CASPER Block Diagram

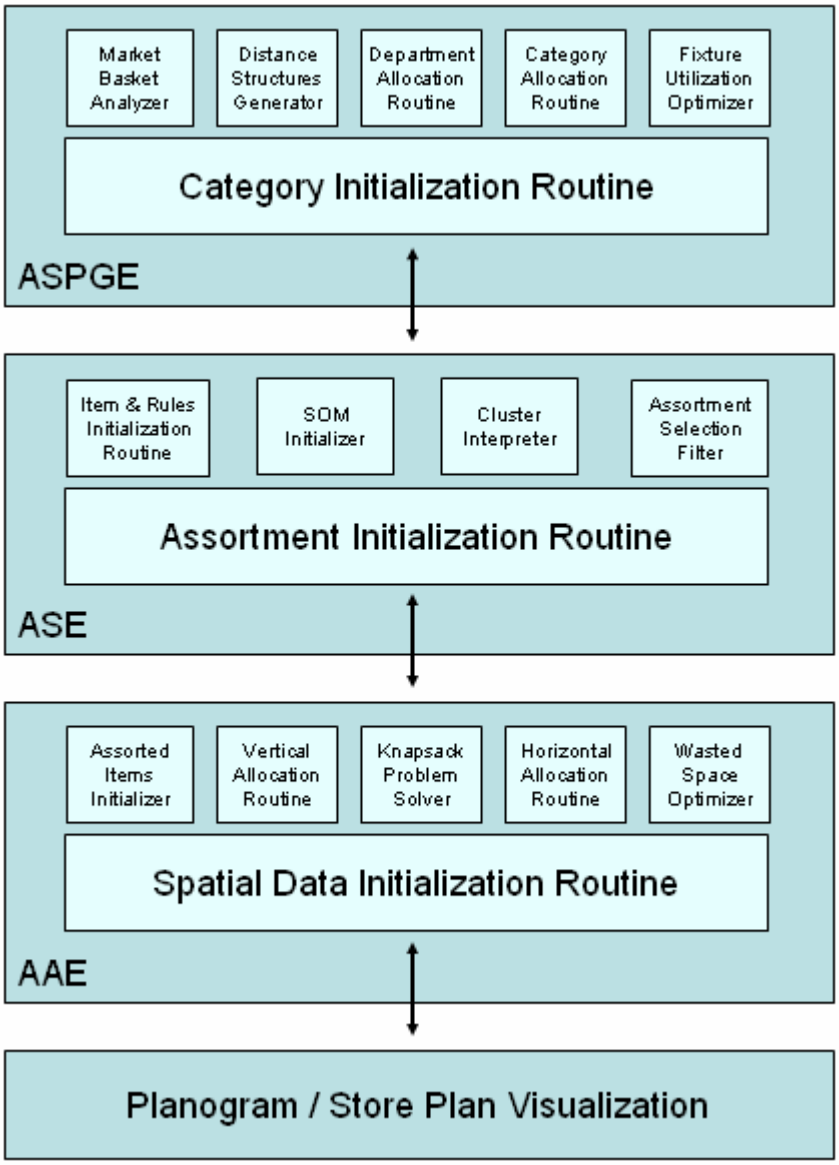


Figure 29 - CASPER: Block diagram

The figure above illustrates the internal architecture of CASPER using a block diagram representation. The different sub-modules of each engine are shown as well as their respective interaction routines: the "Category Initialization Routine" which passes the output of ASPGE to ASE, the "Assortment Initialization Routine" which passes the output of ASE to AAE and the "Spatial Data Initialization Routine" which passes the

output of AAE to the corresponding spatial visualization tool. It is to note that this tool is not part of the internal architecture of CASPER.

4.4 Technical considerations

The following technical aspects need to be satisfied by CASPER to ensure the cross-platform nature of the proposed engine:

- Service-Oriented Architecture (SOA): by implementing a "web service" layer to ensure interoperability with other heterogeneous systems.
- Cross-platform programming language: satisfied by using Java programming language to implement the engine.
- XML Data Source: Given that the input data is gathered from multiple external systems (i.e. Master Data Management, POS system, Business Rule Repository …) and output data is passed to other visualization systems, CASPER needs to store both its input and output in XML files, instead of using proprietary Database Management Systems.

5 Conclusions

5.1 Main results

The main results of this research are highlighted below:

- Finding an automated process covering the complete retail space planning process starting from store plan management (using ASPGE), passing by product assortment (using ASE) and finishing by solving the product-to-shelf allocation problem (using AAE).
- Providing a custom algorithm to generate the store plan based on results of the market basket analysis and the KPI provided by the category manager.
- Adopting a special type of Artificial Neural Networks, the Self-Organizing Maps (SOM), to cluster the items of a given category based on multiple dimensions and using the results to select the optimal product assortment.
- Providing a custom algorithm to allocate products on shelves by mapping the process to a Knapsack Problem, taking into consideration a dynamic set of merchandising rules (vertical and horizontal) as well as space constraints.
- Suggesting an architecture for the proposed solution, knowing that the engine needs to interoperate with a multitude of external modules and applications.

5.2 Main contributions

Our proposed solution helps retailers to take advantage of the following capabilities:

- Managing the complete space planning process using a unified solution, which maintains consistency and integrity throughout the whole process.
- Saving resources (time and personnel) by automating this complex task especially when the retailer owns a large chain of stores.
- Benefiting from the dynamicity of the engine which allows retailers to continuously manage and optimize space planning by simulating multiple scenarios which include different sets of rules and objectives.

- Locally controlling the whole process without needing to "wire" results between different modules (i.e. using a module to perform product assortment and manually inputting the results into another to generate the corresponding planogram).

5.3 Performance results

5.3.1 Introduction

Multiple tests were applied on the different modules of CASPER using data from CVS Caremark. The actual names were changed for privacy concerns. Below is the hardware architecture of the testing environment:

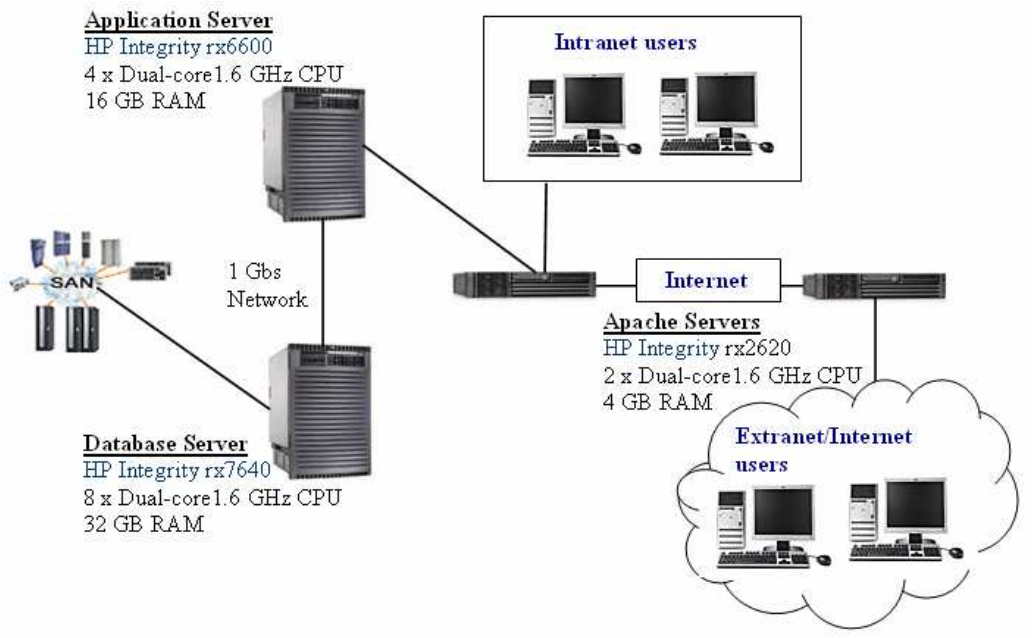


Figure 30 - Performance Tests: Hardware environment

HP Integrity rx6600 Server

Microprocessor: 4x Dual-core1.6 GHz Intel Itanium 2 processors with 18 MB L3 cache

Memory: 16 GB

Internal storage devices: Hot-plug Serial Attached SCSI HDD drive, 146GB with RAID

Operating system: RedHat Enterprise Linux 1 AS4 or HP-UX 11i v3

HP Integrity rx7640 Server

Microprocessor: 8x Dual-core1.6 GHz Intel Itanium 2 processors with 18 MB L3 cache

Memory: 32 GB RAM

Internal storage devices: Hot-plug Ultra320 SCSI HDD drive, 300 GB with RAID

Operating system: RedHat Enterprise Linux 1 AS4 or HP-UX 11i v3

HP Integrity rx2620

Microprocessor: 2x Dual-core1.6 GHz Intel Itanium 2 processors with 18 MB L3 cache

Memory: 4 GB RAM

Internal storage devices: Any Internal storage capacity

Operating system: RedHat Enterprise Linux 1 AS4 or HP-UX 11i

As for the software implementation, the following platforms were used:

- Application server: Apache Tomcat 6.0
- Database server: Oracle 10g R2
- Frontal server: Apache Frontal Server 2.2.4

5.3.2 AAE

The following scenario is applied to measure the performance of the auto-allocation engine: 8 cases with different number of items, different number of shelves and 8 merchandising rules. The performance results are shown in the figure below:

Case id	Number of items	Total number of rules	Number of shelves	Total processing time (ms)
1	100	8	4	392
2	200	8	4	844
3	300	8	4	1408
4	400	8	8	2808
5	500	8	8	3420
6	600	8	8	4012
7	700	8	12	7416
8	800	8	12	9248
9	900	8	12	12100

Table 12 - AAE Scenario

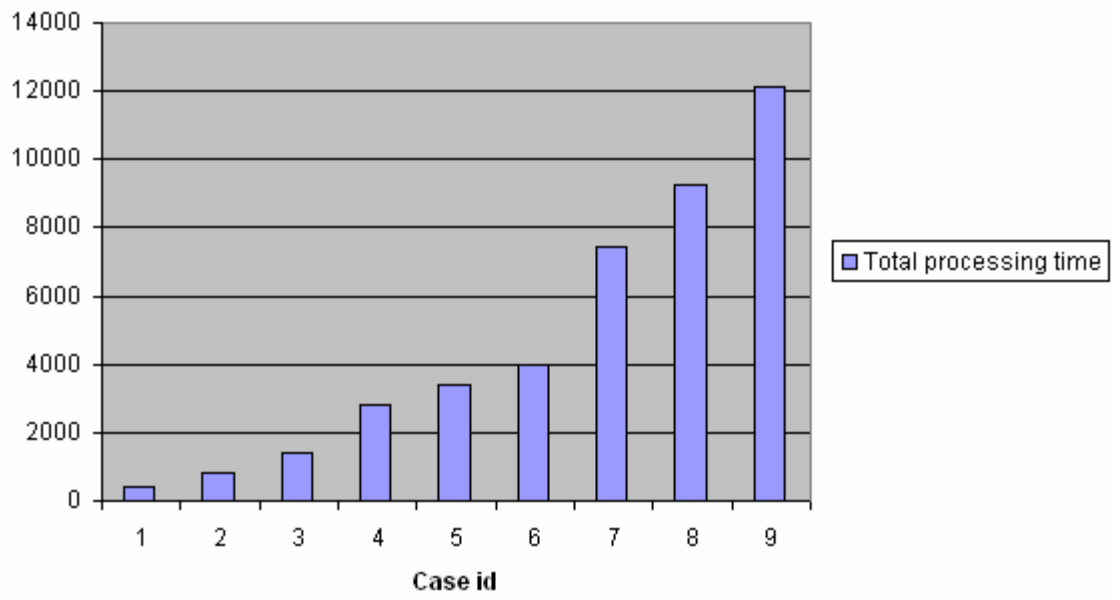


Figure 31 - AAE Scenario: Total processing time

The largest scenario consisting of 900 items that need to be allocated on 12 shelves in respect to 8 merchandising rules is taking around 12 seconds which is acceptable, given the size of the underlying Bounded Knapsack Problem.

5.3.3 ASE

5.3.3.1 Scenario 1

The first scenario of the auto-segmentation engine consisted of inputting 6 different categories in increasing order of size (number of items) as well as 4 variables (selling price, brand, supplier, size). It is to note that category with ID=6 is the largest category for the studied retailer, consisting of 1241 items. The total processing time is shown in figure below:

Category ID	Category size (nb of items)	Number of variables
1	35	4
2	104	4
3	231	4
4	587	4
5	890	4
6	1241	4

Table 13 - ASE Scenario 1: Different category size

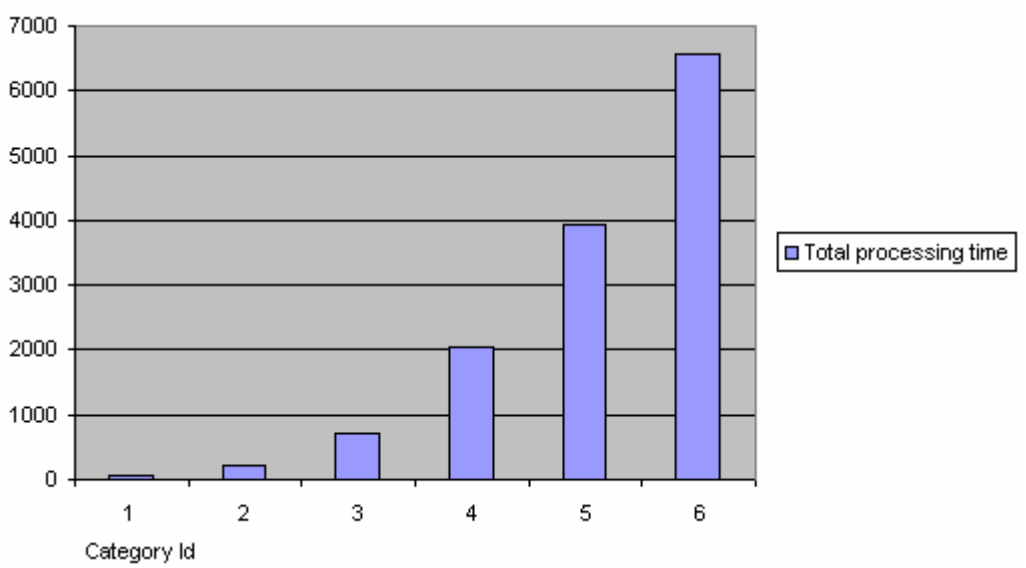


Figure 32 - ASE Scenario 1: Total processing time

The chart above shows that the auto-segmentation engine processed the largest category in less than 7 seconds which is a considerable improvement compared to the days spent in performing the same process manually. As for the quality of the obtained clusters (i.e. segments), no overlapping is noticed. It is to note that, for most retailers, the average number of items per category is around 300 items, which can be processed by the auto-segmentation engine in less than a second.

Category ID	Category size (nb of items)	Number of variables
6	1241	2
6	1241	3
6	1241	4
6	1241	5
6	1241	6
6	1241	7
6	1241	8
6	1241	9
6	1241	10

Table 14 - ASE Scenario 2: Different number of variables

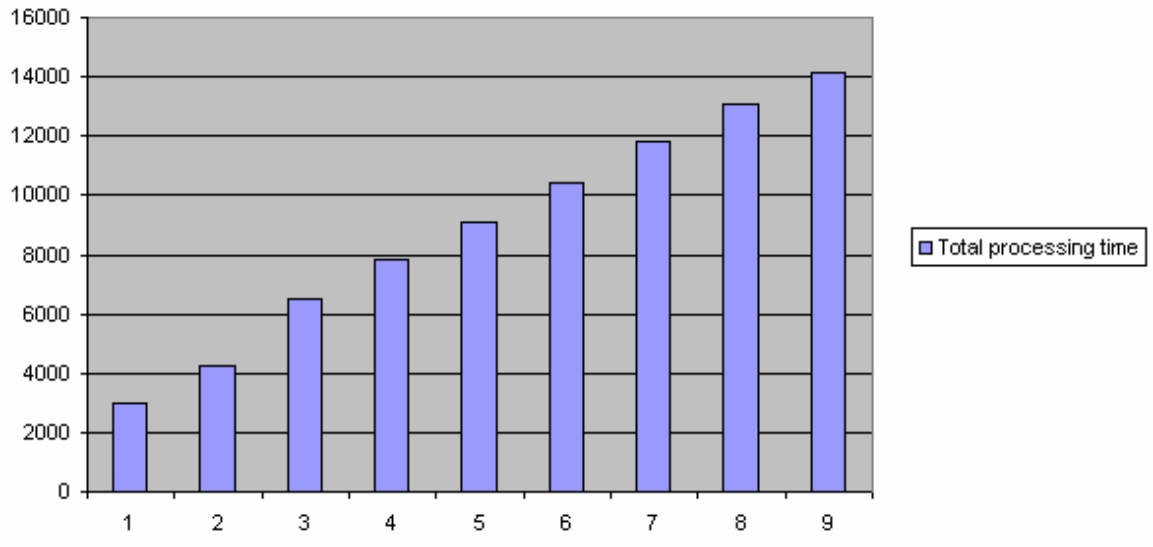


Figure 33- ASE Scenario 2: Total processing time

Based on the results illustrated in the chart above, we can deduce that the performance of the auto-segmentation engine is linearly affected by the number of variables and hence doesn't yield to considerable performance bottlenecks.

5.3.3.2 Scenario 2

The second scenario of the auto-segmentation engine consisted of inputting the largest category (1241) and different number of variables (starting progressively from 2 to 10). The total processing time is shown in figure below:

5.4 Factor table

Factor	Quality scenario	Future work	Impact on stakeholders
Automation	- Eliminating all aspects of user intervention except the definition of the master data and strategy objectives.	- Automating the store expansion process by implementing a data mining algorithm which interacts with real-time Web services related to geographic / demographic data.	- Saving time and resources.
Completeness	- Automating the whole space planning process by generating the store plan	- Handling additional (i.e. special) aspects of	- Enforcing consistency of rules and strategy

	(allocating categories to fixtures), selecting the optimal items from each category and allocating the resulting items on the corresponding shelves.	the space planning process (ex: expansion management – consumer clustering, …)	objectives throughout the whole process.
Dynamicity	- Generating a store plan based on the results of the market basket analysis (which varies from period to period). - Performing the assortment process based on a dynamic number of variables. - Allocating assorted items on the corresponding shelves with respect to a set of dynamic merchandising rules.	- Extending the dynamicity of ASPGE to generate the store plan assuming that the store is empty (no fixed assets)	- Having the ability to re-manage the space planning process under varying factors and circumstances (i.e. seasonality, economic / demographic factors, etc…)
Accuracy / Performance	- Checking the accuracy of the clusters obtained by ASE - Measuring the wasted space on the shelves after allocating items using AAE. - Measuring the processing time of the different automated procedures.	- Tuning the performance of the different modules and applying additional approaches and methodologies.	- Obtaining optimal (or near-optimal) results while saving time and labor.

5.5 Future work

The functionalities of CASPER will be extended in future work to provide the following functionalities:

- Tuning the performance of CASPER and optimizing the resource-consuming algorithms and routines. Given the large-sized retailers targeted by CASPER, load

tests need to applied to the engine to identify possible bottlenecks and performance issues.

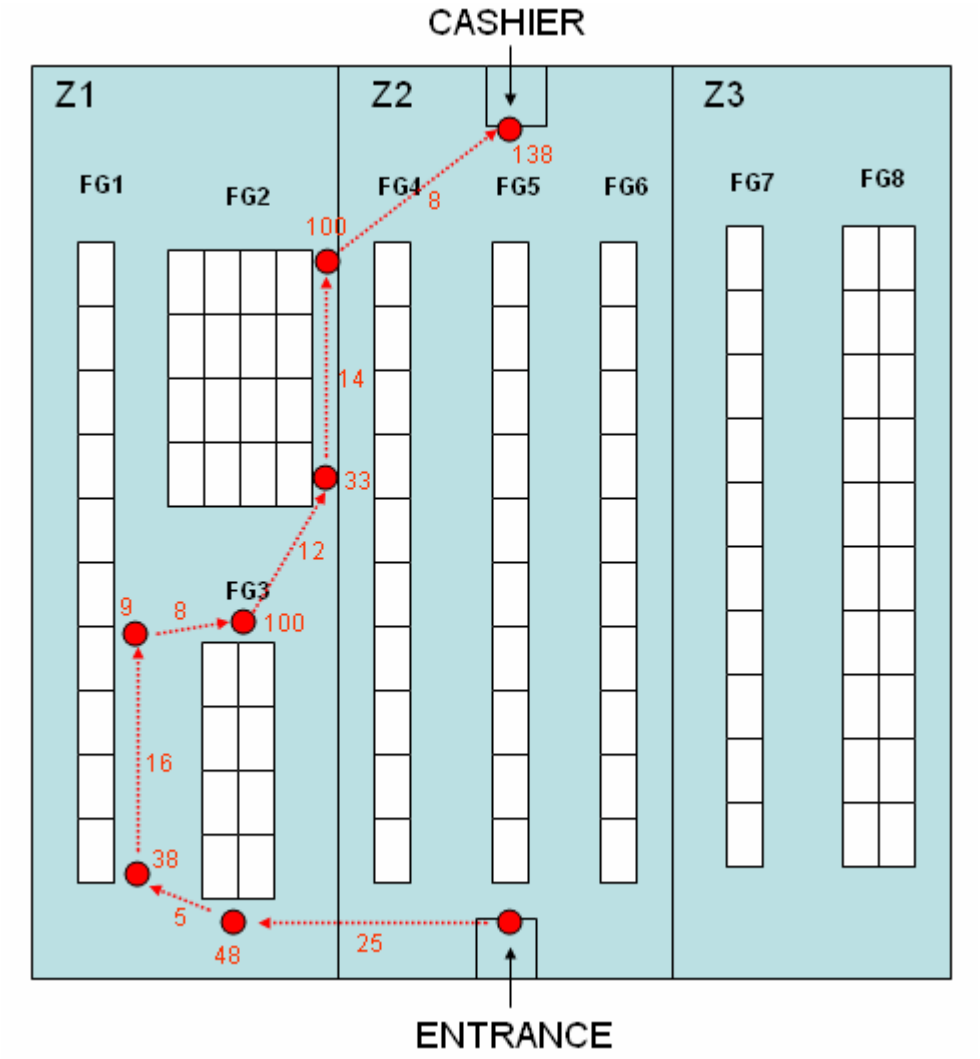


Figure 34 - Future work: consumer trajectory example

- Upgrading the Automatic Store Plan Generation Engine (ASPGE) to process the "consumer behavior" within the store by analyzing the underlying geographical trajectory within the store. Such data can be gathered by implementing RF-ID technology on the shopping carts, hence tracking the movement of the different consumers. Per example (figure above), it will be possible to know how consumers (red spots) move from zone to zone, from fixture to fixture and how much time did they spend at each. This information can extend the scope of ASPGE to optimize

spatial distribution in the store, which affects the consumer satisfaction and enhances the overall profitability.

- Implementing a new module entitled the "Automatic Expansion Advisor" (AEA), which helps retailers to optimally manage their expansions. In other words, AEA analyzes spatial and demographic information of different locations and automatically suggests optimal emplacements for building new stores, based on pre-defined strategies and objectives. Moreover, AEA generates from scratch the most suitable architecture of the newly opened store to ensure enhanced spatial utilization, increased profitability and maximum consumer satisfaction (i.e. shape and size of store, distribution of fixtures, functional store structure…).

6 References:

[1] Moncer Hariga, Abdulrahman Al-Ahmari, and Abdel-Rahman Mohamed, “A Joint Optimization Model for Inventory Replenishment Product Assortment, Shelf Space and Display Area Allocation Decisions,” European Journal of Operational Research, Vol. 181, No. 1, pp.239-251, 2007

[2] Marcel Corstjens, Peter Doyle, "A model for optimizing retail space allocations", Management Science, Vol. 27 No.7, pp.822-33, 1981

[3] Mu-Chen Chen , Chia-Ping Lin, “A data mining approach to product assortment and shelf space allocation”, Expert Systems with Applications: An International Journal, v.32 n.4, pp.976-986, May, 2007

[4] Dario Landa-Silva, Fathima Marikar, Khoi Le, “Heuristic Approach for Automated Shelf Space Allocation”, Proceedings of the 24th ACM Symposium on Applied Computing (SAC 2009), Volume 2, ACM Press, pp. 922-928, 2009

[5] Andrew Lim , Brian Rodrigues , Xingwen Zhang, “Metaheuristics with Local Search Techniques for Retail Shelf-Space Optimization”, Management Science, v.50 n.1, pp.117-131, January 2004

[6] Anna I Esparcia-Alcázar, Lidia Lluch-Revert, Jose Miguel Albarracín-Guillem, Marta Palmer-Gato, and Ken Sharman, “Towards an evolutionary tool for the allocation of supermarket shelf space”. In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), volume 2, pp.1653–1660, Seattle , USA, 2006

[7] Marcel Corstjens, Peter Doyle, "A dynamic model for strategically allocating retail space", Journal of the Operational Research Society, Vol. 34 No.10, pp.943-51, 1983

[8] Alain Bultez, Philippe Naert, "SHARP: Shelf Allocation for Retailers' Profit", Marketing Science, Vol. 7 No.3, pp.211-31, 1988

[9] Fredc Zufryden, "A dynamic programming approach for product selection and supermarket shelf-space allocation", Journal of the Operational Research Society, Vol. 37 No.4, pp.413-22, 1986

[10] Teuvo Kohonen, "Self-Organizing Maps", Springer-Verlag, Leipzig, Germany, 1997

[11] David Pisinger, "Algorithms for Knapsack Problems", Ph.D. thesis, DIKU, University of Copenhagen, 1995

[12] David Pisinger, "A Java Library of Graph Algorithms & Optimization", Chapman & Hall, 2006

[13] Madan G. Singh, Roderick Cook, Marcel Corstjens, " A Hybrid Knowledge-Based System for Allocating Retail Space and for Other Allocation Problems", Interfaces, Vol. 18, No. 5, pp. 13-22, 1988

[14] Sujatha, P.K. Kannan, A. Ragunath, S. Bargavi, K.S. Githanjali, S., "A Behavior Based Approach to Host-Level Intrusion Detection Using Self-Organizing Maps", International Conference on Emerging Trends in Engineering and Technology, pp. 1267-1271, July 2008

[15] Sang-Chul Lee, Yung Ho Suh, Jae Kyeong Kim, Kyoung Jun Lee, "A cross-national market segmentation of online game industry using SOM", Expert Systems Applications 27(4), pp. 559-570, 2004

[16] Koffi Yao, Max Mignotte, Christophe Collet, Pascal Galerne, Gilles Burel, "Unsupervised Segmentation Using a Self-Organizing Map and a Noise Model Estimation in Sonar Imagery", Pattern Recognition 33(9), pp.1575-1584, 2000

[17] M. C. Su, T. K. Liu, and H. C. Chang, "Improving the self-organizing feature map algorithm using an efficient initialization scheme," Tamkang Journal of Science and Engineering, vol. 5, no. 1, pp. 35-48, March 2002

[18] Milind Dawande, Jayant Kalagnanam, Pinar Keskinocak, Sibel Salman, R. Ravi, "Approximation Algorithms for the Multiple Knapsack Problem with Assignment Restriction", J. Comb. Optim. 4(2), pp.171-186, 2000

[19] Ruibin Bai and Graham Kendall, "An Investigation of Automated Planograms Using a Simulated Annealing Based Hyper-heuristics", in Ibaraki, T., Nonobe, K., and Yagiura, M. (Eds.) Metaheuristics: Progress as Real Problem Solvers - (Operations Research/Computer Science Interfaces, Vol. 32), Springer: Berlin, Heidelberg, New York, pp. 87-108, 2005

Appendix A:

Item #	Item Code	Sales	Net Profit Margin	Is Indispensable	Efficiency
1	2274673	528	72	0	0.82
2	2274674	413	35	0	0.42
3	2274675	505	40	0	0.31
4	2274676	621	34	0	0.37
5	2274677	98	54	0	0.3
6	2274678	158	30	0	0.4
7	2274679	387	50	0	0.13
8	2274680	352	15	0	0.56
9	2274681	401	9	0	0.22
10	2274682	831	64	0	0.78
11	2274683	258	60	0	0.73
12	2274684	364	40	0	0.78
13	2274685	88	49	0	0.43
14	2274686	12	68	0	0.44
15	2274687	532	21	0	0.44
16	2274688	301	28	0	0.53
17	2274689	189	17	0	0.27
18	2274690	421	38	0	0.46
19	2274691	497	70	1	0.52
20	2274692	502	25	0	0.45
21	2274693	234	18	0	0.82
22	2274694	567	12	0	0.75
23	2274695	531	8	0	0.34
24	2274696	314	27	0	0.78
25	2274697	419	33	0	0.04
26	2274698	285	21	0	0.04
27	2274699	711	84	0	0.43
28	2274700	102	41	0	0.99
29	2274701	94	30	0	0.23
30	2274702	260	28	0	0.42
31	2274703	397	6	0	0.88
32	2274704	112	53	0	0.91
33	2274705	243	91	0	0.72
34	2274706	329	14	0	0.96
35	2274707	148	12	0	0.89
36	2274708	259	31	0	0.53
37	2274709	352	84	0	0.55
38	2274710	415	81	1	0.08
39	2274711	198	82	0	0.23
40	2274712	89	54	0	0.38
41	2274713	76	42	0	0.76
42	2274714	702	62	0	0.59
43	2274715	290	47	0	0

44	2274716	139	26	0	0.7
45	2274717	428	30	0	0.79
46	2274718	411	55	0	0.18
47	2274719	323	93	0	0.07
48	2274720	184	19	0	0.5
49	2274721	254	45	0	0.02
50	2274722	690	61	0	0.62
51	2274723	522	65	0	0.1
52	2274724	69	32	0	0.87
53	2274725	102	41	0	0.85
54	2274726	680	5	0	0.61
55	2274727	74	72	0	0.08
56	2274728	299	79	1	0.68
57	2274729	309	41	0	0.42
58	2274730	64	59	0	0.81
59	2274731	70	12	0	0.13
60	2274732	362	75	1	0.29
61	2274733	41	63	0	0.93
62	2274734	12	71	0	0.77
63	2274735	902	59	0	0.81
64	2274736	185	41	0	0.62
65	2274737	369	22	0	0.92
66	2274738	157	29	0	0.36
67	2274739	165	34	0	0.26
68	2274740	1025	62	0	0.78