Semi-automated scenario analysis of optimisation models

Jani Strandberg

School of Science

Thesis submitted for examination for the degree of Master of Science in Technology.

Helsinki 1.8.2019

Supervisor

Prof. Antti Punkka

Advisor

D.Sc. Anssi Käki

Aalto University
School of Science

Aalto University
School of Science

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi
Abstract of the master's thesis

Author Jani Strandberg		
Title Semi-automated scenario analysis of optimisation models		
Degree programme Mathematics and Operations Research		
Major Systems and Operations Research		**Code of major** SCI3055
Supervisor Prof. Antti Punkka		
Advisor D.Sc. Anssi Käki		
Date 1.8.2019	**Number of pages** 71+7	**Language** English

Abstract

Mathematical optimisation models have been succesfully used for solving problems across multiple industries. Often, the purpose of these models is to provide decision support by guiding the development of effective plans and decisions. In the process of providing decision support, obtaining the numerical solution of an optimisation model is usually not limiting, and increasingly large and complex models are solvable as a result of increased computation resources and improved solution algorithms. However, analysing and understanding the model results can be difficult for large and complex problems. Analysing model results often involves processing one or several model scenarios with varying parameters, so that related conclusions become available to the user. This part of the mathematical programming process is usually done by the analyst on a case-by-case basis, but this could be aided through computer assisted tools known as Intelligent Mathematical Programming Systems (IMPS).

In this Thesis, we develop a method for analysing and comparing the results of optimisation scenarios. This method forms a basis for a new form of IMPS that can be used to analyse both individual model scenarios and differences between two scenarios. The method is based on the idea of preserving the structure of the mathematical model along with the optimisation results, by representing the results as a graph. While the use of the method is considered in the context of supply chains and linear programming, the approach is fairly general and could be applied in other types of optimisation problems as well. The implementation of the IMPS is done with open-source technologies and can be coupled with any modelling environment and solver.

The usability of the developed method for scenario analysis is evaluated through a case study related to an existing supply chain optimisation model at a large pulp and paper company. We identify some scenario related questions where the developed method has advantages over the traditional approaches where the model structure is not explicitly preserved. The case study illustrates that the developed method has many potential use cases, especially in conjunction with other methods. Furthermore, multiple development needs and avenues for further study are identified.

Keywords Optimisation, mathematical programming, decision support system, model analysis system, scenario, supply chain, graph

Aalto University
School of Science

Aalto-yliopisto, PL 11000, 00076 AALTO
www.aalto.fi
Diplomityön tiivistelmä

Tekijä Jani Strandberg

Työn nimi Semi-automatisoitu optimointimallien skenaarioanalyysi

Koulutusohjelma Matematiikan ja operaatiotutkimuksen maisteriohjelma

Pääaine Systems and Operations Research **Pääaineen koodi** SCI3055

Työn valvoja Prof. Antti Punkka

Työn ohjaaja TkT Anssi Käki

Päivämäärä 1.8.2019 **Sivumäärä** 71+7 **Kieli** Englanti

Tiivistelmä

Matemaattisia optimointimalleja on käytetty menestyksekkäästi ongelmanratkaisuun monilla eri teollisuuden aloilla. Usein optimointimallien tarkoitus on tarjota päätöksenteon tukea ohjaamalla tehokkaiden suunnitelmien ja päätöksen tekemistä. Tässä päätöksenteon tukiprosessissa optimointimallin numeerisen ratkaisun saavuttaminen ei ole useimmiten rajoittava tekijä, ja yhä laajempia sekä monimutkaisempia malleja voidaan ratkaista lisääntyneiden laskentaresurssien sekä kehittyneiden ratkaisualgoritmien avulla. Sen sijaan mallin tulosten analysointi ja ymmärtäminen voi olla vaikeaa laajojen ja monimutkaisten ongelmien tapauksessa. Mallin tulosten analysointiin sisältyy usein yhden tai useamman mallin skenaarion prosessointi siten, että relevantit johtopäätökset tuloksista tulevat käyttäjän saataville. Tämä osa matemaattisesta mallinnusprosessista tehdään usein tapauskohtaisesti analyytikon toimesta, mutta tätä voidaan auttaa tietokoneavusteisilla työkaluilla.

Tässä diplomityössä kehitetään menetelmä optimointimallien skenaarioiden analysointiin ja vertailemiseen. Tämä menetelmä luo perustan uudenlaiselle aputyökalulle, jota voidaan käyttää sekä yksittäisten mallin skenaarioiden että kahden skenaarion välisten erojen analysointiin. Menetelmä pohjautuu matemaattisen mallin rakenteen säilyttämiseen optimiratkaisun lisäksi, mikä tapahtuu tallentamalla tulokset graafimuodossa. Vaikka menetelmän käyttöä tarkastellaan lähinnä toimitusketjujen ja lineaarisen ohjelmoinnin näkökulmasta, lähestymistapa on melko yleinen ja voi soveltua myös muihin optimointiongelmiin. Kehitetyn aputyökalun pohjana on käytetty avoimen lähdekoodin teknologioita, ja se voidaan yhdistää mihin tahansa mallinnusympäristöön ja ratkaisijaan.

Kehitetyn menetelmän käytettävyyttä skenaarioanalyysiin arvioidaan erään toimitusketjun optimointimalliin liittyvän tapaustutkimuksen avulla suuressa paperiteollisuuden yrityksessä. Työssä tunnistetaan skenaarioihin liittyviä kysymyksiä, joissa kehitetyllä menetelmällä on etuja verrattuna lähestymistapoihin, joissa mallin rakennetta ei eksplisiittisesti säilytetä. Tutkimus havainnollistaa menetelmän mahdollisia käyttökohteita, erityisesti muihin menetelmiin yhdistettynä. Lisäksi työssä tunnistetaan menetelmään liittyviä kehitystarpeita ja jatkokehityksen suuntia.

Avainsanat Optimointi, matemaattinen ohjelmointi, päätöksenteon tukijärjestelmä, mallianalyysijärjestelmä, skenaario, toimitusketju, graafi

Preface

This thesis was written for UPM Advanced Analytics. First, I want to thank my instructor Anssi Käki for his invaluable guidance, support, and giving me the opportunity to work at UPM. This thesis would have been impossible without his encouragement and expertise.

I also want to thank the other members of the Advanced Analytics team, Jesse and Joona, who provided valuable feedback and ideas throughout the creation of this Thesis. I feel very privileged for having the opportunity of working with such a talented group of people.

I am also thankful for my supervisor Antti Punkka, for his valuable comments and constructive feedback. His input had a very positive impact on the quality of this work. Thanks also to other faculty at the Aalto University who have provided me with such valuable education.

Last but not least, I wish to express my deep gratitude for my family. I want to thank my parents Mika and Susanna, who have always supported me in everything I have chosen to pursue. Thanks to my sister Jenny for showing me that it's okay to chase your dreams. Finally, special thanks to my partner Aino, who has been a source of inspiration and joy throughout my university studies.

Helsinki, 1.8.2019

Jani Strandberg

Contents

Symbols and abbreviations

Symbols

A	matrix A
A_{ij}	matrix entry in ith row and jth column
$G = (V, E)$	Graph G with node set V and edge set E
$G = (U, V, E)$	Bipartite graph G with disjoint node sets U and V, and edge set E

Operators

$\mathbf{x}^{\mathbb{T}}$	transpose of vector $\mathbf{x}$
$A \cap B$	interesection of sets A and B
$A \cup B$	union of sets A and B
$A \setminus B$	difference of sets A and B

Abbreviations

IMPS	Intelligent Mathematical Programming System
LP	Linear Programming
MILP	Mixed Integer Linear Programming
APS	Advanced Planning System
DMAS	Deductive Model Analysis System
IMAS	Inductive Model Analysis System
DSS	Decision Support System

1 Introduction

1.1 Background

Since the development of mathematical programming, the ability to solve larger and more complex optimisation problems has significantly improved as a result of better solution algorithms and abundant computing power. The numerical computation of a solution is no longer the bottleneck in most mathematical programming processes. Rather, the tasks related to creating, analysing, understanding and communicating these models and their various instances have an increasing role in applications.

While the technical purpose of mathematical programming is to find the optimal value of a given objective function subject to given constraints, in practice this is done to provide some form of decision support by guiding the development of effective plans and decisions. Thus, mathematical models are sometimes embedded into a larger Decision Support System (DSS). In addition to providing decision recommendations, these systems are used to validate whether or not the model is an accurate enough representation of the real underlying system, as well as to establish credibility of the model in the eyes of the often non-technical decision makers with rich reporting and visualisation of the results, for instance.

At the core of both providing sound decision recommendations and establishing model credibility is the process of developing fundamental insights into why the model solution is what it is. This extends far beyond the question of what particular numerical values the decision variables of the optimal solution are. Such insights are important in many applications of mathematical programming, where there seldom exists a single perfect model solution that fully encapsulates the underlying model uncertainties and directly translates into actionable plans.

To enhance model understanding, it is often beneficial to consider several model runs with varying parameters and structures, reflecting the alternative assumptions, objectives and uncertainties regarding the mathematical model. These model instances reflecting various possible realities are often referred to as scenarios (Rönnqvist et al., 2015). The term scenario analysis refers to how these model scenarios are organised and processed so that important conclusions are available to the model user (Greenberg and Chinneck, 1999). Depending on the case, the decision maker may be interested in one particular scenario, or they may be interested in knowing what are the key differences between two, or even multiple scenarios.

In terms of solving the mathematical model to optimality, there is usually a somewhat clear path to victory: Assuming that the model is not ill-structured, one only needs to have an appropriate solver that is capable of solving the problem, and ensure adequate computing resources. However, there is no standard procedure for distilling key insights from the mathematical model, as these are arguably more difficult to generalise or even define in exact terms which are often preferred by mathematically inclined analysts. This is not to say that no quantitative tools exist: For example, most commercial mathematical programming environments are able to perform elementary sensitivity analysis, enabling the analyst to study how small, individual parameter changes affect the model solution. However, this type of analysis can only answer a fraction of the questions that may be relevant to the decision maker.

The process of gathering insights from models and their scenarios is usually done on a case-by-case basis, since there exists no industry standard tool that would aid the analyst in this regard. Furthermore, this process often falls to the analyst responsible for creating and programming the model, as the ability to gather insights often requires understanding the model structure. Clearly, this part of the mathematical programming process could benefit from automated assistance, where some computer program would provide the analyst with ready-made tools to analyse and compare model scenarios. Such systems equipped with computer-assisted model analysis are referred to in the literature as Intelligent Mathematical Programming Systems (IMPS) (Greenberg and Chinneck, 1999). Although numerous systems have been developed for general model analysis, few of these have focused on the aspect of scenario analysis, and computer-assisted tools for scenario analysis reported in literature are virtually nonexistent.

1.2 Thesis objective and scope

This thesis develops a new form of IMPS for analysis purposes in practical applications at an industrial case company. The IMPS is based on ideas previously presented by Greenberg (1983), where optimisation models are modelled as graphs. In this thesis, we extend this notion to the two-scenario case, which allows the comparison of two individual model scenarios. The usefulness of the developed IMPS is validated in a participatory development process with optimisation experts at the case company, where several questions pertaining to the case company's optimisation models were collected and categorised.

Although the problem of gathering insights from models is a very general one, we mostly limit our considerations to optimisation models related to supply chain planning. This is because such models are at the heart of the case company's business, and focusing on a slightly more specific model category is a more approachable problem than trying to build a system that would fit to any type of mathematical model.

In terms of mathematical models, this thesis is limited to Linear Programming (LP) models, as they are the main workhorse at the case company, simpler to analyse than e.g. nonlinear or mixed-integer-linear programming (MILP) models, and the MPS file format, which is a core input of the implemented IMPS, supports only LP models and MILP models. The aforementioned considerations are not to say that the approach suggested in this Thesis would not work for other types of models: Even though we have limited our attention to a more specific level, the same ideas are applicable to other model types as well.

1.3 Thesis structure

The remainder of this thesis is organised as follows. Chapter 2 introduces the background by providing a review of the existing literature regarding automated model analysis and supply chain optimisation. We also review linear programming to provide further context.

Chapter 3 describes some of the common questions related to supply chain optimisation analysis, collected from discussions with multiple optimisation and subject experts. We categorise the questions into different types, and assess for each type the amount of scenarios involved and the difficulty of answering these questions. We also explain and characterise different types of model changes related to these questions.

Chapter 4 describes different general approaches for building an IMPS and explains the foundation behind the approach taken in this Thesis. We present an overview of the developed IMPS, describing its main inputs, outputs and functionalities. In addition, we describe the practical requirements, design choices and the main algorithms used in the IMPS in detail.

Chapter 5 describes the use of the IMPS by performing a case study related to an existing paper supply chain optimisation model at the case company. We illustrate through multiple examples how the IMPS can be used to review and analyse different scenarios and model results.

Finally, Chapter 6 summarises the thesis, provides further development needs and presents the most important conclusions drawn from the entire body of work.

2 Background

2.1 Mathematical programming process

The mathematical programming process can be simplified into the form shown in Figure 1. First, the problem domain is described to include only the objects that are relevant to solving the problem at hand. Then, a mathematical model is extracted from the problem domain by defining the relevant decision variables, constraints and objectives of the problem. In addition, the model often includes parameters, some of which may be uncertain. Thus, several instances of the model, differing in terms of the data used, can be solved. The results for these various instances can then be collected as case studies. Unexpected results or other questions arisen from this case study may then lead to re-examining the problem domain, or modifying the mathematical model. Therefore, the entire process forms a closed loop, as depicted in Figure 1. The mathematical programming process can be subdivided further into seven main functions, based on the taxonomy presented by Greenberg and Chinneck (1999).

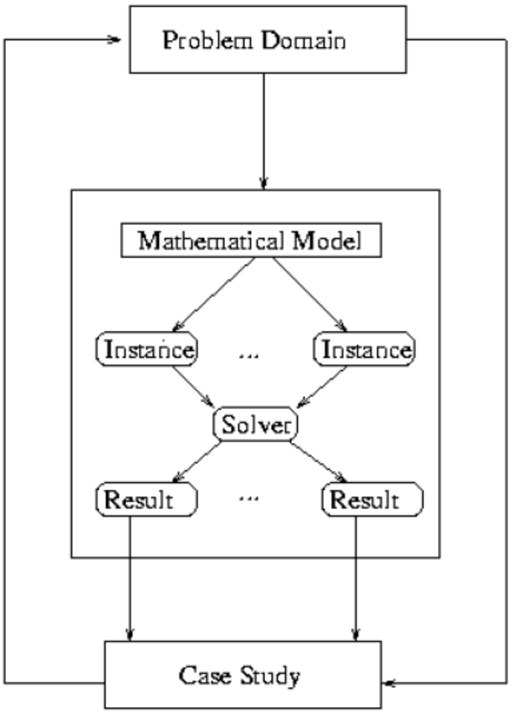


Figure 1: The mathematical programming process (Greenberg and Chinneck, 1999).

1. **Model expression** is related to the various forms in which the mathematical model can be created. Such forms include algebraic modelling languages, spreadsheets, block-schematics, and other representations. While the appropriate form of expression may depend on the problem, it should ideally enable the user to detect possible

errors and omissions. Two important functions related to model expression are documentation and verification. Documentation entails recording a description of the model, such that people other than the original creator may understand it. Verification, in turn, is related to ensuring that the computer-resident model coincides with our functional description.

2. **Model viewing** refers to the forms in which the model and its solutions can be viewed to the user. These views may employ the same structure as the original model expression, although other views based on graphical or natural language representations are possible as well. As is the case in model expression, there is often no "best" view, as this may depend on both the preferences of the modeler, and the problem itself. A related function is reporting, which may include functionalities such as interactive query, the generation of some internal files for further analysis, or creating some other form of report for presentation.

3. **Model simplification** refers to various methods of finding simpler model expressions that still capture the essence of the original problem. Some form of simplification functionality is often included in commercial-quality solvers, but their main purpose is usually to increase optimisation performance, rather than deliver new insights into the problem.

4. **Debugging** is the process of finding and removing the possible mechanical errors in the model. Such errors may lead to problems such as infeasibility, unboundedness or non-viability. Note, however, that the viability of the model as a representation of reality cannot usually be answered by debugging alone.

5. **Data management** refers to the structures and techniques applied to manage the various information that is generated and collected in the mathematical programming process. These structures may have varying complexity, ranging from simple spreadsheets to large databases.

6. **Scenario analysis** refers to various methods of processing and filtering the information generated from multiple model instances, representing various scenarios. An important application is cross-scenario analysis, where the purpose is to find and ideally also explain the main differences between scenarios. It is worth noting that Greenberg and Chinneck (1999) uses the term scenario management to refer to this function. However, this term is also used to describe how and why different types of scenarios are created in the problem context. In the scope of this thesis, we consider the scenarios as given by subject experts, and assume that these scenarios appropriately capture the significant uncertainties related to the problem. Therefore, our main focus is in the function of how different scenarios are analysed and compared, rather than how these scenarios were created. The field of scenario management related to the generation of various scenarios is an interesting field of study in itself, see e.g. Seeve (2018).

7. **General analysis** refers to other general questions one might have about the

mathematical model, unanswered by the previous functions. These include, for example, finding important relationships in the data, and finding the root cause of the price of a certain variable. General analysis tools provide the user with means to explore the model structure in further detail, thus helping the user answer some model-related questions. Related functions include validation and redundancy analysis. The purpose of validation is to determine how well our model represents reality, given the context of decision support that the model is designed to offer. Redundancy analysis, as the name suggests, is related to finding possible redundancies in the model.

Based on this taxonomy, functions 6 and 7 are the main focus of this Thesis. However, it is important to note that these functions are not separate entities in the mathematical programming process, but they are rather intertwined. For example, different model views can be beneficial in general analysis.

2.2 Analysis of supply chains

With the growth of Enterprise Resource Planning (ERP) applications in organisations, supply chain management has become a major area for model-driven decision support systems and mathematical programming (Power and Sharda, 2007). Supply chains are complex entities involving multiple phases, each involving their own planning decisions. Many of these planning problems can be translated into mathematical models, provided that there is sufficient data available to model the supply chain adequately. As ERP systems have made this data more accessible to analysts, the use of mathematical programming has seen increasing use in many supply chain activities, including the planning of logistics, production and demand (Power and Sharda, 2007).

Although supply chains have no single definition in the literature, they can be seen as systems of organisations, people and activities that are involved in moving a product from supplier to customer (Press, 2011). Typical activities in supply chains involve the transformation of resources, such as energy and raw materials, into finished products, transporting those products, and finally distributing them to customers. Between these end points, there may be multiple value adding activities that in combination are necessary in order to deliver the final product (Janvier-James, 2012).

Common entities in the supply chain include suppliers, manufacturers, warehouses, transportation companies, distribution centers and retailers. A supply chain can be represented as a network between these entities. The nodes of such a network may depict these different entities, such as suppliers and production facilities, and links between the nodes depict the material flow through the network, e.g. the procurement of raw materials from suppliers to the production facilities. Figure 2 shows one simple example of a supply chain ,which shows how materials flow through a supply chain network through suppliers to end customers.

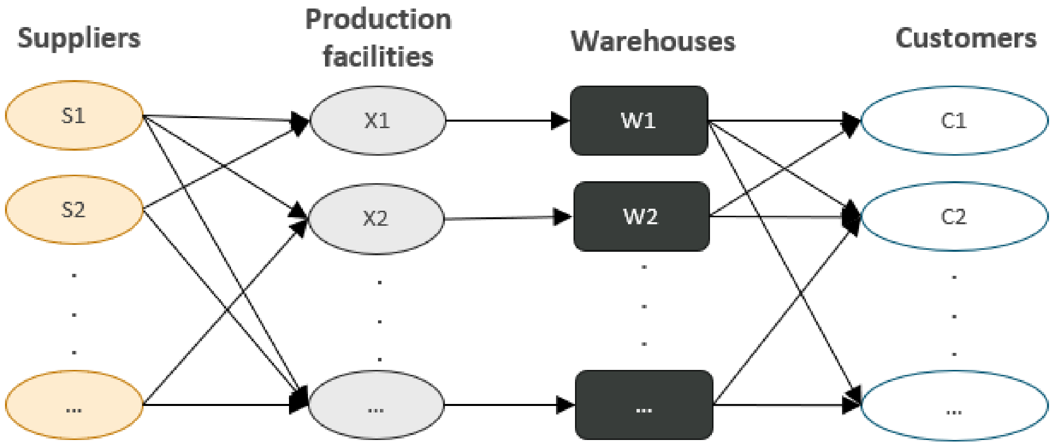


Figure 2: Example of a supply chain.

Supply chain management is the task of integrating the organisational units in the supply chain and coordinating the flows of materials, finances and information such that the ultimate customer demands are fulfilled, and the competitiveness of the supply chain is improved as a whole (Stadtler, 2005).

The ultimate goal of acheiving and maintaining the competitiveness of the supply chain is in turn achieved by directing the supply chain in a strategically viable and sustainable position compared to its competitors. Customer satisfaction is often an important means to achieve this aim (Trkman et al., 2005). The degree of success in achieving these goals depends on two major factors: the integration of organisational units and the coordination of flows (Weintraub et al., 2008). These factors may be further divided into smaller building blocks which further detail the activities necessary for success. The building block that is particular interesting in the context of this thesis is advanced planning. Systems which incorporate mathematical models and solution algorithms are referred to as Advanced Planning Systems (APS) in supply chain management, and they can be viewed as decision support systems that often utilise mathematical programming (Stadtler, 2005). Thus, the benefit of using these systems as decision support tools would be greatly increased if they provided assistance in the main functions related to the mathematical programming process.

Although APS are often proprietary software developed by various vendors, they commonly exhibit a hierarchical architecture, since the planning itself is usually a hierarchical process (Rönnqvist et al., 2015). The main purpose is to support the planning of material flows across the supply chain, including the business-related functions of procurement, production, distribution and sales. The related planning tasks can be done at varying time intervals and levels of aggregation, ranging from aggregated long-term planning to specific short-term planning. Considering the levels of aggregation and material flows through the business-related functions as the axes of supply chain planning, we can describe the various planning tasks through the supply chain planning matrix shown in Figure 3 (Stadtler, 2005). We will use the example supply chain shown in Figure 2 to explain how these planning tasks are related to the different supply chain entities.

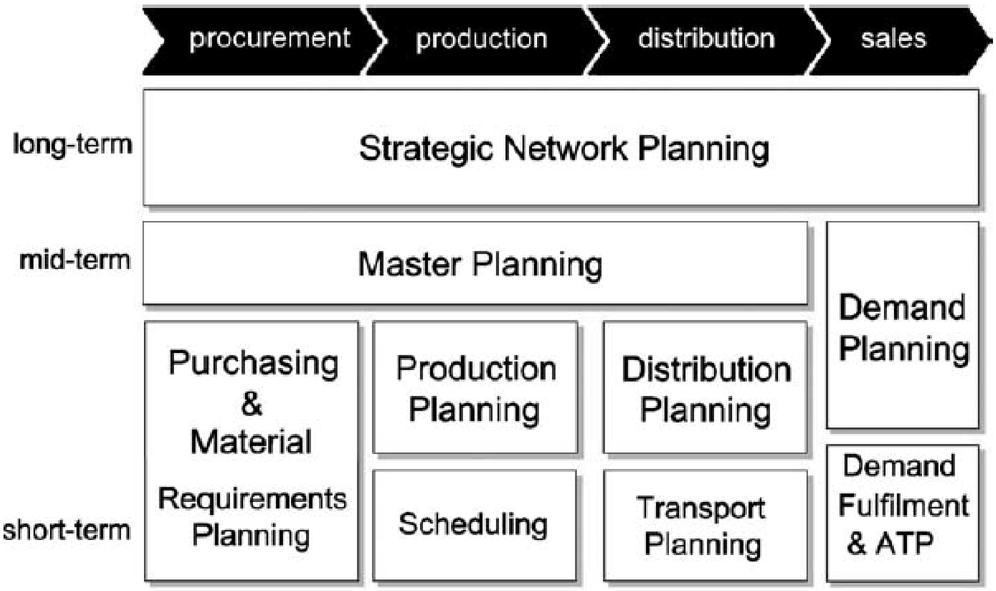


Figure 3: The supply chain planning matrix (Stadtler, 2005).

The hierarchical nature of supply chains is evident in the time dimension: On the longer end, the planning interval may be several years, in which case the goal is often to decide the long-term strategic components of the supply chain, including the capacity and location of production sites, warehouses, transportation means and customer service areas. This corresponds with strategic network planning in the supply chain matrix, where the goal is to design the supply chain so that it enables the best possible economic performance over an extended period of time (Goetschalckx, 2002). In our example supply chain, this would mean planning the structure of the supply chain network itself: This would involve questions such as which suppliers should be included, or how many warehouses are required to serve customers in different areas.

When the structure of the supply chain is set, master planning is done to find the most cost-efficient means to respond to the demand forecasts over some medium-term time interval, often covering a full seasonal cycle. The objective here is to synchronise the flow of materials along the entire supply chain (Rohde and Wagner, 2002). In our example supply chain, this would mean planning all raw material purchases from suppliers, quantities of production at facilities, distribution to different warehouses, and finally sales to customers. The plans at this stage are done to provide an overview of how the supply chain should be operated on an aggregate level, as the more detailed plans are done on separate stages.

After the production flows have been assigned to different sites, more detailed production planning and scheduling is often done within each production facility. The aim with these planning stages is to determine a more detailed production schedule, which shows for example how different machines or flow lines should be operated, and what should be done on different work shifts. (Stadtler, 2002a) In our example supply chain, this planning stage would solve how the different production facilities should be operated in detail, for example on a daily basis.

Based on the directives from master plans as well as shorter-term production planning and scheduling, procurement quantities can be planned with a purchasing and material requirements planning module. This module is necessary for planning non-bottleneck operations, since usually only potential bottlenecks in raw material availability are planned for in production planning and scheduling (Stadtler, 2005). In our example supply chain, this planning stage would solve in detail how and when different raw materials should be purchased from different suppliers.

The distribution planning module considers the flow of goods between sites as well as in the distribution network of the supply chain. Although the master planning phase may include some seasonal stock level requirements at some points of the supply chain, the transports of goods to customers directly or through warehouses and distribution centers are planned in this module in greater detail (Fleischmann, 2002). Even more detailed is transport planning, where specific vehicle loading plans based on production orders to be completed during the next day or shift are formed. This planning phase thus requires order-level knowledge, as well as information on customer-specific needs, including legal restrictions and delivery time windows (Stadtler, 2005). In our example supply chain, these planning stages would show which links between suppliers, warehouses and customers should be used, and what the quantities of flows should be in detail between these entities.

The aforementioned planning tasks clearly indicate the importance of different time intervals in supply chain planning, but other dimensions in the planning process often have hierarchical dimensions as well. For example, in the product dimension, some stock keeping units are different versions of the same product, and similar products may belong in some common product families or groups. On the customer side, some customers operate in different countries, which may in turn be part of different geographical regions (Miller, 2002).

In many cases, the decision maker may not be particularly interested in what is happening to a specific product or customer, but would rather understand the decisions in terms of product families or customer regions (Miller, 2004). As such, an ideal decision support system should enable the user to perform model analysis on different hiearchical levels. Naturally, the level of detail used in the underlying mathematical model sets the lowest possible level of analysis, but the ability to consider the model in terms of its upper hierarchical dimensions, such as product families or customer regions, could provide much more informative decision support in some situations.

2.3 Linear Programming

Linear programming (LP), also called linear optimisation, is a special case of the more general concept of mathematical programming. It is a method for achieving the best numerical outcome in a mathematical model where all relationships are linear. In other words, linear programming refers to techniques for the maximisation or minimisation of a linear objective function, subject to linear equality or inequality constraints. All linear programs can be described with the following form (Bertsimas and Tsitsiklis, 1997):

$$\begin{aligned} \min\ & \mathbf{c}^T\mathbf{x} \\ s.t.\ & A\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{aligned} \tag{1}$$

where $\mathbf{x}$ is the vector of decision variables, $\mathbf{c}$ is the cost coefficient vector, $\mathbf{b}$ is the right-hand side vector and A is the matrix of coefficients related to the problem constraints. In general, we assume that all parameter values A,$\mathbf{b}$, $\mathbf{c}$ are known when the optimisation is run, and the objective is to find the vector $\mathbf{x}$ that minimises the objective function $\mathbf{c}^T\mathbf{x}$, while satisfying the constraints. Note that by "knowing" in this context we do not mean that there must be absolute certainty what the "correct" values for each parameter are. Rather, we mean that for a single model run, each parameter has a certain value that does not change. The actual uncertainty in the parameter values can be addressed by running multiple scenarios, where the uncertain parameters are varied.

The inequalities $A\mathbf{x} \leq \mathbf{b}$ and $\mathbf{x} \geq \mathbf{0}$ are the constraints which specify the feasible region, the set of all possible solutions of the optimisation problem that satisfy all of the constraints. In linear programming, this feasible region is a convex polyhedron, a set defined as the intersection of finitely many half spaces, each defined by a linear inequality constraint. A linear programming algorithm, such as the simplex method, finds the point in the feasible region where the objective function has the smallest value, if such a point exists (Bertsimas and Tsitsiklis, 1997).

Note that any maximisation problem can easily be changed to the corresponding minimisation problem, because minimising some function $f(x)$ is equivalent to maximising the function $-f(x)$. That is, one needs to only multiply the objective function by -1 to change from one to the other.

Linear programming is applicable in a vast field of problems, where quantities can take any real values, and are only restricted by linear constraints. Linear programming has many applications in various industries; In supply chain planning, it has been used for example in solving planning tasks related to master planning and distribution planning. Many powerful solution algorithms for solving linear programming have been developed, and these can be used to solve models involving thousands of constraints and variables in a short period of time (Stadtler, 2002b).

2.3.1 Dual variables and sensitivity analysis

Given any linear programming problem, we can associate with it another problem called the dual linear programming problem. Each variable in the dual problem is associated with a constraint of the original problem, and these can be seen as penalties for violating the constraints (Bertsimas and Tsitsiklis, 1997). The dual of linear program (1) can be formulated as

$$\begin{aligned} \max\ & \mathbf{p}^T\mathbf{b} \\ s.t.\ & \mathbf{p}^T A \leq \mathbf{c}^T \\ & \mathbf{p} \leq \mathbf{0} \end{aligned} \tag{2}$$

Where $\mathbf{p}$ is the vector of dual variables. In linear programming, the optimal objective function value to the primal problem is equal to optimal objective function value of the dual problem, if an optimal solution exists (Bertsimas and Tsitsiklis, 1997).

These optimal dual variable values are important in traditional sensitivity analysis of linear programming models, as they can be used to determine how sensitive the optimal solution is to small changes in model parameters. Specifically, these dual variable values can be used to determine a range for each individual objective function and right-hand side vector coefficient, over which the individual coefficient can vary without having to solve the optimisation again. This information is readily available in most commercial solvers, and can be used to analyse the effect that small individual parameter changes would have on the optimal solution (Bradley et al., 1977).

While the appeal of this traditional sensitivity analysis is apparent as it provides exact quantitative information on the mathematical model, its applicability in practical model analysis setting is limited. This is because traditional sensitivity analysis can only be used to analyse small changes in individual objective function or right-hand side vector coefficients, or multiple smaller changes under some very limiting conditions (Bradley et al., 1977). Especially in large linear programming models, a change in a single model parameter is rarely a topic of interest. Furthermore, such sensitivity analysis does not allow for all types of constraint changes, namely those related to constraint coefficients. Also, many practically relevant changes to the model alter the model structure: For example, adding a new customer or warehouse in the supply chain usually entails the addition of multiple decision variables and constraints. In such cases, traditional sensitivity analysis offers little assistance.

2.4 Model analysis systems

A model created by mathematical programming can be embedded in a model-driven Decision Support System (DSS). By definition, such systems support managers in their decision-making efforts by applying one or several quantitative models to the problem at hand. In addition, model-driven DSS allow users to manipulate model parameters, thus enabling sensitivity analysis of the model outputs as well as more ad hoc "what if" - analyses. Furthermore, DSS can be differentiated from more specific decision analytic studies by two characteristics: Firstly, they make the quantitative model easily accessible to non-technical specialists trough user interfaces, and secondly, they are intended for repeated use in similar decision situations (Power and Sharda, 2007).

One of the purposes of a DSS is to help the decision maker develop a better understanding of the complex system represented by the mathematical model. There are multiple potential sources of increasing this understanding in the mathematical programming process. Perhaps the most direct source is the actual model formulation phase: During model formulation, the decision makers may modify their mental model based on the discoveries of new relationships between key factors, counterexamples of assumed relationships or fallacies in deductive logic (Steiger, 1998).

Another source is the deductive analysis of a single model instance, considering all of the knowledge inherent in the model structure, as well as the solution. For instance, the sensitivity analysis of an LP model solution can help the decision maker understand which parameters are more important than others, and what kind of an effect some small parameter change would have on the solution. An additional source of understanding is the inductive analysis of multiple solved model instances. An example of this would be to solve several model instances while varying some parameters over appropriate ranges, after which these instances can be analysed to assess the relative importance of the individual parameters and their interactions (Steiger, 1998).

The previously mentioned sources of understanding motivate to examine the potential of model analysis systems. Model analysis systems are systems built to enhance the decision maker's understanding of the business environment represented by the model, by helping them interpret and manipulate the output of model solvers and analyse existing knowledge, or extract new knowledge from the business environment the system represents. Model analysis systems can be roughly divided into two classes based on their primary type of processing logic: These classes are deductive analysis systems (DMASs) and inductive analysis systems (IMASs). These systems are distuingished from each other by both their required input as well as the processing logic employed. Whereas DMASs operate on singular model instances and apply existing model-specific knowledge, IMASs operate on many solved instances and apply inductive analysis to generate new knowledge (Steiger, 1998).

DMAS are systems that apply paradigm- or model-specific knowledge to a specific model instance, addressing questions such as "Why is this the solution?", or "why is this instance infeasible?". Perhaps the most prominent example of a DMAS to this date is called ANALYZE, described in detail by Greenberg (1983). It is a computer-assisted analysis system for linear programming models, providing interactive query of the LP matrix and solution values, as well as other functionalities, including model simplification and network path analysis.

IMASs are systems that operate on a set of solved model instances, where some parameter values are varied and the goal is to find the "key parameters" which have the greatest impact on the model solution. One simple form of IMAS is applying regression analysis to multiple model runs, in order to determine the impact of changes in model input parameters to the output. In other words, the goal is to build a metamodel which would help the decision maker build insights into the business environment being modeled (Steiger and Sharda, 1996).

One related approach for generating insights from mathematical models is the use of simplified auxiliary models in conjunction with the original model (Geoffrion, 1976). Geoffrion suggests that these auxiliary models should be both intuitively plausible, and solvable in closed form or by simple arithmetic. Ideally, the solution behavior of the auxiliary model is far more transparent than that of the full model, yet it should yield fairly good predictions of the general solution characteristics of the full model.

3 Scenario analysis for supply chain optimisation models

When supply chain optimisation is used to support decision making, the decision makers typically want to understand the model results and further see how the results change in different scenarios. In this chapter, we outline the principles of scenario analysis for supply chain optimisation. First, some characterisations for typical changes between scenarios are given. Then, a framework for some common analysis questions is presented along with considerations for how these questions can be answered.

3.1 Characterisation of model changes

When comparing various scenarios of an optimisation model, the possible changes can be divided into two categories: parameter-based and structure-based changes. Parameter-based changes are those where the value(s) of some input parameter(s) of the model is(are) changed. In linear programming, such parametric changes are related to either i) the cost coefficients of the problem, represented by the cost vector $\mathbf{c}$, ii) the constraint limits, represented by the right-hand side vector $\mathbf{b}$, iii) the coefficients of the constraint matrix A, or some combination of the above. However, the structure of the model remains unchanged. Structure-based changes are those where the model structure itself is modified in some way. In linear programming, these are related to the addition or deletion of constraints, decision variables, or both at the same time. Note that adding new constraints adds new rows to the constraint matrix and the right-hand side vector, whereas adding new decision variables adds new rows to the decision variable vector, new columns to the constraint matrix and new entries to the cost vector. This obviously involves adding the corresponding parameters to the model as well.

It is important to note that the decision maker does not typically understand how their business questions translate into the mathematical model: The decision maker considers the problem through the physical entities of the supply chain. It is the responsibility of the analyst to ensure that the appropriate changes in model parameters or structure are made to reflect the changes that the business case represents. Depending on the business question and model itself, this translation process may range in difficulty from very simple to prohibitively complex.

Ideally, the model is structured in a way that allows easy manipulation of components that are likely to be of interest to the decision maker. For example, a change in the currency exchange rate may affect multiple separate entities in the supply chain, from raw material procurement to customer sales. Instead of having to change each related cost coefficient separately, the exchange rate could be included as a parameter, which automatically changes all related model parameters accordingly. Another example would be to allow removing a raw material supplier from the supply chain, such that all directly related decision variables and constraints are removed.

3.2 Frequent analysis questions

We next describe some of the common business questions related to supply chain optimisation. These questions were collected from discussions with multiple optimisation and subject matter experts at the case company, including people from different subdivisions and backgrounds. These individuals work with different kinds of supply chains, ranging from wood sourcing to paper production. Our objective is to present the collected questions in the most general setting possible. However, it should be acknowledged that while these questions were identified as relevant for multiple supply chains, the case company is still somewhat limited to a specific industry. Thus, there is a possibility of bias towards the problems related to the specific company and its industry. We believe that these questions may be found in other types of supply chains as well, but their applicability to other settings is not evaluated here.

Table 1 shows example questions that are divided into separate categories based on two characteristics: We assess i) whether the question can be answered with one, two, or multiple scenarios, and ii) whether the question is easy or difficult to answer, based on the analysts' qualitative assessment.

Table 1: Common supply chain analysis related questions

Scenarios	Difficulty	Example questions
Single	Simple	"Who are my top customers by sales?" "What is produced at facility X1?"
Single	Complex	"Who are common suppliers for products P1,P2,P3?" "Which raw materials are most critical in sales?"
Two	Simple	"What's the cost difference between the scenarios?" "Which products are produced less in scenario 2?"
Two	Complex	"How does the cost change affect customer sales?" "How production changes if material R1 is unavailable?"
Multiple	Simple	"Which scenario is the most/ least profitable?" "Which products are produced most across scenarios?"
Multiple	Complex	"What is the order of production bottlenecks?" "What are common patterns across scenarios?"

Based on our discussion with subject matter and optimisation experts, as well as examining the common questions, we were able to distinguish some key elements that are present throughout these questions. Regardless of the amount of scenarios and difficulty of the question, they are all linked to one or both of the following model-related factors:

- The optimal values or cost impacts of individual decisions and constraints
- The relationships between individual decisions and constraints

The first factor is quite clear when we consider that the primary objective in most supply-chain related optimisation models is to minimise costs or maximise profits. Here, the cost impact refers to the total effect a decision has on the objective function, taking into account both the value and the unit cost related to the decision. These allow us to assess the relative importance of individual decisions in terms of costs associated with them. However, for constraints and potentially some decisions, there may not be explicit costs involved. Furthermore, the optimal values of decisions are often important in their own right, as they provide the actual decision recommendations that interest the decision maker. Thus, both the values and cost impacts of the individual model components are often at the center of model analysis related questions, whether we are studying an individual model solution, or comparing the differences between two model scenarios.

While the individual decisions and constraints can be interesting to study, the relationships between different model decisions and constraints are often equally important. This is because most individual decisions in the model are made in conjunction with other decisions, depending on the constraints that bind them together. For example, a decision to produce some quantity of a product P1 in facility X1 requires some capacity, which means there is less capacity available to produce other products, which are governed by separate decisions in the model. Many of the more complicated questions related to supply chains are of this type: The decision maker wants to understand not only what the individual decision recommendations are, but how they are related to each other.

In the following, we provide an overview of the different question categories, and explain how these are related to the aforementioned model factors. While not an exhaustive list by any means, it provides motivation for why these factors can be considered as the focal points for many analysis purposes. How exactly our developed IMPS utilizes these factors is discussed in further detail in Sections 4.2 and 4.4.

Single scenario, Simple questions

This category includes questions that can be answered directly from the model solution, without paying attention to the model structure or relationships between different entities. The ability to sort and filter the results based on desired attributes is sufficient. Examples include obtaining a list of top customers by sales, or a list of products produced at facility X1. Considering our example supply chain in Figure 2, the first example would require us to find the total cost impacts of all decisions related to customer sales, then group and sort these cost impacts by customer. For the second example, we would have to find all production quantity values of decisions related to facility X1. In both cases, only individual decision variable cost impacts and values were required to answer the question.

Single scenario, Complex questions
This category contains questions related to a single model run, but the answer cannot be found without considering the model structure and the relationships between decision variables. The main difference with the previous category is that individual cost impacts or values are no longer enough, as we now need the ability to relate different decisions together. An example would be to find the common raw material suppliers for a list products such as P1, P2 and P3, or finding the raw materials that are involved in the most customer sales. Considering our example supply chain in Figure 2, the first example would require us to find all possible paths from different production facilities, where procuts P1, P2 and P3 are being produced at some non-zero quantity, to all suppliers that supply the raw materials required for these products. The second example would require us find paths from customer sales decisions to the different raw material purchase decisions from suppliers, then group and sort the total cost impacts of the customer sales by each individual raw material. Thus, not only the cost impacts and values of individual decisions are required, but also the connections between them.

Two scenarios, Simple questions
Simple questions related to comparing two scenarios are queries such as "What is the overall cost difference between these scenarios?" which can be answered by comparing the optimal values of the objective function, or "Which products are produced less in scenario 2", which can be determined by comparing individual production quantity decisions between the scenarios. Similar to the single-scenario case, answering these questions is straightforward, requiring only the ability to aggregate the results as desired. In our example supply chain in Figure 2, the first example would require to sum together the total cost impacts of all decisions made in the supply chain: These include raw material costs from suppliers, production costs, transportation costs and sales profits. After this is done for both model scenarios, these sums can be subtracted to obtain a total cost difference between them. The second example would require finding the total values of production quantity decisions made at different production facilities, grouping them by each individual product, and subtracting these scenario-specific values. While perhaps slightly more complex than the individual scenario case, basically the cost impacts and values are again enough to answer these types of questions.

Two scenarios, Complex questions
Many of the simple questions regarding scenario comparison can be considered at a more detailed level. For example, instead of simply calculating the overall cost difference, the decision maker may be interested in the primary sources that cause this difference: maybe the overall logistics costs have increased, but is it because individual routes from facility X1 to customers C1, C2 and C3 have become more expensive, or due to small increase in costs everywhere? Note that the change between

the two comparable scenarios might not have anything to do with logistics costs but rather there could have been a change in customer demands, which eventually leads to increased logistics volumes somewhere. The decision maker may be interested in understanding how certain changes are related to each other: For instance, this may involve questions such as "How does a change in currency rates affect customer sales?", or "How does production change if capacity is increased at facility X1?". In the example supply chain shown in Figure 2, for the first example we would have to identify all the affected customers, for whom there is a difference in sales volumes, in other words the optimal values of the various sales decisions, between the scenarios. Then, the analyst might try to link these changes to the different warehouses, in order to observe how the production flows change throughout the supply chain, or to different production facilities to examine how the product mix has changed as a result of the changes in costs. The second example could be explored by first checking what the differences in production quantity decisions related to facility X1 are between the scenarios, and then try to find connections from these to customer sales, as well as production quantity decisions at other production facilities. Answering these types of questions exhaustively is very difficult without considering the entire supply chain, as there is generally no pattern that is guaranteed to occur.

However, there is one special occurrence between two scenarios that was found prevalent in the case company: These are cases where the optimal value of some decision in one scenario seems to have an approximately opposite effect to some other decision in the other scenario. For example, considering the example supply chain in Figure 2, it could be that the production quantity of product P1 increases at facility X1, while at the same time the production quantity of product P1 decreases at facility X2 by a similar amount. Another example could be that the transportation of product P1 to some customer C1 is switched from warehouse W1 to warehouse W2. These effects may happen due to various reasons, either through a direct or an indirect change in the factors that affect the optimal value of the related decision variables.

Furthermore, these substitution effects can be found at varying hierarchical levels: If substitution seems to happen at a higher level of hierarchy, then it is likely to be a sum of smaller substitutions that happen on the lower levels. As such, these substitutions are often sought first at higher hierarchy levels, since these can be used to explain the changes at lower levels. In terms of decision support, these may provide more valuable information to the decision maker. As an example, the fact that the production of an entire product group is switched from site to another carries more weight than stating that these products individually are produced at different sites. These substitution effects were found pervasive enough that we created a special algorithm for finding these between scenarios. This is explained in further detail in Section 4.5.

Multiple scenarios, simple questions

When comparing multiple scenarios, the simple questions a decision maker may be interested in are such as "How do all these scenarios compare in terms of total profits?" or "What are the most commonly produced products?". In our example supply chain in Figure 2, for the first example we would simply sum over all total cost impacts for each individual scenario, after which they could be sorted in any desired order. For the second example, we would collect the production quantities for each individual product across all production facilities and scenarios, after which we could sort the products by total volume. These questions follow a common theme with other simple cases: As long as there is no need to consider how the supply chain entities (or model elements) are related to each other, there is not much difficulty involved, even if there are multiple scenarios to consider.

Multiple scenarios, complex questions
This is the most general category of questions, where multiple scenarios are compared in ways that require deeper knowledge than the overall results, and some method of linking together decisions between multiple scenarios. As an example, a decision maker could be interested in the order of production bottlenecks that arise when demand is increased. Answering such a question requires solving the optimisation problem multiple times with varying customer demands, as well as information that links these demands to production entities. For our example supply chain in Figure 2, we would then observe how the different production facilities operate with increasing customer demand. Another possibly interesting query would be to find the potential common patterns between all considered scenarios: Such patterns would suggest that the decisions involved are robust and should be made regardless of the considered uncertainties. For the example supply chain, we might find for example that facility X1 always produces product P1, even if the costs or the structure of the supply chain changes. However, this category of questions is very difficult to answer, even more so than two-scenario comparisons, and they will not be explored in more detail in this Thesis.

3.3 Questions as driver for scenario analysis

The mathematical programming process can be viewed as a series of transitions from the real world into the model world: Often, the process starts after some need for change is recognised in the real world (Keisler and Noonan, 2012). In a supply chain optimisation context, this starting step may be the decision maker recognising the need for decision support in managing the supply chain more efficiently. The initial transition from the real world to the model world then happens as the mathematical model is formulated by the analyst.

After the mathematical model has been formulated, it can be used to perform various analyses that interest the decision maker. Scenario analysis can thus be seen as the process of connecting the decision maker's interests into the model world, and translating the corresponding model results or changes back into the real world. This view of scenario analysis emphasises that parsing the model results is only part of the process, and that it is equally important for the analyst to understand what the decision maker's key interests are. The main purpose of gathering business questions from decision makers is to facilitate the scenario analysis process by identifying the relevant matters the analyst should focus on. Moreover, answering these business questions with the mathematical model is by itself a method of providing decision support, thus they are often related to the primary reason for the model's existence.

Furthermore, categorising the business questions can help the analyst obtain a broader understanding of the decision maker's key interests, and what the practical requirements for answering these questions are. For example, in terms of scenarios, are most questions related to a single scenario, or is there a clear need for additional model runs? Additionally, understanding the general difficulty of the questions that interest the decision maker can help the analyst in deciding the best practical approach for scenario analysis: For instance, if the majority of the business questions are simple, there is probably no need for complicated analysis tools, and simple reporting capabilities will suffice.

4 Methodology for building Intelligent Mathematical Progamming System

In this section, we describe different technical approaches to scenario analysis in use of optimisation models, and their benefits and drawbacks. Our focus is not on evaluating commercial optimisation packages (such as IBM ILOG Decision studio, AMPL or AIMMS), but instead focus on general approaches that are based on analysing the information about the optimisation model structure and the solution. Typically, these kinds of systems are based on spreadsheet technologies (e.g., Microsoft's Excel) or programmable environments (e.g., Python, R).

4.1 Approaches to building IMPS

Simple approach
In a simple IMPS, the amount of data saved from the model is minimal and it is used only for simple visual representations and comparisons. An example of this would be to save the model solutions to a spreadsheet, which could then be used to assess simple queries, such as listing the largest changes, or plotting the results based on some preset function. The most significant benefit of such an approach is that it usually requires less model-specific knowledge than its more advanced counterparts. Spreadsheet software have achieved widespread popularity due to their high availability and ease of use, and they are widely used also for model analysis purposes (Grossman, 2008). However, the functionality offered by the built-in tools of spreadsheets is often limited, and writing more complicated functions can become tedious. Furthermore, a supply chain model can have millions of decision variables, which alone can make the use of spreadsheets very difficult (LeBlanc and Galbreth, 2007).

Results oriented approach
A more advanced approach is to use a relational database system to store and manage the information obtained from various scenarios. The various entities related to the problem can be saved as separate data tables, and relationships between different entities can be described with additional tables. This entire database can then be processed in a business analytics platform, such as Microsoft's PowerBI or Tableau Software's Tableau, which allow the user to construct different types of dashboards and views into the model solutions. The main advantage of this approach over the simple one is that the additional data and the relationships between entities of the supply chain, described by relational tables, allow for more advanced queries into the model solutions. In addition, the hierarchical structure of the system can be represented as additional attributes or additional tables, thus preserving the hierarchical structure and allowing the user to view results on any desired level of aggregation.

A typical example of this approach would be to save each of the entities of the supply chain, such as suppliers, products and customers, as a separate table. In addition to providing an identification code for each entity, these tables may contain the hierarchical levels as additional attributes. Then, the optimal solution can be stored into the database as separate tables for each type of decision variable, along with any relevant attributes.

One drawback of this approach is that the user must have model-specific knowledge and experience to find the desired connections in the data. For example, while the data related to the model structure is saved, the structure of the model itself is not preserved. As a result, it becomes difficult if not impossible to determine how two decision variable values, such as production quantities of products P1 and P2, are dependent on each other by some common constraint, such as a production capacity constraint.

Model oriented approach
As previously mentioned, the dependencies between variables are obscured in the results oriented approach: For example, we know the optimum value of a decision variable, as well as its impact on the objective function value. However, based on this information alone, we cannot say which other decision variables are related to this decision variable, and how these relate to the discovery of the optimal solution.

Suppose that in the optimal solution, the value of the decision variable related to the production quantity of product P1 is non-zero. This alone can raise several questions that may be interesting to the decision maker: For instance, why is this exact amount produced? Is there a capacity constraint that is limiting the production, and if so, then what other products are competing for the limited capacity? Then, how does this product flow through the supply chain - which warehouses is it sent to, and which customers' demand is satisfied by this decision? In order to answer such questions, one clearly needs some form of access into the relationships of the supply chain entities.

This lack of information related to the relationships between model components in the previous approach motivates the form of IMPS developed in this Thesis. We call this the model oriented approach. Here, the main difference compared to the previous approach is that the structure of the original model is preserved, which enables queries that are related to the model structure. Whereas the previous approach only considers the optimal decision variable values, in this approach the constraints related to the decision variables, as well as the links between them, are processed and saved as part of the solution data.

In the following section, we present the structure of the IMPS we have developed. We first describe the basic idea behind our model structure based approach, after which we provide a schematic view of the IMPS and a brief description of the used technologies and the IMPS's functionalities.

4.2 IMPS description

4.2.1 Linear program in graph form

Consider again the general linear program

$$\begin{aligned} \min\ & \mathbf{c}^T\mathbf{x} \\ s.t.\ & A\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{aligned}$$

Each row of the LP constraint matrix A corresponds to one constraint in the optimisation problem, whereas each column corresponds to one decision variable. The non-zero coefficients in row A_i of the matrix present which decision variables are directly linked to this constraint, whereas the non-zero coefficients in a column $\mathbf{a}_j$ correspond to links from this decision variable to different constraints. The aforementioned ideas indicate that the structure of a linear programming problem may be given in the form of a graph, which we define as follows (Diestel, 2016).

Definition 4.1. An undirected **graph** is an ordered pair $G = (V, E)$, where V is a set whose elements are called nodes, and E is a set of two-sets of nodes, whose elements are called edges.

Depending on the context, the edges of the graph may link two nodes symmetrically, in which case the graph is called undirected. The edges may also link two nodes asymmetrically, in which case each edge has a distinct start and end node, and the graph is called directed. These directed edges can be given as a set of ordered pairs: For example, an edge from node u to node v would be given as the pair (u, v). The formal definition for a directed graph is as follows.

Definition 4.2. A **directed graph** is an ordered pair $G = (V, E)$, comprising of a set of nodes V, and E is a set of ordered pairs of distinct nodes, whose elements are called directed edges.

Considering also the aforementioned fact that in an LP model, decision variables are directly linked only to constraints and vice versa, we note that the nodes form two disjoint sets U and V, corresponding to the rows and columns of the LP constraint matrix A, respectively. Each edge in the graph links one decision variable and constraint, or a node in U to a node in V. Graphs that satisfy such a condition are called bipartite graphs (Diestel, 2016).

Definition 4.3. A **bipartite graph** is a graph $G = (U, V, E)$ whose nodes can be divided into two disjoint and independent sets U and V, such that every edge in the set of edges E connects a node in U to a node in V.

With the help of these definitions, we can now present the definition of the fundamental digraph of a LP problem, which was originally given by Greenberg (1983).

Definition 4.4. (Greenberg, 1983) The **fundamental digraph** of a LP constraint matrix A is a bipartite, directed graph $G = (U, V, E)$ with node sets U and V, corresponding to the constraints and decision variables of A, respectively. The set of edges E is defined by the non-zero elements of A, and they are directed based on the signs of the values in A: $\forall i \in U, j \in V$: $A_{ij} < 0 \Leftrightarrow (i, j) \in E$, $A_{ij} > 0 \Leftrightarrow (j, i) \in E$.

The directions of the edges in the fundamental digraph can be explained as follows: For each entry in the constraint matrix A, the edge is from the constraint i to the decision variable j, if the corresponding matrix entry A_{ij} is negative, thus the edge is given as the pair (i, j). Likewise, the edge is from the decision variable j to the constraint i, if the corresponding matrix entry A_{ij} is positive, and the edge is given as the pair (j, i). Note that for each constraint node, the directions of incoming edges can be switched around by multiplying the corresponding constraint by -1. In the case of an inequality constraint, this naturally switches the direction of inequality as well. The problem is then no longer in the general form, but the logical structure of the problem is not changed.

We illustrate how the fundamental digraph can be formed by using a simple example. Consider a manufacturing company that can only produce two types of products, $P1$ and $P2$. Producing both products requires two raw materials, $R1$ and $R2$. Each unit of $P1$ requires 1 unit of $R1$ and 3 units of $R2$, and each unit of $P2$ requires 1 unit of $R1$ and 2 units of $R2$. The total availability of these raw materials is 5 units of $R1$ and 12 units of $R2$. Each unit of $P1$ can be sold for a profit of 6 per unit, and each unit of $P2$ can be sold for a profit of 5 per unit. The company wants to maximise is profit. Denoting the production quantities of the products x_{P1} and x_{P2} respectively, we formulate the corresponding linear programming problem as

$$\max\ 6x_{P1} + 5x_{P2} \tag{3}$$
$$s.t.\ x_{P1} + x_{P2} \leq 5 \tag{4}$$
$$3x_{P1} + 2x_{P2} \leq 12. \tag{5}$$

This problem contains two decision variables, x_{P1} and x_{P2}, and two constraints (4) and (5) that limit the availability of each raw material. The corresponding constraint matrix is $A = \begin{bmatrix} 1 & 1 \\ 3 & 2 \end{bmatrix}$, thus both decision variables are linked to both constraints, and since all entries of A are positive, the edges are directed from the decision variables to constraints, as described in the definition of the fundamental digraph. The resulting digraph for this problem is shown in Figure 4.

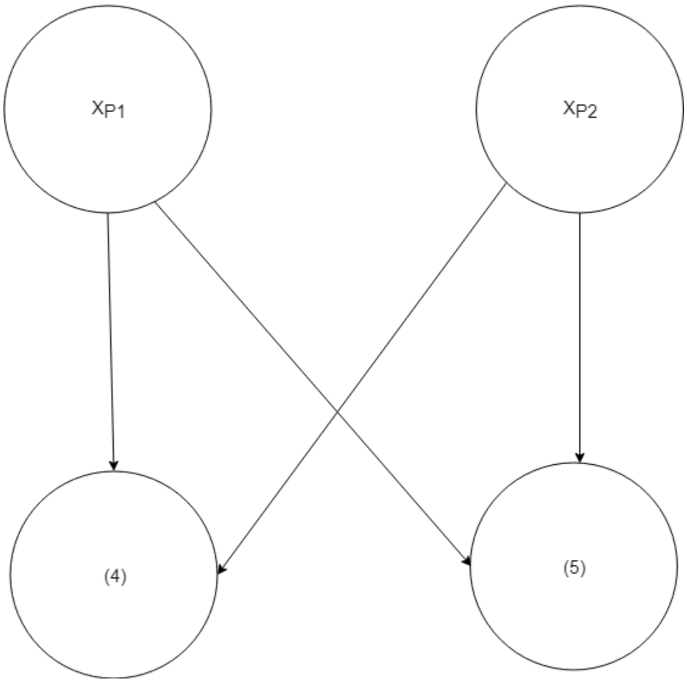


Figure 4: The fundamental digraph of the example problem.

4.2.2 Expanding the graph information

The fundamental digraph of the constraint matrix serves as a starting point for our graph structure. However, this structure offers only little information outside of the links between model components. To enable more detailed queries into a single model solution and the comparison of different scenarios, we need additional information related to each node. Thus, our purpose is to expand this digraph structure by allowing each node to contain additional information by themselves, thus removing the need to store this information somewhere else. In the following, we describe the attributes that are saved as part of this augmented solution graph.

For decision variables, each node should store information on the following:

1. The optimal value of the variable in the corresponding solution.

2. The cost coefficient of this variable in the objective function.

3. The cost impact of this variable to the objective function.

The need to store the optimal values is obvious, especially when comparing separate model scenarios. However, these decision variables may have varying magnitudes and different units. Therefore, when considering the relative importance of different changes in decision variable values, the change in decision variable values by itself is not always a good metric.

Instead, the cost impact of each decision variable provides a more unified metric for assessing how impactful each change is, since the units of all decision variables are converted to the unit of the objective function. In linear programming, this cost impact is simply calculated as the corresponding cost coefficient multiplied by the decision variable value.

It may be unapparent at first glance why we would need to store information on the cost coefficient separately. However, when comparing scenarios where some of the cost coefficients change, explicitly determining this change can be beneficial, as this cannot be exactly deduced from the changes in value and cost impact alone.

For constraints, each node should store information on the following:

1. The row sum of the constraint, or $A_i\mathbf{x}$.
2. The right-hand side (RHS) of the constraint.

The row sums of the constraints provide valuable information on how each constraint is satisfied. Furthermore, when comparing scenarios, a change in the row sum corresponds to a change in the flows through that constraint.

A change in the right-hand side for an inequality constraint corresponds to change in the upper or lower bound of the constraint, e.g. the upper bound for maximum capacity, or the lower bound for a minimum production constraint. In situations where we want the focus of the analysis is on such changes, such information is essential.

Consider also the edges between nodes in the fundamental digraph: They are created solely based on the signs of elements in the constraint matrix A, and there is no information on the exact value of the dependency between the constraints and decision variables, which are given by the entries in the constraint matrix. To enable comparisons between solutions where these values are changed, it makes sense to store these values exactly as they are, instead of simply connecting the nodes based on non-zero entries in the constraint matrix. Thus, we augment each edge in the graph to contain the exact value of the corresponding entry in the constraint matrix.

The aforementioned attributes form what we generally refer to as the attribute vector. Each node and edge in the graph is associated with exactly one attribute vector, and the contents depend on whether the entity in question is a constraint node, a decision variable node, or an edge.

Definition 4.5. The **attribute vector** associated with **decision variable** x_j is

$$\mathbf{s}_{x_j} = \begin{bmatrix} x_j^* \\ c_j x_j^* \\ c_j \end{bmatrix},$$

where x_j^* is the optimal value of the decision variable, and c_j is the cost vector component associated with the decision variable x_j.

Definition 4.6. The **attribute vector** associated with **constraint** y_i is

$$\mathbf{s}_{y_i} = \begin{bmatrix} b_i \\ \sum_j A_{ij} x_j^* \end{bmatrix},$$

where b_i is the right-hand side vector component associated with constraint y_i, and A_{ij} is jth column (decision variable) of ith row (constraint) of the constraint matrix A.

Definition 4.7. The **attribute (scalar)** associated with **edges** is $s = A_{ij}$, where the edge is (y_i, x_j) if $A_{ij} < 0$ and (x_j, y_i) if $A_{ij} > 0$

For the aforementioned attribute vectors, we only need the decision variable values of the optimal solution, and the right-hand side and cost vectors **b** and **c**, in addition to the constraint matrix A. However, to further enrich our ability to find specific patterns and relationships in the model, it is useful to map the decision variables and constraints to the real entities of the supply chain. The specifics obviously depend on the particular problem, but we attempt to give some general guidelines here based on our experiences.

Each node should have a unique identification code that links it to the specific decision variable or constraint. This is necessary, since otherwise we would be unable to match the same nodes between different scenarios. This clearly requires that the identification scheme should also be consistent across scenarios. One should be especially careful when using automatic numbering, since these may change after the addition or deletion of a decision variable or constraint. The MPS file format supports variable naming, so this is not an issue in practice.

Furthermore, a classification based on different node types is useful. In a supply chain context, these often include information on whether the node is related to e.g. the acquisition of raw materials, production of goods, or satisfying customer demand. The purpose of including such information is to incorporate the logical structure of the problem, enabling the IMPS to look for more specific patterns the user might be interested in. The node types could coincide with the mathematical model formulation, so that each different type of decision variable and constraint is included as the type of the respective node.

In addition, nodes should have more specific information on their distinct dimensions, based on the node type. For each node in the graph, we define the dimensions as follows.

Definition 4.8. The **dimensions** of node n are defined as a function D_n: $K_n \to L_n$, where K_n is the set of keys for node n, and L_n is the set of values for node n.

We assume that the node keys match between all nodes of equal type, which means that when comparing the dimensions of two nodes of the same type, we can use the set of keys from either of these nodes. These dimensions should also be quite clear from the mathematical model, as these coincide with the properties each decision variable and constraint has. For example, consider a decision variable node n_1 for the production quantity of some product $P1$, in some production facility $A1$, at some time period $T1$. Then, the keys of D_{n_1} are the names of these dimensions, such as *productID*, *facilityID*, and *periodID*, with the corresponding values: $D_{n_1}(productID) = P1, D_{n_1}(facilityID) = A1, D_{n_1}(periodID) = T1$. For another production decision variable node n_2, the production quantity of the same product at the same facility, but at time period $T2$, the corresponding dimensions would be $D_{n_2}(productID) = P1, D_{n_2}(facilityID) = A1, D_{n_2}(periodID) = T2$.

For constraint nodes, we may include the type of the constraint as an additional key-value pair in its dimensions: In other words, each constraint node's dimensions can include a special key, with a corresponding value depending the type of the constraint: '$\leq$' (L), '$=$' (E), or '$\geq$' (G). This can be useful in situations where we want to filter results based on binding inequality constraints, for instance.

As discussed in Section 2.2, supply chains often include multiple hierarchies. To enable queries where some of these hierarchies are the focal point of interest, the nodes should also carry information on the hierarchies relevant to the node. For example, a node related to a product would contain information on the related product family, in addition to the specific product. This approach is inefficient in terms of space usage, since this hierarchical information is duplicated for each node that share some of their attributes. However, if we want to maintain the graph as an exact representation of the model structure, this approach is necessary. These can be then included as additional key-value pairs into the corresponding node dimensions D_n.

Finally, it may be beneficial to distinguish between two separate types of entities in the optimisation model: i) entities that can be linked to physical entities of the supply chain, such as decision variables related to production, or material flow constraints, and ii) virtual entities that are included in the problem formulation, but do not have a physical interpretation. Entities such as minimum production levels or penalties whose goal is to guide the solution in some direction are part of this category.

We provide an example to illustrate how the aforementioned information would be augmented into the fundamental digraph. Consider again the simple linear programming problem (3)-(5), and its fundamental digraph. Obtaining the optimal solution results in decision variable values $x^*_{P1} = 2$, $x^*_{P2} = 3$. The cost coefficients of these decision variables are obtained directly from the first row, thus we have $c_{P1} = 6, c_{P2} = 5$. The cost impacts of the decision variables are easily calculated as $x^*_{P1}c_{P1} = 2 \times 6 = 12$, and $x^*_{P2}c_{P2} = 3 \times 5 = 15$. For the constraints (4) and (5), we observe that both of them are less-than constraints, and their right-hand sides are 5 and 12, respectively. Their row sums can be calculated as $x^*_{P1} + x^*_{P2} = 2 + 3 = 5$, and

$3x^*_{P1} + 2x^*_{P2} = 2 \times 3 + 3 \times 2 = 12$. Thus both constraints are binding at the optimal solution. From the constraint matrix A, we obtain the following attributes for the edges between nodes: $s(x_{P1}, (4)) = 1, s(x_{P1}, (5)) = 3, s(x_{P2}, (4)) = 1, s(x_{P2}, (5)) = 2$.

This example problem basically contains nodes of two types: Both decisions could be classified as decisions related to production quantities, and both constraints could be classified as constraints related to raw material availability. For the dimensions, the production quantity decisions could have a key *productID* that maps to the product of the corresponding node: $D_{x_{P1}}(productID) = P1, D_{x_{P2}}(productID) = P2$. For the raw material constraints, the dimensions could have a key *RMID* that maps to the raw material of the corresponding node: $D_{(4)}(RMID) = R1, D_{(5)}(RMID) = R2$.

The augmented digraph for the example problem is shown in Figure 5. For each edge and node, the corresponding numeric attribute vector is illustrated as a rectangle, and the node dimensions as key-value pairs inside the node. Note that the structure of the original graph has not changed, but it now includes much more information that can be used in analysing the model solution.

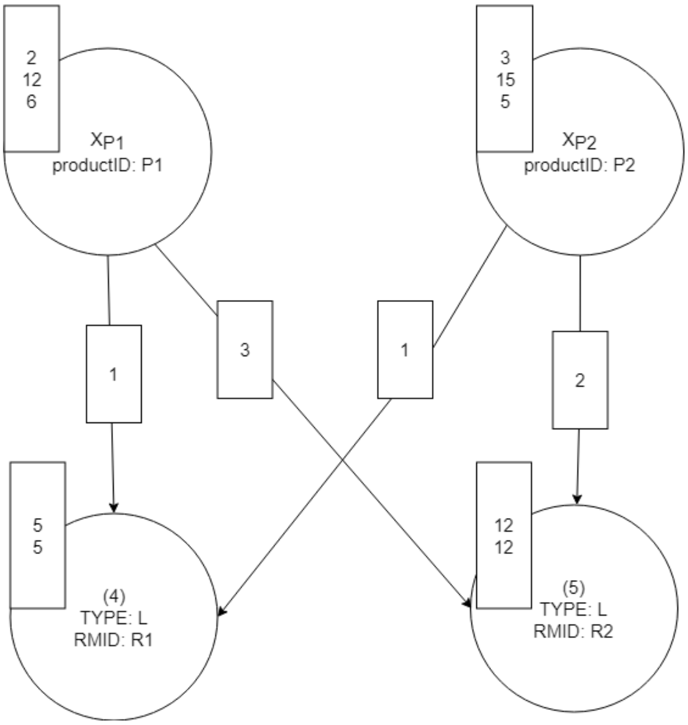


Figure 5: The augmented digraph of the example problem.

4.2.3 Schematic view and file structure

The schematic view of the IMPS is shown in Figure 6. The necessary inputs for the IMPS consist of two separate files: the model file (.MPS) and solution file (.SOL),

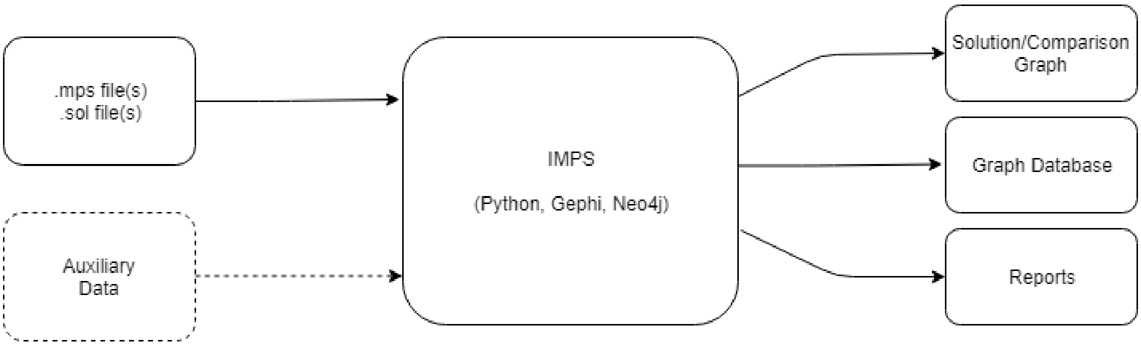


Figure 6: Schematic view of the IMPS

which will be described in more detail later. Each model run generates a pair of these files, and most commercial solvers allow the user to save these files separately. In addition, we may include auxiliary files which contain additional information related to the problem. The currently supported auxiliary files are the following:

i) a metafile, a text file containing a list of all decision variable and constraint types, along with the dimensions related to each of these types. This metafile is based on the same naming convention that is used with the variables and constraints of the problem, so different parts of the variable or constraint name can be directly matched with the rows from the metafile.

ii) a hierarchical mapping of some dimensions to upper hierarchical levels. This file specifies the dimensions with hierarchical levels, listing each value at the lowest level with the related upper levels. For example, for each unique product identification code, we may identify the related product family.

iii) a list of virtual constraints. In some cases, the model may include some virtual constraints or penalties that do not represent any physical flow or material constraint in the supply chain, but instead are meant to guide the solution in some direction. Specifying these separately allows the user to either ignore or specifically look for relationships related to such constraints.

The technologies used in the IMPS include Python, Gephi and Neo4j. All of the technologies applied in the IMPS are free to use and widely accessible, which makes further development easy. Python was chosen as the main programming language of the IMPS, due to the familiarity of the language at the case company, as well as its powerful data and graph manipulation libraries. Gephi is an open-source software package used for graph and network analysis. It includes a 3D rendering engine that allows the user to display and explore large graphs in real time. Furthermore, the user has broad access to the data inside the graph, and the program enables multiple methods of filtering, manipulating and clustering the graph. As such, it provides a powerful tool for the visual analysis of graphs (Bastian et al., 2009).

Neo4j is a graph database management system developed by Neo4j, Inc. Compared to traditional relational database management systems, the foundational element behind this system is a graph that represents relationships in the data. In Neo4j, all entities are stored as nodes, edges, or attributes. Each node and edge can have multiple attributes, and they can be labelled with different names to narrow graph searches to desired labels. The representation of optimisation model runs as graphs directly translates into a graph database of this form: each decision variable and constraint is a node in the graph database, with the attributes and labels corresponding to its type. Relationships between nodes are the same as the connections between decision variables and constraints.

4.2.4 MPS Parser

MPS (Mathematical Programming System) is a file format for presenting LP and MILP problems. The format is column-oriented, meaning that the structure of the file is based on the decision variables, as opposed to row equations. Furthermore, all model components receive names, which can be used advantageously in the post-processing steps as we describe below. The format is compatible with essentially all commercial LP solvers, which is why it was chosen as the standard input of our IMPS. To convert the MPS file into a feasible data structure, an MPS parser is built. We briefly describe how our MPS parser functions through the various fields of the MPS file.

The MPS file starts with a NAME record, which our MPS parser skips since this is arbitrary and has no significance in model structure. Then, the parser reads through the following ROWS section, corresponding to each constraint in the model, saving each row name as a new key in a hash table. The type of the constraint, E for equality ($=$) rows, L for less-than ($\leq$) rows, and G for greater-than ($\geq$) rows, is saved as an attribute. Note that the objective function is specified as a special row with name 'OBJ' of special non-constraining type N. Then, the following COLUMNS section contains the non-zero coefficients of the constraint matrix. Here, the parser checks each entry, and saves the column-coefficient pair as a key-value pair under the corresponding row key. Next, the RHS section contains the right-hand sides of the constraints. The parser saves these as attributes under the corresponding row key. Missing entries are assumed to have a right-hand side of zero. The optional sections of BOUNDS and RANGES are not considered in the current implementation, since they are only specific forms of constraints that could be described in the COLUMNS section as well. As a final result, our parser results in a hash table- type data structure with keys corresponding to the rows of the model, and columns related each row form an inner hash table of key-value pairs of column names and coefficients, along with the other row-related attributes. This is similar to an adjacency list representation of the graph, which is efficient when the underlying graph is sparse.

Note that the MPS format we have used is an extended format, which allows for variable and constraint names longer than 8 characters, the maximum limit in the traditional MPS format. This format allows the use of whitespace characters as field limiters, but the names used cannot contain whitespaces. Furthermore, the naming convention of variables and constraints we currently use is based on the format automatically generated by some commercial solvers, e.g. Gurobi. Here, the variable or constraint name starts with the type, followed by square brackets that contain comma-separated values for each related attribute. The pseudocode for this algorithm is given in appendix A.

4.3 Constructing individual graphs

After obtaining the parsed MPS (PMPS) file, the graph construction is straightforward. The only additional information necessary to create an augmented solution graph is the solution (.sol) file, containing the optimal decision variable values for the specific model run. Assuming that the decision variable names are consistent with the ones specified in the MPS file, we simply look up the optimum value for each decision variable from the solution file when creating the corresponding node. These can then be used to calculate the cost impacts of each decision variable, as well as the row sum of each constraint. Finally, if any auxiliary data is included, these files are parsed and the relevant attributes are added to each node.

As a final result, we obtain a graph data structure in Python that can either be transferred into Gephi for visualisation, or into a graph database from which we can query any model-related information that is relevant to the business questions of the decision maker.

The pseudocode for the graph construction algorithm is given in appendix B. In the algorithm, the word 'row' refers to the set of key-value pairs indentified by a row key, and a specific attribute 'val' in this row is referred to as 'row[val]'. Thus, for columns, 'col' refers to the column name, while 'row[col]' refers to the multiplier of the decision variable 'col' in the constraint 'row'. Similarly, we refer to the optimal decision variable values and cost coefficients by referring the respective mappings by the column name. Notice that the row with key 'OBJ', identifying the cost coefficients of decision variables, is treated separately from the other rows.

4.4 The comparison of graphs

Given two augmented digraphs for the same model, G and H, we then wish to compute their difference to produce a comparison graph. We denote this new graph as $C = G - H$. In this section we describe how this operation is done. It is important to note here that there may be alternative methods for constructing such a graph, but this version has proved to be reasonable for our purposes.

The main idea behind the construction of the comparison graph is performing a node-by-node comparison of the graphs. This is done by subtracting the attribute vector of each node in H from the corresponding node's attribute vector in G. For example, consider two nodes n_G, n_H, which are the corresponding nodes from graphs G and H, respectively. Then, the corresponding node in the comparison graph, denoted n_C, is going to have the attribute vector:

$$\mathbf{s}_{n_C} = \mathbf{s}_{n_G} - \mathbf{s}_{n_H} \tag{6}$$

The comparison graph is then constructed based on the difference in each decision variable's attribute vector. Specifically, if the optimum value or the cost impact of a decision variable has changed between scenarios, it is included as a node in the comparison graph. It is important to consider these changes simultaneously, since there may be cases where the change can only be seen in one of these elements. We motivate this further by considering several examples.

For instance, some decision variables may not carry explicit costs by themselves: Consider a supply chain case where we have separate decision variables for the purchasing of raw materials and the consumption of these raw materials. Naturally, the cost incurred by purchasing some raw material should only be accounted for once, therefore the decision variables that control the consumption of these materials are zero cost. If we were to consider only the cost changes between the scenarios, we would only see that some raw material was purchased more or less, and not how this affects its consumption.

Likewise, some decision variables may have the same optimum value in the scenarios, even though the cost impact of these decision has changed. Consider again a supply chain example, where the margin profit for selling some product has changed between scenarios. In such a situation, it may be that the optimum decision is still to sell the same amount of that product as before. Thus, if we were only checking the changes in volume, we might miss a significant part of the change in the objective function value.

Note that there is also a potential case where the only difference in the attribute vector is in the cost coefficient. However, this is only possible if the optimal value of the decision variable is zero in both scenarios. This implies that the decision is not taken in either of the considered scenarios, thus it can be considered irrelevant in terms of scenario differences.

It is worth mentioning that comparing these total cost impacts may be somewhat unreasonable in cases where the cost coefficient has also changed. For example, consider a case where the marginal profit obtained by selling some product is "worsened" by some change between scenarios. As a result, the optimum solution may be to either reduce or completely stop selling this specific product. Then, this decision by itself has a negative cost impact, since less profits are obtained due to this decision.

However, the cost impact obtained simply by comparing the profits obtained by selling this product in different scenarios are misleading, because it does not take into account the fact that if we were to continue selling this product, our profits would have also been less than previously. Thus, a fair comparison would be to consider the loss based on the new cost parameter too, not just the old one.

The decision variable nodes, where neither the cost impact nor optimum value has changed, are not included in the comparison graph. This includes both cases where optimum variable values are zero and non-zero, and there is no distinction being made between these cases. If the optimum value of a decision variable stays the same, and its cost impact remains the same as well, then we can safely ignore it. This is because such decisions clearly have no impact on the difference between the model scenarios, which are the primary reason for studying the comparison graph in the first place. In practice, we set some small tolerance value, and consider all absolute changes less than this as equivalent. This tolerance allows us to ignore very small changes that are either rounding differences, or otherwise have a negligible impact on the solution difference.

In addition to the relevant decision variable nodes, the comparison graph includes all incoming and outgoing edges from these nodes, as well as the constraint nodes at the other end point of each edge. This means that all constraints that are directly affecting any relevant decision variable are included in the comparison graph. Note that similar to irrelevant decision variables, there may be constraints that are irrelevant in terms of scenario difference: For instance, there may be constraints that are linked only to these irrelevant decision variables, thus they can be ignored.

There are a few special cases worth discussing. First, to enable the addition or deletion of decision variables and constraints between scenarios, we must be able to include nodes that are only part of one model scenario. Our approach with such nodes is as follows: For decision variable nodes, we set the missing attribute vector to the zero vector $\mathbf{0}$. In other words, if the decision variable node n_G is only in G but not in H, we set $\mathbf{s}_{n_H} = \mathbf{0}$, and similarly for decision variables only in H but not in G, we set $\mathbf{s}_{n_G} = \mathbf{0}$, then compute the difference in attribute vectors as described in equation 6. As a result, all decision variable nodes that are only part of one scenario are included in the comparison graph, if the decision variable has a non-zero value, cost impact or both. For constraint nodes, we can use the same approach for calculating the difference in the attribute vector, but only include the constraint nodes that share an edge with a relevant decision variable node in the comparison graph.

With the aforementioned considerations, we can define our comparison graph formally as follows. Note that the two node sets are not treated equally in the definition: The decision variables define the primary structure of the comparison graph.

Definition 4.9. Let $G = (U_G, V_G, E_G), H = (U_H, V_H, E_H)$ be two augmented digraphs of the same model, where U_G and U_H are the sets of constraint nodes in

G and H, while V_G and V_H are the sets of decision variable nodes in G and H, respectively. The **comparison graph** is the graph $C = G - H = (U_C, V_C, E_C)$, where $V_C = \{n_C \in V_G \cup V_H | \, \mathbf{s}_{n_C} \neq \mathbf{0}\}$, $E_C = \{e_C \in E_G \cup E_H | \, \exists n_C \in V_C : n_C \in e_C\}$, and $U_C = \{n_C \in U_G \cup U_H | \, \exists e_C \in E_C : n_C \in e_C\}$.

In the comparison graph, the set of decision variable nodes V_C is the subset of all decision variable nodes in the union of V_G and V_H for which the attribute vector difference is nonzero. This in turn defines the set of edges E_C, which is the subset of the union of E_G and E_H which include any node in V_C as one of its end points. Finally, the set of constraint nodes U_C is the subset of all constraint nodes in the union U_G and U_H, for which there is at least one edge in E_C that the node is a part of.

Regarding the information in these comparison graphs, there is valuable information we can include in the node and edge dimensions: As described previously, the comparison graph can include nodes or edges that are only part of one scenario, and we want the comparison graph to include information on which solution each node is from. Usually, the vast majority of the nodes should be the same between scenarios, since otherwise we are comparing models whose structures differ significantly, and direct comparisons are not a reasonable approach. However, we can generally allow some decision variables and constraints to be different between scenarios. This allows analysing the effect that these nodes have on the solution, especially if other model parameters are not simultaneously changed. An example would be setting a minimum production constraint for some product type, in which case our comparison graph would show how the solution changes due to the addition of the constraint. Another special case is when a decision variable could have potentially been included in the optimal solution for both scenarios, but the optimum value is zero in one of them. Such changes are of particular interest, because they relate to decision variables that are relevant in only one of the scenario solutions. In both of these cases, we can include this information as additional key-value pairs to each node's dimensions: For example, consider again a node n_G that is in G but not in H, and a node n_B that is in both G and H. We may then add to each node's dimensions in the comparison graph a special key such as '*sol*', and set the corresponding value based on which scenario the node is from. We thus have in this case that $D_{n_G}(sol) =$ '*G*', and $D_{n_B}(sol) =$ '*Both*'.

The rest of this section will explain the algorithm used to create the comparison graph in practice. The pseudocode for this algorithm is given in appendix C. In the algorithm, we refer to the attributes vector of node or edge e of graph G as $G[e]$. In order to simplify notation, we do not make a distinction between numeric attributes and non-numeric dimensions. We assume the non-numeric dimensions to be equal between the scenarios (for nodes that are included in both scenarios). The algorithm consists of two main steps: First, we perform the node-by-node comparison for all nodes. Denoting the graphs as in the definition of the comparison graph, this comparison yields potentially three separate sets of decision variable nodes: $V_G \cap V_H$,

$V_G \setminus V_H$ and $V_H \setminus V_G$. Each decision variable node in the comparison graph belongs to exactly one of these sets, and we update the node's dimensions based on which set it belongs to. As the difference in attribute vectors can be calculated only for nodes in the first set, for nodes in the other sets we calculate the difference by setting the missing attribute vector to zero, as described previously. For decision variables in both scenarios, we also check if the optimum value is zero in one of the scenarios, and update the node dimensions accordingly. We then include only those decision variables where the difference in attributes is non-zero. We apply similar same steps for constraint nodes, but without this inclusion step. After all relevant decision variable have been included, we proceed to creating the edges between nodes.

For each decision variable node in the comparison graph, we proceed differently based on which set the node belongs to. For each node in $V_G \setminus V_H$, we know that all edges can only be part of G, thus we can directly add all edges to the neighboring constraint nodes, and update the edge dimensions accordingly. Similarly for nodes in the set $V_H \setminus V_G$, all edges are known to be only part of the other scenario with graph H. For decision variable nodes in the set $V_G \cap V_H$, the situation is slightly more difficult. While most edges are again part of both scenarios, we allow for the possiblity of some edges belonging to only one of the scenarios. For example, consider some external production constraint that limits the production of certain products. The same constraint may be included in both scenarios, but the decision variables linked to this constraint may not be the same across the scenarios. Considering some specific node, its neighboring edges are in one of the following three sets: $e_G \cap e_H$, denoting neighboring edges from both scenarios, $e_G \setminus e_H$, denoting neighboring edges only from G, and $e_H \setminus e_G$, denoting neighboring edges only from H. Note that neighboring edges refer to both incoming and outgoing edges in this case, as there is no need to further differentiate between these two sets. The edge comparisons between the scenarios can then be performed in a similar fashion to the node comparisons: For edges in both scenarios, we directly calculate the difference between numerical attributes, and for other edges we update the edge's dimensions accordingly, and calculate the difference by setting the missing numerical attributes to zero.

After the edges between nodes have been created, there graph may still include isolated constraint nodes with no incoming or outgoing edges. These are constraints that are not directly related to any of the changes between scenarios. Thus, as a final step, we remove all isolated nodes from the comparison graph. As the end result, we obtain the comparison graph, which can then be used as the basis of analysis similarly to the solution graph of a single scenario. Note that this procedure does not guarantee that the comparison graph is fully connected: there is a possibility of producing a disjoint union of subgraphs. If such subgraphs exist, this implies that there are some changes that are not directly related to each other, and these changes can be analysed separately.

4.5 Detecting substitutions

As previously discussed in Section 3.2, one pattern that can be utilized in scenario analysis is finding potential substitutions between the decision variables in two scenarios. To automatically find these, we have created a special substitution detection algorithm. While we believe there exists no exact mathematical definition for this occurrence, we give an explanation of our approach followed by a formal definition.

We define substitution as two opposite changes of roughly equal magnitude, that is, of similar absolute value. The chosen attribute can be either the optimal values or the total cost impacts of the decisions, depending on the context. These are not limited to pairwise switches, as one of these changes may also be the composite of several smaller changes. Our algorithm relies on two additional assumptions: 2) The substitutions must happen in the same node type, and 3) only nodes with marginally equal dimensions are considered. The second assumption means that different types of decision variables or constraints should not be compared, while the third assumption requires some further explanation. We formally define marginally equal dimensions as follows:

Definition 4.10. Two nodes n_1 and n_2 of the same type in the comparison graph have **marginally equal dimensions** if there is exactly one key for which the node dimensions have different values: $\exists! k \in K_{n_1} : D_{n_1}(k) \neq D_{n_2}(k)$.

Note that since the keys are always assumed to be common between nodes of the same type, the key in the definition can come from either of the nodes. We will further motivate this assumption through an example. Consider a decision variable for the quantity of product P1 sent to customer C1 from warehouse W1. Then, if this quantity has changed between scenarios, one might ask questions such as: i) has the quantity of product P1 sent to customer C1 from warehouses other than W1 changed, ii) has the quantity of product P1 sent from warehouse W1 to customers other than C1 changed, or iii) has the quantity of products other than P1 sent from warehouse W1 to customer C1 changed? In all of these cases, our goal is to find the changes where all but the one dimension in question are the same. For example, for i) we would find all changes related to the same customer C1 and product P1, but a different warehouse. Using definition 4.10 and the aforementioned conditions, we define substitutions in the context of the comparison graph as follows.

Definition 4.11. Let n_1 be a node and N some set of nodes in a comparison graph, and denote the attribute vector of node n as $\mathbf{s}_n$. Then, n_1 and N can be considered as potential **substitutions** in attribute i if the following conditions apply:

1. The attribute i of n_1 and sum of the attribute in N are opposite and roughly of equal magnitude, and all nodes in N have the same sign:

$$\mathbf{s}_{n_1}[i] + \textstyle\sum_{n \in N} \mathbf{s}_n[i] \approx 0, \;\; \mathbf{s}_{n_1}[i]\mathbf{s}_n[i] < 0 \; \forall n \in N.$$

2. The node n_1 and all nodes in N are of the same type.

3. The node n_1 and all nodes in N have marginally equal dimensions.

Given the comparison graph of two scenarios, the user of the algorithm can select a desired node type and numerical attribute, within which these substitutions are searched for. This can be applied at any desired level of aggregation, using the same hierarchical mapping as in the creation of the graph. As a prerequisite, the comparison graph must include information on node types and dimensions, and optionally on potential hierarchies. Furthermore, the user can select a desired tolerance for the maximum difference that is still to be considered as a substitution effect: For example, the user might establish that the positive change must account for at least 90% of the corresponding negative change.

The procedure of the algorithm is to first select only the nodes of the desired type from the graph. Then, if the comparisons are desired at some specific aggregation level, we remove the attributes not part of the aggregation level: For example, if the nodes are related to products, we may remove a product ID dimension, if we want to focus on product families instead of individual products. We then end up with some nodes with duplicate keys, and we can simply sum their attributes together. This aggregation approach is similar to the one suggested in Section 4.8, but here we do not directly alter the structure of the underlying graph.

Then, for each of the nodes, we search for other nodes of marginally equal dimensions, where the direction of change is opposite to the original change. The algorithm then attempts to find some combination of these changes that is closer to the original change than the specified tolerance. This is done by taking a cumulative sum of the changes in decreasing order of magnitude, and checking this sum every time a new change is added to it. If the cumulative sum of changes is closer to the original change than the specified tolerance at any point of this process, then these are reported as potential substitutions to the user. It is also possible that there are multiple potential combinations for substitution, but the current version of the algorithm only returns the first one that is found. The pseudocode for this algorithm is given in appendix D.

4.6 Automatic report writing

An additional functionality often seen in IMPS is the capability of producing some automatic reports to the user. Such reports can further lighten the burden of the analyst by automatically calculating or otherwise presenting some meaningful information of the scenario, so that the analyst does not have to spend additional time trying to find these specific pieces of information manually. These can be especially powerful in situations where similar pieces of analysis would be repeated for different scenarios.

Naturally, there are many potential points of interest when analysing supply chain scenarios, thus no single report can address all potential questions a decision maker may have. Nevertheless, some pieces of information can certainly be generalised, at least in the context of the supply chain. Focusing on the capabilities of the model oriented approach, the simplest informative reports that can be written are those related to constraint information. For example, on the production side, we can report the capacity bottlenecks that are occurring in the supply chain, or contrastingly where some capacity is left unused. On the customer side, we can report unsatisfied demands for specific products or customers. Such reports can be directly generated from the solution graph, and the user only has to specify which node types (machine capacity, customer demand) they want to see.

In order to fully utilise the capabilities offered by the model oriented approach, reports based on more complex relationships can be created on top of the solution graph. These reports are more oriented towards answering specific questions the decision maker may have, and they often need to be more customised to the specific optimisation model. An example of such a report is ranking the most critical raw materials in the supply chain, in terms of total sales of products that are reliant on the specific raw material. Another example is ranking the unmet demands in the supply chain, in terms of total profits that are currently not obtained.

The main difference between these automatic reports and ad hoc queries made into a graph database is that these automatic reports are designed such that they attempt to fully answer some pre-specified question, whereas the ad hoc queries are more flexible but also require the analyst to specify the model relationships they are looking for.

4.7 Visual network analysis

In scenario analysis, the ability to simply visualise the model graph as a network of nodes is potentially useful, especially in cases where the main concern is to assess the impact of different decisions in one scenario, or the various changes that have happened between scenarios. In this visual graph representation, we can perform different types of highlighting and manipulation based on the subject of interest. For example, we can make node size reflect their respective cost coefficient or cost impact, thus drawing attention to the nodes where most significant cost contributions happen. Another possibility is considering the actual values of the decision variables, which correspond to volumes flowing through the supply chain. Thus, highlighting decision variable nodes based on their values could be used to show the most significant flows of materials through the supply chain. These can be applied both when analysing a single scenario, in which case we visualise the individual solution graph, and when comparing two scenarios, where we visualise the comparison graph instead. There is obviously a slight difference in the interpretation of these cases: with a single scenario, we are highlighting the most impactful decisions in terms of cost or volume, and with two scenarios we are actually considering the most impactful changes regarding these decisions. In both cases, these visualisations can provide the analyst with a meaningful overview of the model results.

Although the emphasis of costs or volumes are perhaps the most obvious use cases, there are many other possibilities of using network analysis tools in a model analysis setting. For example, in single scenario analysis, the information saved on constraint nodes can be used to highlight binding inequality constraints. In the case of less-than constraints, these often indicate bottlenecks in the supply chain, such as limited warehousing or production capacity. Emphasising these bottlenecks can provide the decision maker with useful information on how the supply chain operates, at least according to the underlying optimisation model. Furthermore, these results can provide ideas for new interesting scenarios that could be explored.

In scenario comparison, nodes can be filtered based on the flags that indicate in which scenarios the node is active (non-zero) or even present at all. This would allow the analyst to observe which parts of the supply chain are most affected by the change between scenarios, not in absolute terms but rather by showing which previously unused options have become attractive, or which previously attractive decisions are no longer made. This can highlight many interesting facts, such as previously unused production capabilities that have become active, or previously unmet customer demands that are now being satisfied. Similarly, we may highlight production decisions that are no longer made, or customer demands that are no longer profitable enough to satisfy.

Furthermore, any of the other node attributes can be used in filtering the network to show only the specific parts of the graph that the analyst is interested in. For example, if the analyst wants to know what is happening to a specific product, customer, or raw material, they can filter the graph to show only the nodes with matching information on the desired attribute, such as a specific product identification code. Such filtering can be very useful if the interest of the decision maker can be narrowed down to a more detailed level that corresponds to some of these attributes. However, the limitation with this filtering approach is that it does not include nodes without the relevant attributes, even though they might be considered relevant otherwise.

Some of the aforementioned visualization capabilities are illustrated in figures 7, 8 and 9 below. These figures are visualizations of a particular supply chain related scenario comparison graph, the details of which are omitted here. In all of these figures, the size of the nodes is scaled relative to their cost impact, such that decisions with larger cost impacts (either positive or negative) have a larger size. Although the figures are all related to the same scenario comparison, they highlight different factors.

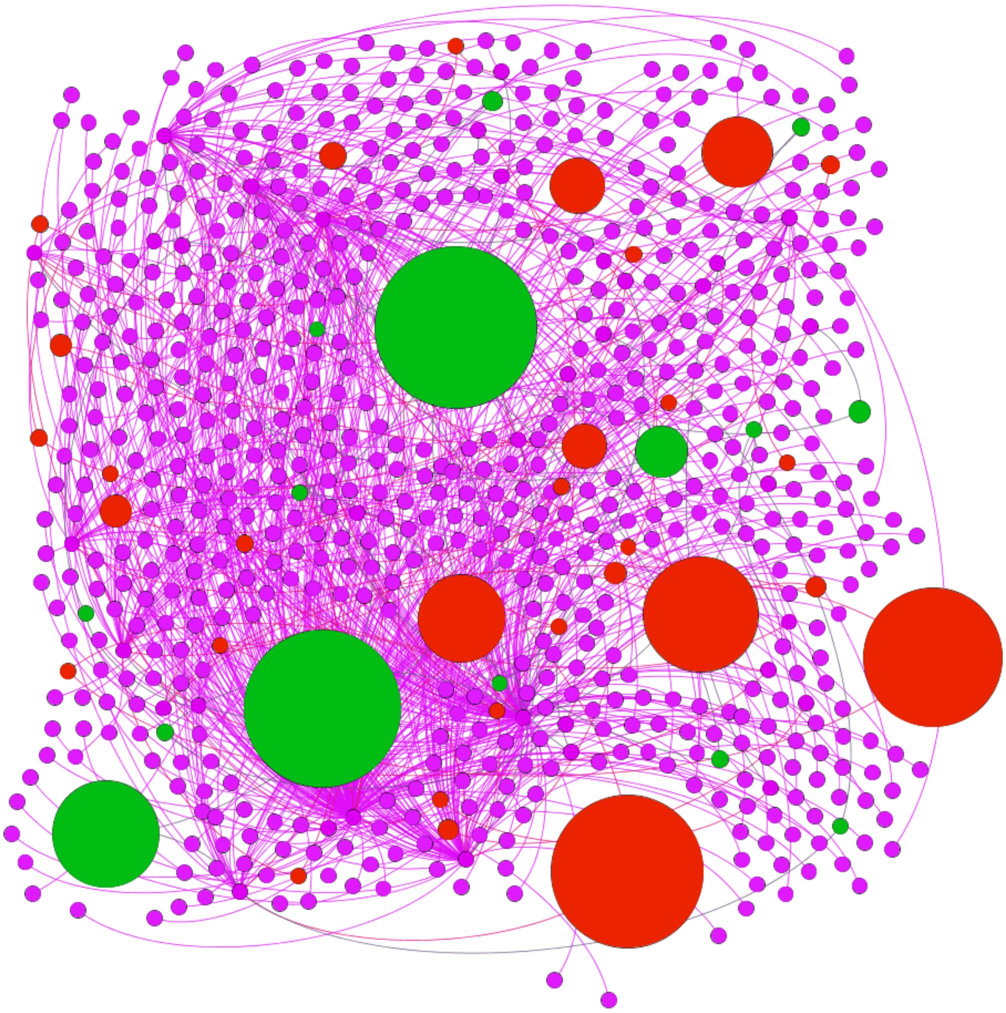


Figure 7: Example visualization of cost impacts

In Figure 7, the direction of cost impact is highlighted with a specific color: Positive and negative changes are shown with green and red, respectively, indicating changes such as increased or decreased sales of certain products. The third color is used for nodes with no explicit cost impacts. A figure of this type shows how the different changes contribute to the total cost difference between the scenarios. In this example, there is only a handful of changes with significant cost impact, and while there are few decisions with large positive cost impacts, these are largely offset by the negative cost impacts.

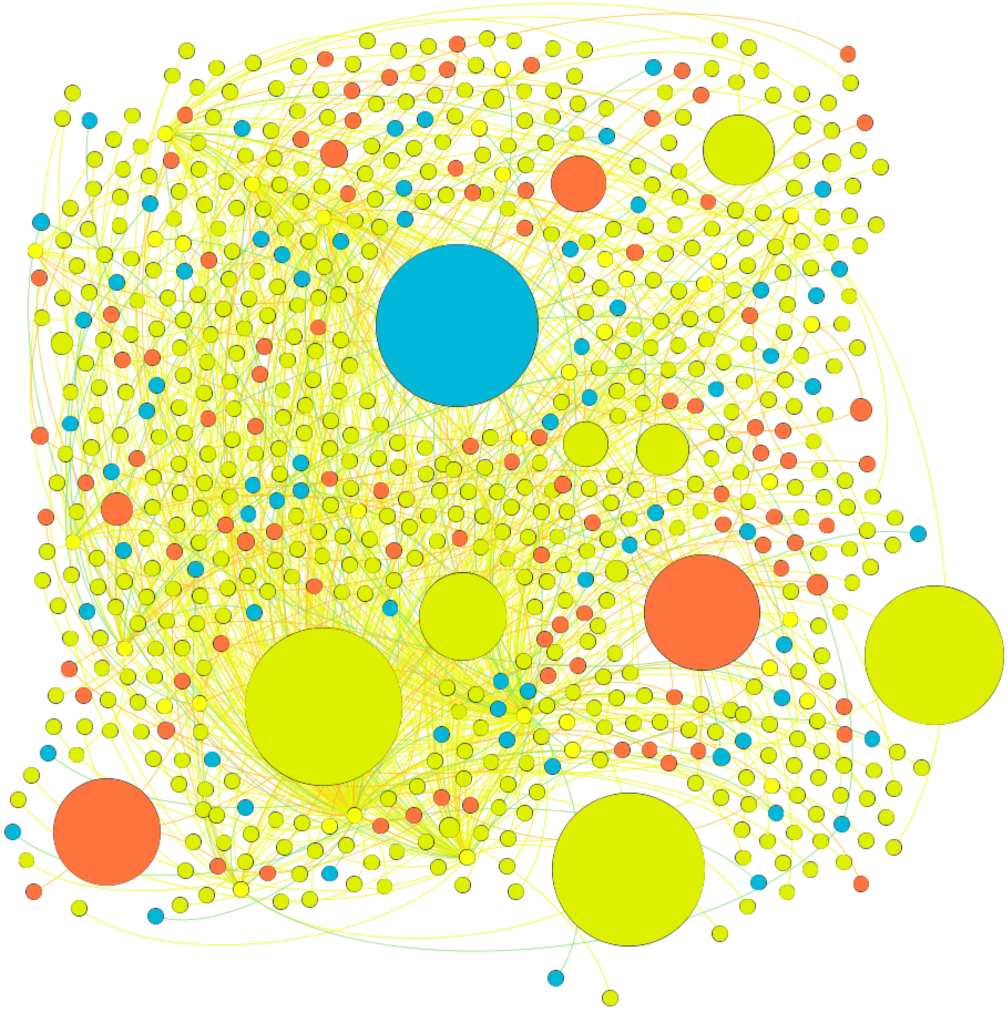


Figure 8: Example visualization of active decisions

In Figure 8, the same comparison graph is highlighted in a different manner. Here, the colors indicate the related scenario for each decision: The yellow nodes are those that are active in both scenarios, whereas the the orange and blue nodes indicate decisions that are only made in the first or the second scenario, respectively. This figure gives us a different view into the scenario differences: In this example, we observe that some of the largest changes in terms of cost impact are only related to one of the scenarios, indicating that some significant production changes should occur in order to minimize costs.

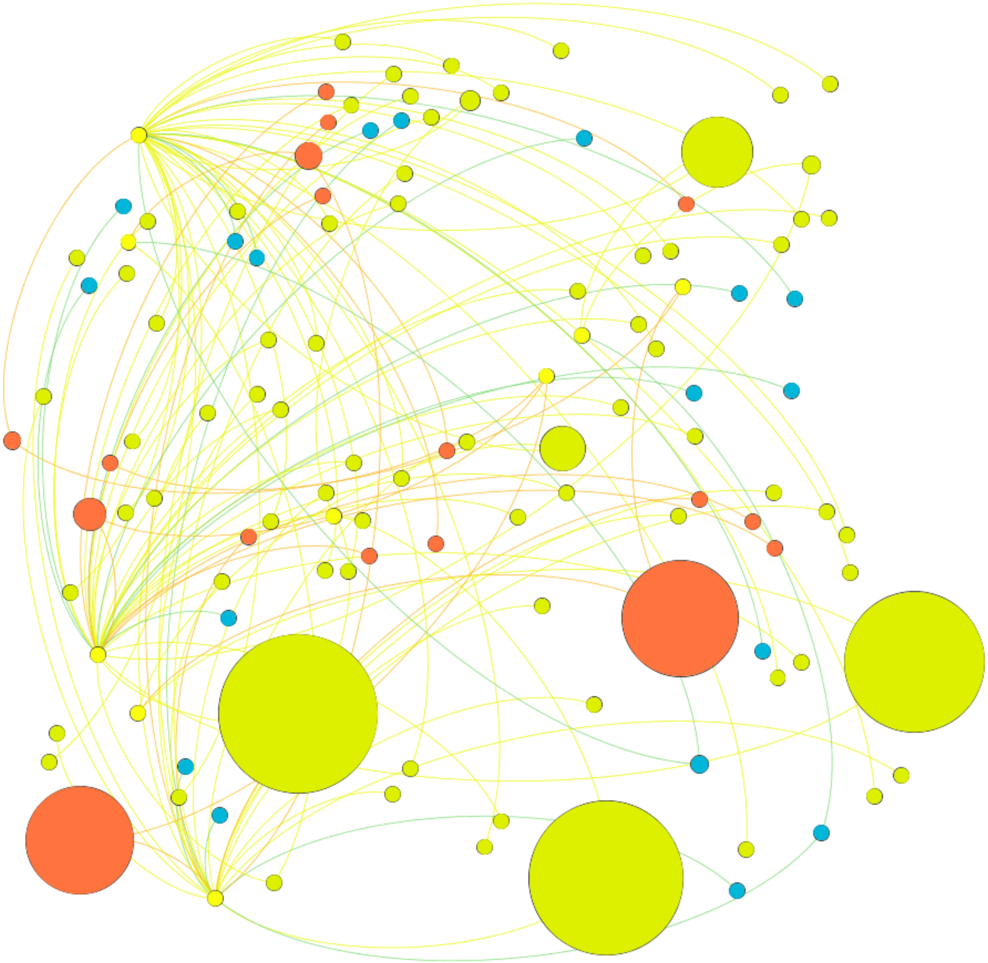


Figure 9: Example visualization of filtered graph

In Figure 9, the colours are the same as previously, but the graph is filtered to include only decisions and constraints related to a specific product, e.g. product P1 as in the previously considered examples. This filtering makes the resulting graph more sparse, as only decisions and constraints directly related to product P1 are shown. This also makes the results easier to interpret, as there is less clutter from nodes that are not interesting to the decision maker. In this example, we observe that many of the impactful decisions are actually related to the product that we have chosen to filter here, which means that the production and sales of this particular product are largely affected by the scenario changes.

4.8 Aggregation of graphs

As discussed previously, supply chains are often embedded with hierarchical structures, such as products belonging in product families and customers in different regions. In many situations, we may not be interested in specific changes in a single decision variable, but would rather know what the important changes are on a higher hierarchical level. Here, we consider one possibility of comparing results hierarchically using a process known in graph theory as node contraction, or vertex identification (Pemmaraju and Skiena, 2003). We will first give a formal definition, and then explain how this process could be used in diagnostic aggregation.

Definition 4.12. Let u, v be a pair of nodes in the graph $G = (V, E)$. The **contraction** of nodes u, v consists of merging them together into a new node w, such that w is adjacent to the union of nodes to which u and v were adjacent. This results in a new graph $G' = (V', E)$, where $V' = (V \setminus \{u, v\}) \cup \{w\}$.

We assume that the nodes are not required to share an incident edge between them. Note also that in general, this operation may result in a graph with self-loops or multiple edges between nodes, even if the original graph did not contain these.

Consider the graph $G = (V, E)$, that contains all decision variable changes as well as the related constraints. Let us assume that we would like aggregate the results over some specific hierarchical dimension h that is one of the keys for node dimensions, related to an original dimension d. Aggregation over this hierarchical node dimension is then equal to merging together any two nodes of the same type n_1,n_2 that i) have the same value in dimension h: $D_{n_1}(h) = D_{n_2}(h)$, and ii) have the same values in all other dimensions except the original dimension: $\forall k \in K_{n_1} \setminus \{d\} : D_{n_1}(k) = D_{n_2}(k)$. In other words, we combine nodes of marginally equivalent dimensions, where the marginal difference is in dimension d. After the two nodes have been merged together into a new node, the edges incident to this new node correspond to the edges that were incident to the original nodes. This aggregation process can be done by first removing the original dimension d from all nodes that contain it, after which some nodes have identical dimensions. Then, for each subset of nodes that have identical dimensions, we can perform pairwise node contraction in any order, until there is only one node left in the subset.

In the comparison graph setting, the process of node contraction can be applied to the comparison graph to produce a smaller graph, where each node corresponds to an aggregation of several decision variables or constraints. Naturally, after such operation the result is no longer an exact representation of the underlying system. However, in the case that we are interested in the changes on an aggregate level, this operation may indeed provide us with a simpler and more manageable representation of the original problem.

One important and nontrivial question is what node contraction does to node attributes. There is no clear answer as to what should happen, since by definition of node contraction, the result of contracting two nodes is always a new node, which is simply incident to the union of nodes that were indicent to the old two nodes. However, in an optimisation setting, we are minimising some objective function, and the cost effects of each change on the graph is known. Therefore, a natural numerical representation of the cost effect of a contracted node is to sum the individual cost effects together. With this notion, the aggregated graph remains consistent with the original one in the sense that the total difference in the objective function remains the same.

Clearly, this notion of summing the values together could be applied to all other numerical node attributes as well, although their interpretation may not be quite as straightforward. For example, the sum of cost coefficients between contracted decision variable nodes does not provide meaningful information. However, an average or median over these changes may be more informative. Following this idea, perhaps it would be possible to define a separate aggregation function for all different types of numerical node attributes, such that the result from aggregation is then conducive to the analysis of these aggregate results.

However, it is also possible that some attributes cannot be reasonably described in aggregate form. Consider, for example, the non-numeric attributes that flag the solutions in which the decision variable was active. Then, after performing node contraction over some dimension, the resulting supernode may include nodes that were active in different solutions. There is no clear answer what would be the best form of aggregation.

In the case when the optimisation problem consists of several time steps, one special form of aggregation would be to aggregate the results over time. If we are not interested in what happens in the model at a specific time step, but rather want to know what the model does on a more aggregate level, then this approach can simplify the graph by making it considerably smaller. However, as with other forms of graph aggregations, there are potential cases where such aggregation is problematic: For example, if some model parameters change over time, we lose the information regarding when these changes happen.

5 Case study

The case study focuses on paper supply chain optimisation which will be described first in detail. Then, the use of developed IMPS is illustrated along with some insights from the analyst who has developed the model and presented its results to case company managers earlier.

5.1 Supply chain description

Figure 10 shows the paper supply chain in a graphical form. Notice the similarity with the general example shown in Figure 2: The main difference is that the amount of differents supply chain entities has been fixed, and production facilities are referred to as mills.

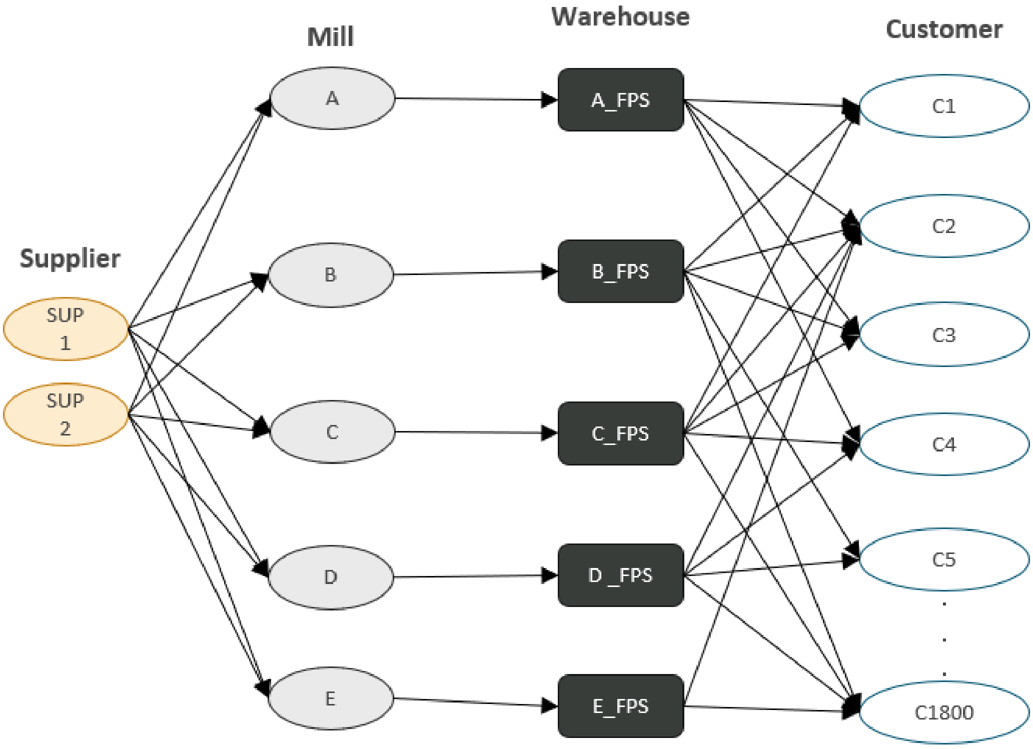


Figure 10: Case study paper supply chain

In terms of different supply chain planning phases described in Section 2, the model is an example of master planning, with the added ability to analyse the impact of different first tier decisions as well. The optimisation model contains the supply chain from the manufacturer's point of view. The chain begins with suppliers, who supply different raw materials to mills in different regions. Each mill has 2-4 machines of limited capacity, and these machines are used to produce various products using machine- or mill-specific production recipes. These recipes each have their own specific raw material requirements, and some products can be produced with multiple recipes. The products are then processed into finished products and sent to warehouses. From the warehouses, the products are finally sent to customers. Specifically for the customer and product dimensions, there is additional hierarchical information: Each customer belongs to some specific region, and each product is part of a larger product family.

Each decision variable and constraint in the model is related to a monthly time period, and the model includes twelve monthly periods. The overall objective of the optimisation model is to maximise the total sales margin by allocating the estimated customer demand to the different production facilities and their machines. In other words, the goal is to maximise profits obtained by sales, taking into account the variable costs related to factors such as raw material procurement and logistics.

In more detail, the model includes the following variables:

- P_{khqt} - Amount of raw material k purchased from supplier h to mill q in period t
- D_{krt} - Amount of raw material k consumed on recipe r in period t
- Q_{rit} - Amount of product i produced with recipe r in period t
- I_{pit} - Stock level at finished product storage p for product i at the end of period t
- X_{ipct} - Customer c's demand for product i, satisfied by finished product storage p in period t

The following constraints are included in the model:

$$\text{Machine\_material\_flow}[k,q,t] : \sum_{h} P_{khqt} = \sum_{r \in R(q)} D_{krt} \tag{7}$$

$$\text{Machine\_production}[k,r,t] : D_{krt} = p_{kr} Q_{rit} \tag{8}$$

$$\text{Machine\_capacity}[l,t] : \sum_{r \in R(l)} s_{lr} Q_{rit} \leq m_{lt} \tag{9}$$

$$\text{Storage\_flow}[p,i,t] : \sum_{r \in R(q,i)} Q_{rit} + I_{pi(t-1)} = \sum_{c} X_{ipct} + I_{pit} \tag{10}$$

$$\text{Demand}[i,c,t] : \sum_{p} X_{ipct} \leq d_{ict} \tag{11}$$

Here, $R(q)$ denotes the recipes that belong to the mill q. p_{kr} is the amount of raw material k needed to produce one unit of recipe r. $R(l)$ is the set of all recipes that can be produced with machine l, s_{lr} is the capacity used with machine l for each unit of recipe r, m_{lt} is the maximum capacity of machine l at period t. $R(q,i)$ denotes all recipes that can be produced in mill q whose end product is i, and finally d_{ict} is the customer c's estimated demand for product i at period t.

The constraints (7) state that all material bought to the mill at each time period must be consumed by some recipe that the mill is using during the same period. Constraints (8) state that each recipe has a specific conversion rate from each raw material to product. Constraints (9) state that each recipe has some capacity requirement for each unit produced, and that the total capacity used should not exceed the given maximum capacity. Constraints (10) state that the flows of material through each finished product storage should be conserved. Finally, constraints (11) state that there is some specified demand for each customer, product and time period, and that this demand is an upper limit of what can be satisfied through the different storages. In addition to these physical constraints, there are additional virtual constraints, which are not discussed here. The exact details of the model parameters are not crucially important: Rather, the key is to understand how the different model components are connected to each other. The immediate connections between the nodes of the model can be seen in the constraints, as they show exactly which constraint is linked to which decision variables.

In this study, we consider both the case where we only have one scenario, and the case where we compare two scenarios. All of the single-scenario examples refer to the same baseline scenario, where the supply chain is operated according to the best current information on all related factors, such as demand forecasts and raw material availability. For the two-scenario comparisons, various different scenarios based on real business questions are compared against this baseline scenario. The objective here is to broadly illustrate different possibilities of analysing these model results by using the functionalities implemented in the IMPS. Without explicitly mentioning this in each separate case, when we only have one scenario, the graph is constructed by parsing the corresponding MPS and solution files, and then running the algorithm described in Section 4.3. When comparing two scenarios, we first create the single graphs, after which we create the comparison graph using the algorithm described in Section 4.4. In addition, we include the node types, their respective dimensions and the hierarchies for the product and customer dimensions as additional properties for each node in the graphs. These steps enable us to perform the analyses described below.

5.2 Visual network analysis

Figure 11 shows a visualisation of a complete scenario comparison graph between two supply chain optimisation scenarios for the model. This visualisation was created

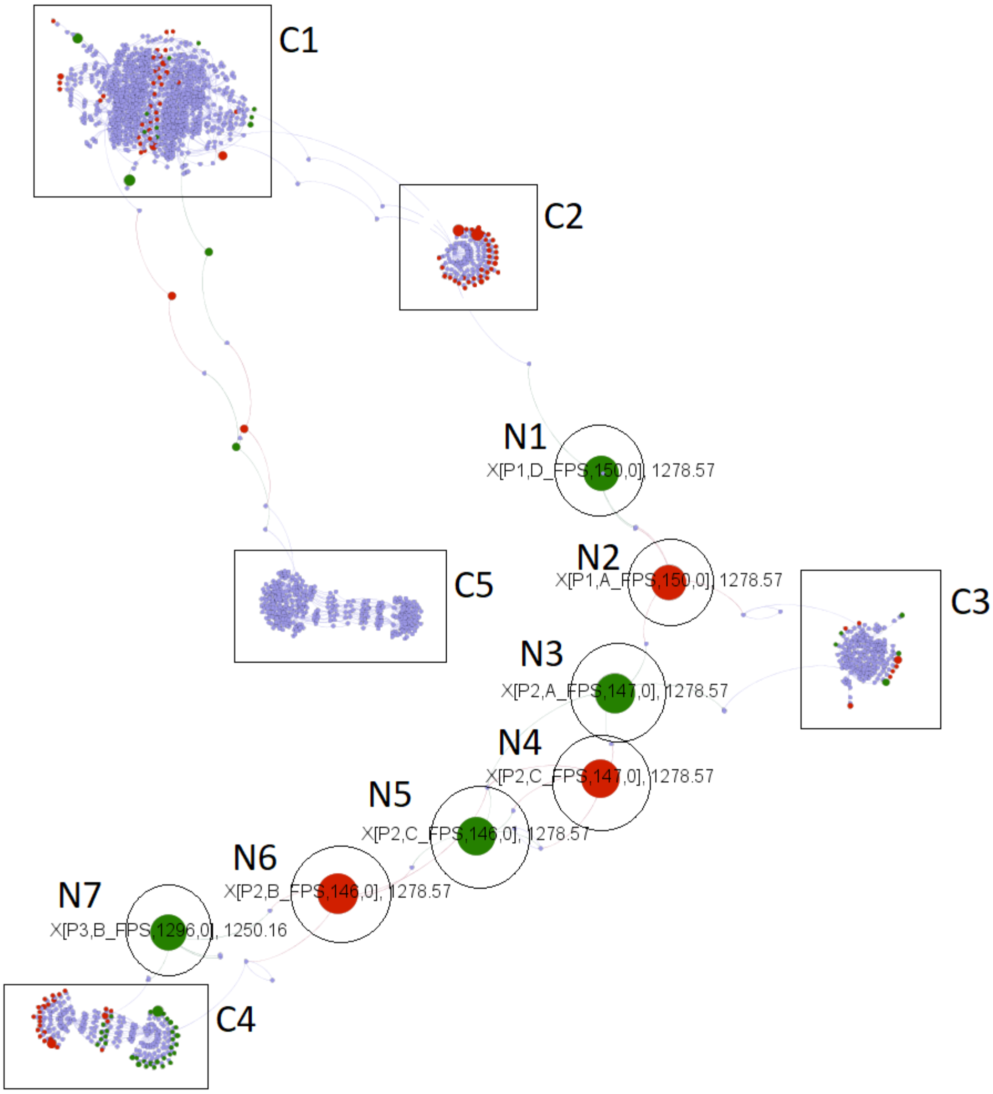

C1
C2
N1
X[P1,D_FPS,150,0], 1278.57
N2
X[P1,A_FPS,150,0], 1278.57
C5
C3
N3
X[P2,A_FPS,147,0], 1278.57
N4
X[P2,C_FPS,147,0], 1278.57
N5
X[P2,C_FPS,146,0], 1278.57
N6
X[P2,B_FPS,146,0], 1278.57
N7
X[P3,B_FPS,1296,0], 1250.16
C4

Figure 11: Scenario comparison graph

in Gephi, using one of their preset layout algorithms. The main difference between these scenarios is that in the second scenario, the production capacity of one of the machines in mill D was increased in the beginning of the first period, compared to the baseline scenario. The decision maker is interested in the additional profit the new capacity would generate. This can be answered by comparing the total objective function values between the scenarios. However, the decision maker is also interested in how this additional capacity would actually be used, and the comparison graph shows in detail how individual decisions change. Considering the taxonomy described in Section 3.2, this question is an example of a question belonging in the category (Two, Complex).

In this specific visualization, the size of decision variable nodes is relative to their absolute cost impact, and colored based on the direction of change: Green color denotes positive cost impacts, either through less costs or more profits, and red color denotes negative cost impacts through more costs or less profits. Nodes of the third color are either constraints or have no explicit cost impact. While there are clearly quite many changes that happen as a result of the capacity increase at mill D, attention is immediately drawn to the few nodes with significantly larger size than others: these are decisions on how specific product demands for customers are satisfied from different warehouses. The labels of these nodes have been highlighted along with the absolute change in the decision variable values: Since more products sold corresponds to more profits and vice versa, the colors highlighting cost impacts correspond to the change in volume. Note that the opposite is true for decision variables for raw material purchases: Increased production volume increases the need of raw materials, leading to increased costs. The nodes between the highlighted nodes are the demand satisfaction constraints, showing how these changes in decision variables are linked together. The hubs of smaller nodes marked as C1 to C5 in the Figure are the various production and material flow constraints for each production facility, as well as the decision variable nodes related to changes in raw material purchases.

From the graph and the labels, we observe that the increased capacity at mill D is used mainly to increase production of product P1, which is sent to customer 150 from mill D's warehouse. This decision corresponds to the node marked in the Figure as N1. Increased production of P1 leads to increased purchases of some raw materials: the cost impacts of these decision variable changes are shown as the hub of red-colored nodes marked as C2, connected to the aforementioned node through the model constraints. Interestingly, the volume of P1 sent from warehouse D_FPS to customer 150 matches exactly the amount that was previously sent from warehouse A_FPS. Now, mill A produces less product P1 (node N2) and more product P2 which is sent to customer 147 (node N3). While the changes in volumes seem to match across the products in this case, the raw material requirements are not quite the same as they were previously, shown by the hub of smaller nodes marked as C3, connected to the these two nodes.

Continuing the chain of nodes further, we observe that the demand for customer 147 was previously satisfied from warehouse C_FPS (node N4). Now, the same volume of the same product is sent to customer from C_FPS to customer 146 (node N5). In this case, there are actually no production changes at mill C, but the end customer for the product changes. Finally, we see that customer 146 previously received the same volume of product P2 from warehouse B_FPS (node N6). Now, the freed capacity at mill B is used to produce more of product P3, which is then sent to customer 1296 (node N7). The raw materials required only for product P2 are bought less, whereas the materials required only for product P3 are bought more. This is shown by the connected hubs of smaller green and red nodes marked as C4. As an overall result, the graph representation shows to the decision maker that increasing capacity at mill D is most valuably used by satisfying demands that were already met from other mills, and through several links throughout the supply chain, increased profits are mostly generated by satisfying previously unmet demand through a different mill. These sorts of insights are valuable to the decision maker when planning capacity increases or any other related activities where an overview of the scenario differences is required.

5.3 Ad hoc queries

In this section, we illustrate how the graph database representation can be used to answer more complex questions related to analysing the supply chain, using different ad hoc queries on the graph database. We first present a few examples for the single scenario case, then a few examples for the two-scenario comparison case. All of the examples shown use the syntax of Cypher, the native graph query language used by Neo4j. An extensive user manual, covering the language in detail, is maintained by the Neo4j team and can be downloaded from their website at `https://neo4j.com/docs/`.

One question considered is finding the common material suppliers for products P1,P2 and P3 in the baseline scenario, which is an example of a question of category (Single,Complex) in our question taxonomy described in Section 3.2. In this case, the problem can be broken into the following steps: First, find the decision variable nodes that are related to the production of the specified products. Then, find all direct paths from this node to decision variable nodes that are related to the procurement of raw materials. From these nodes, collect the distinct supplier identification codes for each product. As a result, we then obtain a list of suppliers for each product of interest, which we can further check for differences or commonalities. Note that in this question, the interest is not in exact values or cost impacts, but finding what connections exist between the production decisions and suppliers.

In a graph database, performing such a query is quite straightforward. One needs only the ability to describe the path of decision variables and constraints that is assumed between the end points. In this case, the path is through the production decision variable (Q), a Machine production constraint, raw material consumption variable (D), Machine material flow constraint and finally the decision variable for purchasing of raw material (P). The corresponding query with Cypher is as follows. As a result, we obtained that all these products use the exact same supplier, which was the expected outcome.

```
WITH [P1,P2,P3] as products
MATCH (q:Q)-(:Machine_production)-(:D)-(:Machine_material_flow)-(p:P)
WHERE q.ProductID in products and q.value <> 0
RETURN q.ProductID, collect(DISTINCT p.SupplierID)
```

Another interesting question of the same category was the following: Which of our customers would potentially suffer in the baseline scenario as a result of a shortage in raw material R1? In order to answer such a question, we obviously have to first find out which products require this raw material. This alone requires the ability to link all possible paths from the relevant raw material procurement nodes to different production nodes, in a similar fashion to the previous example. In addition, we have to proceed from these production nodes to the storage flow nodes, and finally to the nodes that show which customers receive the related final products. Overall, this question amounts to finding all related paths through the supply chain, all the way from raw material purchases to the selling of finished products to customers. Similar to the previous question, the main interest is in the connections, not exact values or cost impacts, although these could be calculated if necessary. While slightly more complicated than the previous example, the corresponding Cypher query is still quite manageable. Denoting the raw material of interest as 'R1', the corresponding query is as follows. This query showed that there was a large population of customers who were reliant on the availability of this raw material, which would suggest to the decision maker that there is significant risk of losing customers, should this shortage become a reality.

```
MATCH (x:X)-(:Storage_flow)-(q:Q)-(:Machine_production)-(:D)
-(:Machine_material_flow)-(p:P)
WHERE p.RMID = 'R1'
RETURN DISTINCT x.CustomerID
```

Note that answering these types of questions could be possible with a result oriented approach using relational databases. However, this would likely require several join operations between different tables. For the examples above, forming a comparable answer would likely include joining together multiple tables with information on the different supply chain entities, such as suppliers, raw materials, production facilities, as well as the relationships between them. For an entire supply chain, this approach can become unmanageable.

In scenario comparison, the decision makers were interested in a few questions that can be categorized as (Two,Complex) in our question taxonomy presented in Section 3.2. First, they were interested in knowing how additional capacity at mill D would be used in production, and ultimately increasing profits. We can begin answering such a question by starting with the capacity constraint nodes where the RHS attribute has increased. We first check whether this new capacity is actually used, by checking that the row sum attribute has increased as well. Then, from these constraints, we easily obtain the connected nodes, the decisions variables related to production, where the value and thus production amount has increased. In case we are also interested in where these products are shipped, we can include material flow constraints and customer- specific shipping decision variables. An example of such a query is given below. The result is a list of products and customers, with the corresponding volume increases. This is a similar example to the one presented in Section 5.2, but here we obtain a list of volumes, instead of a visual representation. Furthermore, this form of query does not show the entire chain of events through all different warehouses like the visual representation does.

```
MATCH (m:Machine_capacity)-(q:Q)-(:Storage_flow)-(x:X)
WHERE m.RHS < 0 and m.RSUM < 0 and q.value < 0 and x.value < 0
RETURN x.ProductID, x.CustomerID, x.value
```

Another question of interest was the case where raw material R1 has become unavailable, thus rendering some production recipes unusable. The contrast to the similar one-scenario case is that here, instead of trying to find the affected customers, our objective is to explore how this change actually affects how the supply chain should be operated. The natural starting point here is the raw material nodes that were flagged to be available only in the first solution, in other words with the dimension 'SOL1'. We then find the corresponding material flow constraints, through which we find the affected nodes related to resource consumption, production recipes, and ultimately the production quantity variables that are affected by this change. Such a query would be

```
MATCH (q:Q)-(:Machine_production)-(:D)-(:Machine_material_flow)-(p:P)
WHERE p.sol = 'SOL1'
RETURN q.ProductID, q.value
```

We thus find the directly affected decisions and the corresponding volumes, but there are multiple other questions that may arise as a result. First, if there are multiple recipes for the same product, then we might want to know if there exists a recipe that does not use these lacking raw materials. If such a recipe exists, then we can find out if it is actually used to supplement the previously available production capabilities. There are several possibilities that arise: It may be that such actions are not as profitable, and the unused capacity, assuming that capacities have not changed simultaneously, is used to produce some completely different products to satisfy some previously unmet demand. Another possibility is that no other products are profitable to produce, and the lack of raw materials results in excess capacity. In this example, we found that there were no additional recipes in use, and that the unused capacity would be mostly used to produce other products that were not reliant on the missing raw material. The increased production volumes were found out through an additional query, where we used the previously found production quantity decisions as a starting point. Referring to the list of affected product quantity decisions as `AffectedProducts`, the query is

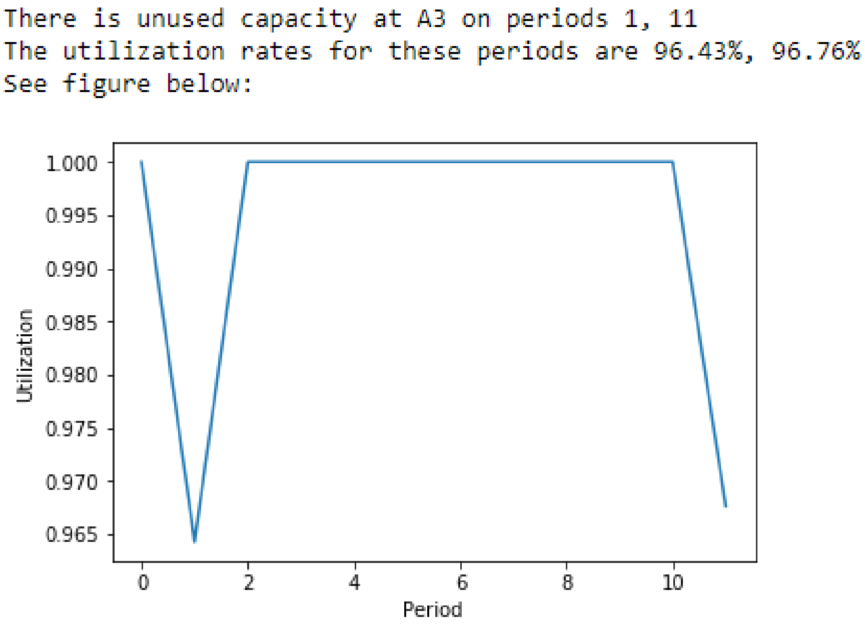
```
MATCH (q1:Q)-(:Machine_capacity)-(q2:Q)
WHERE q1 IN AffectedProducts and NOT q2 IN AffectedProducts
RETURN q2.ProductID,q.value
```

The example above illustrates the difficulty of finding a general approach that could answer any type of question, regardless of the underlying changes between scenarios. However, such queries can aid the analyst to find possible directions of further analysis, which can be further enhanced by chaining multiple queries together.

5.4 Automatic report writing

In this section, we demonstrate a few of the automatic reports that can be generated from the solution and comparison graphs. As previously mentioned, reports that require little model-specific knowledge can be made almost directly from the graph, whereas more complicated reports require some customisation. Nevertheless, the same graphs serve as the starting point for all the reports discussed here.

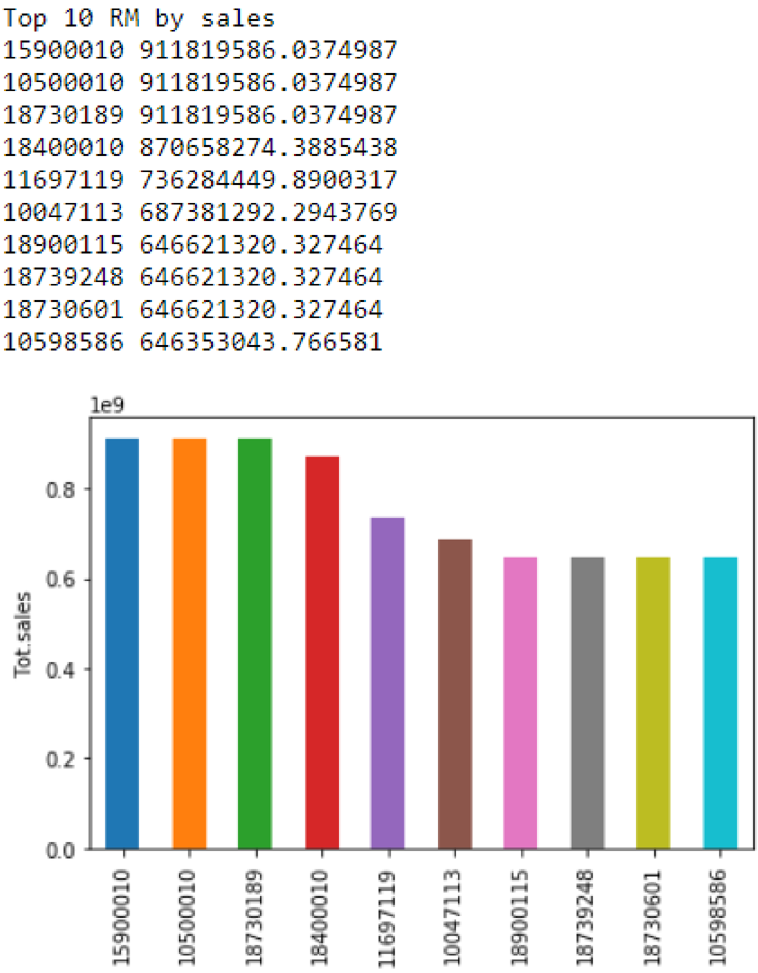
Since the operation environment of the case study is such that the maximum production capacity is often reached, the decision maker is interested in knowing if there are some machines that are specifically not operated with maximum capacity at some time periods. In terms of the question taxonomy presented in Section 3.2, this question is of category (Single, Simple), although it would be impossible to answer without storing the constraint information. The unused capacities can be reported directly from the solution graph by checking all machine capacity nodes, and comparing their respective row sum and right-hand side (RHS) attributes. In this example, the nodes also contain the identification codes for each specific machine and the current time period, which we can use to group and display the results separately for each machine over time. The baseline scenario has only one machine that is not completely utilised all the time, namely machine A3. When the corresponding solution graph is processed by our report writer, the output is the one shown in Figure 12. As a result, the analyst quickly obtains interesting information that they can then analyse further.



```
There is unused capacity at A3 on periods 1, 11
The utilization rates for these periods are 96.43%, 96.76%
See figure below:
```

Figure 12: Example report for underused capacity

An already previously mentioned example of a question of category (Single,Complex) would be finding the most valuable raw materials in terms of sales of products that require them. Creating such a report requires several steps: First, we must be able to calculate the cost impacts generated by each product from each warehouse, which can be done by summing the individual cost impacts of product sales. In other words, for each warehouse flow constraint, we sum the cost impacts of neighboring product sales nodes. Then, this total revenue must be allocated to the production decision variables, neighboring the same warehouse flow constraint on the opposite side. This can be done by considering the relative volumes produced by different recipes. Then, once the cost impacts have been allocated among different production variables, we traverse the graph from these production variables to each raw material supply node, and store the same (positive) cost impact of production to each of them. As a result, we obtain a report indicating which raw materials are involved in the most valuable sales, and what their relative impact is. For the example scenario, this report is shown in Figure 13. The results show that a few raw materials are clearly the most significant in terms of cost impact, suggesting that their availability plays a significant role in generating profits.

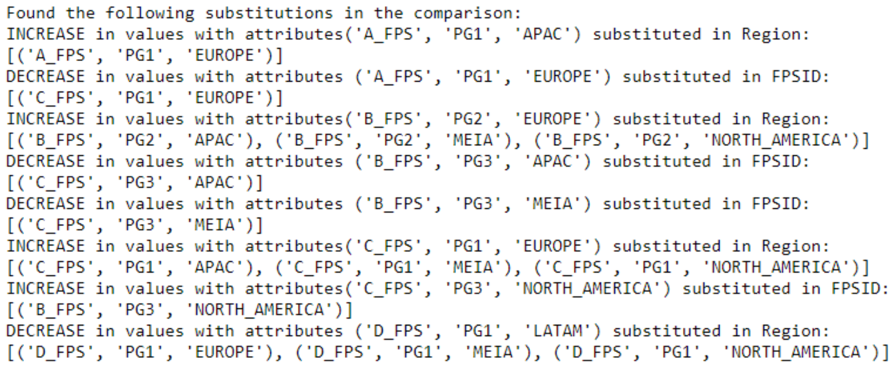


```
Top 10 RM by sales
15900010 911819586.0374987
10500010 911819586.0374987
18730189 911819586.0374987
18400010 870658274.3885438
11697119 736284449.8900317
10047113 687381292.2943769
18900115 646621320.327464
18739248 646621320.327464
18730601 646621320.327464
10598586 646353043.766581
```

Figure 13: Example report for critical raw materials

For the scenario comparison case, we present an example of applying the substitution detection algorithm to two scenarios, where in the second scenario the currency rate between two major operating currencies has changed significantly. Here, the decision maker wants to find out what kind of production or customer demand switches are caused by this change. This is a special example of a question of category (Two,Complex) in our question taxonomy in Section 3.2. We focus on the decision variable nodes that quantify the amount of demand satisfied for each product and customer from each warehouse (X). This analysis could focus on either costs or volumes, but since in this case the costs are largely affected by the currency rate change, we are more interested in how the actual volumes would change. We first attempt to find substitutions on the highest hierarchical level: This means finding potential substitutions on a regional and product family level, between the different warehouses. As a result, we obtain the a list of potential substitutions, which is shown in Figure 14.

In this specific case, the analyst can immediately notice that some significant changes have occurred between the scenarios: Firstly, the volumes of product family PG1 sent from warehouse A_FPS have been shifted from customer region EUROPE to APAC. Furthermore, this decrease in volumes from warehouse A_FPS of product family PG1 to EUROPE is now being countered by another warehouse, C_FPS, which has increased its sales of PG1 in EUROPE. Interestingly, this volume increase is offset by the decrease in volumes from warehouse C_FPS to other customer regions, namely APAC, MEIA and NORTH AMERICA. A similar story seems to happen for product family PG2, for which we observe that warehouse B_FPS has shifted customer sales from regions APAC, MEIA and NORTH AMERICA, such that these combined volumes are matched by the increased volume of sales in EUROPE. These substitutions suggest that these customer regions other than EUROPE may be less lucrative if the currency rate does change. This was indeed the case, thus this substitution report could be used to confirm the assumptions of the analyst, as well as providing a more detailed view into how the supply chain operations should change if the speculated scenario became a reality.



```
Found the following substitutions in the comparison:
INCREASE in values with attributes('A_FPS', 'PG1', 'APAC') substituted in Region:
[('A_FPS', 'PG1', 'EUROPE')]
DECREASE in values with attributes ('A_FPS', 'PG1', 'EUROPE') substituted in FPSID:
[('C_FPS', 'PG1', 'EUROPE')]
INCREASE in values with attributes('B_FPS', 'PG2', 'EUROPE') substituted in Region:
[('B_FPS', 'PG2', 'APAC'), ('B_FPS', 'PG2', 'MEIA'), ('B_FPS', 'PG2', 'NORTH_AMERICA')]
DECREASE in values with attributes ('B_FPS', 'PG3', 'APAC') substituted in FPSID:
[('C_FPS', 'PG3', 'APAC')]
DECREASE in values with attributes ('B_FPS', 'PG3', 'MEIA') substituted in FPSID:
[('C_FPS', 'PG3', 'MEIA')]
INCREASE in values with attributes('C_FPS', 'PG1', 'EUROPE') substituted in Region:
[('C_FPS', 'PG1', 'APAC'), ('C_FPS', 'PG1', 'MEIA'), ('C_FPS', 'PG1', 'NORTH_AMERICA')]
INCREASE in values with attributes('C_FPS', 'PG3', 'NORTH_AMERICA') substituted in FPSID:
[('B_FPS', 'PG3', 'NORTH_AMERICA')]
DECREASE in values with attributes ('D_FPS', 'PG1', 'LATAM') substituted in Region:
[('D_FPS', 'PG1', 'EUROPE'), ('D_FPS', 'PG1', 'MEIA'), ('D_FPS', 'PG1', 'NORTH_AMERICA')]
```

Figure 14: Example report for substitutions.

These substitutions can also be visualized: Figure 15 shows part of the comparison graph, where node contraction has been performed to combine nodes with the same product group and customer region. For clarity, only the decision variable nodes related to customer sales and the constraints directly connected to these decisions are shown here. The size of the nodes in this case is relative to the volume, and the green and red colors of the nodes indicate more and less volume, respectively. From this figure, we identify the same substitutions that were obtained in textual format. The first three substitutions from the report are denoted in the figure as node clusters C1, C2 and C3, respectively. In each of these node clusters, we observe how the counterbalancing decision variable values are connected to each other through constraints: For example, in cluster C1 the decision variable nodes are connected through a warehouse flow constraint, whereas in C2 the connecting constraint is one related to demand satisfaction. This case also illustrates that a specific decision variable may be considered a substitution in several ways: For example, clusters C1 and C2 share a common decision variable node, indicating that the change in this specific decision variable can be considered as a substitution in two ways.

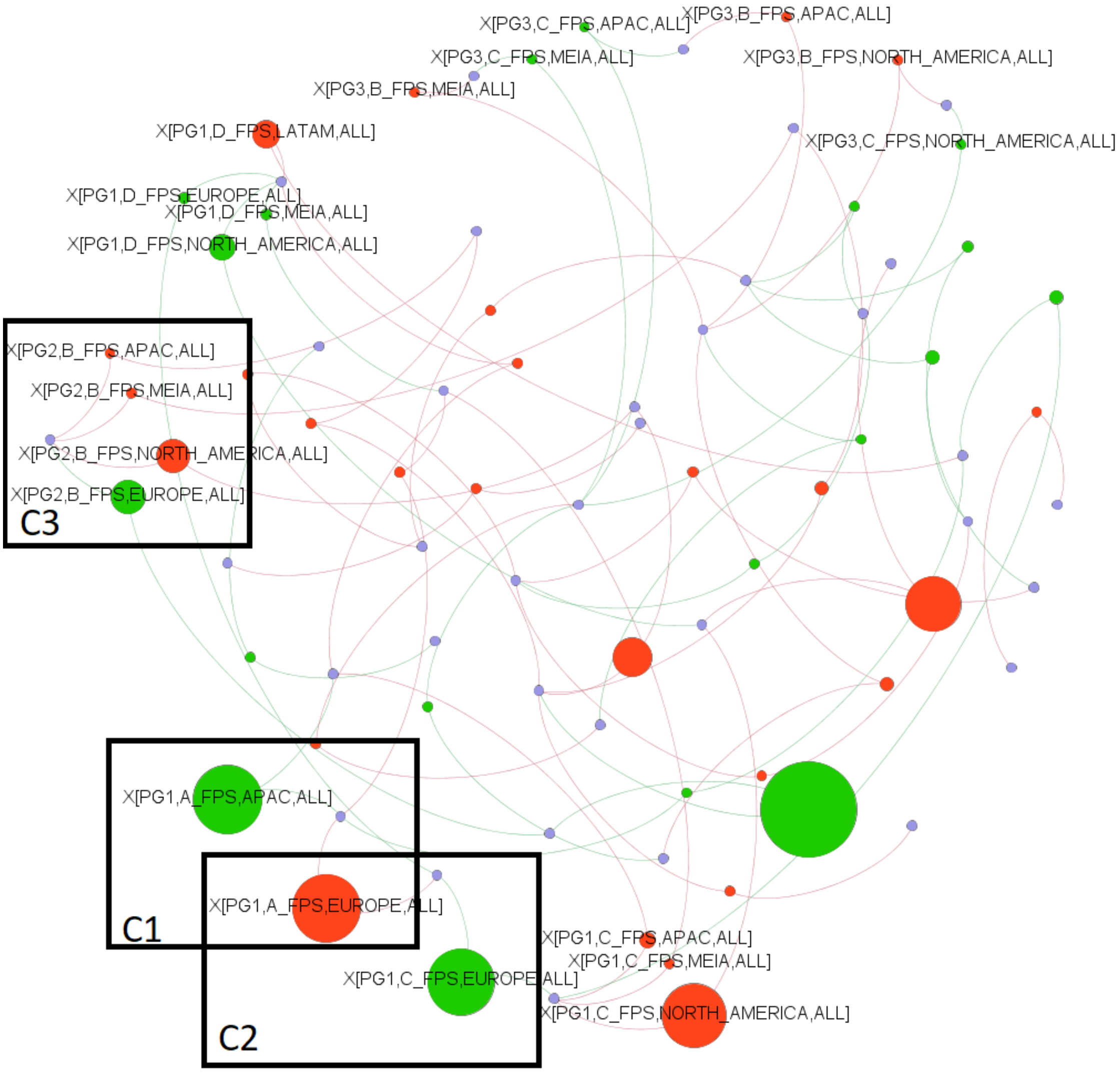


Figure 15: Substitutions in visual format

6 Conclusions and further developments

This Thesis explores the potential of providing semi-automated assistance in supply chain planning problems by creating an Intelligent Mathematical Programming System (IMPS) to aid the analyst. The approach is based on using the underlying mathematical model as the primary driver for the analysis by translating it to a graph, in addition to the actual optimisation results. This differs from other traditional approaches, where the structure of the model is not explicitly used as a tool for analysis. We refer to this methodology as the model oriented approach.

The presented algorithm for creating a graph out of a single LP model solution is based on an extension of the fundamental digraph described by Greenberg (1983) and in this Thesis, it was further extended to the case where two separate model scenarios are compared by creating a special comparison graph out of individual model graphs. This comparison graph algorithm allows analysis of both simple parameter-based and more complex model structure-based changes. In addition to technical details of the approach, we explored some potential uses by considering different types of questions decisions makers may have related to the supply chain. The main objective was to explore how the approach could help an analyst answer such questions. Potential uses included visual network analysis of scenarios, automatic report writing, and storing the graph into a graph database, which the analyst can use for ad hoc or structured graph queries.

Based on the initial results of the case study and discussions with the case company experts, at least some of the business questions related to the supply chain are directly related to the underlying model structure. These questions include finding connections between various entities of the supply chain, and they can extend through the entire chain from suppliers to final customers. In these cases, the model oriented approach is a potential alternative to the traditional approaches, since the model oriented approach maintains the connections between model entities. The Thesis classified typical business questions in Section 3, and most of the questions considered in the case study belong to the "complex" category considering either one or two scenarios. While the distinction between "simple" and "complex" questions is debatable and the amount of interesting questions in our case study is limited, the few successes nevertheless demonstrate the values of the proposed approach. It is also clear that from the case company experts' perspective that the model oriented approach is not a substitute to the traditional results oriented approach, but rather a complementary one.

There are challenges related to the proposed approach: for example, visual analysis becomes computationally challenging for large problems with millions of decision variables. Also, the effectiveness of querying the graph database relies heavily on the analyst's ability to translate the business questions into the appropriate model components and their relationships, as there is no automated process in place that would automatically select only the relevant parts of the model into the query.

Furthermore, these queries still require the user to understand the model structure quite extensively, therefore limiting the usability of the system. These problems can be somewhat alleviated by manipulating the directions of links between nodes to match the direction of physical flows in the supply chain, but this again requires model expertise. Finally, the automatic reports that can be written with the current system are very limited in their functionality, as there were few simple reports that were seen as both i) useful in a practical setting, and ii) where the implementation with the model oriented approach was considered to be easier than other approaches. Our aim was not to recreate the capabilities already offered by spreadsheets or BI platforms, but rather try to explore some previously unutilised methods of reporting model results. More complex automated reports may require a more model-specific approach.

While our approach is quite general and easily applicable to different types of supply chain optimisation models, and theoretically to any LP model, making conclusions on the benefits offered by the system requires more use cases. It also remains to be seen how general this kind of tool can be: the less individual model-specific details and quirks can be accounted for, the more limited is the depth of analysis. The lack of a unifying framework between the different suggested forms of scenario analysis, supply chain optimisation models and various business questions is also evident throughout the Thesis: almost all choices and methodologies are motivated through specific examples instead of attempting to explain them more generally.

6.1 Future research directions

In addition to abovementioned further testing of the IMPS, the approach could be developed further from its theoretical foundations, too. First, none of the core components of the IMPS specifically utilise graph-theoretic results. Here, both the network visualisation and some presented and new algorithms could potentially benefit from existing advances in graph theory. Examples include finding specific graph structures and interpreting them; generating richer automated textual reports that follow "paths" or exploit the graph structures in some other way; making the presented substitution algorithm smarter by allowing more complex comparisons or by grouping similar substitutions together.

Regarding the structure of comparison graphs, it would be beneficial to somehow be able to estimate which changes are most strongly related to each other. Currently, it remains unclear whether specific changes in the optimal solution between scenarios can be attributed to some small subset of other changes, or if the changes must always be considered as a whole. As there is often a large amount of individual changes that result from even a few parameter changes between scenarios, the ability to cluster related changes together, instead of having to consider them on an individual level, could provide significant improvements on the current analysis capabilities. As a prerequisite, this would require us to find an appropriate measure for "relatedness" between different nodes in the context of the comparison graph. One simple candidate would be to measure the distance between different nodes as the shortest path of undirected links between them. Intuitively, it would make sense that the shorter this path between two nodes, the more likely it is that they are related: Consider, for example, changes in production quantity decision variables that are neighbors to the same capacity constraint. While this approach seems reasonable at first, it would neglect the possibility of significant interaction between entities that are more distant from each other. This is an undesirable result, as many of the examples presented in this Thesis already show that such distant interactions do exist. Therefore, we believe more appropriate measures for node relatedness need to be developed.

The aggregation of graphs is another interesting subject where further study could provide a more robust method of simplifying the often large solution and comparison graphs that are created from optimisation models. The main concerns here are not related to practicalities, as we managed to describe a procedure that could be applied to produce such aggregated graphs. Rather, it is their interpretability that is still unclear: For instance, what exactly do the aggregate nodes and edges represent in general, or must these always be interpreted in the specific context of the underlying mathematical model? Moreover, can we define an appropriate aggregation function for all types of attributes, such that all the resulting aggregate values have a meaningful interpretation?

In terms of model analysis, the main focus of this Thesis was in considering one or two scenarios at a time, thus adopting the deductive model analysis paradigm similar to the one in systems such as Greenberg's ANALYZE. The alternative paradigm that we shortly described in Section 2 is inductive model analysis, where a large number of scenarios is processed to increase understanding of model relationships. While the obvious drawback of such an approach is that it requires more model runs, there are potential benefits over the deductive paradigm as well. First and foremost, an inductive approach could allow us to create a metamodel on top of the original optimisation model, which could then be used to explain how different model parameters affect the solution, ideally enabling the analysis of both individual and multiple scenarios. Since many of the decision makers' complex questions can be translated to finding and understanding the causal relationships between different entities of the modelled system, it is clear why such a metamodel could be a useful tool for the analyst. There are multiple methods with which such metamodels could be created: For example, the use of neural networks, a field with much ongoing research, was considered much earlier as a tool for inductive learning of mathematical models for example by Steiger and Sharda (1996). It would be interesting to see if some of the recent advances in this field could be transferred to the field of model analysis.

References

M. Bastian, S. Heymann, and M. Jacomy. Gephi: An open source software for exploring and manipulating networks. In *International AAAI Conference on Web and Social Media*, 2009. URL https://www.aaai.org/ocs/index.php/ICWSM/09/paper/view/154/1009.

D. Bertsimas and J. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, 1st edition, 1997. ISBN 1886529191.

S. Bradley, A. Hax, and T. Magnanti. *Applied Mathematical Programming*. Addison-Wesley Publishing Company, 1977.

R. Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer-Verlag, 5th edition, August 2016.

B. Fleischmann. Distribution and transport planning. In H. Stadtler and C. Kilger, editors, *Supply Chain Management and Advanced Planning*, pages 195–210. Springer, Berlin, Heidelberg, 2nd edition, 2002. ISBN 978-3-662-10142-1. doi: 10.1007/978-3-662-10142-1_12. URL https://doi.org/10.1007/978-3-662-10142-1_12.

A. Geoffrion. The purpose of mathematical programming is insight, not numbers. *Interfaces*, 7(1):81–92, November 1976.

M. Goetschalckx. Strategic network planning. In H. Stadtler and C. Kilger, editors, *Supply Chain Management and Advanced Planning*, pages 105–121. Springer, Berlin, Heidelberg, 2nd edition, 2002. ISBN 978-3-662-10142-1. doi: 10.1007/978-3-662-10142-1_7. URL https://doi.org/10.1007/978-3-662-10142-1_7.

H. Greenberg. A functional description of analyze: A computer-assisted analysis system for linear programming models. *ACM Transactions on Mathematical Software*, 9(1):18–56, March 1983.

H. Greenberg and J. Chinneck. Intelligent mathematical programming software: Past, present and future. *INFORMS Computing Society Newsletter*, 20(1):1,4–9, Spring 1999.

T. Grossman. A primer on spreadsheet analytics. In *EuSpRIG: In Pursuit of Spreadsheet Excellence*, pages 129–140. European Spreadsheet Risks Interest Group, 2008. ISBN 978-905617-69-2. URL http://arxiv.org/abs/0809.3586.

A. Janvier-James. A new introduction to supply chains and supply chain management: Definitions and theories perspective. *International Business Research*, 5(1):194–207, January 2012.

J. Keisler and P. Noonan. Communicating analytic results: A tutorial for decision consultants. *Decision Analysis*, 9(3):274–292, 2012. doi: 10.1287/deca.1120.0238. URL https://doi.org/10.1287/deca.1120.0238.

L. LeBlanc and M. Galbreth. Implementing large-scale optimization models in excel using vba. *Interfaces*, 37(4):370–382, 2007. ISSN 00922102, 1526551X. URL http://www.jstor.org/stable/20141515.

T. Miller. *Hierarchical Operations and Supply Chain Planning*. Springer, London, 2002. ISBN 978-1-4471-1110-8. doi: 10.1007/978-1-4471-0305-9. URL https://doi.org/10.1007/978-1-4471-0305-9.

T. Miller. A hierarchical supply chain planning approach, business briefing. *Global purchasing & Supply Chain Strategies*, pages 42–57, January 2004.

S. Pemmaraju and S. Skiena. *Computational Discrete Mathematics: Combinatorics and Graph Theory with Mathematica*. Cambridge University Press, 2003.

D. Power and R. Sharda. Model-driven decision support systems: Concepts and research directions. *Decision support systems*, 43:1044–1061, April 2007.

Cambridge University Press. *Cambridge business English dictionary*. Cambridge University Press, 2011.

J. Rohde and M. Wagner. Master planning. In H. Stadtler and C. Kilger, editors, *Supply Chain Management and Advanced Planning*, pages 143–160. Springer, Berlin, Heidelberg, 2nd edition, 2002. ISBN 978-3-662-10142-1. doi: 10.1007/978-3-662-10142-1_9. URL https://doi.org/10.1007/978-3-662-10142-1_9.

M. Rönnqvist, S. D'Amours, A. Weintraub, A. Jofre, E. Gunn, R. Haight, D. Martell, A. Murray, and C. Romero. Operations research challenges in forestry: 33 open problems. *Annals of Operations Research*, 232(1):11–40, September 2015. ISSN 1572-9338. doi: 10.1007/s10479-015-1907-4. URL https://doi.org/10.1007/s10479-015-1907-4.

T. Seeve. A structured method for identifying and visualizing scenarios. Master's thesis, Aalto University School of Science, 2018.

H. Stadtler. Production planning and scheduling. In H. Stadtler and C. Kilger, editors, *Supply Chain Management and Advanced Planning*, pages 177–193. Springer, Berlin, Heidelberg, 2nd edition, 2002a. ISBN 978-3-662-10142-1. doi: 10.1007/978-3-662-10142-1_11. URL https://doi.org/10.1007/978-3-662-10142-1_11.

H. Stadtler. Linear and mixed integer programming. In H. Stadtler and C. Kilger, editors, *Supply Chain Management and Advanced Planning*, pages 391–401. Springer, Berlin, Heidelberg, 2nd edition, 2002b. ISBN 978-3-662-10142-1. doi: 10.1007/978-3-662-10142-1_25. URL https://doi.org/10.1007/978-3-662-10142-1_25.

H. Stadtler. Supply chain management and advanced planning - basics, overview and challenges. *European Journal of Operational Research*, 163(3):575–588, June 2005.

D. Steiger. Enhancing user understanding in a decision support system: A theoretical basis and framework. *Journal of Management Information Systems*, 15(2):199–220, 1998.

D. Steiger and R. Sharda. Analyzing mathematical models with inductive learning networks. *European Journal of Operational Research*, 93(2):387–401, September 1996.

P. Trkman, M. Stemberger, and J. Jaklič. Information transfer in supply chain management. *Issues in Informing Science and Information Technology*, 2, January 2005. doi: 10.28945/851.

A. Weintraub, S. D'Amours, and M. Rönnqvist. Using operational research for supply chain planning in the forest products industry. *INFOR: Information Systems and Operational Research*, 46, November 2008. doi: 10.3138/infor.46.4.265.

A MPS parsing algorithm

```
Algorithm 1: MPS Parsing algorithm
  Data: MPS file
  Result: Graph as adjacency list
  initialise hash table h = $\emptyset$
  open MPS file
  begin reading MPS file line by line
  while line is not ROWS do
      Skip line
  end
  while line is not COLUMNS do
      Split line into rowtype and rowname
      Save rowname as new key in h
      Save 'TYPE':rowtype as new key-value pair under rowname
      Read new line
  end
  while line is not RHS do
      Split line into colname, rowname, val
      Save colname:val as new key-value pair under rowname
      Read new line
  end
  for rowname in h do
      Set 'RHS':0 as new key-value pair under rowname
  end
  while line is not BOUNDS do
      Split line into rowname and val
      Set val as the new value of 'RHS' under rowname
      Read new line
  end
  return h
```

B Graph Construction algorithm

```
Algorithm 2: Graph construction algorithm
  Data: PMPS-file, sol-file, auxiliary files
  Result: Solution graph
  Create mapping from column names to solution values → sol
  Save PMPS['OBJ'] as a separate mapping → obj
  Initialise graph G = ∅
  for row in PMPS do
      add row key as new node into G
      add row['RHS'] into node as new attribute
      add row['TYPE'] into node as new attribute
      for col in row do
          if col not in G then
              add col as new node into G
              add sol[col] and sol[col] into node as new attributes
              add obj[col]*sol[col] into node as new attribute
          if row[col] > 0 then
              add edge from row to col with value row[col]
          else
              add edge from col to row with value row[col]
          end
      end
  end
  if Metafile in auxiliary files then
      create mapping from each node type to dimension names → M
      for node in G do
          split node name into type and dimensions
          add type into node as new attribute
          get corresponding dimension names → M[type]
          for dval,dname in (dimensions, M[type]) do
              add dval into node as new attribute with name dname
          end
      end
  if Hmap in auxiliary files then
      for node in G do
          for key in Hmap do
              if key in node then
                  for subkey in Hmap[key] do
                      get matching value for node[key] → hval
                      add hval into node as new attribute with name subkey
                  end
          end
      end
```

```
if virtcons in auxiliary files then
    for node in G do
        if node[type] in virtcons then
            get neighboring edges of node → $e_G$
            for edge in $e_G$ do
                add label 'Virtual' into edge as new attribute
            end
    end
    return G
```

C Graph comparison algorihm

```
Algorithm 3: Graph comparison algorithm
  Data: $G$,$H$
  Result: Comparison graph $C$
  Initialise empty graph $C = \emptyset$
  Split decision variable nodes into sets $V_G \cap V_H$, $V_H \setminus V_G$, $V_G \setminus V_H$
  dvars = $(V_G \cap V_H) \cup (V_H \setminus V_G) \cup (V_G \setminus V_H)$
  for node in dvars do
      if node in $V_H \setminus V_G$ then
          h = H[node]
          g = 0
          sol = 'H'
      else if node in $V_G \setminus V_H$ then
          h = 0
          g = G[node]
          sol = 'G'
      else
          sol = 'BOTH'
          h = H[node]
          g = G[node]
      end
      if g - h $\neq$ 0 then
          Add node to $C$
          Add sol to node dimensions
          Set g - h as node attribute vector
          if sol == 'BOTH' then
              if g[0] == 0 then
                  add key-value pair active = 'H' to node dimensions
              end
              else if h[0] == 0 then
                  add key-value pair active = 'G' to node dimensions
              else
                  add key-value pair active = 'BOTH' to node dimensions
              end
          end
      end
  end
```

```
Split constraint nodes into sets $U_G \cap U_H$, $U_H \setminus U_G$, $U_G \setminus U_H$
cons = $(U_G \cap U_H) \cup (U_H \setminus U_G) \cup (U_G \setminus U_H)$
for node in cons do
    if node in $U_H \setminus U_G$ then
        h = H[node]
        g = $\mathbf{0}$
        sol = 'H'
    else if node in $U_G \setminus U_H$ then
        h = $\mathbf{0}$
        g = G[node]
        sol = 'G'
    else
        sol = 'BOTH'
        h = H[node]
        g = G[node]
    end
    Add node to $C$
    Add sol to node dimensions
    Set g - h as node attribute vector
end
for node in $C$ do
    if node in $V_G \setminus V_H$ then
        get neighboring edges of node from $G \rightarrow e_G$
        sol = 'G'
        for edge in $e_G$ do
            g = G[edge]
            add edge to $C$
            add sol to edge dimensions
            add g to edge attributes
        end
    end
    else if node in $V_H \setminus V_G$ then
        get neighboring edges of node from $H \rightarrow e_H$
        sol = 'H'
        for edge in $e_H$ do
            h = H[edge]
            add edge to $C$
            add sol to edge dimensions
            add h to edge attributes
        end
end
```

```
for node in $C$ do
    else if node in $V_G \cap V_H$ then
        get neighboring edges of node from $G$ and $H$, → $e_G$,$e_H$
        split edges into sets $e_G \cap e_H$, $e_G \setminus e_H$, $e_H \setminus e_G$
        all = $(e_G \cap e_H) \cup (e_H \setminus e_G) \cup (e_G \setminus e_H)$
        for edge in all do
            if edge in $(e_H \setminus e_G)$ then
                sol = ’H’
                h = H[edge]
                g = 0
            end
            else if edge in $(e_G \setminus e_H)$ then
                sol = ’G’
                h = 0
                g = G[edge]
            else if edge in $(e_G \cap e_H)$ then
                sol = ’BOTH’
                h = H[edge]
                g = G[edge]
            end
            add edge to $C$
            add sol to edge dimensions
            add g - h to edge attributes
        end
end
for node in $C$ do
    get neighboring edges of node → $e_C$
    if $e_C = \emptyset$ then
        remove node from $C$
    end
end
```

D Substitution detection algorithm

```
Algorithm 4: Substitution detection algorithm
  Data: Comparison graph $C$, type $n$, attribute $a$, substitution threshold $\epsilon$,
        aggregation level $L$
  Result: Report of potential substitutions
  Initialise list of hash tables H = ∅
  for node in C do
      if node[type] == n then
          get all node dimensions as key-value pairs → node[d]
          get node attribute $a$ → node[a]
          h = node[d] + node[a]
          append h to list H
      end
  end
  if L != ∅ then
      Initialise list of hash tables H' = ∅
      for elem in H do
          for key,value in elem do
              if key not in L then
                  remove key from elem
              end
          end
      end
      for elem in H do
          if elem not in H' then
              append elem to H'
          end
          else
              H'[elem][a] += H[elem][a]
          end
      end
      H = H'
  end
```

```
for $e_i$ in $H$ do
    Separate $e_i$ into attribute $a_i$ and set of dimensions $d_i$
    for $e_j$ in $H$ do
        Separate $e_j$ into attribute $a_j$ and set of dimensions $d_j$
        for $dim$ in $d_i$ do
            Initialise hash table s = $\emptyset$
            Initialise sum c = 0, list of keys $k = \emptyset$
            if $a_i a_j < 0$ and $d_i \cap d_j = d_i \setminus dim$ then
                add $d_j$:$a_j$ as new key-value pair into s
            end
            for key,val in s do
                c += val
                append key to $k$
                if $(1 - \epsilon)|a_i| < |c| < (1 + \epsilon)|a_i|$ then
                    Report $k$ as potential substitution set for $e_i$ in $dim$
                end
            end
        end
    end
end
```