CHALMERS

Big data algorithm optimization

Case study of a sales simulation system

KASPER KARLSSON
TOBIAS LANS

Chalmers University of Technology
Department of Computer Science & Engineering
Computer Science - Algorithms, Languages & Logic
Master of Science Thesis
Göteborg, Sweden 2013

BIG DATA ALGORITHM OPTIMIZATION
CASE STUDY OF A SALES SIMULATION SYSTEM

KASPER KARLSSON
TOBIAS LANS

© KASPER KARLSSON & TOBIAS LANS, July 2013.

Examiner: PETER DAMASCHKE

Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering Göteborg, Sweden July 2013

Abstract

When sales representatives and customers negotiate, it must be confirmed that the final deals will render a high enough profit for the selling company. Large companies have different methods of doing this, one of which is to run sales simulations. Such simulation systems often need to perform complex calculations over large amounts of data, which in turn requires efficient models and algorithms.

This project intends to evaluate whether it is possible to optimize and extend an existing sales system called PCT, which is currently suffering from unacceptably high running times in its simulation process. This is done through analysis of the current implementation, followed by optimization of its models and development of efficient algorithms. The performance of these optimized and extended models are compared to the existing one in order to evaluate their improvement.

The conclusion of this project is that the simulation process in PCT can indeed be optimized and extended. The optimized models serve as a proof of concept, which shows that results identical to the original system's can be calculated within $< 1\%$ of the original running time for the largest customers.

Acknowledgements

We would like to start by thanking Pierre Sandboge for his help with getting the systems up and running and his never-ending patience for our questions.

We would also like to thank Peter Damaschke for acting as our supervisor and making this project possible.

A final thanks goes out to Hoi-Man Lui, Peter Borg and all of the other great people at the Omegapoint office in Gothenburg.

Contents

List of Figures

List of Tables

List of Algorithms

1

Introduction

1.1 Motivation

Omegapoint is an IT consulting company focusing on business development, high quality system development and information security.

One of Omegapoint's clients wish to improve and extend their current order system. This system is called PCT, which stands for Price revenue management Client Tool. PCT is used by sales representatives to find suitable discount rates for different items when negotiating with customers. As a central part of this process, a sales representative will run simulations over different discount rates in order to evaluate their expected profit.

When the system performs a simulation, it starts by making an estimation of the customer's future order quantities based on their order history. This estimation serves as a basis when calculating the expected marginal profit which the given discount rates will yield. However, running such simulations takes too long time in PCT and as such an optimization of the simulation process is needed.

1.2 Big data

Big data is a slightly abstract phrase which describes the relation between data size and data processing speed in a system. A comprehensible definition of the concept is "data whose size forces us to look beyond the tried-and-true methods that are prevalent at that time." [1]. This means that a scenario where innovative optimization of both models and algorithms is required to handle large amounts of data might well be classified as a big data problem.

In PCT, the big data challenge arises from the huge amounts of data needed in order to run simulations for large customers. In some cases more than fifty thousand historical order rows may have to be handled, with multiple possible conditions and discount rates applied to every single one of them. While the data set itself is not extremely large by today's standards, the complex operations and calculations which have to be performed on each one of them adds new dimensions to the simulation procedure. Discounts are for example inherited through a large tree structure containing tens of thousands of nodes and the results must be presented to the user within a reasonable amount of time.

The reasonable time limit has been defined as ten seconds for the simulation procedure in PCT. This value is based on research [2, 3] showing that a system user who has to wait even further for results of complex calculations will lose focus - something which could prove devastating during a negotiation with a customer.

An ideal simulation procedure would always return the results within just a few seconds, since this would mean that simulations could take place during normal conversation without requiring any waiting at all.

1.3 Goals and limitations

The first goal of this project is to optimize the existing discount simulation algorithm in order to reduce its running time. The discount simulation's purpose is to apply given discounts to articles and article categories, in order to evaluate whether they will generate an acceptable profit for the selected customer.

The second goal is to create a model with associated algorithms for a scaling extension of the system's simulation functionality. The purpose of this extension is to make it possible to apply different discount rates depending on the volume of individual orders. This will encourage customers to place a few large orders every year instead of several small ones, thus decreasing shipping and warehouse charges for the company without reducing the sales volumes.

In order to achieve these goals, this report focuses on two possible areas of improvement - optimizations of the models and algorithms themselves and improvements of the underlying SQL database. Other possible improvements such as hardware upgrades on machines running the algorithm, implementations of the algorithm in programming languages other than Java or other database solutions than SQL are not considered.

1.4 Thesis outline

The rest of this report is divided into four chapters - Simulation, Method, Results and Discussion.

The Simulation chapter begins with a detailed description of how discount rate simulations work and the problems which the current implementation has introduced. The second part contains a specification of the scaling simulation functionality and an explanation of the technical difficulties which are introduced by this extension.

The Method chapter describes the models and algorithms which have been developed in this project. It also contains a theoretical analysis of these and comparisons between the current implementation in PCT and our solution.

In the Results chapter, the performance of PCT as well as of our solutions for both the optimized customer discount model and the scaling extension are presented. This is split up into a set of test cases, with motivations of their relevance for actual usage scenarios.

The report is wrapped up with a Discussion chapter. This is where the results are discussed and conclusions and ideas for future work are presented.

2

Simulation

When a sales representative negotiates with a customer, one can think of it as a sort of balancing problem. The sales representative wishes to maximize the profit gained by keeping discounts at a minimum, while the customer wants to minimize his or her costs by maximizing the discounts. This is where the simulation process comes in handy - by simulating the effects of new discounts, it is possible to decide whether they are profitable enough or not.

When both the sales representative and the customer are satisfied with the results, they can save the discounts as conditions in the system's database. Discount rates from such conditions will then be applied to the customer's future orders.

This chapter is divided into two main parts. The first one describes how customer discount simulations work and how these are currently implemented in PCT. The second part focuses on a scaling extension to the simulation process, which makes it possible to use different discount rates depending on individual order volumes.

The models and algorithms presented in this chapter are not necessarily identical to the ones implemented in PCT or the optimized system. They are supposed to be read as explanations of the expected functionality of an arbitrary implementation, unless anything else is explicitly stated.

2.1 Customer discount simulation

Customer discount simulations are currently fully implemented in PCT. By running a simulation over the data described in section 2.1.1, a sales representative will find out which profit would be gained if the customer bought the same articles as in the historical period but using current pricing conditions. Even more importantly, new discount rates can be applied to the simulation meaning that the sales representative can see which effects they will give and whether they seem profitable enough or not.

The details of the simulation process are described first in section 2.1.2, but reading the chapter in the presented order is highly recommended. Understanding of the underlying concepts is a great advantage when trying to gain insight into the workings of the simulation process.

2.1.1 Data needed for a customer discount simulation

A simulation is based on data from the following sources:

- Article tree - A tree structure where branch nodes represent article categories and leaf nodes represent articles
- Sales history - A set of aggregated order rows, containing information about previous sales history
- Existing customer conditions - Agreed discount rates from existing contracts, which set a certain discount rate to a specific node in the article tree
- User input - Various parameters that specify which historical data and discount rates to use in the simulation

Since the contents of these data sources are very central to the simulation process, a quick description of each one of them is presented below.

The article tree

The article tree categorizes all of the company's articles into article groups. These are in turn grouped together into more general categories in three "price levels", where level 3 is the most specific and level 1 is the most general category. An example tree using this structure is presented in figure 2.1.

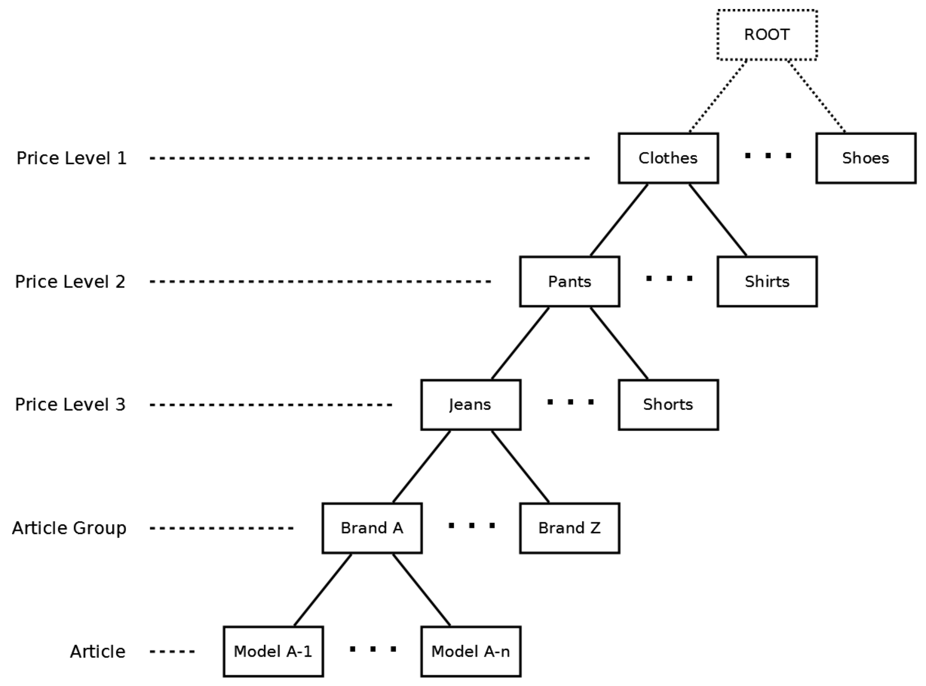


Figure 2.1: An example article tree containing clothes and accessories

As seen in this figure, leaf nodes contain articles while branch nodes represent article categories. The most general categories are stored in price level 1 (Clothes and Shoes), subcategories of these in price level 2 (Pants and Shirts are both subcategories of Clothes) and so on. In the current article database, each price level 3 node contains exactly one article group meaning that these two levels are equally specific.

While the tree in the figure is just an example (using made up names of clothes and accessories instead of the more cryptical category codes from the real system), it should be enough to explain the concept of the article tree structure used in PCT. The amount of nodes for each level in the reduced tree provided for this project is shown in table 2.1. The corresponding numbers for the actual system's tree are even larger. Since internal company policies do not allow sharing of the full article tree, this reduced tree has been used throughout the whole project.

Node level	#Nodes
Price level 1	8
Price level 2	64
Price level 3	802
Article Group	802
Article	9,706

Table 2.1: The amount of nodes for each level of the article tree provided for this project

Sales history

The sales history consists of a database containing a large set of historical order information, aggregated on a per month basis. An example aggregated order row is shown in figure 2.2.

Period	**Customer ID**	**Article no**	**Actual discount**
201207	123456	88084686	257.87
Agreed discount	**Avg. target discount**	**c0**	**Currency**
0	184.76	56.25	EUR
Customer level	**Price**	**Organisation**	**Target discount**
8	422.8	1,000	184.76
Value	**Volume**	**Weight unit**	**Market code**
82.465	55.44	KG	DE

Table 2.2: An example order row from the aggregated sales history database

This example row shows that customer *123456* in the German market bought a total of 55.44 kg of article *88084686* during July 2012. One can also see what the total value of the sold articles were, which discount the customer received and so on. The fields which are relevant for the simulation process are described in greater detail in section 2.1.2.

To provide the reader with a perspective of the amount of data stored here, the sales history database for the German market alone stores around 750,000 such aggregated order rows during a single year.

Existing customer conditions

When a sales representative and a customer agree on a discount rate for a certain article or article category, this is added to a database of *customer conditions*. An example condition is shown in figure 2.3.

ID	**Opt lock**	**Aggvalorvol**	**Channel**
abcdef0123456789	0		01
Command	**Dirty**	**Discount**	**Eff. stop date**
1	FALSE	10.5	
Freeze enddate	**Freeze pl date**	**Freeze start date**	**New freeze**
			FALSE
Note	**End date**	**Start date**	**Status**
This is just an example	201310	201211	
Contract ID	**Created by**	**Customer ID**	**Pricelevel ID**
fedcba0987654321	SalesRep01	123456	DEPL3_10
Unfreeze cond. ID			

Table 2.3: An example customer condition

In this example, we can see that the sales representative *SalesRep01* has agreed to give customer *123456* a 10.5% discount on all articles in the category *DEPL3_10*. Other fields indicate the ID of the condition, the ID of the contract which the condition belongs to, whether or not the condition is temporarily disabled ("frozen"), an optional note specified by the sales representative and so on. Once again, the fields that are relevant for the simulation process are described in greater detail in section 2.1.2.

User input

The final data needed for a simulation is provided by the user. This data consists of a *customer*, a *path* leading from the root down to an arbitrary node in the article tree, a *time period* and a set of *discount rates* for the nodes in the path.

The customer is specified as a reference to the ID of a customer in the customer database. Each sales representative has a set of assigned customers whom he or she can choose from.

The path is as a set of selected nodes in the article tree, where the first selected node lies on price level 1 and any node added afterwards must be a child of the last selected node. This means that there is always a price level 1 node in the path and that the sales representative may choose to add a price level 2 node as well. If a price level 2 node was added, the user can choose to proceed by adding a price level 3 node and so on. The shortest possible path has the length 1 (meaning that the path consists of a price level 1 node) and the longest possible path has the length 5 (containing one node each from price levels 1-3, an article group node and finally an article node), which is equal to the height of the article tree.

The time period is represented by a start date and an end date, each represented as a combination of a year and a month. When a simulation is run, historical order data whose period parameter lies inside of this interval will be used and any data outside of the interval will be ignored. The end date must of course lie after the start date and the start date must lie within the last 13 months. This limit ensures that a full year's history can always be used, since the sales history database may not contain the current month's full history yet.

Finally, the user will specify a discount rate for each node in the path. A discount rate is a decimal number between 0.0 and 99.9 with one decimal value, representing the discount percentage. It is also possible to let a node inherit its parent's value by not assigning a discount rate to it. Since the price level 1 node does not have a parent node to inherit from, its discount is set to 0.0% if no discount rate is entered on this level. The discount rates are typically modified multiple times during the simulation process, since the sales representative must simulate over multiple configurations in order to find a suitable set of discount rates to add to a contract.

2.1.2 The simulation process

The sales representative starts by entering which customer he is negotiating with and selecting a path in the article tree (see section 2.1.1) for which discounts will be entered. Next up, a start and stop month is specified and now the system is ready to run the first simulation.

Since no discount rates have been entered at this point, all nodes in the path will use their existing discount rates if any such exist in the active conditions and 0.0% otherwise. All price level 1 nodes which are not affected by the existing conditions will also have their discounts set to 0.0%. Due to the concept of *discount inheritance* (see section 2.1.3), all other nodes will inherit their parent's discount rate top-down if they do not have an existing condition. This means that the results of the first run will always show the economical results that will follow if the same item quantities are sold as in the historical data used for the simulation, taking only currently active conditions into account.

Conditions may have been added or removed since the historical orders were handled, so it is not enough to just aggregate the values and profits from the history database. Instead, the "base value" (which one can think of as the price for the order rows if no discounts had been applied) must be calculated for each article. By applying discount rates from existing conditions to these base values, the system finds out how much the customer would have to pay for the same orders if they had been placed using current conditions.

In the next step, the sales representative sets discounts for the nodes in the selected path and runs another simulation over the same data. Any conditions affecting discount rates for the path nodes will be overrun by the discount rates set by the sales representative, while conditions affecting other nodes will still be taken into consideration. The user specified discount rates will then be inherited down through the article tree just like the ones from the conditions. The result will thereby correspond to the profit which would be achieved if these new rates were added to the conditions database and the same orders as in the historical data were then placed again by the customer.

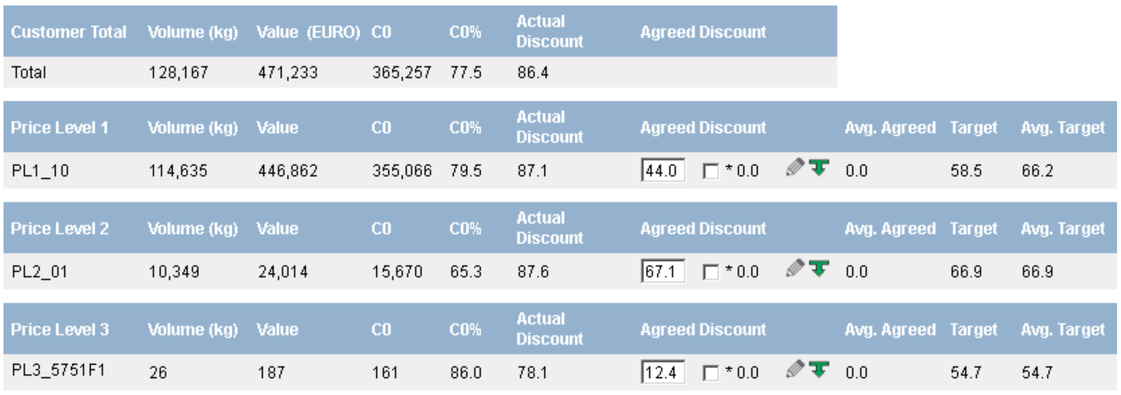
This simulation step will typically be run multiple times with different discount rates for the nodes in the path, until they are balanced in such a way that both the customer and the sales representative are satisfied with the results. Running multiple simulations with different discount rates for the same time period and historical data until one gets satisfying results is referred to as going through a *simulation process*.

Simulation output

So far, the output of simulations has been described in terms of "profit" and "value". The actual values computed during a simulation are of course more specific than that and as such, the specification of requirements presents guidelines for the output data layout.

The specification indicates that the output should be presented as a table, where each node in the selected *path* is represented as a row. There is also a top row labeled "Total", which shows the total simulation values of all articles in the whole article tree.

A print screen showing how this looks in the current version of PCT is shown in figure 2.2. The columns of each row are described in table 2.4.



Customer Total	Volume (kg)	Value (EURO)	C0	C0%	Actual Discount	Agreed Discount
Total	128,167	471,233	365,257	77.5	86.4	

Price Level 1	Volume (kg)	Value	C0	C0%	Actual Discount	Agreed Discount	Avg. Agreed	Target	Avg. Target
PL1_10	114,635	446,862	355,066	79.5	87.1	44.0 ☐ * 0.0	0.0	58.5	66.2

Price Level 2	Volume (kg)	Value	C0	C0%	Actual Discount	Agreed Discount	Avg. Agreed	Target	Avg. Target
PL2_01	10,349	24,014	15,670	65.3	87.6	67.1 ☐ * 0.0	0.0	66.9	66.9

Price Level 3	Volume (kg)	Value	C0	C0%	Actual Discount	Agreed Discount	Avg. Agreed	Target	Avg. Target
PL3_5751F1	26	187	161	86.0	78.1	12.4 ☐ * 0.0	0.0	54.7	54.7

Figure 2.2: A print screen from PCT showing how simulation output is presented in the current system

Field name	**Unit**	**Type**	**Description**
Node name	n/a	String	The name of the row's node
Volume	kg	Integer	The total volume of all orders for articles under the row's node in the article tree
Value	Euro	Integer	The total amount of money which the customer would have to pay if all historical orders for articles under the row's node were placed again, with the new discounts applied
C0	Euro	Integer	The profit which the company would gain if all historical orders for articles under the row's node were placed again, with the new discounts applied
C0%	Percent	Decimal number	Shows how many percent of the row's value C0 corresponds to, i.e. (Value/C0)*100
Actual discount	Percent	Decimal number	The average historical discount for articles under the node in the simulation period
Above target	n/a	Boolean	A warning flag which shows whether the agreed discount is higher than the node's target discount
Agreed discount	Percent	Decimal number	The discount used for the row's node in the current simulation
Avg Agreed	Percent	Decimal number	The average agreed discount for articles under the row's node in the article tree
Target discount	Percent	Decimal number	A recommended target discount for the node, based on the customer's pricing level
Avg Target	Percent	Decimal number	The average historical target discount of articles under the row's node

Table 2.4: The columns which are used to structure the output from a simulation

The five last columns are empty for the "Total" row, since these values are considered irrelevant to display for the whole article tree.

2.1.3 Discount inheritance

Discounts can be applied to nodes on any level of the article tree - from price level 1 down to specific articles. It is intuitive that a discount which is set for a single article will only affect the price of that specific article. When it comes to discounts set on article groups or price level nodes, the system uses a concept called "discount inheritance" to let this affect underlying nodes. In order to determine which discount rate to apply to a given node, the method presented in algorithm 2.1.1 is used.

Algorithm 2.1.1: findDiscountRate(Node n)

```
Input: A node n from the article tree
Result: The discount rate which should be applied to n
1  if n is a node in the path for which a discount rate d is set then
2      return d
3  else if n is not a node in the path AND n has an active condition c then
4      return the discount rate from condition c
5  else if n is a price level 1 node then
6      return 0.0%
7  else
8      parent := n's parent node in the article tree
9      return findDiscountRate(parent)
10 end
```

The concept of discount inheritance is easy to visualize due to the tree structure of the article database. An example tree with some existing discount rates is shown in figure 2.3. Existing discount rates are written directly onto the grey nodes to which they belong, while nodes without such rates are white. The final result of the discount rate inheritance in the same tree can be seen in figure 2.4, where arrows show how discount rates are passed down through the tree.

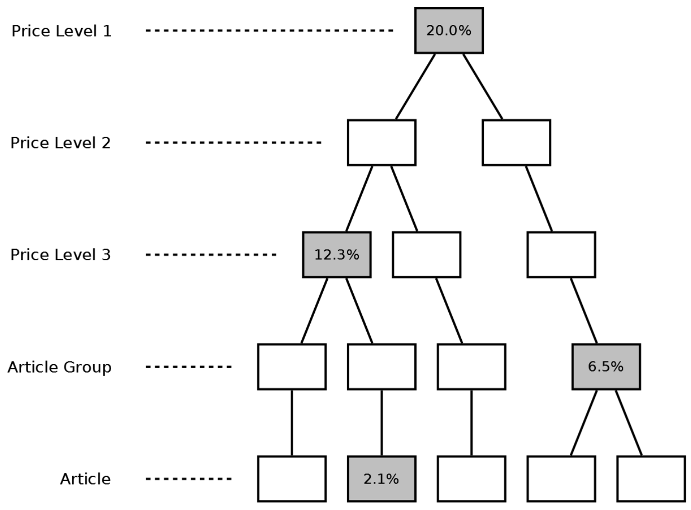


Figure 2.3: An example article tree where discount rates have been set for four nodes

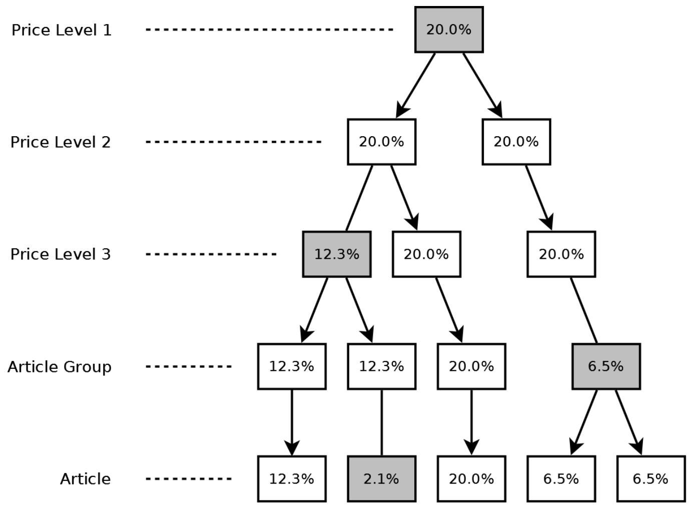


Figure 2.4: Discount inheritance in the example article tree from figure 2.3

2.1.4 Current implementation

As mentioned in the project motivation in section 1.1, the current implementation of PCT suffers from critical performance issues. Since the source code of this system is not allowed to be included in this report, the problems of its algorithm have to be explained in terms of bad structure choices and complexity rather than examples and excerpts from the actual code.

A (very) rough outline of the algorithm structure used to perform simulations in PCT is presented in algorithm 2.1.2. While it does not motivate or explain the details of each step, it does provide enough information to analyse its complexity. To give the reader some sort of idea of the actual magnitude of the implementation of this algorithm, its Java source code takes up several hundred kilobytes (not including GUI, server connections, database handling and other parts which are not directly related to the algorithm). In other words, a line describing e.g. criteria matching means running a separate algorithm which in turn has a complexity worth mentioning.

Algorithm 2.1.2: Structure of the simulation process in PCT

```
if this is the first run of the simulation process then
    initialize connection to each input data element in the GUI [$\mathcal{O}(k)$]
end
foreach price level in the article tree [$\mathcal{O}(k)$] do
    match condition level [$\mathcal{O}(k)$]
    match price level [$\mathcal{O}(k)$]
    foreach item in the customer's cache [$\mathcal{O}(n)$] do
        match criteria [$\mathcal{O}(k)$]
    end
    retrieve target discount [$\mathcal{O}(k)$]
    foreach article in the article tree [$\mathcal{O}(a)$] do
        foreach article in the customer's cache [$\mathcal{O}(n)$] do
            match criteria [$\mathcal{O}(k)$]
            foreach price level in the article tree [$\mathcal{O}(k)$] do
                retrieve data and calculate results
            end
        end
        retrieve agreed discounts [$\mathcal{O}(k)$]
        compare discounts to target discounts [$\mathcal{O}(k)$]
    end
end
foreach article in the customer's cache [$\mathcal{O}(n)$] do
    calculate results for articles under price level 1 nodes $\notin$ path
end
```

In the pseudo code above, the complexity has been included on each line where $\mathcal{O}$ notation is applicable. The meaning of each occuring variable in the $\mathcal{O}$ notation is presented in table 2.5.

Variable	Magnitude	Description
a	$\sim 30{,}000$	The amount of articles in the article tree (exact numbers for the reduced tree used in this project can be seen in table 2.1)
k	5	Height of the article tree (constant in this program, but may vary between implementations)
n	$1 \leq n \leq 13a$	Distinct (month, article) tuples in the selected customer's sales history for the last 13 months

Table 2.5: The meaning of different complexity variables in algorithm 2.1.2

The total complexity of the implementation of the current simulation algorithm is

$$\mathcal{O}(k+k(k+k+nk+k+a(n(k+k))+k+k)+n) = \mathcal{O}(k+5k^2+nk^2+2ank^2+n) = \mathcal{O}(ank^2)$$

It should also be noted that the complexity of repeated runs of the algorithm is

$$\mathcal{O}(k(k+k+nk+k+a(n(k+k))+k+k)+n) = \mathcal{O}(5k^2+nk^2+2ank^2+n) = \mathcal{O}(ank^2)$$

This is barely an improvement from the first run at all - one single k term is removed since the initialization step on line 2 does not need to be run again. The total complexity of $\mathcal{O}(ank^2)$ is high in itself, since both a and n can hold quite large numbers and the k term is used at multiple places. However, this is not the only reason behind the high running times of the algorithm.

Another big problem is the on-demand usage of database resources. Every time a set of values from the database is needed, a new connection to the database is opened. The sought values are then retrieved by an SQL query and afterwards the database connection is closed again. Repeatedly opening and closing database connections takes time and this is done in many parts of the algorithm, including the data retrieval methods mentioned in line 15. This means that $\mathcal{O}(ank^2)$ database connections and SQL queries may have to be opened and run in the worst case. Some values are even retrieved from the database multiple times during a single execution, since they are used in multiple places in the code but are not saved after being retrieved the first time.

It should however be noted that some actions have been taken in order to reduce the amount of data retrieved from the database per simulation. Every customer's order history for the last year is stored as a list in a hash map indexed by the customer ID, which thereby works as an in-memory database. This makes retrieval of a customer's historical data (without direct database access) in $\mathcal{O}(1)$ time possible. Of course, iterating over the resulting list will still take $\mathcal{O}(n)$ time. This cache is created on server startup and updated regularly, so creation and updates of the cache do not affect the running time of the simulation algorithm. Some loops in PCT are still performed over all distinct values in certain database tables when information from this cache could have been used instead, leading to even more unnecessary database lookups.

A third cause of the high running time is the redundant calculation of certain values. The foreach loop on lines 4-21 in the algorithm above runs once for each price level and calculates the results for all articles under the node on that level. This means that the results for all articles under the price level 1 node will be calculated first, followed by a recalculation of all articles under the price level 2 node and so on for a total of up to k calculations of the same values for some articles.

Let the selected path in the simulation be called P and the path from an arbitrary article a up to its price level 1 ancestor be called P_a. Then, $|P \bigcap P_a|$ (the amount of nodes which are both in P and in P_a) equals the number of times article a's results has to be calculated during each simulation. Calculating the same value more than once is of course redundant and adds unnecessary running time. This is visualised in figure 2.5.

Nodes in the path P are marked with thick outlines in the figure. Values for articles within the blue box ($A_1 - A_5$) are calculated once, while articles within the green box ($A_1 - A_4$) are calculated once more and articles within the red box (A_2,A_3) yet another time. The purple box (A_6) marks articles that lie under a price level 1 node $\notin P$, whose values are always calculated only once.

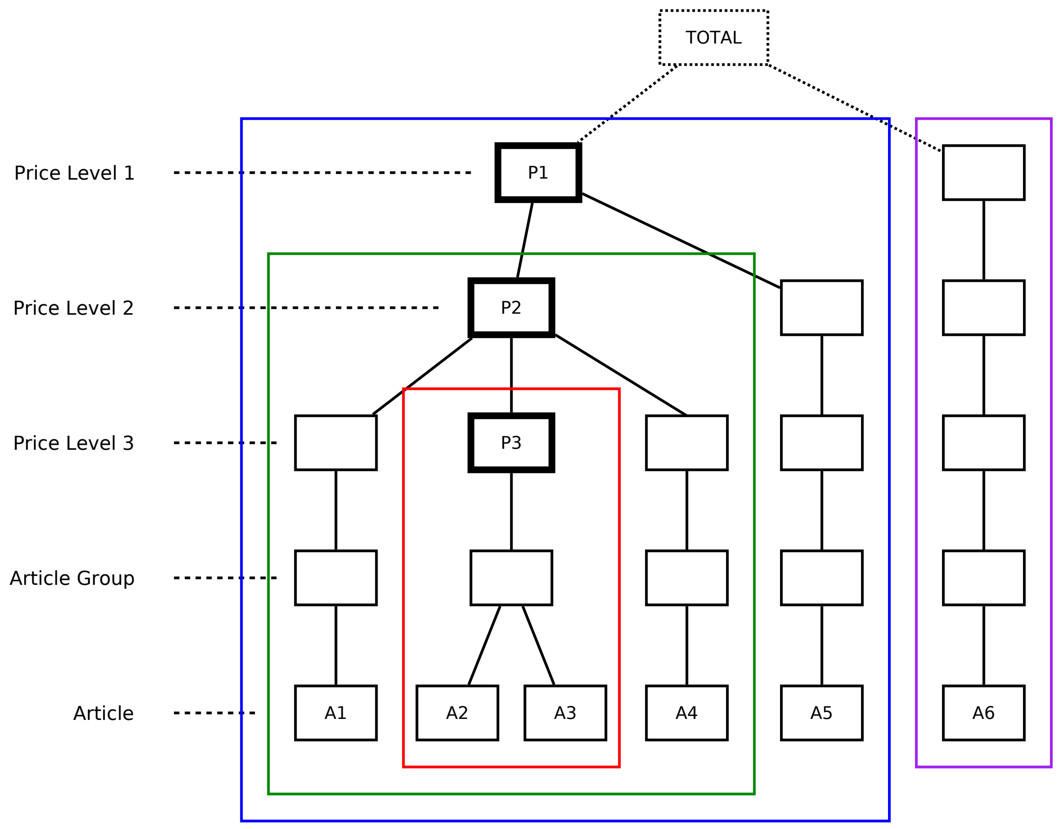


Figure 2.5: Redundant calculation of values in PCT

The combination of a high time complexity, inefficient database usage and redundant calculations cause the running time of each simulation to grow rapidly for increasing values of n.

2.2 Scaling simulation

As mentioned in section 1.1, the company wants a scaling extension of the simulation process and conditions in PCT. The purpose of this extension is to make it possible to apply different discount rates depending on the volume of each individual order. If larger order volumes are rewarded with higher discounts, customers will be more likely to place large orders a few times per year instead of small orders every week or month, thus decreasing shipping and warehouse charges.

The extension's specification is built around a concept called "discount stairs". These are set by sales representatives on a per customer and node basis, in order to define which discount rates will be applied to orders of certain volumes. This concept is described in greater detail in section 2.2.2. The data sources which are needed for scaling simulations are in turn described in section 2.2.1.

2.2.1 Data needed for a scaling simulation

A scaling simulation is based on data from the following sources:

- Article tree - A tree structure where branch nodes represent “price levels” (article categories) and leaf nodes represent articles
- Sales history - A set of order rows, containing information about previous sales history. The aggregated sales history from the customer discount model is also required.
- Existing scaling conditions - Agreed scaling conditions from existing contracts, which set a certain discount stair to an article, article group or price level 3 node in the article tree
- User input - Various parameters that specify which historical data to use, which discount stair to use and various other simulation settings

The article tree

This is exactly the same tree as the one used for customer discount simulations, which is described in section 2.1.1.

Sales history

The sales history used for scaling simulations consists of a database containing a large set of historical orders. Note that these order rows are not aggregated, as opposed to the ones used for customer discount simulations. An example order row is shown in table 2.6.

Invoice date	**Invoice no**	**Invoice value**	**Invoice volume**
2012-07-03	a138215	40.345	35
Article ID	**Customer ID**	**Unit ID**	**ID**
DE88001104	123456	kg	379311

Table 2.6: An example order row from the order history database

However, in some situations data from the aggregated database table (described in section 2.1.1) is used as well. This means that scaling simulations require both of these database tables to be present.

Existing scaling conditions

Agreed discount stairs for specific article tree nodes and customers are added to a database of *scaling conditions*. An example scaling condition is show in table 2.7.

Price level ID	**Customer ID**	**v0**	**v1**	**v2**	**v3**	**v4**
$DEPL3\_5690F1$	123456	5	10	20	30	50
v5	**d0**	**d1**	**d2**	**d3**	**d4**	**d5**
	12.1	15.0	17.5	20.0	25.3	

Table 2.7: An example scaling condition

In this example, the scaling condition covers the price level 3 node $DEPL3\_5690F1$ for customer 123456. The stair has five thresholds, whose volume limits are shown in columns $v0 - v4$ and their respective discounts in columns $d0 - d4$. Since there is no sixth threshold in this example, $v5$ and $d5$ are left empty. A description of how these values are used in a scaling simulation is presented in section 2.2.3.

According to the scaling extension specification, all customers can be expected to have a total of at most ten active scaling conditions. Most discounts are in other words still expected to be handled through customer discount conditions when the scaling extension has been implemented.

User input

The final data needed for a scaling simulation is provided by the user. This data consists of a *customer*, a *time period*, a *node* and a *discount stair*.

The customer is a reference to the ID of a customer in the customer database, just like in customer discount simulations.

The time period is represented by a start date and an end date, which is slightly different from the time period in customer discount simulations. Since the historical order rows used for scaling simulations are not aggregated per month like the ordinary order history, these dates specify a day of the month as well.

The node is a reference to either an article, an article group or a price level 3 node in the article tree. This is quite different from the path used in customer discount simulations - not only because only a single node is selected, but also because price level 1 and price level 2 nodes can not be used.

A discount stair is a way of defining different discount rates for the same node, depending on individual order volumes. This is described in greater detail in section 2.2.2.

2.2.2 Discount stairs

As mentioned in section 2.2.1, discount stairs make it possible to apply different discount rates for orders depending on their volumes. Just like in customer discount simulations, discount inheritance (see section 2.1.3) is applied. However, scaling conditions can not be set for price level 1 or price level 2 nodes in the article tree. The reason behind this is that articles who do not share the same ancestor on price level 3 are generally considered too diverse to share volume limits. In other words, it would not always make sense to apply the same discount rate for i.e. orders between 20 and 25 kg on two articles of very different types.

A discount stair consists of between one and six volume thresholds and a discount rate for each one of these. The thresholds indicate boundaries between weight intervals, meaning that i thresholds define $i+1$ intervals. A set of such thresholds is shown in table 2.8 and its resulting interval limits are shown in table 2.9. Since there are six thresholds in this example, there are seven discount intervals.

Threshold volume	**Threshold discount (%)**
5	12.1
10	15.0
20	17.5
30	20.0
50	25.3
70	30.1

Table 2.8: An example set of discount thresholds

Volume v	**Discount (%)**
$v < 5$	0.0
$5 \leq v < 10$	12.1
$10 \leq v < 20$	15.0
$20 \leq v < 30$	17.5
$30 \leq v < 50$	20.0
$50 \leq v < 70$	25.3
$70 \leq v$	30.1

Table 2.9: The resulting discount intervals from the thresholds in table 2.8

From the last table we can easily see that an order with e.g. volume $v = 4$ would get 0.0% discount if this stair is used, while an order with volume $v = 27$ would receive a 17.5% discount.

2.2.3 The scaling simulation process

The general workings of scaling simulations are similar to those of customer discount simulations, but some differences are of course present. A scaling simulation begins with a sales representative selecting a customer. In excess of this, a price level 3 node, article group node or article node in the article tree (see section 2.1.1) is selected and a time interval specified. Furthermore, a set of one to six volume thresholds is specified. Note that only volumes are entered before the first run - actual discount rates for these are not entered until later. The system is now ready to run the first scaling simulation.

During the first run, the selected node n will use the discount rate 0.0% for each volume interval specified in the input step. After each run, the user can modify the discount rates for each interval of n's discount stair, apart from the lowest one which is always locked to 0.0%. The values for all articles are calculated according to the method described in algorithm 2.2.1.

Algorithm 2.2.1: getValues(Node n_a)

```
   Input: An article node $n_a$ from the article tree
   Result: Total simulated cost, value and volume for article $n_a$
 1 current := $n_a$
 2 repeat
 3     if current is the selected simulation node with discount stair $d_s$ then
 4         return scalingSimulation($n_a$, $d_s$)
 5     else if current has an existing discount stair $d_a$ in a scaling condition then
 6         return scalingSimulation($n_a$, $d_a$)
 7     end
 8     current := current's parent in the article tree
 9 until current is a price level 3 node
10 return Total cost, value and volume from aggregated (i.e. not scaling) history
   database for $n_a$ during selected time period
```

As line 10 in the algorithm above shows, values for articles that are not affected by the scaling node or scaling conditions are retrieved directly from the aggregated database table (which is also used for customer discount simulations). This works since the aggregated data for a month per definition equals the sum of all individual orders from the same month.

Even if only a part of the month is covered by the selected time interval for the simulation, the whole month's history will still be used in this case. It should also be noted that the calculation of base prices and application of customer discounts are ignored - the aggregated values are used directly in order to lower the scaling simulation's complexity.

Algorithm 2.2.2 shows how the *scalingSimulation()* function which is called in algorithm 2.2.1 works.

Algorithm 2.2.2: scalingSimulation(Node n_a, DiscountStair d_s)

```
Input: An article node $n_a$ from the article tree and a discount stair $d_s$
Result: Simulated values (cost, value and volume) for article $n_a$
1  totalCost := 0
2  totalValue := 0
3  totalVolume := 0
4  orderRows := all order rows from the scaling sales history (see section 2.2.1) for
   article $n_a$ within selected time period
5  foreach row r in orderRows do
6      rowVolume := $r$'s volume
7      aggrPrice := price for $n_a$ in $r$'s month in the aggregated history database
8      aggrCost := cost for $n_a$ in $r$'s month in the aggregated history database
9      aggrVolume := volume for $n_a$ in $r$'s month in the aggregated history database
10     listPrice := aggrPrice / aggrVolume
11     listCost := aggrCost / aggrVolume
12     rowDiscount := The discount from $d_s$ whose volume interval covers rowVolume
13     rowCost := rowVolume * listCost
14     rowValue := rowVolume * listPrice * (1 - rowDiscount*0.01)
15     totalCost := totalCost + rowCost
16     totalValue := totalValue + rowValue
17     totalVolume := totalVolume + rowVolume
18 end
19 return (totalCost, totalValue, totalVolume)
```

Since the specification of historical order rows does not include any columns for price and cost, these values have to be calculated from the aggregated historical data. First off, the article's "list price" and "list cost" are calculated as a sort of base values on the form currency unit/volume unit (e.g. Euro/kg) for the specified article and month. These values are then multiplied by the current order row's volume in order to get its price and cost.

Next up, the row's simulated value is calculated. The row's volume is matched to a volume interval in the discount stair (as seen in section 2.2.2) and the corresponding discount is applied to the row's price in order to find its value.

Finally, the sum of all row's costs, values and volumes are returned. The simulation results are obtained by aggregating the resulting values for all articles, including the ones where values are retrieved from the aggregated historical database.

2.2.4 Scaling simulation output

The output of a scaling simulation should be presented just like the output of a customer discount simulation, which is described in section 2.1.2. The bottom row covers the scaling node, all of its ancestor nodes have a row each in the middle and the top row contains the total values for the whole article tree.

2.2.5 Technical difficulties

Scaling simulations have not yet been implemented in PCT. Adding this functionality has been considered impractical, since scaling simulations run over far larger data sets than customer discount simulations (which already have problems with high running times). The non-aggregated data is even too large to be held in an in-memory database cache, which means that every historical order row will have to be retrieved from an ordinary database. This slows down the data handling even further.

Customers are estimated to place orders for the same article up to once a week and the time period used for a scaling simulation can be at most one year. This means that one can assume a maximum of 52 order rows per article in a single scaling simulation.

As shown in table 2.1, the reduced article tree contains 9,706 distinct articles while the full article tree has around 30,000 nodes. It is deemed possible that a single customer buys up to a thousand different articles regularly.

Scaling simulations can only be run over article nodes or price level 3 nodes in the article tree, but if scaling conditions are specified for other such nodes than the selected scaling node, these require separate scaling simulations on their own. The running time of a scaling simulation over an arbitrary node for a customer will thereby increase for every scaling condition added for the same customer. Price level 3 nodes have an average of 12.3 and a maximum of 172 underlying articles, so adding a single condition could increase the number of required historical order rows by almost 9,000.

This means that running a scaling simulation can require multiple separate scaling simulations for articles with existing scaling conditions. In total, these could require computations over as many as 52,000 historical order rows.

3

Method

This chapter explains the optimized models and algorithms which we have developed during this project. Just like the previous chapter, this is divided into two parts. The first part shows how the customer discount model can be optimized and the second part describes an efficient model for scaling simulations.

3.1 Customer Discount Model

If we examine the current simulation process described in section 2.1.2 and remember the issues from section 2.1.4, we note that the major bottlenecks in the existing implementation are that:

- Simulations suffer from a high time complexity
- Data retrieval from the database and in-memory cache is done in an inefficient way
- The same data is redundantly calculated and aggregated several times in the simulation process

Intuitively, a good place to start in order to reduce the time complexity is to eliminate redundant calculations in the simulation process. Doing this would not only improve the performance of single simulations, but also lead to shorter running times for repeated simulations. An example of this is to save the parts that remain constant during repeated simulations, in order to avoid reaggregation and recalculation of the same values in each run.

Apart from the three bottlenecks there is another issue which can be addressed, i.e. the structure of the article tree. From table 2.1 we can see that there is an equal number of price level 3 and article group nodes in the article tree. As one might suspect, each price level 3 node categorizes exactly one article group. This means that the article group level does not fill any function and can be entirely removed from the article tree. The resulting article tree will as such consist of four levels, i.e. price level 1-3 and the article level.

A rough outline of our optimized model is to first construct a path tree, where we aggregate parts of the sales history prior to actually performing any simulation calculations. Since we know which articles will be used in the simulation process after the simulation path has been selected, it is a simple matter to tag the articles whose values will remain constant and the ones which will be affected by the simulation. This means that we only need to iterate over the articles in the article tree once. After this, it is a simple and relatively fast procedure to apply the user set discounts and merge the results of the simulation. In order to avoid redundancy, we will use a bottom up approach to merge the results of the simulations.

Our solution consists of two major parts. In the first step, we perform the expensive part of tagging and aggregating the data needed for the simulation. In the second step we perform a small series of calculations on the aggregations to run the simulations and merge the results.

3.1.1 Preparation step

The preparation step consists of retrieving the sales history for the selected customer and time period and merge this with the existing article tree to construct a path tree, which is shown in figure 3.1. Here all the necessary data for the simulation will be aggregated without any user specified discounts applied. This means that we can apply the discounts and calculate the results in a later procedure.

This new tree structure consists of the selected simulation path nodes with dependent and constant parts attached to each node. The dependent part contains the values of articles under the path node that will be directly affected by the discount set at that node. Respectively, the constant part contains the values of articles under the path node that will remain constant regardless of the discount set at the node. This includes articles that lie under another price level one node and articles covered by existing conditions. In the figure we note that the total level does not have a dependent part, since we can only simulate up to price level one. Similarly, the article level does not have a constant part since articles cannot have any underlying nodes.

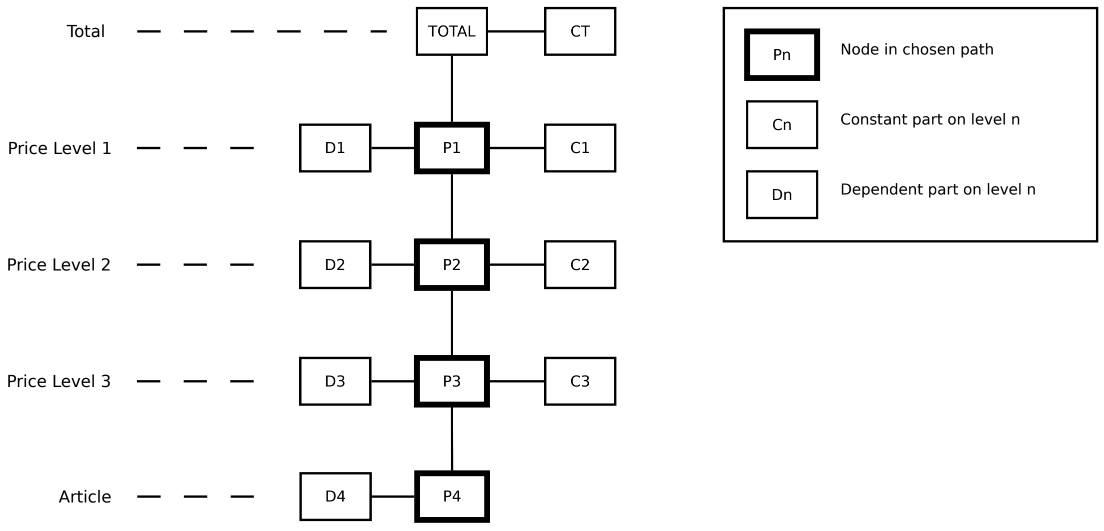


Figure 3.1: Path tree with aggregated sales history

The aggregation of the relevant history from the in-memory cache into the correct dependent or constant node in the path tree is described in algorithm 3.1.1.

By aggregating the customer's historical data from the PCT cache (which was described in section 2.1.4) into a temporary customer cache, a smaller set of historical rows is obtained without affecting its total values. The idea here is to have the order history aggregated for each article and customer on a time period basis. This is a big improvement from the original solution, since we can now access the total aggregated sales history for an article in constant time. Previously, this would have required iteration through the customer's entire sales history each time data for an article was needed. We also created a cache for all of the conditions for the selected customer, in order to make fast lookup of discounts from conditions possible.

Algorithm 3.1.1: Prepare simulation

```
Data: An empty path tree treepath with the same number of levels as specified
      from user input
Result: treepath populated with aggregated sales history
        history populated with all sales history for the selected customer and
        time period
        conditions populated with all conditions that exist for the selected
        customer
1  Construct and populate a hash map history that contains aggregated sales history
   for the selected customer and time period, with the article number as key
2  Construct and populate a hash map conditions that maps nodes in the article
   tree to existing conditions for the customer
3  foreach AggregatedSalesHistory art in history do
4      (index,isConstant,discount) := findDiscountAndDependency(art)
5      current := chosen path node at level index in treepath
6      if isConstant then
7          constVal := apply discount to sales history for art
8          add constVal to current's constant node
9      else
10         add sales history for art to current's dependent node
11     end
12 end
```

The function call to findDiscountAndDependency that is used in algorithm 3.1.1 finds which node in the path tree to add the article's sales history to and which discount to apply. This is done by traversing the original article tree upwards from the article, examining if there are any conditions present for the current node in each step. For each article there are three possible outcomes:

- The article's discount is dependent on a node in the simulated path
- The article is in another price level 1 subtree without a condition
- The article's discount is inherited from an existing condition

When the article's discount depends directly on the simulated path, we simply aggregate the sales history into the correct dependent node in the path tree. We don't have to find a discount in this case, since we will use the one specified by the user. If the article is in another price level one subtree and we do not have a condition specified, we aggregate the sales history to the total constant node with a discount of 0%. Finally, if the article's discount depends on a node with a specified condition we begin by finding the condition and retrieving its discount. We then locate the path node which this constant article's data should be aggregated to. Pseudo code for this is presented in algorithm 3.1.2.

Algorithm 3.1.2: findDiscountAndDependency(AggregatedSalesHistory *art*)

```
Input: Aggregated sales history for an article art
Result: Returns a tuple consisting of
        • the discount that should be applied to the article
        • a boolean indicating whether the article is constant
        • the level in the path tree which art should be added to

discount := 0.0
isConstant := false
foreach node n in the path in the article tree from art to price level 1 do
    for i from 1 to 5 do
        if n equals the node at level i in path tree then
            return (i,isConstant,discount)
        end
    end
    if n has an existing condition in the conditions hash map then
        discount := existing condition of n
        isConstant := true
    end
end
/* Node lies under a price level 1 node outside of the path */
return (0, true, 0.0)
```

3.1.2 The simulation step

After running the preparation step, the heavy computations have been performed and all that remains is to apply the discounts to the dependent nodes and merge the results. We apply a bottom up strategy as shown in figure 3.2, where arrows show how values propagate upwards in the path tree. This is far more efficient than calculating from the top down, as was the case in PCT (see section 2.1.4).

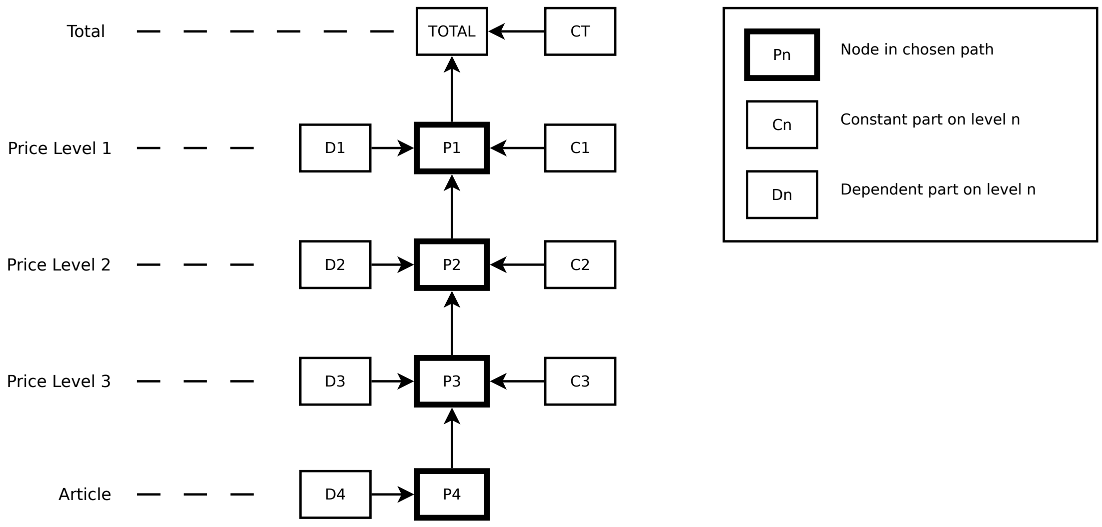


Figure 3.2: Propagation of simulated values

The simulation calculations consist of applying the discount from the path node to the data values in the dependent node and calculating the values specified in table 2.4. The next step is to add the values from the constant and child nodes to the simulated dependent value and reiterate this process for each path node until the total level is reached. A high level description of the propagation of the simulated values and the mathematical formulas used to perform these calculations are shown in figure 3.3.

$$\begin{aligned} V_4 &= d_4(D_4) \\ V_3 &= d_3(D_3) + C_3 + V_4 \\ V_2 &= d_2(D_2) + C_2 + V_3 \\ V_1 &= d_1(D_1) + C_1 + V_2 \\ V_T &= C_T + V_1 \end{aligned}$$

V_i	The values of path node i
$d_i(D_i)$	The discount applied to dependent of node i
C_i	The constant values of node i

Figure 3.3: Customer discount simulation formulas

The procedure used when running a simulation is presented in algorithm 3.1.3.

Algorithm 3.1.3: Customer discount simulation

```
Data: A populated path tree
Result: Calculates the values of all simulation path nodes
1  foreach node n in the simulation path tree from lowest to highest level do
2      if n is on lowest level in path then
3          child := null
4          constant = null
5      else
6          child := path node on the next level
7          constant = n.constant
8      end
9      if n is on highest level in path then
10         dependent := null
11     else
12         dependent := n.dependent with n's discount applied
13     end
14     calculate n.values based on constant, dependent and child
15 end
```

To summarize the customer discount model, we have made four major changes to the original model from PCT. The first is that we perform all of the data aggregation from the in-memory cache in a prepare step and aggregate the data from each article into the correct node in the simulation path. The second change is that the path tree is also stored between simulations to allow for faster repeated simulations. The third one is that article groups have been removed from the article tree. The last change is that we merge the results of the simulations in a bottom up fashion, in order to remove the redundancy caused by recalculation of values for each level in the path.

3.1.3 Complexity analysis

In section 2.1.4 we performed a complexity analysis of the PCT implementation. In order to get an idea of how this model relates to PCT, we will perform a complexity analysis of our model as well and compare them. Since the preparation step is where the heavy calculations occur in our model, it is most suitable to compare the complexity of that part to the original model. A very rough outline of the preparation step's complexity in our model is given in algorithm 3.1.4.

Algorithm 3.1.4: Structure of the prepare simulation step

```
1  Build article cache [$\mathcal{O}(n)$]
2  Build condition map [$\mathcal{O}(c)$]
3  foreach article in article cache [$\mathcal{O}(u)$] do
4      foreach price level in path from article to price level 1 [$\mathcal{O}(k)$] do
5          Find discount from possible existing condition
6      end
7      Find path node to aggregate article to [$\mathcal{O}(k)$]
8  end
9  Calculate initial values [$\mathcal{O}(m)$]
10 Set target and agreed discounts [$\mathcal{O}(k)$]
```

The pseudocode above contains the complexity in $\mathcal{O}$ notation on each line where it is applicable. A description of each variable is presented in table 3.4.

Variable	Magnitude	Description
a	$\sim$ 30,000	The number articles in the article tree (exact numbers for the reduced tree used in this project can be seen in table 2.1)
k	4	Height of the article tree
n	$1 \leq n \leq 13a$	Distinct (month, article) tuples in the selected customer's sales history for the last 13 months
c	< 50	The number of active conditions for the selected customer
m	$1 \leq m \leq k$	The height of the selected simulation path
u	$1 \leq u \leq a$	The number of distinct articles in the customer's sales history for the selected period

Figure 3.4: Explanation of the complexity variables in algorithm 3.1.4

From algorithm 3.1.4 we can deduce that the complexity for the preparation step is

$$\mathcal{O}(n + c + u(k + k) + m + k) = \mathcal{O}(n + c + 2uk + m + k) = \mathcal{O}(uk)$$

Once the preparation step has been executed, simulations will have a complexity of $\mathcal{O}(m)$. The analysis of the original implementation in section 2.1.4 showed that the time complexity for each simulation in PCT is $\mathcal{O}(ank^2)$, where a is the number of articles, n is the number of articles with sales history and k is the height of the article tree. The complexity is the same both for single and repeated simulations in PCT.

By comparing the complexity of the preparation step in our model to PCT, we can see that there is a factor of $\mathcal{O}(nk)$ difference in favor of our model. In repeated simulations there is a difference of factor $\mathcal{O}(ank)$, since m is bounded by k. This means that at least in theory, our model should work significantly faster than PCT.

3.2 Scaling model

As we saw in section 2.2, the scaling functionality requires an extension and a restructuring of the existing simulation procedure. There are however several aspects that remain the same from the customer discount model. The user will still have to choose a path in the article tree to simulate over and it must be possible to run repeated simulations over the same chosen path. Discount inheritance still applies, with the difference that we now have to apply a discount stair (see section 2.2.2).

In order to organize the results, we can reuse the path tree from figure 3.1. This method has already proven efficient in the customer discount model and could as such be used to make quick repeated simulations possible in the scaling model as well.

The main difficulty of the scaling feature is that we now need to work with individual order rows instead of using data aggregated on a per month basis. This means that we can no longer rely solely on the in-memory cache, but that we rather have to retrieve most of the data needed for a simulation directly from the database. The reason for this is that the scaling simulation applies different discounts based on the volume of each specific order row and that this data is too large to be kept in a cache.

Since it is generally slower to read from disk than from memory, one bottleneck in this model is the number of accesses to the database. This means that two of the main optimization goals are to improve the performance of the database and to minimize the number of accesses to it.

As the specification dictates, we can only have one scaling node in a scaling simulation. Articles that do not depend directly on this node can be treated in the same way as constant article values were treated in the customer discount model. Since the dependent nodes in the path tree have been replaced by this single scaling node, the path tree used in the scaling model will have the structure shown in figure 3.5.

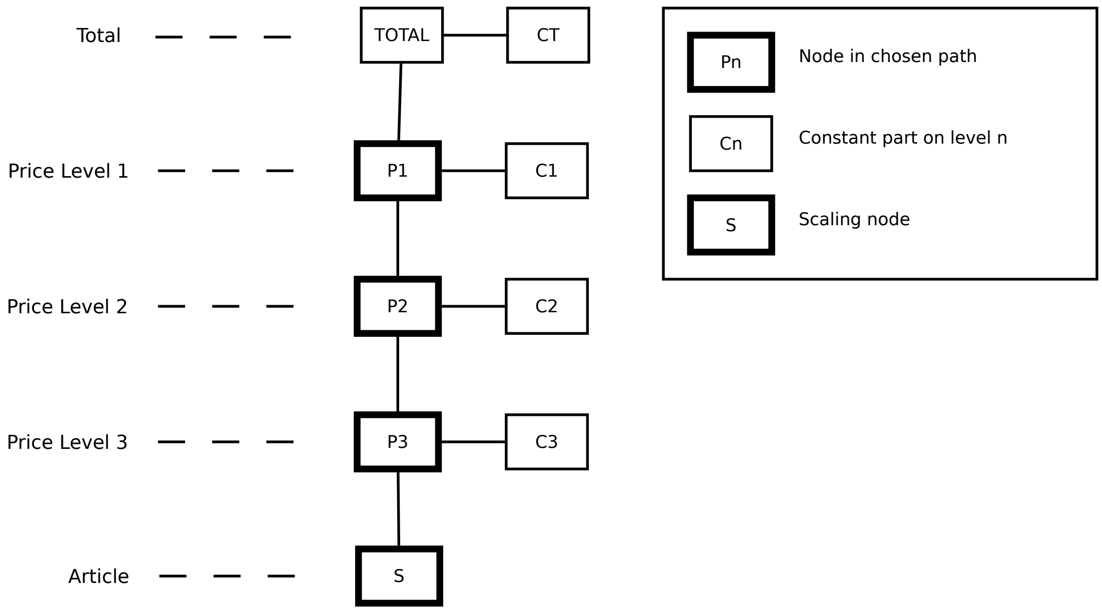


Figure 3.5: Path tree with scaling functionality

3.2.1 Model overview

Since the new functionality demands an extension of the old model, we can originate from our optimized customer discount model. We then need to modify some of its steps and add some new features. The scaling model consists of these three steps:

- Calculate the constant values of the article tree
- Perform precalculations for the scaling node
- Run a simulation over the scaling node

The constant values can be calculated in a manner similar to the prepare simulation step from the customer discount model. The main difference is that scaling conditions are used instead of customer conditions.

The primary idea of the precalculation step is to create an on-demand cache, which will hold all of the order rows needed for a scaling simulation over the scaling node. This new cache will then be used to speed up repeated simulations when the step volumes are changed between runs. After the step volumes have been set, the order rows will be aggregated into the correct step in preparation for an actual simulation. When the discounts in the discount stair have been set as well, a simulation can be run in close to no time in a fashion similar to the method from the customer discount model. This means that repeated simulations with the same step volumes will have very low running times.

3.2.2 Calculate constants

The constant calculation step is quite similar to the preparation step used in the customer discount model. We iterate over each article in the in-memory sales cache for the selected customer and aggregate the article's values to the correct path node in the path tree. The difference is that we ignore the articles that are used in the scaling part of the simulation, because we need to use the individual order rows of these articles. The basic outline of the simulation preparation is shown in algorithm 3.2.1.

Algorithm 3.2.1: Prepare scaling simulation

```
Data: An empty converted tree treepath with the same number of levels as
      specified from user input plus one for the "Total" level
Result: treepath populated with aggregated sales history with discounts applied
        conditions populated with all scaling conditions that exist for the
        selected customer
1 Construct a hash map history that will contain aggregated sales history for the
  selected customer and time period with the article number as key
2 Construct a hash map conditions that contains the discount stairs for the nodes
  that have scaling conditions with the node as key
3 foreach AggregatedSalesHistory art in history do
4     (values,pathlevel) = getValuesAndPathLevel(art)
5     current := node at pathlevel in treepath
6     add values to current's constant node
7 end
8 prepareScalingNode()
```

The algorithm described above is very similar to the one that we saw in the customer discount model, but with one major difference. In the customer discount model, the calculation of the constant parts was a simple and fast method where the discount could be retrieved from the condition and then applied to the aggregated data. However, using scaling conditions is not that straightforward.

As we saw in section 2.2.2, a scaling condition consists of a scaling stair where the discount of the correct step needs to be applied to each of the affected articles' order rows. First we need to find the concerned order rows in the database, then retrieve the correct discounts for each such order row and finally apply the discounts to them. The major downside with this method is that each time we encounter a scaling condition, we more or less need to perform a scaling simulation on all articles that the condition applies to.

Having to handle scaling conditions also means that we need to redesign the algorithm that finds the discount and corresponding node in the simulation path (algorithm 3.1.2). The reason for this is that we can no longer apply the condition discount to aggregated values of each article but must apply different discounts depending on the volume of each individual order. To solve this, we have redesigned the previously developed algorithm to one that calculates the values of all constant articles and places the result in the correct node in the path tree. This is described in algorithm 3.2.2.

Algorithm 3.2.2: getValuesAndPathLevel(artAgg)

```
Input: Aggregated sales history for an article artAgg
Output: A tuple consisting of the values of the article and the path level in the
        path tree which the article's values should be added to.
conditionFound := false
pathLevelFound := false
art := article of artAgg
foreach node n from art to highest ancestor of art in article tree do
    if conditionFound and pathLevelFound then
        break loop
    end
    if n equals scaling node then
        pathLevel := scaling node level
        pathLevelFound := true
        conditionFound := true
        break loop
    end
    foreach node m in tree path do
        /* n is in tree path */
        if n equals m then
            pathLevel := level of m
            pathLevelFound := true
            break loop
        end
    end
    /* n has a scaling condition */
    if not conditionFound and conditions has an entry for n then
        values := scaling condition applied to the artAgg
        conditionFound := true
    end
end
/* Node lies under a price level 1 node outside of the path */
if not pathLevelFound then
    pathLevel := 0
end
/* No condition found; use values without any new discounts applied */
if not conditionFound then
    values := values from artAgg
end
return (values, pathLevel)
```

The algorithm on the previous page is used to calculate the values for all articles that do not depend on the scaling node's discount stair. This procedure has three cases to consider:

- The article is affected by the scaling node's discount stair
- The article is affected by an existing scaling condition
- The article's values do not depend on any scaling features

When an article inherits its discount stair from the scaling node, we ignore the article altogether and leave it for the prepare scaling method to handle. Otherwise, we need to apply the scaling condition by performing a separate simulation for the article using its discount stair. Finally, if the article does not depend on either the scaling simulation or a previous condition we use the data from the in-memory cache and add its values to the correct node.

3.2.3 Prepare scaling node

The scaling node preparation is a two step procedure. In the first step we create a cache of all order rows that are relevant for the simulation and in the second one we place the order rows in the correct discount step after the step volumes have been set by the user. The reason for creating the first cache is to speed up repeated simulations if the user decides to change the step volumes. If the first cache had not been created, all order rows would have to be retrieved from the database each time the step volumes were changed by the user. This would drastically decrease the system's overall performance in such cases.

Apart from simply storing the order rows as they are in the cache, we use the fact that different articles with the same volume will be placed in the same step in a simulation regardless of the volume steps. This means that all order rows with the same integer volume can be preaggregated in the cache. An example of this is shown in figure 3.6.

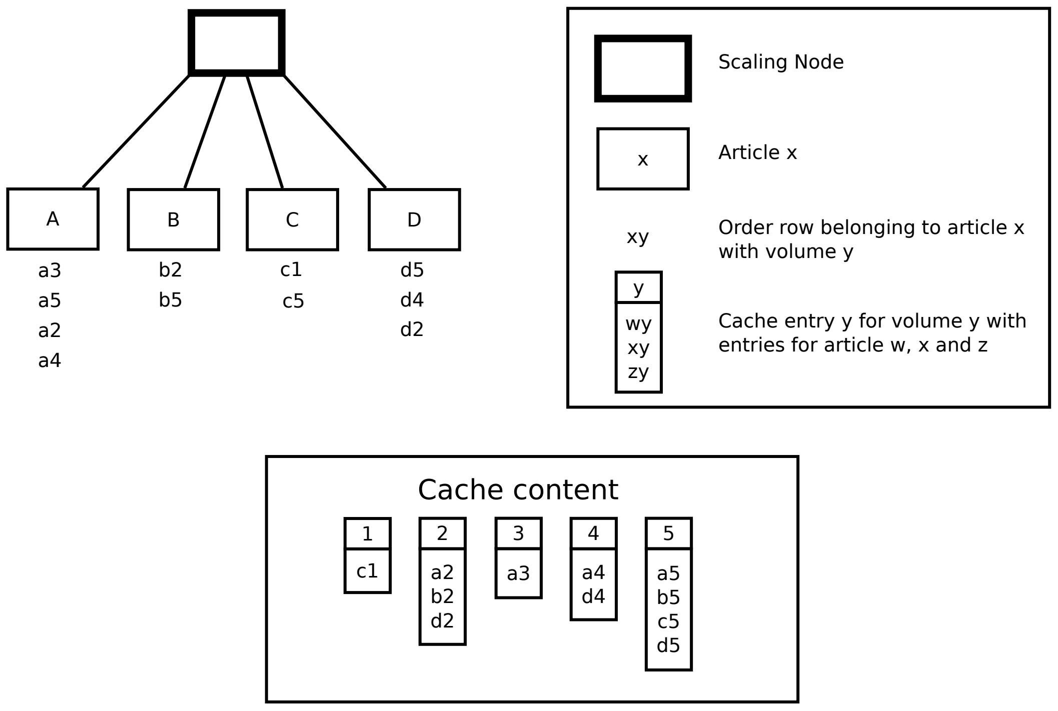

A
B
C
D
a3
a5
a2
a4
b2
b5
c1
c5
d5
d4
d2
Scaling Node
x
Article x
xy
Order row belonging to article x with volume y
y
wy
xy
zy
Cache entry y for volume y with entries for article w, x and z
Cache content
1
c1
2
a2
b2
d2
3
a3
4
a4
d4
5
a5
b5
c5
d5

Figure 3.6: Aggregation of order rows for the cache

After the user has chosen the step volumes, the cached order rows are placed in the correct discount step in the manner described in section 2.2.2. Assuming that the cache looks like the one described in the figure above and that the user sets the volume steps to three and five, all cached order rows with a volume less than three would be placed in the lowest step. Order rows with volumes three or four will be placed in the second step and finally, order rows with a volume of five will be placed in the last step.

In order to effectively perform this sorting procedure, we need a fast method for finding the correct volume interval for each order volume in the cache. Luckily, this is a well-known problem called one-dimensional range search [4] which can be solved in $\mathcal{O}(\log n)$ time for a set of n intervals using binary search.

The scaling preparation is performed according to the procedure described in algorithm 3.2.3.

Algorithm 3.2.3: Prepare scaling node

```
  Data: An empty scaling node sim
  Result: A scaling node ready for simulation
1 Create a cache rowCache of all relevant order rows
2 Let user define stepVolumes
3 Create a scaling step for each volume in stepVolumes
4 Create a scaling step for order rows with volumes less than $min(stepVolumes)$
5 foreach entry e in rowCache do
6     Place e in correct scaling step of sim using binary search
7 end
8 return sim
```

Once all of the order rows have been placed in the correct steps along with the discounts we are ready to perform the actual simulation.

3.2.4 Scaling simulation

The actual simulation of a scaling node works as described in section 2.2.3. For each step in the discount stair, the step discount is applied to the order rows and the result is aggregated to a result for the entire stair. The outline of the procedure is specified in algorithm 3.2.4.

Algorithm 3.2.4: Scaling node simulation

```
  Data: A discount stair ds with corresponding order rows
  Result: The values of all steps aggregated with the correct discounts applied
1 Initialize stairValues vector holding cost, value and volume
2 foreach step s in ds do
3     stepValues := Calculate simulated cost, value and volume of s by applying its
      discount
4     Aggregate values from stepValues into stairValues
5 end
6 return stairValues
```

After the simulation of the scaling node has been performed, the result is propagated upwards in the tree in a fashion similar to the one used in the customer discount model. The full procedure of the tree propagation is shown in figure 3.7. In this example, it is assumed that we are simulating on the article level.

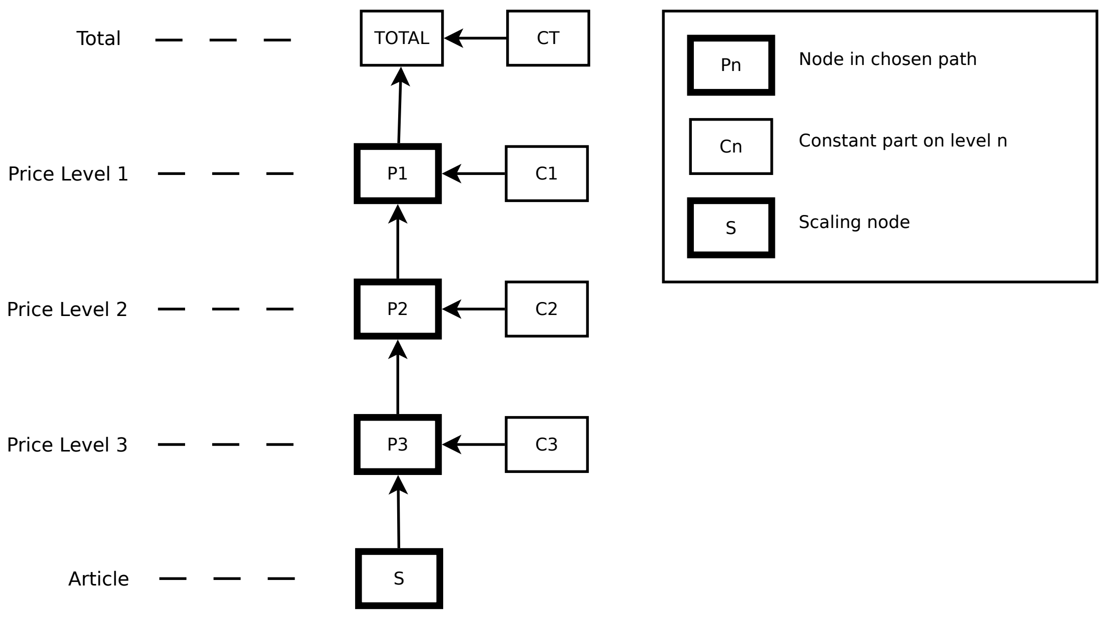


Figure 3.7: Overview of scaling simulation procedure

Just as in the discount model the simulation over the path tree is done in a bottom up fashion where the values of each child node is aggregated into the parent node according to the figure above. The scaling simulation can be summed up in the following formulas:

$$
\begin{array}{rcl|ll}
V_4 & = & V_S & V_i & \text{The values of path node } i \\
V_3 & = & C_3 + V_4 & V_S & \text{The values from the scaling node} \\
V_2 & = & C_2 + V_3 & C_i & \text{The constant values from node } i \\
V_1 & = & C_1 + V_2 & & \\
V_T & = & C_T + V_1 & &
\end{array}
$$

Figure 3.8: Scaling simulation formulas

The procedure used to perform a scaling simulation is shown in algorithm 3.2.5.

Algorithm 3.2.5: Scaling simulation procedure

```
Data: An aggregated article tree
Result: Calculates the values of all simulation path nodes
1 foreach node n in the simulation path from lowest to highest level do
2     if n is on lowest level in path then
3         n.values := result of scaling node simulation
4     else
5         calculate n.values based on n.constant and child
6     end
7     child := n
8 end
```

3.2.5 Database indexing

Since we can not get away from the fact that we need to continually access the database during a scaling simulation, some optimizations on the database are needed. An effective method for improving the performance of a relational database is to use an index, in order to lower the running time of lookups [5].

A database index makes it possible to find a set of rows in a database table sharing a set of common properties, without having to examine each row in the table. Since we are only interested in sales history data for a certain customer during a simulation, an index over the customers in the table is suitable. Every query to the database from our model is also limited to a specific article id. By creating an index over the tuple *(Customer ID, Article ID)* in the sales history table, it is possible to retrieve the set of all order rows with a given article ID and customer ID pair in significantly shorter time than before.

4

Results

This chapter contains tables and plots showing the performance of actual simulations run in the different systems described in earlier chapters. Since correct results are always required from both PCT and our implementations, the interesting part is the running time in different scenarios rather than accuracy or correctness.

The first section contains running times of customer discount simulations. Running times for our implementation are shown together with corresponding running times for PCT for the same underlying data.

In the second section, running times of our implementation for scaling simulations are shown. This feature is not yet supported by PCT, so this part does not have any reference values from the original system.

4.1 Customer discount results

This section contains running times for customer discount simulations. Every test case in this section has been run under the same conditions using both our implementation of our own model and PCT on the same server, in order to get comparable results.

Test case one - Live data

The first test case shows how the running time of the first simulation changes as the number of distinct articles in the customer's sales history grows. The "live" data used comes from PCT's real database, meaning that these simulations represent actual scenarios which sales representatives could encounter in PCT. The amount of distinct articles and corresponding running times are plotted in figure 4.1 and the actual values are shown in table 4.1.

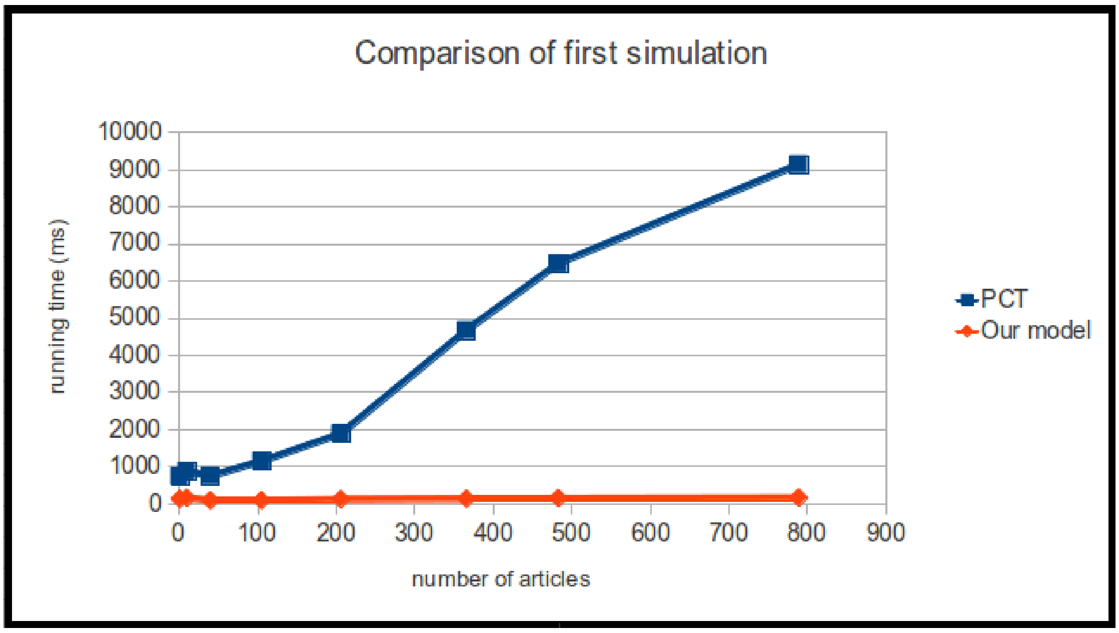


Figure 4.1: Comparison of running time for first simulation between PCT and our model

#articles	Running time PCT [ms]	Running time our model [ms]
1	723	130
10	876	156
40	741	89
105	1,142	91
206	1,879	118
366	4,671	131
483	6,473	148
789	9,141	161

Table 4.1: Running time for first simulation in PCT and our model

Test case 2 - Repeated simulations

The second test case is based on the same historical data as the previous one, but shows how running times change during three repeated simulations for each customer. Since repeated runs are very common in the simulation process, any improvements for repeated runs directly affect the time such processes take to complete. Performance changes for repeated runs in PCT only depend on database caching, while the improvements in our model depend on the fact that the preparation step has already been run. Running times for PCT are shown in figure 4.2 and corresponding times for our model are shown in figure 4.3.

It is worth mentioning that the Y axis magnitude differs a lot between the two plots. Exact running times can be seen in table 4.2.

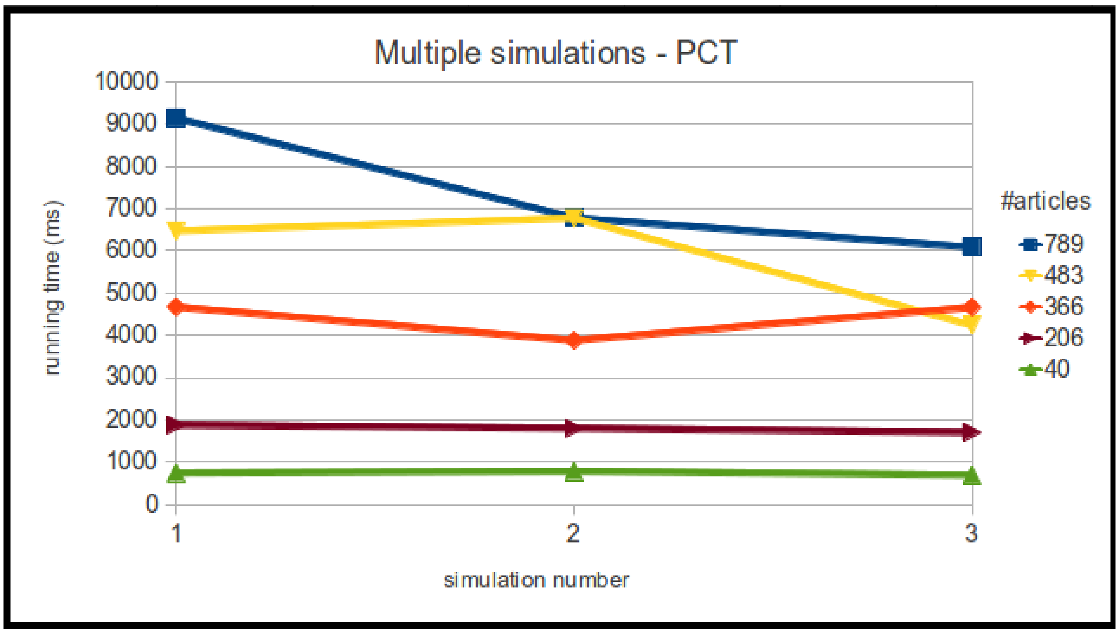


Figure 4.2: Running time for repeated simulations over the same datasets in PCT

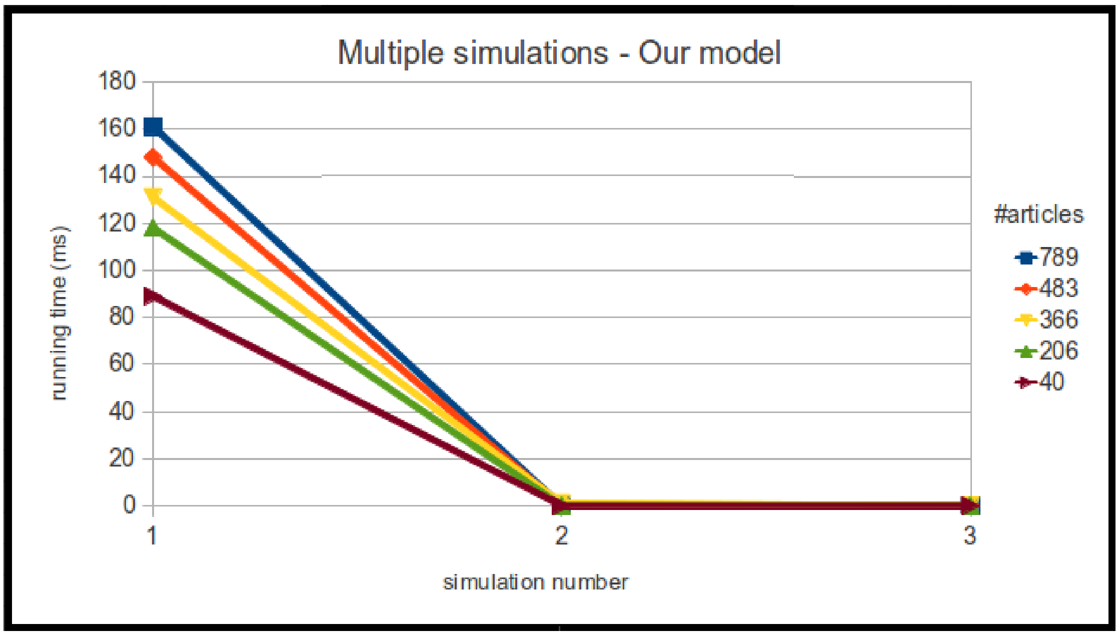


Figure 4.3: Running time for repeated simulations over the same datasets in our model

Running time [ms]	PCT			Our model		
#articles	**Run 1**	**Run 2**	**Run 3**	**Run 1**	**Run 2**	**Run 3**
40	741	782	692	89	< 1	< 1
206	1,879	1,801	1,707	118	< 1	< 1
366	4,671	3,879	4,665	131	< 1	< 1
483	6,473	6,773	4,240	148	< 1	< 1
789	9,141	6,780	6,087	161	< 1	< 1

Table 4.2: Running time for repeated simulations in PCT and our model

Test case three - Large data sets

In the third test case, running times of simulations over large sets of sales history entries for the same customer are measured. Due to the large amounts of order history required for this test, the database had to be filled with generated data. Thus, this test does not focus on the performance of realistic scenarios but rather on the performance of simulations over large data sets.

As can be seen in figure 4.4, the running time for PCT increases rapidly compared to the running time of our model. A separate plot which only shows the running time of our model (whose increase is impossible to see in the previously mentioned figure) is presented in figure 4.5. Exact running times are shown in table 4.3.

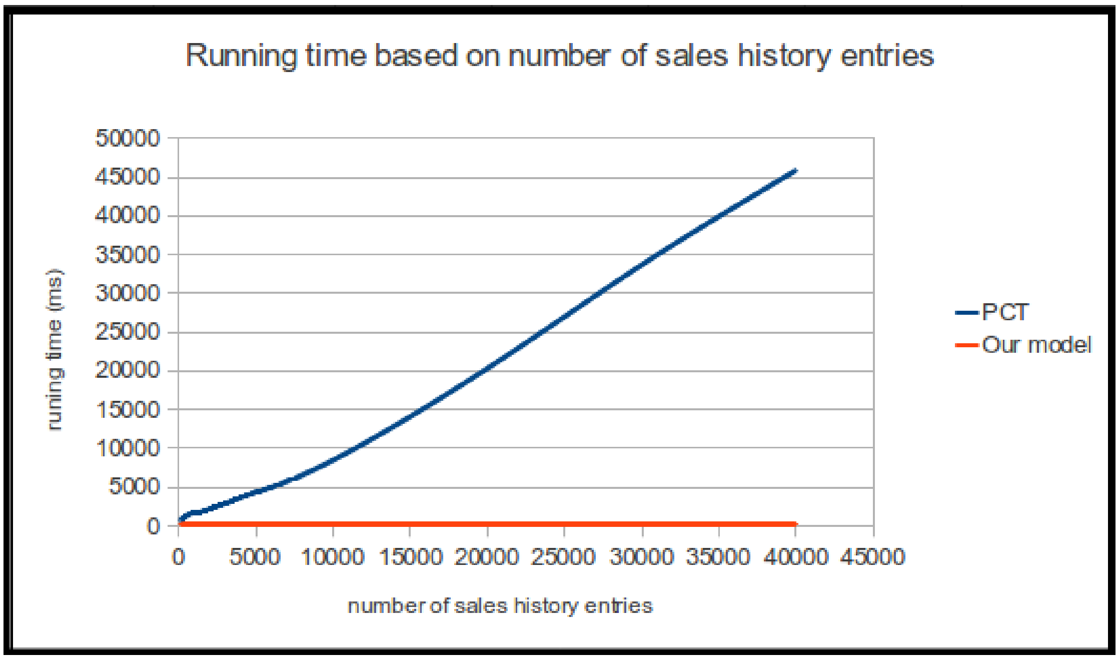


Figure 4.4: Comparison of running time for first simulation between PCT and our model

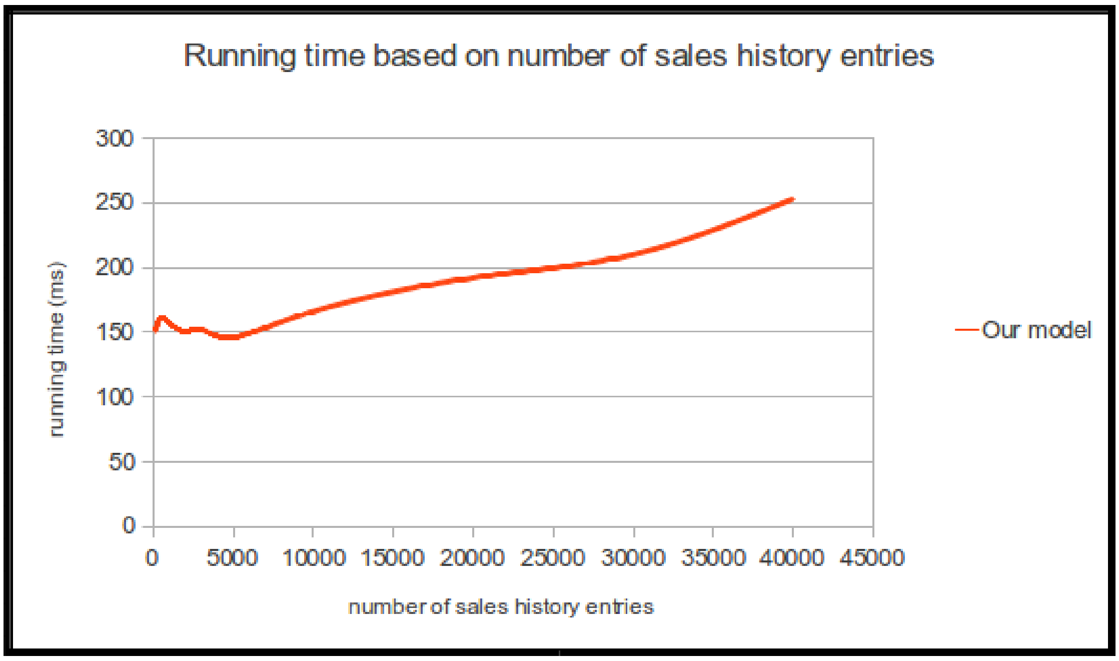


Figure 4.5: Running time for first simulation using our model

#articles	**Running time PCT [ms]**	**Running time our model [ms]**
100	736	150
500	1,295	161
1,000	1,694	157
1,500	1,835	153
2,000	2,161	150
3,000	2,884	152
5,000	4,335	146
10,000	8,463	166
20,000	20,314	192
30,000	33,671	210
40,000	45,892	253

Table 4.3: Running time for first simulation in PCT and our model over generated data

Test case four - Article distribution

In the fourth test case, a constant amount of 2,400 historical order rows is used (corresponding to 200 articles, which have all been sold at least once to the customer each month during the last year). However, the distribution of these articles has been changed between every run. In the first run, all articles belong to a price level 1 category outside of the path. In the second run, 25% of the articles lie under the price level 1 node in the path and the remaining 75% under other price level 1 nodes and so on.

It is common for customers to buy articles lying under different price level 1 nodes, so even though this data has been generated it shows how different realistic data distributions affect the simulation's running time. The results of this test are shown in figure 4.6 and the measured values can be seen in table 4.4.

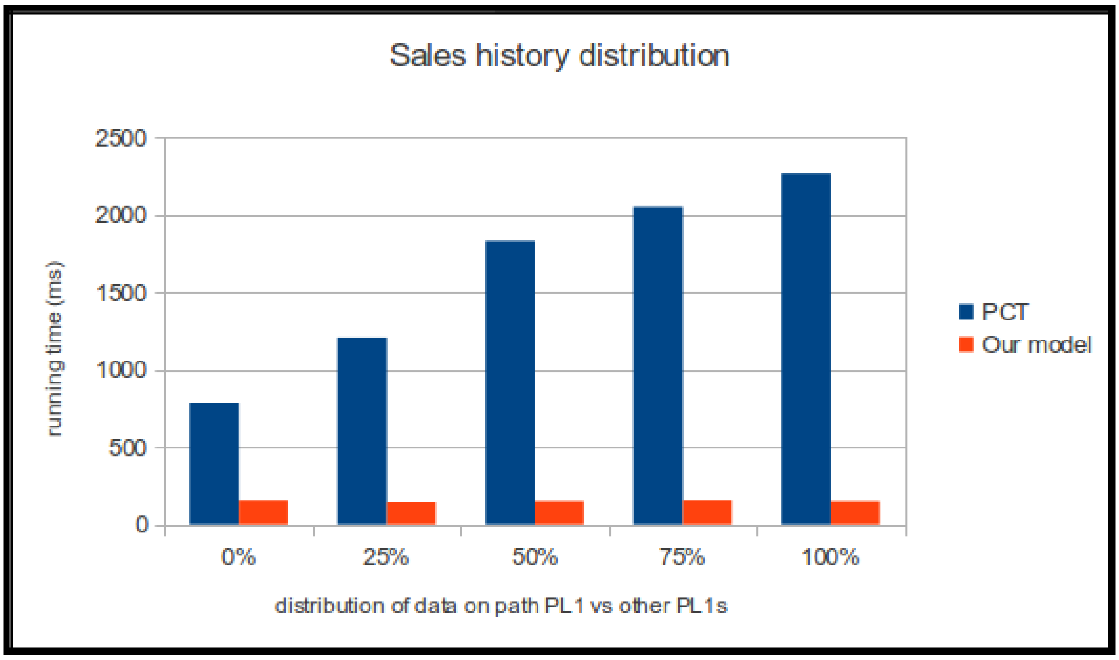


Figure 4.6: Comparison of running times depending on the proportion of sales history under the chosen price level 1 node

% of articles under PL1 node $\in$ path	**Running time PCT [ms]**	**Running time Our model [ms]**
0	786	156
25	1,206	144
50	1,830	151
75	2,051	157
100	2,265	150

Table 4.4: Running time for different history distributions in PCT and our model

Test case five - Conditions

The fifth test case shows how the amount of existing conditions in the database affect the simulation's running time. The data is generated so that the same sales history (covering 12 order rows each for 300 articles, for a total of 3,600 historical order rows) is used in each run. The only thing that changes is the amount of these articles which are covered by existing conditions. The exact values are shown in table 4.5.

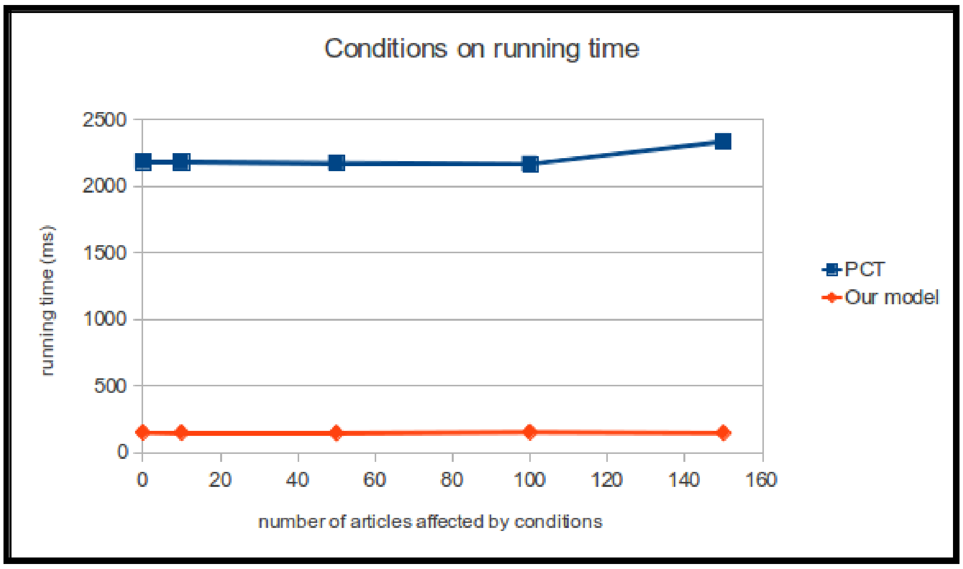


Figure 4.7: Comparison of running times where conditions affect the specified number of articles

#articles covered by conditions	Running time PCT [ms]	Running time Our model [ms]
0	2,181	150
10	2,178	148
50	2,172	145
100	2,162	152
150	2,333	146

Table 4.5: Running time depending on the amount of articles covered by existing conditions

4.2 Scaling results

Unlike the customer discount model where we could compare the results of our implementation to PCT, the scaling model does not have an existing implementation to compare our results to. Instead we have tested the implementation on a non-PCT server, which means that the running time of the scaling results are not directly comparable to the results in section 4.1.

Since no implementation of the scaling functionality exists in PCT there is no data in the database to test our implementation on. Subsequently, this means that all data used for the scaling tests has been generated specifically for this purpose.

Test case six - Large number of order rows

The sixth test case consists of simulations where the number of order rows for the customer increases. The test runs from 40 order rows up to and beyond the maximum number of 8,944 order rows for a simulation. The result of this test is shown in figure 4.8.

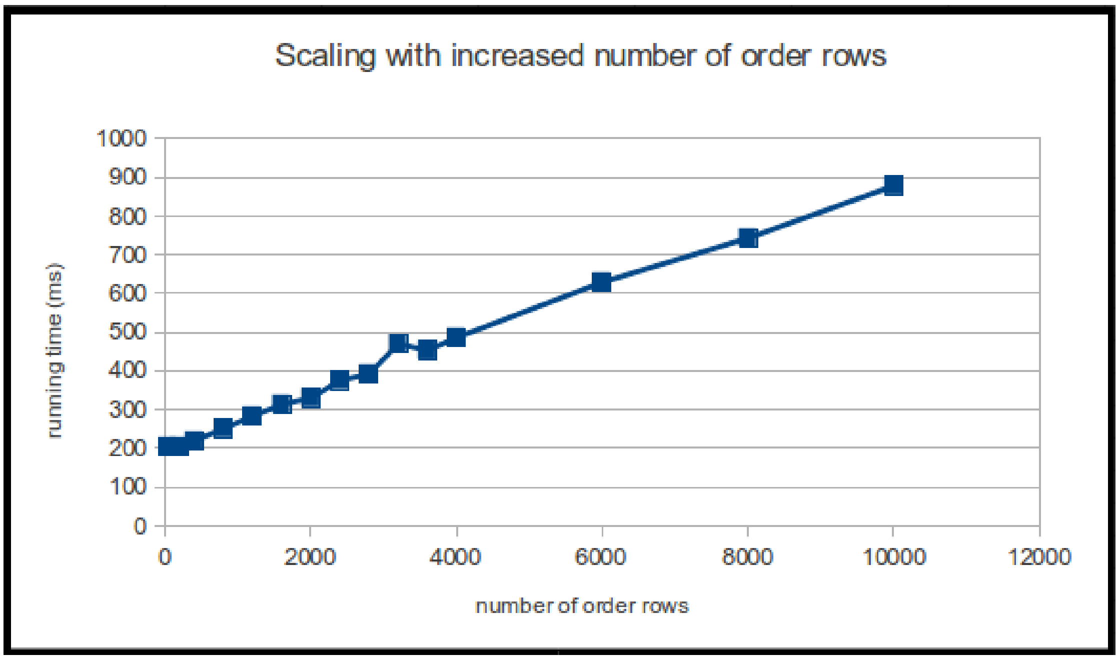


Figure 4.8: Running time for scaling simulations where the number of order rows increases

Test case seven - Sales history distribution

This test case shows how the running time of a simulation changes as the proportion of a customer's sales history affected by the scaling node's discount stair increases. Two hundred articles are used, with exactly twelve order rows per article for the customer.

In the first case, 0% of the customer's sales history is used in the scaling simulation, in the second case 25% and so on. Figure 4.9 shows the result of the test, where the blue part marks the constant calculations and the red part marks the scaling simulation. The data points used for the plot are displayed in table 4.6.

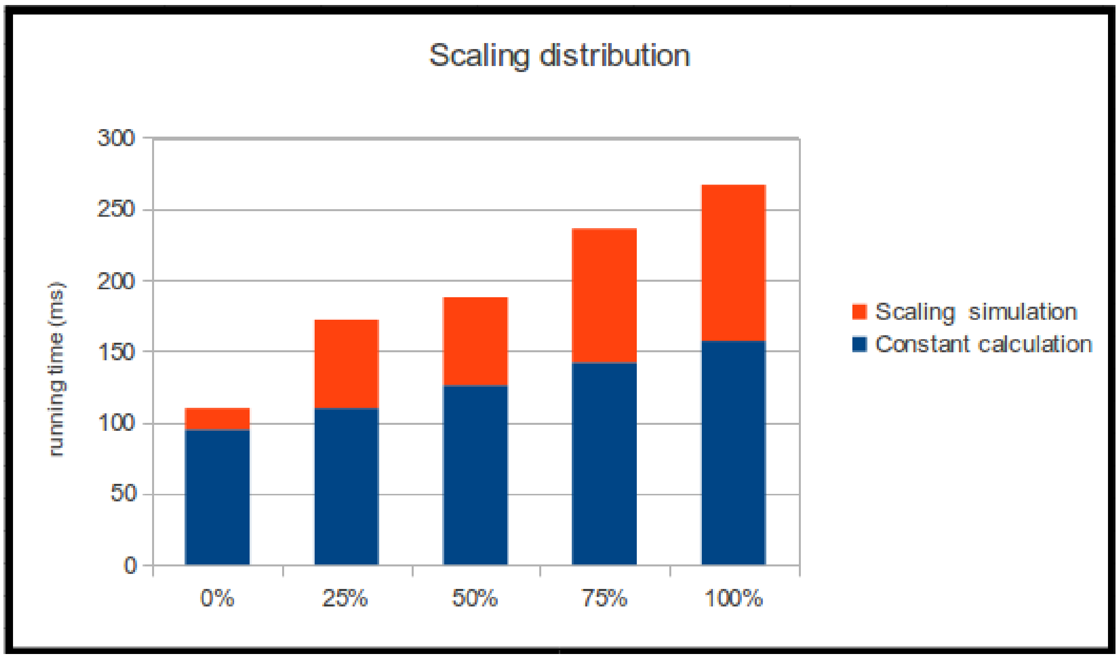


Figure 4.9: Running time of scaling simulations where the percentage of total sales history used for scaling increases

Distribution	**Running time Constant calculation [ms]**	**Running time Scaling simulation [ms]**
0%	95	15
25%	110	62
50%	126	62
75%	142	94
100%	157	110

Table 4.6: Running time for the constant and scaling parts of the simulation

Test case eight - Scaling conditions

The final test case investigates how the number of articles affected by conditions will influence the running time of a simulation. This test was run both prior to and after the introduction of the database index. The result of the test is shown in figure 4.10.

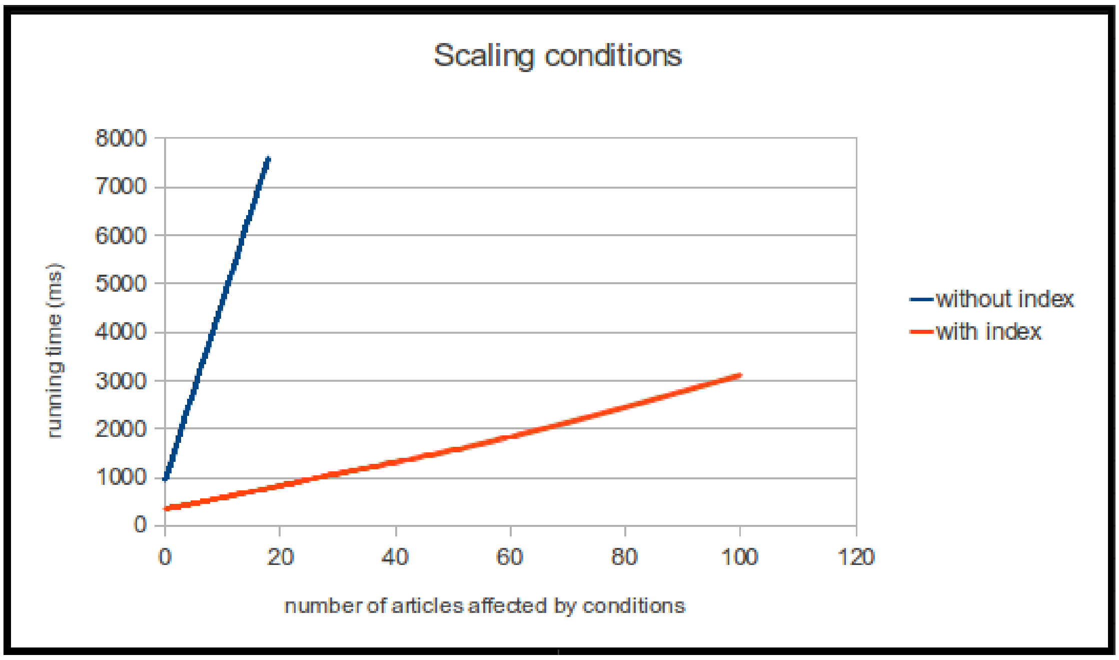


Figure 4.10: Correlation between the number of articles that are affected by conditions and running time, prior to and after the introduction of an index

5 Discussion

The purpose of this chapter is to analyse the project's results and to discuss how our solution can improve the simulation process in PCT.

The first two sections discuss the results from chapter 4 and how these correlate with our optimizations. This is followed by some ideas about possible future work and a summary of the conclusions from this project.

5.1 Customer discount simulation

As specified in section 2.1.4, the long running times of customer discount simulations in PCT are caused by three major problems. These are the current algorithm's high time complexity, its inefficient usage of database resources and the redundant calculations caused by PCT's inefficient model. This section aims to explain how these problems are handled in our model and how this can be seen in the results.

Time complexity

The improvements of the time complexity are easy to identify due to the complexity analysis of PCT in section 2.1.4 and of our model in section 3.1.3. PCT's complexity of $\mathcal{O}(ank^2)$ should be compared to our model's $\mathcal{O}(uk)$, where u is limited by a and $u \ll a$ in any realistic scenario. This indicates that the calculations in our model are performed in a far more efficient way.

The actual decrease of the running time caused by the reduced complexity is of course present in all simulations, but it is very easy to see in test cases one and three. In the latter of these, the running time for the largest data set using our model was roughly 99.4% lower than the one for PCT. If we compare the running times for the smallest data sets in test case one instead, we can see that the improvement is still 82.0%.

It is therefore reasonable to assume that simulations run using our model will in general be $> 80\%$ faster than the same simulations run using PCT and that the relative improvement will get even larger as the underlying simulation data grows.

Database usage

As described in section 2.1.4, PCT suffers from an ineffective method of retrieving data from the underlying database. The same data is often retrieved multiple times during a single simulation and loops are sometimes run over database rows instead of the cache entry for the customer. In the worst case scenario, PCT has to run $\mathcal{O}(ank^2)$ SQL queries.

When developing and implementing our model, we have made sure to avoid these problems by saving data which has to be used multiple times and by using values from the in-memory cache as often as possible. All active conditions for the chosen customer are retrieved using a single query and stored in a hash map using the condition nodes as keys. Any succeeding checks against the user's conditions can then be performed in $\mathcal{O}(1)$ time, without accessing the database at all. This means that the increased simulation time caused by increasing numbers of conditions are barely noticeable - something which is indeed reflected in the results of test case five.

Test case five also shows that the amount of conditions in PCT make a negligible impact on the running time; at least in realistic scenarios where customers have far less than a hundred conditions. The reason behind this is that other (about equally slow) calculations are performed instead for articles without existing conditions.

The removal of the article group nodes from the original model decreased both the article tree's height and the maximum path length from 5 down to 4. Because of this, iterations over these parameters run faster using our model even if the actual code would otherwise look identical.

Furthermore, our model never iterates over a list of articles retrieved directly from the database. Such loops are instead run over the key set of another hash map, whose keys represent only the articles which the customer has bought during the specified time period. This means that fewer database queries are needed, plus that unnecessary iterations are ignored which in turn improves our solution's complexity as well.

The total amount of database queries per simulation in our model is one for the conditions plus an additional query for every node in the path to retrieve their respective target discounts, for a total of $2-5$ queries. The difference between this low number and the $\mathcal{O}(ank^2)$ queries for the worst case scenario in PCT mentioned above is obviously very big.

Redundant calculations

PCT often calculates the exact same values multiple times during a simulation, as described in section 2.1.4. Furthermore, it does not reuse results between repeated runs in any way - it simply runs all calculations again (in the same redundant way as before).

The main idea of our model and its associated algorithms was to avoid calculating the same value more than once, as described in section 3.1.2. By calculating from the bottom up in the selected path, higher nodes can inherit their children's values directly instead of having to recalculate them.

The removal of redundant article value calculations reduces the complexity of the first simulation (or rather, the preparation step) for any data set. It is thereby very closely connected to test cases one and three, as mentioned in the time complexity section above. Test case four shows how our model behaves in a scenario where an increasing quota of articles is moved in under the price level 1 node in the path, thus increasing the redundancy of PCT's calculations. As expected, our running time remains more or less constant regardless of this distribution.

Since repeated simulations are only carried out over the path tree in our model, repeated simulations always take less than one millisecond to perform. This can be seen in test case two.

5.2 Scaling simulation

The PCT developers originally deemed the scaling functionality presented in section 2.2 too complex to be implemented with a reasonable running time.

The main issue that the scaling model suffered from was that scaling requires the use of each individual order row instead of monthly aggregated sales history. Because of this, it was impossible to use the in-memory cache solution from the customer discount model with the scaling extension for two reasons. The first was that the data size of all order rows was greater than the cache can hold. The second was that it should be possible to switch between the two simulation types, so the in-memory cache needed to remain in its existing state. This meant that instead of reading in all of the sales history once and storing it in a cache, the database had to be accessed each time that sales data was needed.

There was also the issue of having reasonable running times of the system. Research has shown that the general threshold for the time that a user is willing to wait for feedback from a system is ten seconds [2, 3]. Therefore, a running time lower than this is required for any implementation.

Since there is no existing implementation of the scaling functionality, each test will use the reasonable response time as a reference point to assess the quality of our implementation.

Test case six shows that the running time of scaling simulations with as many as 10,000 order rows does not exceed 900 ms. Since a scaling simulation can only be performed at the article level or price level 3, the maximum number of order rows that can be used in a simulation is 8,944. This number comes from the maximum number of articles under a single price level 3 node and the maximum number of order rows for a single article, as explained in section 2.2.5. This means that a scaling simulation for a customer with maximum sales history for a greater number of articles than the specification requires can be run in a reasonable time.

Test case seven shows how the running time of the simulation will change, based on the distribution of sales history for articles that depend on the scaling node. When the percentage of sales history affected by the scaling node's discount stair increases, so does the total running time of the simulation. In order to keep the running times low, there are no existing conditions on any of the articles.

Test case eight shows how the running time will be affected by the number of articles that have scaling conditions. To show the actual impact of indexing on the running time, this test was done both with and without indexes on the database tables.

In figure 4.10 the blue graph illustrates the results of the unindexed tests and the red graph shows the same tests with indexes. When there are no indexes, we can see that a simulation with no affected articles takes almost twice as long time to run than it does with indexes and the running time for each added article grows faster than the running time of the same test with an indexed database. We also note that at 100 affected articles, the test with database indexes still runs in a reasonable time, whereas the unindexed tests will hit the reasonable time limit at approximately 10 articles.

This is an adequate result, since the specification from section 2.2 states that a customer should have a maximum of 10 existing scaling conditions. Depending on whether the conditions are set for article nodes or price level 3 nodes, these will on average affect 10 to 123 articles.

These tests show that our model sufficiently solves the scaling problem in a reasonable running time. In fact, they even demonstrate that the implementation will often perform beyond the scope of the specification.

5.3 Future work

The optimized models described in this report are all implemented in Java, just like the original PCT system. Further studies of performance gain for implementations written in other programming languages would be interesting - perhaps a functional programming language would be able to run simulations even faster?

Another interesting approach would be a comparison between the performance of our models using different database solutions. NoSQL database systems could prove effective in handling the big data problems introduced by the scaling extension. Particularly, an implementation using a graph database would be interesting due to this technology's great performance when dealing with tree structures. For example, the graph database Neo4j has shown promising results in multiple studies such as [6], where Neo4j is concluded to be up to ten times faster than MySQL for traversals and [7], where the results show that running times for MySQL increase much faster than for Neo4j as the data magnitude grows.

5.4 Conclusions

This project has consisted of analysis, optimization and implementation of the existing simulation algorithms in PCT as well as modelling and implementation of its upcoming scaling extension. The results show that the implementation of the optimized customer discount model provides large enough performance improvements to guarantee reasonable running times even for the largest customers. The results for the scaling extension prove that implementation of the desired functionality in PCT is possible as well, as long as the big data issue is handled in an efficient way.

A final conclusion of this project is that optimization of existing algorithms is not always sufficient in order to improve the performance of a system. Creating new, optimized models and developing fast algorithms for these can prove far more efficient than optimization of existing algorithms based on improficient models.

Bibliography

[1] A. Jacobs, The pathologies of big data, Commun. ACM Vol. 52 (8) (2009) pp. 36–44.

[2] R. B. Miller, Response time in man-computer conversational transactions, in: Proceedings of the December 9-11, 1968, fall joint computer conference, part I, AFIPS '68 (Fall, part I), New York, NY, USA, 1968, pp. 267–277.

[3] S. C. Seow, User and system response times, in: Designing and Engineering Time: The Psychology of Time Perception in Software, Addison-Wesley Professional, 2008, pp. 33–48.

[4] M. de Berg, O. Cheong, M. van Kreveld, M. Overmars, 1-dimensional range searching, in: Computational Geometry: Algorithms and Application 2ed, Springer Berlin Heidelberg, 2008, pp. 96–99.

[5] Y. Manolopoulos, Y. Theodoridis, V. J. Tsotras, Advanced Database Indexing, Springer US, 2000, Ch. 3. Fundamental Access Methods, pp. 37–59.

[6] C. Vicknair, M. Macias, Z. Zhao, X. Nan, Y. Chen, D. Wilkins, A comparison of a graph database and a relational database: a data provenance perspective, in: Proceedings of the 48th Annual Southeast Regional Conference, ACM SE '10, ACM, New York, NY, USA, 2010, pp. 42:1–42:6.

[7] S. Batra, C. Tyagi, Comparative analysis of relational and graph databases, International Journal of Soft Computing and Engineering (IJSCE) Volume 2 (Issue 2) (2012) pp. 509–512.