A HEURISTIC FOR LARGE SCALE MULTI-PERIOD DISASSEMBLY LEVELING AND SCHEDULING

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
IZMIR UNIVERSITY OF ECONOMICS

BY
ŞERBETCİOĞLU, CEMRE

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
IN
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

AUGUST 2017

Approval of the Graduate School of Natural and Applied Sciences

[illegible]

(Assoc. Prof. Dr. Devrim ÜNAY)

Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

[illegible]

(Assoc. Prof. Dr. Selin ÖZPEYNİRCİ)

Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

[illegible]	[illegible]
(Asst. Prof. Dr. Mahmut Ali GÖKÇE)	(Assoc. Prof. Dr. Selin ÖZPEYNİRCİ)
Co-Supervisor	Supervisor

Examining Committee Members	Date: 24.08.2017
Asst. Prof. Dr. Mahmut Ali GÖKÇE	[illegible]
Asst. Prof. Dr. Kamil Erkan KABAK	[illegible]
Asst. Prof. Dr. Erdinç ÖNER	[illegible]
Assoc. Prof. Dr. Özgür ÖZPEYNİRCİ	[illegible]
Assoc. Prof. Dr. Selin ÖZPEYNİRCİ	[illegible]

ABSTRACT

A HEURISTIC FOR LARGE SCALE MULTI-PERIOD DISASSEMBLY LEVELING AND SCHEDULING

Şerbetcioğlu, Cemre

M.Sc. in Industrial Engineering
Graduate School of Natural and Applied Sciences

Supervisor: Assoc. Prof. Dr. Selin ÖZPEYNİRCİ
Co-Supervisor: Asst. Prof. Dr. Mahmut Ali GÖKÇE
August 2017, 76 pages

In recent years, with increasing regulations and laws on environment, both consumers and manufacturers start getting more conscious in order to prevent the depletion of natural resources. Due to this awareness, manufacturing industry starts to change its direction towards product recovery instead of disposing them. Since products need to be separated systematically in order to be recycled or remanufactured, disassembly has a major role in product recovery process. There are various problems on the disassembly domain but in this study, the main focus is on the disassembly leveling and scheduling field. The problem of disassembly scheduling considered in this thesis can be defined as determining the quantity and timing of disassembling used/end-of-life products and/or subassemblies while satisfying the demand for parts/subassemblies over a planning horizon and at the same time the problem of disassembly leveling, that is, determining disassembly level for the end-of-life product for different periods. This study addresses the basic problem without parts commonality i.e., none of the items in used/end-of-life product share its parts and/or subassemblies, with the extension of considering capacity restriction and unsatisfied demand, unlike the existing studies. A mixed integer programming (MIP) model is developed first, then in order to have a more practical application which is fast enough to solve large scale disassembly scheduling operations, a heuristic algorithm is suggested with its computational results.

Keywords: Disassembly, disassembly scheduling, disassembly leveling, mathematical programming

ÖZ

ÇOK PERİYOTLU ÇOK PARÇALI DEMONTAJ ÇİZELGELEME İÇİN SEZGİSEL YÖNTEMLER

Şerbetcioğlu, Cemre

Endüstri Mühendisliği, Yüksek Lisans
Fen Bilimleri Enstitüsü

Tez Yöneticisi: Doç. Dr. Selin ÖZPEYNİRCİ
Ortak Tez Yöneticisi: Yrd. Doç. Dr. Mahmut Ali GÖKÇE
Ağustos 2017, 76 sayfa

Geçtiğimiz birkaç yıl, doğal kaynakların gittikçe artan azalma ve doğanın kirlenme hızını azaltmaya yönelik, tüketici ve üreticiler için çıkan artan sayıda kanuni düzenlemelere tanık olmuştur. Bu artan düzenlemeler, tüketicideki bilinç artışı ve maliyet unsurları ile birlikte, şirketlerin yarı mamül/ parça/ hammadde geri kazanımı konularında daha çok uğraş vermelerine sebep olmuştur. Geri kazanım işlemlerinin verimli yapılabilmesi, artık endüstriyel boyutta demontaj işlemlerinin iyi yönetilmesi ile yakından ilgilidir. Şimdiye kadar demontaj çizelgeleme problemi (ne zaman, hangi üründen, kaç tane ve ne kadar demonte edileceği) için çeşitli matematiksel modeller geliştirilmiş, fakat gerçekçi boyutlardaki problemlerden makul sürelerde makul çözümler elde edebilecek sezgisel yöntemler geliştirilmemiştir. Bu çalışmada, gerçekçi boyutlardaki demontaj çizelgeleme probleminin çözümü için geliştirilmiş sezgisel yöntemler ve bu yöntemlerin gerçekçi bir problem veri seti üzerinden performansları, bilinen optimumlarla karşılaştırmalı olarak sunulmaktadır.

Anahtar Kelimeler: Demontaj çizelgeleme, demontaj seviyeleri, sezgisel yöntemler, matematiksel programlama

Acknowledgments

I would like to thank my co-advisor Asst. Prof. Dr.Mahmut Ali Gökçe for continuous support during my research, for his patience, motivation and enthusiasm. His guidance helped me all the time during research and writing of this thesis. I am grateful to him for enlightening me from the first very beginning of this study. He was a great teacher, educator, and advisor, who provided a lot of the ideas and inspiration for the thesis, and without his assistance and dedicated involvement in every step throughout the process, this thesis would have never been accomplished.

My sincere thanks also goes to my advisor Assoc. Prof. Dr. Selin Özpeynirci for her support and guidance over years. Besides my advisor and co-advisor, I would like to thank the examining committee for their insightful comments and contributions to my thesis.

Also, my deepest gratitude and love belong to Tolgay Hıçkıran, whose encouragement, inspiration and support were with me all these years.

Last but not least, I would like to express my sincere gratitude to my family for their love and support during all my education years as well as my thesis process.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In recent years, with increasing regulations and laws on environment, both consumers and manufacturers became more conscious to prevent the depletion of natural resources (Lambert and Gupta, 2002). A major contributing factor for this increased public awareness is the accumulating product waste, which is yielded by rapid development in technology and decreasing product life-cycles. This speed in the improvement of products led people to desire the latest technology and caused additional demand, which yielded an increase in amount of waste while decreasing the lifetime of products. With increased quantity of scrapped products, landfill capacity also reached an alarming rate and as a result, green products gained a great deal of attention. Hence, people start to behave in more environmentally responsible manner. Due to this environmental awareness and government legislations, manufacturing industry is forced to use recycled materials more and more, and they start to change their direction towards product recovery instead of simply disposing them, since total breakdown of disposed end-of-life(EOL) products takes quite long time in nature.

Besides the societal pressure, companies started to realize the existence of economic benefit to be gained from recovery of resources. Many firms, such as Audi, BMW, Daimler-Chrysler, Ford and Volkswagen recognized the importance of efficient use of resources and they have their own disassembly/recovery plants.

In addition to automobile manufacturers, product recovery has also given rise in the military field, since governments no longer allow scrapping the weapons and ammunitions (Gungor and Gupta, 1999). In this framework, the objective is mainly to recover products considering ecologic and economic benefits while keeping the waste within desirable level. A schematic demonstration of product recovery is shown in Figure 1.1.

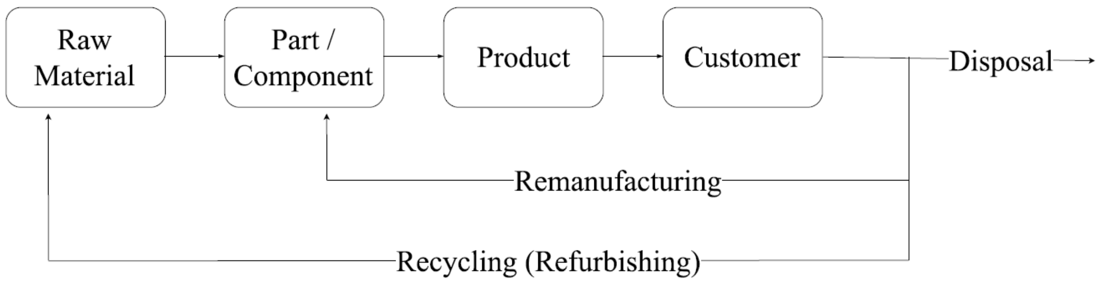


Figure 1.1: Product Recovery Schema

Product recovery is defined as reutilizing used/EOL products in terms of recycling or refurbishing and remanufacturing where recycling is the material regain from used/EOL products via some disassembly operations while remanufacturing preserves the functional content of used/EOL products and improves their quality up to desired level (Sasikumar and Kannan, 2001). Since products need to be separated systematically in order to be recycled or remanufactured or even disposed of, disassembly has a major role in product recovery process. According to Jovane et al.(1993), disassembly can be stated as separating a product into its parts and/or subassemblies with necessary sorting, while extracting hazardous substances to satisfy environmental standards and requirements.

Vongbunyong and Chen (2015) divides disassembly in two types based on its completeness: complete disassembly and incomplete disassembly. Complete disassembly is disassembling product fully, while incomplete disassembly is disassembling partially. In addition, there are three classifications of disassembly operations which are non-destructive, semi-destructive and destructive. Non-destructive disassembly is the case which is preferred for reuse and remanufacture since it does not damage the disassembly while semi-destructive disassembly leaves only the main items and it may damage them during the operation. On the contrary, destructive disassembly aims to reach most valuable parts inside while destroying the rest and is the most common operation technique. Although disassembly setting has many similarities with the assembly concept, the divergent structure of disassembly is highly challenging due to its variations and uncertainties such as product's condition, market demand and technology changes.

A number of previous studies are performed on disassembly systems and their challenges. Those various problems can be covered in several disassembly domains such as design of disassembly, disassembly line balancing, disassembly process planning, disassembly leveling and disassembly scheduling. In this study, our interest is mainly in disassembly leveling and scheduling. Herein, disassembly leveling is deciding whether to disassemble or not and determine depth of the level while; disassembly scheduling is the problem of determining the quantity and timing of disassembling used/EOL products and/or subassemblies while satisfying the demand of parts/subassemblies over a planning horizon (Gungor and Gupta, 1999). The focus of this thesis is on disassembly scheduling, that is, to determine the quantity and time of disassembling used/EOL products and/or subassemblies.

From theoretical point of view, disassembly scheduling can be considered similar as reversed material requirement planning (MRP); however, in our case reversed manufacturing resource planning (MRP II) is more likely to the problem's characteristic in terms of capacity consideration. Even though they seem similar, because of the divergent structure of the bill of material (BOM) and different demand sources it differs from those concepts in more complicate way (Taleb, Gupta and Brennan , 1997). Figure 1.2 illustrates disassembly and assembly structures in detail.

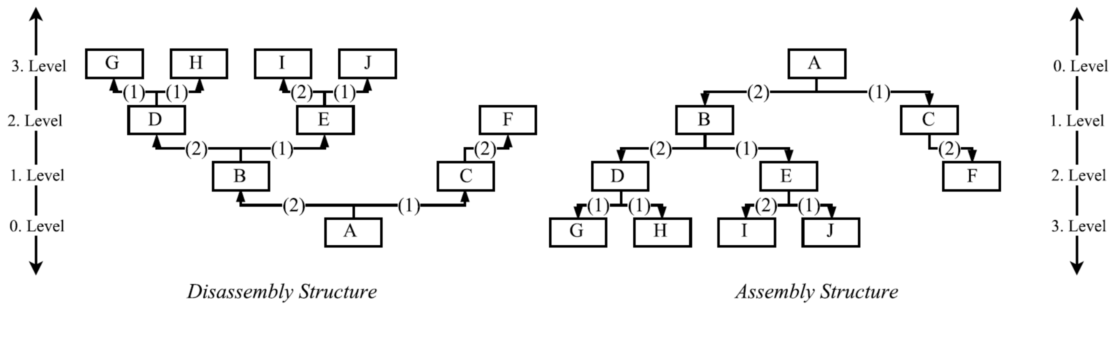


Figure 1.2: Difference of Disassembly and Assembly Structure

Figure 1.2 also shows the difference among single and multiple demand sources for both structures. In more detail, items G, H, I and J are the demand sources in disassembly structure while in the assembly structure A is the the only independently demanded item. On the contrary, in disassembly structure there exists only one procurement source which is A, while assembly structure has multiple procurement sources that are items G, H, I and J. This figure exemplifies the complexity of disassembly structure compared to the assembly structure. Main reason of this is the dependency between items which limits disassembly operation.

Additionally, level or depth of the product tree is another major decision criterion in disassembly structure that differs from assembly structure. Therefore, an integrated problem of disassembly leveling and scheduling that can be defined as a capacitated disassembly scheduling problem on the case of single product type without parts commonality is addressed in this study. Among the various disassembly scheduling problems with heuristic approach in literature, this study contributes existing articles with following extensions:

- capacity restriction
- allowing unsatisfied demand but penalizing it as a cost
- disassembling, selling and holding items all at the same time
- various costs consideration such as purchasing, setup operation, inventory

holding and penalty cost

- giving opportunity to sell not only parts but also subassemblies

To represent a model with above extensions mathematically, an integer programming model is demonstrated with the objective of maximizing revenue which also differs from most of the existing studies. Although the integer programming model with above extensions gives optimal solution for small-sized problems, for large scale problems including a wide range of problem specification in terms of product structure; mathematical programming approach can become infeasible for operational planning purposes and therefore, a faster solution approach is needed in order to be able to run a disassembly operation in practice. As illustrated in more detail in Chapter 2, there is also a lack of such approaches for this problem in previous studies. Hence, in this study, a heuristic algorithm is proposed for the above mentioned disassembly leveling and scheduling problem space. Moreover, a testbed problem generator is developed with specific design configurations, which is the main contribution to the existing studies in terms of heuristic algorithm solution approach. To show the performance of the suggested algorithm, computational experiments are performed on a number of randomly generated problems and the test results are reported.

The remainder of this thesis is organized as follows. In Chapter 2, relevant previous articles on disassembly leveling and scheduling are reviewed. The problem is defined and the model considered in this study is demonstrated in Chapter

3. The suggested heuristic algorithm is represented in Chapter 4 and in Chapter 5, experimentation set design is explained first. Following, computational results for randomly generated test problems are reported for both solution approaches and experimentation results are discussed. Lastly, in Chapter 6 the study is concluded with a summary and future study issues.

Chapter 2

Literature Review

Following the introduction, in this chapter relevant articles in literature are reviewed. In disassembly domain, various research articles published through years. Jovane et al. (1993) reviewed current recycling activities according to process, product and system design frames. Bok et al. (1998) covered government legislations and EOL product strategies for three regions; Japan, the United States and Western Europe. Gungor and Gupta (1999) focused on Environmentally Conscious Manufacturing and Product Recovery (ECMPRO) related issues in their study. Recently, Vongbunyong and Chen (2015) emphasized the importance of economical feasibility in disassembly systems. The difficulties that are mainly the variations of the basic problem and demand uncertainities covered in the corresponding book chapter. For other related research articles, the interested reader is referred to Neuendorf et al. (2001), Tang et al. (2002) and Ilgın and Gupta (2010).

The focus of this thesis is mainly on disassembly leveling and scheduling scope. Disassembly scheduling can be stated as the problem of determining quantity and timing of EOL products to be disassembled in order to fulfill certain or uncertain demand for parts and/or subassemblies of the corresponding EOL item in a planning horizon, with or without capacity restriction. Since disassembly is a costly operation, obtaining optimal or near optimal schedules are highly important. Also, improving efficiency for companies who are interested in this field, should be flexible in terms of disassembly operations and satisfactory solutions should be provided for necessary conditions.

Most frequently used objectives in disassembly scheduling can be listed as:

- Minimizing number of disassembled items
- Minimizing total cost
- Maximizing overall profit

The previous articles on disassembly scheduling can be classified by assembly product structure (with or without parts commonality), number of products (single or multiple), capacity constraints and demand uncertainty (deterministic or stochastic) (Neundorf et al., 2001). Mainly, in this thesis literature review is classified in two major categories that are, capacitated and uncapacitated disassembly scheduling problem.

2.1 Disassembly Scheduling Models without Capacity Constraint

Capacity restriction considered in this study can be stated as resource or workload limitation. Compared to the capacitated problem, to the best of author's knowledge, not much work done on uncapacitated problem. From the original work of the uncapacitated problem that is characterized by Gupta and Taleb (1994), many research has been published on disassembly scheduling. Here, Gupta and Taleb (1994) suggested a reverse form of MRP algorithm for the basic problem. Basic problem can be considered as the deterministic problem with single product without parts commonality where there are not any capacity restrictions and no explicit objective function such as cost variety. The parts commonality here implies that products may have the same parts. In other words, multiple occurrences of parts may happen among products. In their study, their MRP-like algorithm determines the ordering and disassembly schedule for EOL items and subassemblies.

Several extensions suggested after Gupta and Taleb (1994) introduced the basic model. Taleb, Gupta and Brennan (1997) extended the previous studies with materials and parts commonality allowance for single product and suggested another MRP-like algorithm where the objective was to determine the procurement strategy that minimizes number of ordered products. This extended model makes the problem more complex than the basic one since parts commonality provided a dependency between the parts. Later, Taleb and Gupta (1997) extended the problem with parts commonality proposed by Taleb, Gupta and Brennan (1997), considering multiple product structures. In this model, they proposed two-phase algorithm that determines the disassembly schedule while attempting to minimize disassembly cost. Later, Neuendorf et al. (2001) extended the problem presented by Taleb, Gupta and Brennan (1997) assuming regular MRP with an improved Petri-net-based algorithm. Imtanavanich and Gupta (2002) studied on single period multi-objective disassembly scheduling problem for multiple products with parts commonality using integer goal programming. Later, Kongar and Gupta (2006) considered the demand uncertainty using fuzzy goal programming.

Lambert and Gupta (2002) considered disassembly graph approach and integer programming on the extended problem with parts commonality for both single product and multiple product types. Kim et al. (2003) suggested a heuristic based on linear programming (LP) relaxation approach for the multiple product type with parts commonality. Lee and Xirouchakis (2004) proposed a two-stage heuristic algorithm for the basic model with a cost-based objective function consideration as an extension of Gupta and Taleb (1997). Then, Lee et al. (2004) presented integer programming models for basic model and variants of it. They represented three integer programming models with all having the objective of minimizing various costs. Later, Lee (2005) developed an integer programming model with a cost-based objective for the basic problem and compared it with the existing MRP-like algorithms in order to emphasize the importance of various costs in the objective function.

In another research study by Kim et al.(2006), an algorithm is developed based on Lagrengian relaxation for the basic problem with a cost-based objective. Then, Kim, Lee and Xirouchakis (2006a) extended this proposed model from single product to multiple product. They first used LP relaxation and then, improved it using a dynamic programming algorithm with the objective to minimize the setup, disassembly operation and inventory holding costs. Langella (2007) modified the Taleb and Gupta (1997) model with cost variety. Purchasing and disposal costs are considered for multiple product with parts commonality. Barba-Gutierrez, Adenso-Diaz and Gupta (2009) addressed the connection between disassembly scheduling with lot sizing. They demonstrated a heuristic for the original Gupta and Taleb (1994) model and solved it using three lot sizing rules which are Economic Order Quantity(EOQ), Period Order Quantity(POQ) and Lot for Lot(L4L). Additionally, also a genetic algorithm is developed by Gao and Chen (2008) for the basic problem as an extension.

Kim et al. (2009) showed the complexity of basic problem with cost-based objective and proved that it is NP-hard. Then, they suggest another branch and bound algorithm. Kang et al. (2012) integrated disassembly leveling and scheduling problems and developed an integer programming model for single period. Two types of problems are considered in the study which are basic problem without parts commonality and extended version with parts commonality for both single and multiple products. Then, a heuristic is suggested and solved for both cases. As an extension of integration of disassembly leveling and scheduling study, Kim and Lee (2011), considered the problem for multi-period and suggested their own heuristic. In fact, the backward heuristic suggested by Kang et al. (2012) and Kim and Lee (2011) are the inspiration of our proposed algorithm in this study. Particularly, the backward fashion in their algorithm is the inspiration behind the proposed heuristic algorithm for the multi-period capacitated dissassembly scheduling problem. Additionally, in this thesis some other varieties of the problem is considered such as capacity restriction, multi-period planning horizon and demand consideration for not only leaf items but also subassemblies. A detailed problem description is mentioned in Chapter 3.

2.2 Disassembly Scheduling Models with Capacity Constraint

Although previous research addressed several cases of the disassembly scheduling problem as it is shown in Table 2.1, most of the heuristic algorithm studies is concerned with the uncapacitated problem. Unlike regular MRP, most of the earlier research studies considered uncapacitated problem which implies that there aren't any restrictions on available resources or workload capacity. The first study on capacitated disassembly scheduling was by Lee et al. (2002) who proposed an integer programming model with capacity restriction as an extension of the basic disassembly scheduling problem without parts commonality for single product that is developed by Gupta and Taleb (1994). They considered cost-based objective function unlike basic model and solved a case study on inkjet printers. Kim, Lee and Xirouchakis (2006b) extended this study with respect to two points that are setup cost consideration and Lagrangean heuristic algorithm development to solve large size problems. They emphasized the importance of setup costs in this study. For the same problem, Hyong-Bae et al. (2006) demonstrated and integer programming model and suggested a two-stage heuristic with a cost-based objective. Kim et al. (2006) suggested an algorithm with the objective of minimizing number of disassembled items and capacity consideration as an extension of the uncapacitated version of the problem developed by Kim, Lee and Xirouchakis (2005) .

Prakash, Ceglarek and Tiwari (2011) studied a metaheuristic approach with capacity constraint while allowing parts coming from external sources. Among existing studies on capacitated disassembly scheduling, this one is the first that considers a variety of original model that is parts commonality. They proposed a Constraint-Based Simulated Annealing (CBSA) algorithm and compared results with Simulated Annealing (SA) and Genetic Algorithm (GA) approaches. Recently, Gokgur et al. (2015) suggested a mathematical model for large scale problems which considers all varieties of the problem with capacity restriction. As an extension to existing studies, they considered additional costs for the first time such as penalty cost for unsatisfied demand, selling price, disposal cost. In this study, they allowed demand for not only leaf items but also subassemblies which makes the problem a bit more complex. For the capacitated disassembly scheduling problem, most recently Ji et al.(2016) extended previous studies considering start-up cost to conduct managerial sights and industrial applications. They developed Lagrangean heuristic and compared the results with previous researches.

There also exists some research studies in capacitated disassembly scheduling with stochastic demand consideration. For more detailed review on stochastic disassembly scheduling problems, see Kongar and Gupta (2006), Barba-Gutierrez and Adens-Diaz (2009) and Kim and Xirouchakis (2010).

Table 2.1 presents a summary of disassembly scheduling literature review including the proposed algorithm in this thesis. In the table PC denotes Parts Commonality (with parts commonality) where NPC implies No Parts Commonality (without parts commonality). SI and MI are the item numbers which are single item and multiple items, respectively. Capacity condition is stated with C and D letters where C implies capacitated and U states uncapacitated. Demand uncertainty is symbolized with D for deterministic and S for stochastic. Lastly, solution approaches are divided into three that are, MP, H and MH which implies Mathematical Programming, Heuristic and Metaheuristic, respectively.

Table 2.1: Summary of Literature Review

Literature		Structure		Item		Capacity		Uncertainty		Solution Approaches		
Authors	Year	PC	NPC	SI	MI	U	C	D	S	MP	H	MH
Gupta and Taleb	1994		x	x		x		x			x	
Taleb, Gupta and Brennan	1997	x		x		x		x			x	
Taleb and Gupta	1997	x			x	x		x			x	
Neuendorf et al.	2001	x		x		x		x			x	
Lambert and Gupta	2002	x		x	x	x		x		x		
Lee, Xirouchakis and Zust	2002		x	x			x	x		x		
Kim et al.	2003	x			x	x		x		x	x	
Imtanavanich and Gupta	2004	x			x	x		x		x		
Lee and Xirouchakis	2004		x	x		x		x		x	x	
Lee et al.	2004	x	x	x	x	x		x		x		
Kim, Lee and Xirouchakis	2005		x	x		x		x		x		
Lee	2005		x	x		x		x		x		
Kim, Lee and Xirouchakis	2006		x	x			x	x		x	x	
Hyong-Bae et al.	2006		x	x			x	x		x	x	
Kim et al.	2006		x	x			x	x		x		
Kongar and Gupta	2006	x			x	x			x	x		
Kim, Lee and Xirouchakis	2006	x			x	x		x		x	x	
Langella	2007	x			x	x		x			x	
Barba-Gutierrez , Adenso-Diaz and Gupta	2008		x	x		x		x			x	
Gao and Chen	2008		x	x		x		x		x		x
Kim et al.	2009		x	x		x		x		x		
Barba-Gutierrez and Adens-Diaz	2009		x	x			x		x		x	
Kim and Xirouchakis	2010		x		x		x		x	x	x	
Kim and Lee	2011	x			x	x		x		x	x	
Prakash, Ceglarek and Tiwari	2011	x		x			x	x		x		x
Kang et al.	2012	x	x		x	x		x		x	x	
Sung and Jeong	2014	x			x	x		x		x	x	
Gökgür, Gökçe and Özpeynirci	2015	x	x	x	x		x	x		x		
Ji et al.	2016	x			x		x	x			x	
Kim et al.	2016	x			x	x		x		x	x	
The proposed model	**2017**		**x**	**x**			**x**	**x**		**x**	**x**	

In Table 2.1, it is shown that capacitated version of the problem with heuristic solution approach has limited number of research articles. In fact, all of the studies with these specifications consider minimizing total cost where maximizing revenue is the objective of this study. Because, this thesis also allows shortage while penalizing it which causes additional decision variables such as number of sold items. Moreover, there exists demand both subassembly and part/component which differs from existing studies especially in terms of capacitated heuristic approaches. The focus of this thesis is on capacitated disassembly scheduling problem with both mathematical model and heuristic approaches. While most of the studies on capacitated problem with heuristic approach considers limited number of design parameters, in this study the performance study is extended with an extensive experimentation study that provides more insight about the problem dynamics.

To sum up, this study contributes to the existing problem that demonstrated by Gokgur et al. (2015) with a faster solution algorithm suggestion in terms of computation time. It is also worth to mention that, the proposed algorithm is also a contribution to the existing backwards heuristic suggested by Kang et al. (2012) and Kim and Lee (2011).

Chapter 3

Problem Definition

The problem under consideration is, disassembly scheduling with product structure of single product type without parts commonality. According to Taleb and Gupta (1997), disassembly product structure is divided into three types, that are, single product without parts commonality, single product with parts commonality and multiple products with parts commonality. These three types are adopted in Figure 3.1 below. In this thesis, a multi-period capacitated disassembly leveling and scheduling problem is studied with the consideration of the first type of structure shown in figure, which is the single product type without parts commonality. In this type of product structure, there exists only one EOL product and none of the items in this structure share its parts/components. In the second structure, that is single product type with parts commonality, items can share same parts/components. For clarity, common parts in the product structure are designated with dark colored boxes while yields of items are notated with numbers in parenthesis. Third structure which is multiple product type with parts commonality, allows both part sharing between different products and part sharing in the same product. In other words, part E in the third figure below, is an example of part sharing in same product that is obtained from disassembling parts B and C since they are part of product A. Apart from this type of parts commonality, again in the same structure below, part F is obtained from disassembling parts C and part H where part C belongs to product A and part H belongs to product G which is an example of part sharing in different products.

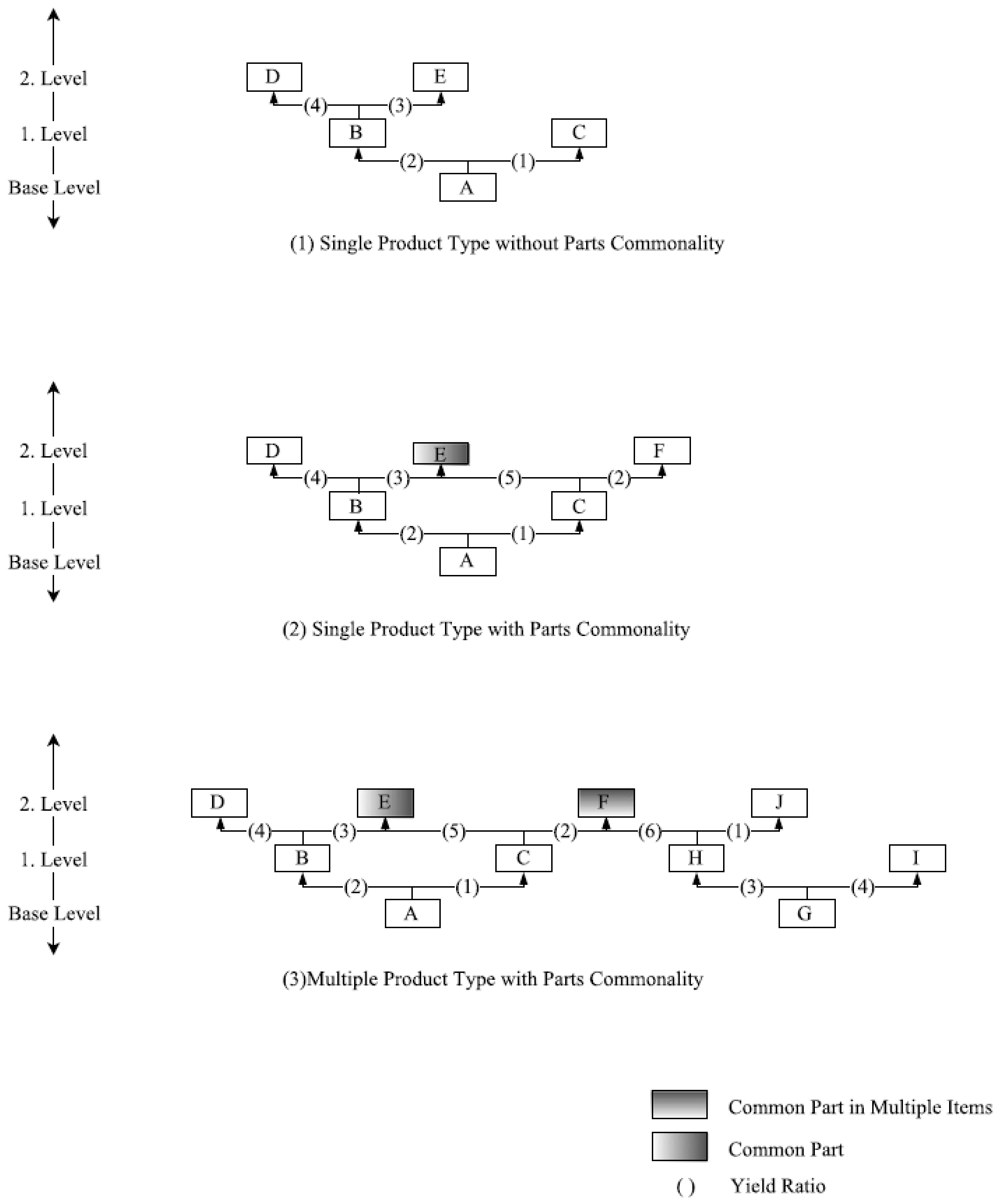


Figure 3.1: Disassembly Product Structure Examples

Disassembly product structure consists of EOL product and its parts/subassemblies. These parts/components and subassemblies are also described as root, intermediate and leaf items where root items represent the product to be disassembled or in other words, product itself. Leaf items on the other hand, are the parts or components that cannot be disassembled further. Unlike most of the existing studies, independent demand occurrence of not only parts/components but also subassemblies which are named as intermediate items, is considered in this thesis. In other words, demand occurs in both child and parent levels except root item which differs this study from existing heuristic approaches on capacitated disassembly scheduling problem.

A child parent relationship is where one or a number of children is obtained directly by immediately disassembling the parent. As it is stated before, single product without parts commonality type of problem is taken into account of which is also known as the basic problem. Therefore, in our case each child item has at most one parent item. Figure3.2 shows the disassembly structure and parent-child relationship which is also one of the products considered in our computational experiments (See Chapter 5 for related experiments and results).

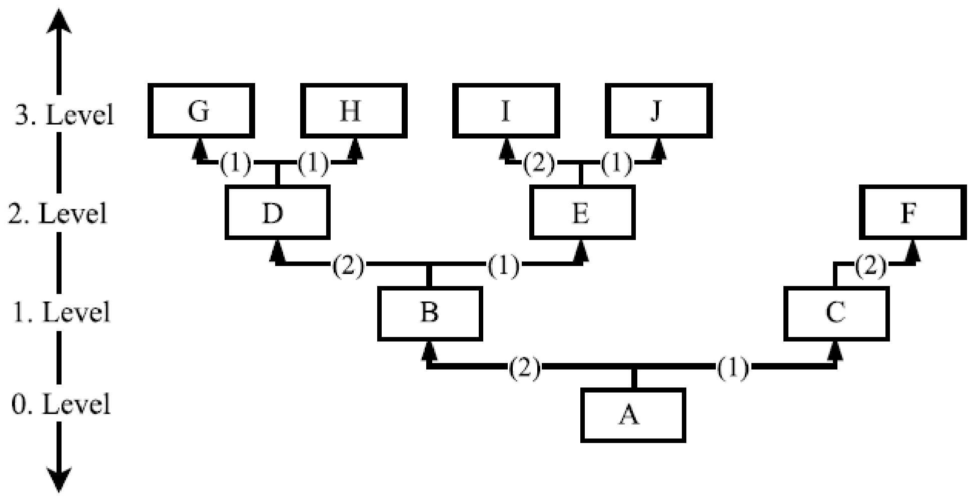


Figure 3.2: Disassembly Product Structure

In above disassembly bill-of-material (d-BOM) structure, the single product is the EOL item which is named as A, the root item. Next, B, C, D, E are the subassemblies which are also called as intermediate items. Finally, F, G, H, I, J are the parts/components that are described as leaf items. Yield ratio between root and leaf items are represented with the number in each parenthesis in figure below, which denotes the quantity of parts or components obtained while disassembling one unit of root item.

Disassembling one unit of the parent item A, results in 2 units of its child item B and 1 unit of its other child item C. Additionally, item B is disassembled into 2 units of D and 1 unit of E while item C is disassembled into 2 units of F. Hence, subassemblies or intermediate items such as B and C, are both children and parents. Root item A, on the other hand, is only a parent item while leaf items F, G, H, I and J are child items.

Demand for both parts and subassemblies assumption leads us to another important decision variable; disassembly level. Capacitated disassembly leveling and scheduling problem studied in this thesis is an integrated model which consists of both disassembly leveling and scheduling.

The problem considered in this study can be defined as follows: for a given disassembly product structure with single product without parts commonality, problem of determining the quantity and timing of disassembling used/EOL products and/or subassemblies while satisfying the demand of parts/subassemblies over a planning horizon and considering capacity restriction. In other words, deciding the timing and quantity of disassembly operations with capacity restriction while determining the disassembly level/depth. Here, a single time period may be a minute, hour etc. depending on the problem characteristics.

The objective function considered is to maximize revenue while considering capacity constraints over a planning horizon. Revenue is composed of sales and disassembly process related costs which considered in this study are; purchase, setup, operational, inventory holding and penalty costs. As it is mentioned in the previous chapter, most of the related articles in literature considered minimization problem while maximizing revenue is the objective in this study. Note that some of the costs considered in this model are time-variant, i.e. purchase cost. Purchase cost is the cost of buying one unit of the EOL product, or root item and it may vary in different time periods. Since disassembly operations are mostly performed manually, setup cost is also considered in the model which implies the cost required for the preparation of disassembly operation including the costs related to equipment/tooling during setup. If at least one disassembly operation is conducted in corresponding time period, then setup cost occurs.

Different from setup cost mentioned above, there also exists another disassembly operation related cost which is the operational cost. Operational cost is the cost required for the disassembly process and assumed to be deterministic and given in advance, while inventory holding cost occurs if there are any items stored. Since a multi-period problem is considered, inventory holding cost is also included which occurs if there are any items stored. In other words, it is the cost to carry one unit of part or subassembly from one period to other and it is computed at the end of each period. Besides, the nature of disassembly structure may cause a nondemanded part or subassembly to be disassembled as well. In more detail, in order to obtain demanded item in a disassembly structure it is needed to disassemble its parent item first which may cause a nondemanded or less demanded item from corresponding family disassembly. This foregone conclusion causes nondemanded child or children gain with the amount of corresponding yield. Hence, these nondemanded items will eventually be held in inventory as well.

Unlike most of the studies in the literature, in this study, penalty cost is also taken into account since shortage is allowed. Note that, backlogging is not allowed but shortage may occur in corresponding time period which ends with lost sales. Shortage cost is considered as the penalty cost of not satisfying one unit of demanded item. Therefore, in each period decision maker needs to consider the consequences of not satisfying demanded item and/or satisfying demand while having on-hand inventory.

In disassembly structures, based on yield information of product tree, most of the time disassembly operations conclude with nondemanded items on-hand. This leads us to the take into account of the trade-off between holding items in inventory or not satisfying demanded items while penalizing them. Before an explicit demonstration of main model that considers these variables as decision variables, first a MIP model with an objective of maximizing revenue is introduced.

Yet, another extension among most of the research studies in literature, the disassembly leveling and scheduling problem mentioned above considers capacity restriction. Capacity limit implies the availability of disassembly operation assigned in a period of the planning horizon. Hence, the number of disassembled items has an upper limit specified by each period's capacity restriction. Also, disassembly product structure is assumed to be given in advance for the corresponding planning horizon.

Other assumptions made in this study are summarized as follows:

- No defective parts are considered.
- Demand of leaf items and intermediate items are considered as deterministic and known in advance
- There is no shortage of root items.
- Disassembly lead time and ordering lead times are assumed negligible.
- Backordering is not permitted.
- Unsatisfied demand is allowed and penalized.

3.1 Mathematical Model

According to disassembly product structure described in previous section, an integer programming model is developed in order to describe the problem clearly. Definition of sets, indices used in the mathematical model are summarized as follows:

In the d-BOM structure, items are represented by integers 1, 2. . . , i_r, i_{r+1},. . . , i_{l-1}, i_l,. . . , I where root items are represented with integers from 1 to i_r, intermediate items are defined with integers from i_{r+1} to i_{l-1} and leaf items are notated with integers from i_l to I. Rest of the notations used are given as follows:

Sets and Indices

i, j	Item number, $i, j \in I$ =\{1,...,$\|I\|$\}
t	Period number, $t \in T$ =\{1,...,$\|T\|$\}

Parameters

h_i	inventory holding cost of item i
o_i	disassembly operation cost of item i
p_t	purchasing cost of EOL product in period t
c_i	setup cost of item i
pc_i	selling price of item i
pn_i	penalty cost of not satisfying one unit of item i
d_{it}	demand of item i in period t
cap_i	aggregated capacity for item i
PS_i	parent set of item i
a_{ji}	number of units of item i obtained from disassembling one unit of j
β_{it}	fill rate of item i in period t
M	arbitrary large number

Decision Variables

X_{it}	number of item i disassembled in period t
Z_{1t}	number of EOL product purchased in period t
S_{it}	number of item i sold in period t
I_{it}	inventory level of item i at the end of period t
U_{it}	number of unsatisfied demand i in period t

Y_{it}	1,	if setup occurs for item i in period t
	0,	otherwise

Mathematical model is given below:

$$\text{Maximize} \qquad \sum_{t=1}^{T}\sum_{i=1}^{I} pc_i S_{it} - \Bigg[\sum_{t=1}^{T}\sum_{i=1}^{i_{l-1}} c_i Y_{it} + \sum_{t=1}^{T}\sum_{i=1}^{i_{l-1}} o_i X_{it} + \sum_{t=1}^{T}\sum_{i=1}^{I} h_i I_{it} + \sum_{t=1}^{T}\sum_{i=1}^{I} pn_i U_{it} + \sum_{t=1}^{T} p_t Z_{1t} \Bigg]$$

subject to

$$X_{it} \le cap_i \qquad \forall i \in I, t \in T \qquad (1)$$

$$X_{it} \le MY_{it} \qquad \forall i \in I, t \in T \qquad (2)$$

$$\beta_{it} d_{it} \le S_{it} \le d_{it} \qquad \forall i \in I, t \in T \qquad (3)$$

$$I_{it} = I_{i,t-1} + Z_{1t} - X_{it} \qquad i{=}1,\ 2,...,i_r \qquad (4)$$

$$I_{it} = I_{i,t-1} + \sum_{j \in PS_i} a_{ji} X_{jt} - X_{it} - S_{it} \qquad i{=}i_{r+1},...,i_{l-1} \qquad (5)$$

$$I_{it} = I_{i,t-1} + \sum_{j \in PS_i} a_{ji} X_{jt} - S_{it} \qquad i{=}i_l,...,I \qquad (6)$$

$$Z_{1t} \ge 0 \qquad \forall t \in T \qquad (7)$$

$$X_{it} \ge 0 \qquad \forall i \in I, t \in T \qquad (8)$$

$$S_{it} \ge 0 \qquad \forall i \in I, t \in T \qquad (9)$$

$$U_{it} \ge 0 \qquad \forall i \in I, t \in T \qquad (10)$$

$$I_{it} \ge 0 \qquad \forall i \in I, t \in T \qquad (11)$$

$$I_{i0} = 0 \qquad \forall i \in I \qquad (12)$$

$$Y_{it} \in 0, 1 \qquad \forall i \in I, t \in T \qquad (13)$$

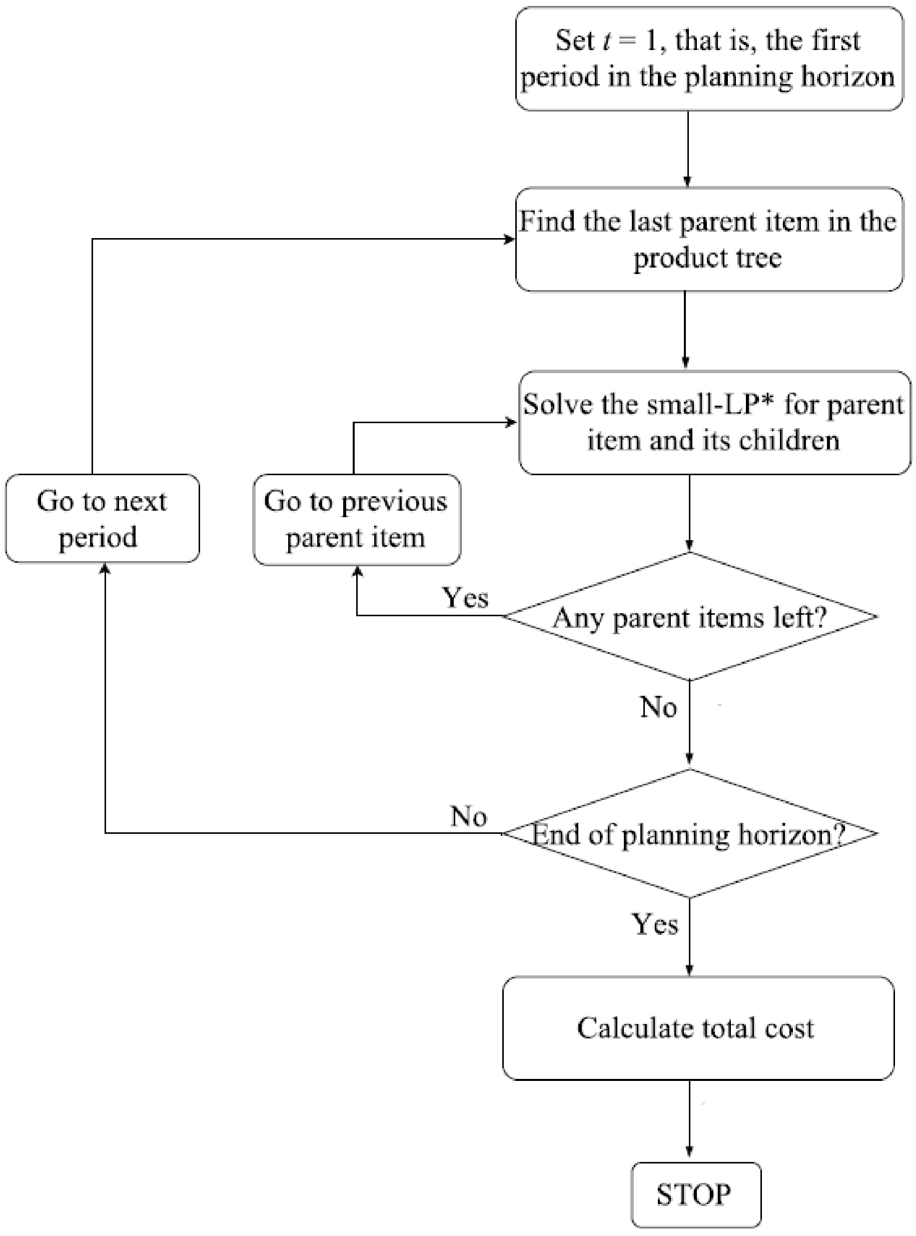
The objective function maximizes revenue as it is stated in previous page. Revenue here, is the difference between sales and disassembly process related costs that are setup, disassembly operation, purchasing, inventory holding and penalty cost. Constraint (1) guarantees that setup occurs if there are any disassembly operations done in corresponding period. Constraint (2) represents the capacity limitation in each period. That is, total number of disassembled items in a period cannot exceed given capacity for corresponding item same period. Constraint (3) ensures that the amount of items sold cannot exceed demand for same item while it also cannot be less than pre-defined unsatisfied demand allowance. This is defined by fill rate parameter, which ensures that at least demand's fill rate should be satisfied. In other words, lower limit of not satisfying a demanded item is set by fill rate in each period. Constraints (4), (5) and (6) define inventory balance level for root item (EOL product), intermediate (subassemblies) and leaf items (parts/components). Particularly, constraint (4) specifies inventory balance level for root items which differs from remaining inventory level constraints with not including amount of disassembly coming from its parent. As it is stated above, this is because root items do not have any parent items. Additionally, constraints (7), (8), (9,), (10), (11), (12) and (13) represent decision variable conditions. In particular, constraint (10) ensures that backlogging is not allowed while constraint (12) implies that initial inventory for all items equal to 0.

Chapter 4

Proposed Heuristic Solution

The capacitated disassembly leveling and scheduling problem that is described in this study can be solved using the mathematical model described in detail in Chapter 3. However, solving the mathematical model using a commercially available integer programming software takes significantly large amount of computation time especially for large size practical problems. Therefore, a faster heuristic that can give near-optimal solutions within short amount of computation times is suggested. Next, the mathematical model demonstrated above and suggested algorithm's performance are compared.

This chapter explains the heuristic algorithm suggested in this study. Before explaining the algorithm in detail, Figure4.1 is demonstrated to clarify the overall procedure of the heuristic algorithm.



Set t = 1, that is, the first period in the planning horizon
Find the last parent item in the product tree
Solve the small-LP* for parent item and its children
Go to next period
Go to previous parent item
Yes
Any parent items left?
No
No
End of planning horizon?
Yes
Calculate total cost
STOP

*LP that is developed for the subproblems.

Figure 4.1: Flow of the Proposed Heuristic Algorithm

Since a problem that allows shortage is studied, the trade-off between cost of not satisfying demanded item on fill rate limitation with satisfying the demanded amount and the cost of holding nondemanded items in inventory is focused in this heuristic algorithm. In other words, if demand is not satisfied, there exists penalty cost. In contrast, although demand is satisfied, because of dependency of the disassembly structure explained earlier, there may also exists on-hand inventory. Hence, the objective is to decide whether to lose sales or hold items in inventory if there are any.

In this scope, a myopic mathematical model that solves the subproblems in a backward attitude is developed. In other words, the suggested heuristic solution runs in a bottom up trend. Particularly, the algorithm divides the main problem into small problems based on parent-child relationship. Since each parent and its children composes a family, a small-sized problem or subproblem is basically a family itself.

Figure 4.1 shows the flow of the algorithm which begins from last parent item, and solves this small-sized LP. Note that, unlike the mathematical model demonstrated in Chapter 3.1 before, in this small-LP only inventory and penalty costs are considered. Therefore, the small-LP here does not include other costs such as setup, operation, purchasing. More specifically, subproblems tend to minimize the total cost that is, summation of inventory holding and penalty costs.

From last parent to first parent item subproblems are solved for each parent and its children. This is why the proposed heuristic algorithm has a myopic sight because it runs in a backward fashion in terms of parent numbers. This solution gives information of the amount of children items to be sold and inventory held which provides the necessary parent item disassembly amount. The amount of parent item to be disassembled is set in order to use in next family which includes this specific parent item's parents. Here, calculating amount of parent item to be disassembled is the the main decision behind inventory and penalty concepts for d-BOM structures, since it is to decide whether to disassemble or not and if yes, decide the amount to be disassembled.

Following the small-LP solution, the proposed heuristic algorithm continues solving the small-LP for rest of the subproblems or families in the product structure until all items are calculated. When small-LP is solved for all items in corresponding product tree, algorithm continues with the next period in the planning horizon and again starts to solve subproblems in a backward fashion. This shortsighted approach is followed until all of the periods in planning horizon is comprised. Then, using solutions obtained, the objective value that is demonstrated in previous chapter, also stated in detailed procedure of the algorithm below, is calculated.

The detailed procedure of proposed algorithm is as follows:

Algorithm Proposed Heuristic Algorithm

```
1: Set $t = 1$ (period number)
2: Set $p = P$ (parent item)
3: Solve below mathematical model for corresponding parent and its children
```

$$\text{Minimize} \qquad \left[\sum_{i=1}^{I} h_i I_{it} + \sum_{i=1}^{I} pn_i U_{it}\right]$$

subject to

$$
\begin{aligned}
X_{it} &\le cap_i & \forall i \in I, t \in T && (1)\\
X_{it} &\le MY_{it} & \forall i \in I, t \in T && (2)\\
\beta_{it} d_{it} \le S_{it} &\le d_{it} & \forall i \in I, t \in T && (3)\\
I_{it} &= I_{i,t-1} + Z_{1t} - X_{it} & i{=}1,\ 2,...,i_r && (4)\\
I_{it} &= I_{i,t-1} + \sum_{j \in PS_i} a_{ji} X_{jt} - X_{it} - S_{it} & i{=}i_{r+1},...,i_{l-1} && (5)\\
I_{it} &= I_{i,t-1} + \sum_{j \in PS_i} a_{ji} X_{jt} - S_{it} & i{=}i_l,...,I && (6)\\
Z_{1t} &\ge 0 & \forall t \in T && (7)\\
X_{it} &\ge 0 & \forall i \in I, t \in T && (8)\\
S_{it} &\ge 0 & \forall i \in I, t \in T && (9)\\
U_{it} &\ge 0 & \forall i \in I, t \in T && (10)\\
I_{it} &\ge 0 & \forall i \in I, t \in T && (11)\\
I_{i0} &= 0 & \forall i \in I && (12)\\
Y_{it} &\in 0, 1 & \forall i \in I, t \in T && (13)
\end{aligned}
$$

```
4: Set $p = P - 1$
   **if** $P = 0$ go to **step 5**
   **else** go to **step 3**
5: Set $t = t + 1$
   **if** $t = T$ go to **step 6**
   **else** go to **step 2**
6: Calculate total cost as
```

$$\sum_{t=1}^{T}\sum_{i=1}^{I} pc_i S_{it} - \Bigg[\sum_{t=1}^{T}\sum_{i=1}^{i_{l-1}} c_i Y_{it} + \sum_{t=1}^{T}\sum_{i=1}^{i_{l-1}} O_i X_{it} + \sum_{t=1}^{T}\sum_{i=1}^{I} h_i I_{it} + \sum_{t=1}^{T}\sum_{i=1}^{I} pn_i U_{it} + \sum_{t=1}^{T} p_t Z_{1t}\Bigg]$$

```
7: **STOP**
```

In more detail, the algorithm suggested in this study works as follows: Here, T denotes the number of periods in the planning horizon where P denotes parent items in the EOL product, e.g. in Figure 3.2 there exists 5 parent items which are A, B, C, D and E as it is mentioned in Chapter 3. Since the heuristic runs in a backward fashion, it starts from the last parent item and calculates related variables for each family. For instance, if d-BOM structure in Figure 3.2 is considered, the heuristic starts from last parent item; item E and it solves the model for its children I and J, so E, I and J composes a family or a subproblem in other words. This small-LP with minimization objective solves the subproblem and then number of parent item, E in our example, is set for further use in the following steps. Next, the algorithm continues with (P-1)th parent item which is D in corresponding example and its children G, H. With this backward attitude, until all of the subproblems are solved, the algorithm continues with following period in the planning horizon. Until the end of the planning horizon is reached, these steps are repeated in given behavior. When planning horizon is completed, total cost, which is same as the mathematical model demonstrated before, is calculated and the algorithm ends.

Chapter 5

Computational Experiments

This chapter presents the experimentation and results on comparison of both solution approaches. To evaluate the performance of both solution approaches, computational experiments are performed on a testbed composed of randomly generated problems. Random test instances are generated based on carefully selected various design parameters which are explained in following subsections. According to specific design parameters, first, a d-BOM generator is developed in order to generate random instances practically. Then, rest of the parameters and settings for the capacitated disassembly scheduling problem considered in this thesis are described. With the results of the experiments, design parameter effects on percentage deviations from both solution approaches are discussed.

5.1 Experimental Design Parameters

The lack and need of consideration of various design parameters in disassembly scheduling field in literature led us to investigate which parameters affect and how they affect the solution. Gokgur et al. (2015) is first to mention the importance of other design parameters in a product tree, which also provided us to see our problem's settings and limits clearly. Since this study integrates mathematical model and heuristic approaches at the same time, interpreting the design parameter effects on both approaches in terms of execution time and percentage deviation of objective function from optimal can help learning more about the problem's structure, difficulty and performance of the proposed heuristic in general. As it is explained more detailed in Chapter 4, due to proposed heuristic algorithm's execution scheme, number of levels configuration is needed to be questioned since proposed heuristic algorithm runs in backward fashion from bottom level to top in the product tree. Additionally, since a multi-period disassembly scheduling problem is studied in this thesis, effect of a change in number of periods is also investigated as a design parameter.

The product tree structure considered here can be fully described by the following design parameters:

1. Number of items

2. Number of levels

3. Number of periods

5.1.1 Number of items

In a product structure, *number of items* denotes total number of items in an EOL product including intermediate and leaf items. Since single product without parts commonality product structure is considered in this study, there exists only one root item in all of the problems that are generated with different number of items. Particularly, a d-BOM structure with 10, 20 and 30 items both have one root item that is the EOL product. This product tree design parameter is essential and also the most commonly found in literature. For instance, Ji et al. (2016) considered 10, 20 and 30 item design levels which is the same number of item assumption as in this thesis.

5.1.2 Number of levels

This product tree-based parameter represents the depth of d-BOM structure. *Number of levels* in a product tree specifies the depth that a product structure can expand. For instance, the first d-BOM shown in Figure 3.2 root item A's level is the 0^{th} level where B and C are in the 1^{st}, D and E are in the 2^{nd} level. Hence, a total of 3 levels exist in the product tree as it is shown in corresponding figure.

This specific design parameter is investigated because the deeper a product tree expands; more complex the problem gets in terms of execution time and percentage gap both. As number of levels increase, dependencies between items due to disassembly structure are bound to make the problem's solution more complex. Besides, the proposed solution algorithm runs in a backward fashion while solving subproblems not taking account of the whole problem. So, this subproblems are not able to see far away as it is stated in previous section. In other words, it has lack of foresight which leads us to question how this level depth design affects the solution quality. Three d-BOM levels (2, 3 and 4) are considered in this study. To the best of author's knowledge, this study is first to consider number of levels in in a product tree structure design considering heuristic approach in disassembly domain.

5.1.3 Number of periods

Number of periods parameter determines the number of time in the planning horizon. It does not affect and change product structure, however, it affects problem structure with time dependencies. Therefore, it is directly related with other parameters which varies in time such as demand, fill rate and capacity. The planning horizon designs are specified with 3,5 and lastly 7 in order to be able to construct weekly plans. Design parameter values that are considered as number of periods here also used in the studies of Kim et al. (2016). Table 5.1 summarizes design parameter values in detail.

Table 5.1: Design Parameter Values

Design Parameters	Values		
number of items	10	20	30
number of levels	2	3	4
number of periods	3	5	7

Based on these design parameters, a d-BOM generator is developed in order to create various product structures in a practical way which is explained in following subsection and with this generator random test instances are illustrated.

5.2 d-BOM Generator

The focus of this study is, single product without parts commonality type of disassembly scheduling problem is considered. In other words, product structure studied in this thesis has only one root item (EOL product) while having more than one parent item is not allowed. Accordingly, design parameters are decided first and based on these parameters an application is coded via Microsoft Office Visual Basic for Applications (VBA). This application gives random product trees with user-determined (preferred) configuration, that is, input specification mentioned in previous subsections. The d-BOM generator works as follows:

Step 1: Set number of items that an EOL item have

Step 2: Set yield ratio

Step 3: Generate d-BOM

5.3 Experimentation and Results

For the test on test instances with different sizes, 135 test instances are generated, i.e. 5 instances for each combination of three levels of the number of items (10, 20, and 30), three d-BOM levels (2, 3 and 4) and three levels of the number of periods are generated (3, 5 and 7). In addition to the design parameters investigated in previous section, there are also other parameters that need to be determined in order to generate test instances for the experimentation set. A capacitated disassembly scheduling problem with cost-based objective function requires parameters such as demand, capacity and other various costs. Distribution of such parameters are determined as follows:

In the d-BOM structures, the yields from parent items were generated from DU(1, 3), as Kim et al. (2006), where DU(a, b) denotes the discrete uniform distribution with range [a, b]. There is demand for each item except root item and with 0.1 probability there is no demand for an item in a period. With 0.9 probability demand is generated from DU(0, 100). Time-invariant costs, that are, setup cost, disassembly operation cost and inventory holding cost are generated from DU(10, 30), DU(1, 5) and DU(10, 20), respectively. These specific distributions are also used before in the studies of Kim et al. (2016), Gokgur et al. (2015) and Kim and Lee (2011). Selling price of an item is generated from DU(200, 400) except root item which leads to another parameter, fill rate. The parameter of fill rate is generated between DU(80, 99)/100. Taking into account of unsatisfied items, which is limited by fill rate parameter, there exists penalty cost for each unsatisfied demanded item below the fill rate. There is a trade off between penalty cost and selling price and it is set as follows:

$$pn_i = pc_i \times 0.10 \qquad \forall i \in I \tag{5.3.1}$$

where

pc_i	selling price of item i
pn_i	penalty cost of not satisfying one unit of item i

Capacity parameter considered in this thesis is the maximum amount of item to be disassembled in a period. It is a time-invariant parameter however, it differs from item to item. This is practically true as the disassembly of an item depends on the way it is assembled more than the value of the item itself. This workload is restricted with fill rate and demand variables and calculated as follows:

$$cap_i = \sum_{i=1}^{I} \sum_{t=1}^{T} \beta_{it} \times d_{it} \qquad \forall i \in I, t \in T \tag{5.3.2}$$

where

cap_i	aggregated capacity for item i
β_{it}	fill rate of item i in period t
d_{it}	demand of item i in period t

To demonstrate the performance of both mathematical model and proposed heuristic algorithm, using above parameters two performance measures are used. They are: (a) percentage gap of heuristic solutions from optimal solution values; and (b) CPU seconds. Here, the percentage gap from an optimal solution value for a problem is defined as

$$\frac{(Z - Z^*)}{Z} \times 100$$

where Z* is the solution value obtained from the proposed heuristic algorithm in this study and Z is an optimal solution value from mathematical model. Here, the optimal solution values are obtained by solving mathematical model using CPLEX 12.7.1.0, a commercial integer programming software package whereas Visual Studio 2015 C++ language version is used for heuristic algorithm. The tests were done on the same personal computer with an AMD A6 3400M 1.40GHz clock speed and results are summarized in following table. Note that, each row represents results from five instances with same design parameter configuration.

Table 5.2: Experimentation Results

				GAP*			**CPU Seconds****			**CPU Seconds*****	
i	***l***	***t***	**Min**	**Mean**	**Max**	**Min**	**Mean**	**Max**	**Min**	**Mean**	**Max**
10	2	3	0.012	0.211	0.455	<0.1	<0.1	<0.1	14	19.2	23
10	2	5	0.139	0.370	0.573	<0.1	<0.1	<0.1	19	24.6	30
10	2	7	0.370	0.492	0.646	<0.1	<0.1	<0.1	28	29.4	32
10	3	3	0.022	0.253	0.605	<0.1	<0.1	<0.1	17	21.4	26
10	3	5	0.125	0.440	0.783	<0.1	<0.1	<0.1	21	26	29
10	3	7	0.168	0.642	0.935	<0.1	<0.1	<0.1	29	31.4	33
10	4	3	0.003	0.278	0.868	<0.1	<0.1	<0.1	20	22	24
10	4	5	0.313	0.470	0.603	<0.1	<0.1	<0.1	24	26.4	29
10	4	7	0.416	0.485	0.589	<0.1	<0.1	<0.1	30	33.6	39
20	2	3	0.274	0.545	0.835	<0.1	<0.1	<0.1	24	26.2	28
20	2	5	0.291	0.693	1.004	<0.1	<0.1	<0.1	27	30.2	32
20	2	7	0.647	0.817	1.148	<0.1	<0.1	<0.1	34	35.8	38
20	3	3	0.369	0.557	1.005	<0.1	<0.1	<0.1	25	26.8	29
20	3	5	0.128	0.778	1.313	<0.1	<0.1	<0.1	30	30.6	32
20	3	7	0.301	0.819	1.644	<0.1	<0.1	<0.1	34	37.2	39
20	4	3	0.298	0.657	0.991	<0.1	<0.1	<0.1	27	28.8	30
20	4	5	0.429	0.806	1.323	<0.1	<0.1	<0.1	30	33.2	35
20	4	7	0.611	0.873	1.112	<0.1	<0.1	<0.1	36	38.2	40
30	2	3	0.773	1.000	1.334	<0.1	<0.1	<0.1	29	30.8	33
30	2	5	1.021	1.151	1.330	<0.1	<0.1	<0.1	37	38.4	40
30	2	7	0.703	1.187	1.669	<0.1	<0.1	<0.1	44	45.4	47
30	3	3	0.289	1.296	2.517	<0.1	<0.1	<0.1	32	33.8	36
30	3	5	0.633	1.352	2.362	<0.1	<0.1	<0.1	38	40	41
30	3	7	1.083	1.416	2.268	<0.1	<0.1	<0.1	48	49.6	51
30	4	3	0.825	1.327	1.937	<0.1	<0.1	<0.1	38	39.6	41
30	4	5	1.037	1.492	2.251	<0.1	<0.1	<0.1	44	44.8	46
30	4	7	1.227	1.811	2.459	<0.1	<0.1	<0.1	50	50.8	52

i number of items
l number of levels
t number of periods
* Percentage gap from optimal solutions
** CPU seconds of Visual Studio 2015
** CPU seconds of CPLEX 12.7.1.0

In Table 5.2 it is shown that overall average of mean percentage gaps is less than 1% for 135 instances. Percentage gap values differ between a minimum of 0.003% and a maximum of 2.517% which denotes that even for the most complex design levels in terms of number of items, levels and periods, the percentage gap between both solution approaches does not exceed 2.517%. Based on these experimentation results, it can be concluded that solution algorithm proposed in this thesis can give near-optimal solutions for all cases.

On the other hand, as anticipated as the problem difficulty (in terms of the number of periods) increases the CPU seconds increase significantly for CPLEX. However, the heuristic gave solutions for all the test instances within less than 0.1 sec while optimal solution requires slightly longer computation times than heuristic solution. This implies that heuristic can be applied to situations in which the computation time is critical. In addition, the heuristic can also be used for solving subproblems since algorithm solves subproblems first in a backward fashion. Following subsection examines effects of design parameters in detail.

5.4 Analysis of effects of design parameters

While designing product trees for computational experiments three different parameters are considered that are number of items, number of levels and number of periods. Based on each of these parameters, three different level designs are selected. In other words, we have different levels of different parameters that we wish to compare. For instance, the effect of a single parameter such as three levels of number of items (2, 3 and 4) on percentage deviation. It is done with the hope that we can get better insights into what most affects the performance of the proposed heuristic by means of analysis of variance (ANOVA).

However, in order to be able to use ANOVA, according to Hines et al. (2003), three basic assumptions have to be simultaneously satisfied that are *independency of cases*, *normality* and *homoscedasticity.* Note that, effect of design parameters on CPU seconds of Visual Studio 2015 is not analyzed in this subsection since heuristic algorithm solves the problem in less than CPU seconds 0.1 for all instances. Additionally, mathematical model's execution time increases as the problem gets more difficult (in terms of item number, level number and period number), which is analyzed. Additionally, percentage gaps of heuristic solution's objective function from the optimal are investigated in this subsection.

In this scope, first, independency of cases validity is questioned for this data set. Since test instances constructed in our study are generated randomly (using different random seeds each time) for each instance independently, this condition is satisfied. Normality requirement on the other hand, is tested with Kolmogorov-Smirnov test using Minitab 17. This test is based on the empirical cumulative distribution function (ECDF) and compares ECDF of the sample data with the distribution expected if the data were normal. According to the test results, the p-value of this test is 0.017, that is, less than our chosen significance level (0.05), hence, the null hypothesis of population normality is rejected. Related results from Kolmogorov-Smirnov test with a normality plot and a histogram is shown in figures 5.1 and 5.2.

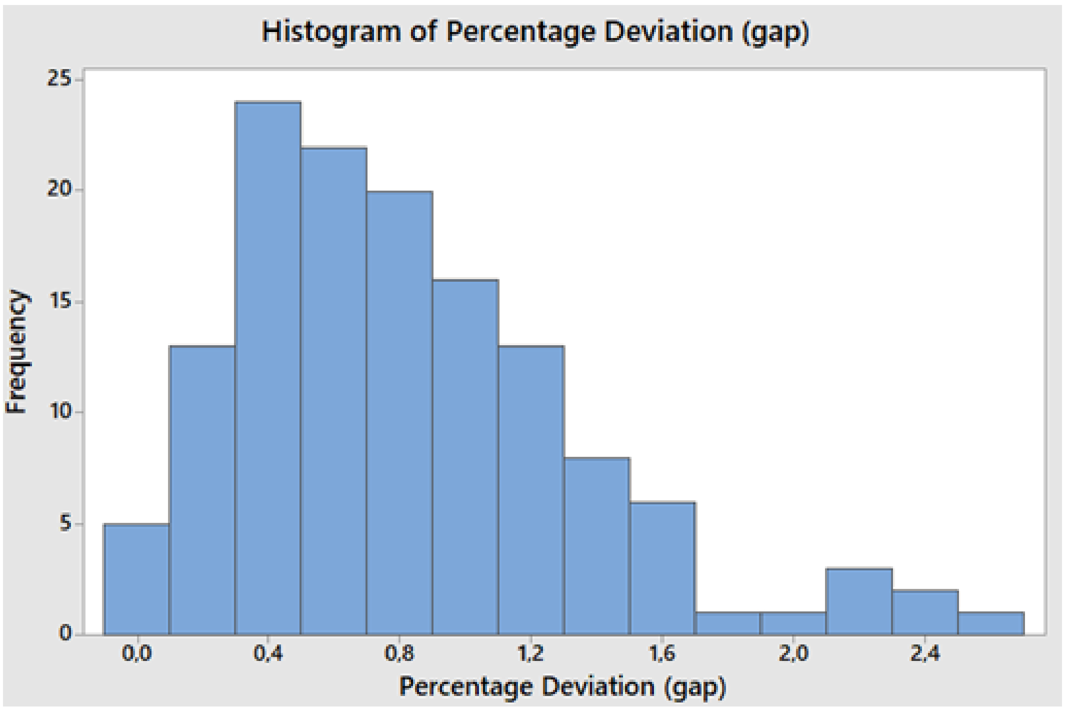


Figure 5.1: Histogram of Percentage Deviation (gap)

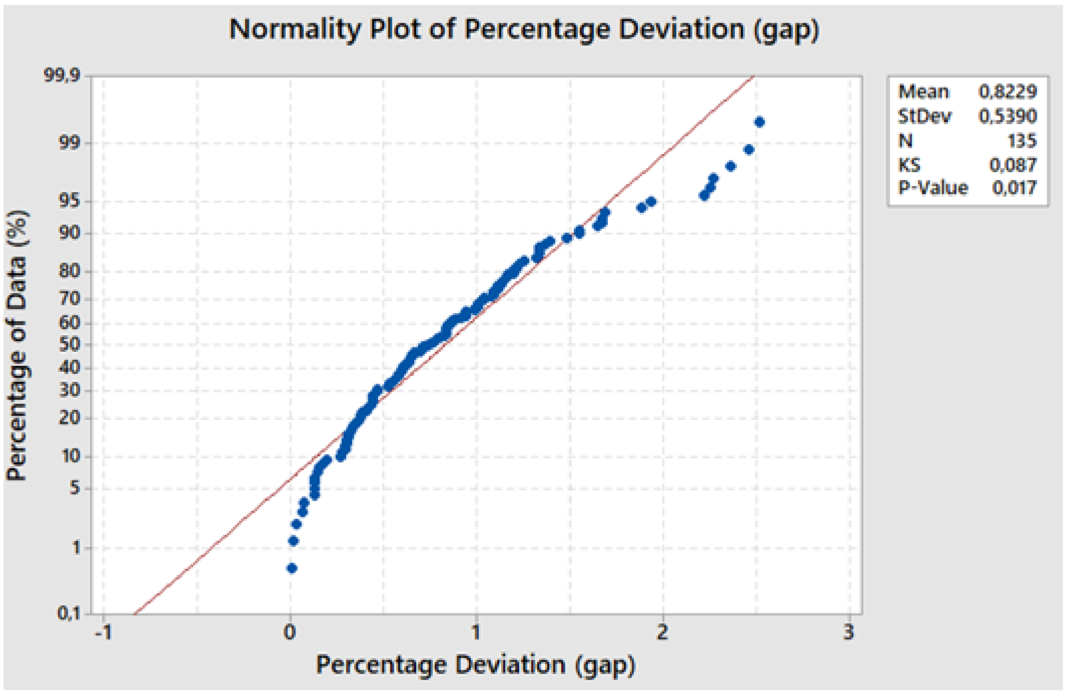


Figure 5.2: Normality Plot of Percentage Deviation (gap)

To reduce the variability in this data, logarithmic base transformation was applied and Kolmogorov-Smirnov test was repeated with this new logarithmic based data. Figure 5.3 demonstrates the normality plot gathered from logarithmic based percentage deviation data. The points do not form an approximate straight line and p-value is less than 0.05 with a 95% confidence interval. Therefore, it is concluded for the second time that, population does not satisfy normality.

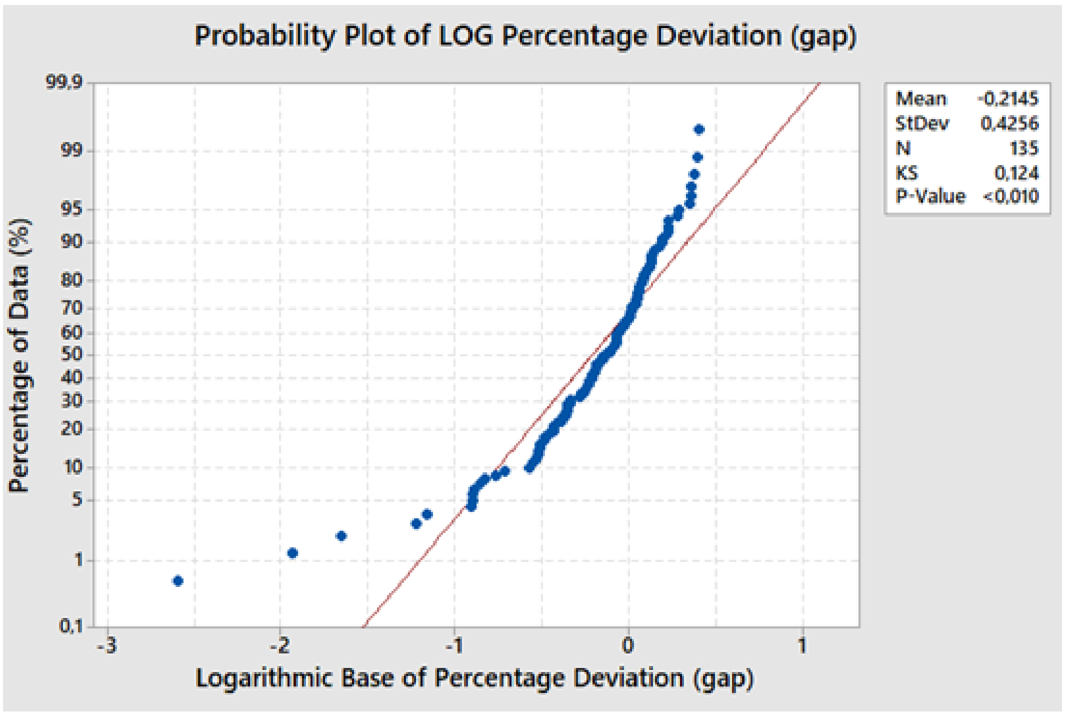


Figure 5.3: Normality Plot of Logarithmic Base Percentage Deviation (gap)

Since three conditions that ANOVA requires are not fulfilled with the data observed in this study, ANOVA cannot be conducted in order to find out effect of design parameters in terms of percentage deviation. Therefore, a main effect analysis is conducted since the aim is to investigate a single independent variable's effect on a dependent variable i.e. number of items effect on percentage deviation.

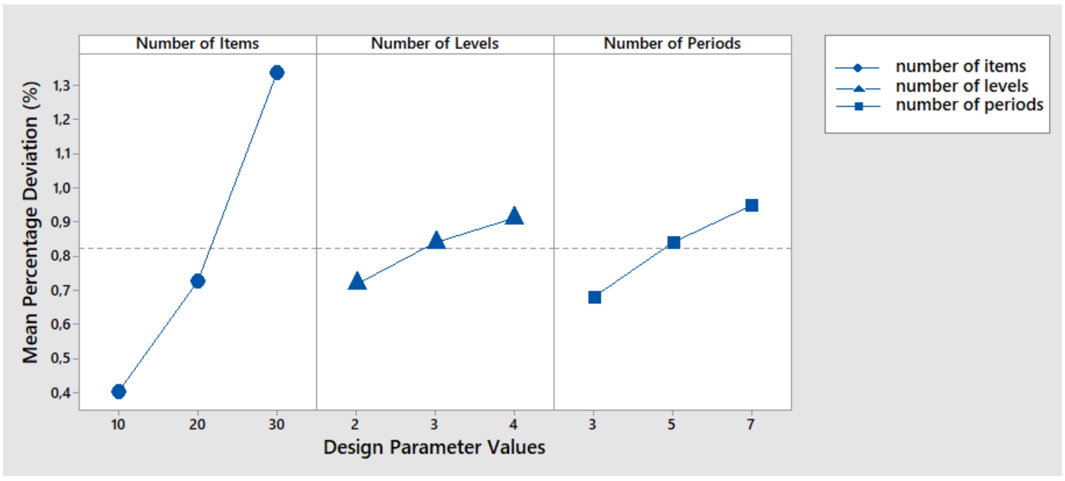


Figure 5.4: Main Effect Analysis Output on Percentage Deviation

In Figure 5.4, main effect analysis is shown graphically that is, each design parameters' effect on solution quality of both approaches in terms of objective value. As it is expected, number of items in a product structure directly affects solution quality in terms of percentage gap. Especially while solving mathematical model via CPLEX. In particular, increasing number of items from 10 to 20 has less impact than increasing it to 20 to 30.

When the number of levels design parameter is examined, d-BOM depth in other words, as the product tree gets deeper, gap between both solution values gets bigger. However, this increase in percentage deviation is not as big as number of items effect.

Number of periods effect that is shown in the third graph gives us the information that as number of periods increases, percentage gap between two solutions also increases. This was also an expected result since problem expands with increasing number of periods. Figure 5.4 illustrates that, according to this data, change in number of items makes the most difference comparing to number of levels and number of periods parameters.

Additionally, the design parameter effect on CPU seconds of CPLEX is graphically presented in Figure 5.5. As it is mentioned this was an expected result especially for number of items and number of periods designs. Number of levels, on the other hand, was an unpredictable result in terms of its effect on optimal solution. Because since our heuristic algorithm runs in a backward fashion which causes a myopic sight, it was expected for us to see the execution time increase proportionally with the level depth. However, since we have reasonable number of test instances in this study, even the deepest level problem is solved under 0.01 seconds. Although this is a satisfactory output in terms of time scale, we still need to widen our experimentation set in order to observe the solution quality in terms of near optimality.

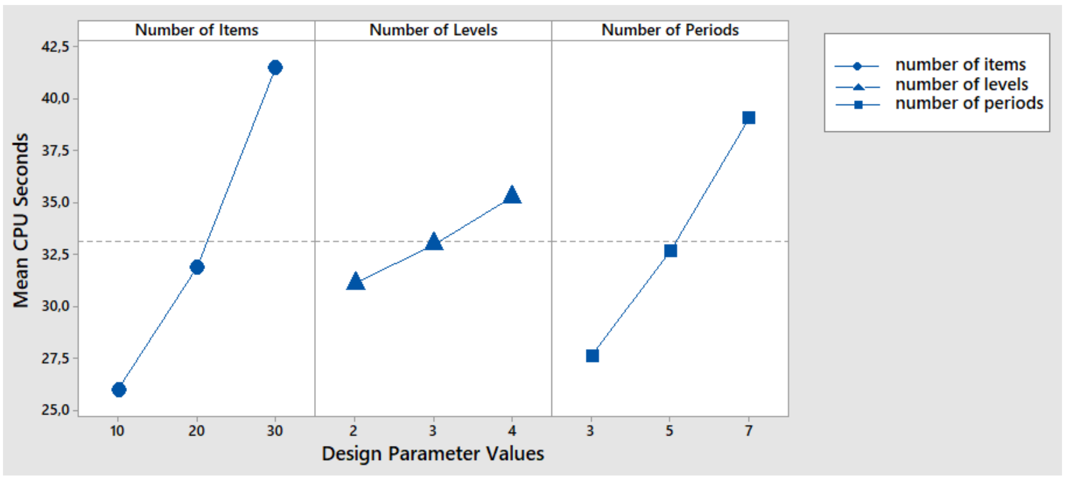


Figure 5.5: Main Effect Analysis Output on CPU Seconds of CPLEX

It can be concluded that Figure 5.5 shows distinctly that problem complexity in terms of number of items, number of levels and number of periods design parameters directly affects the CPU seconds of CPLEX. Since highest increase in CPU Seconds is because of the number of items design parameter, this shows us that for large scale problems, mathematical model performance gets worse in terms of execution time. Proposed heuristic approach on the other hand, solves the problem less than 0.1 seconds even for most complex design levels. For a more detailed representation of number of items design parameter effect, Figure 5.6 is constructed additionally.

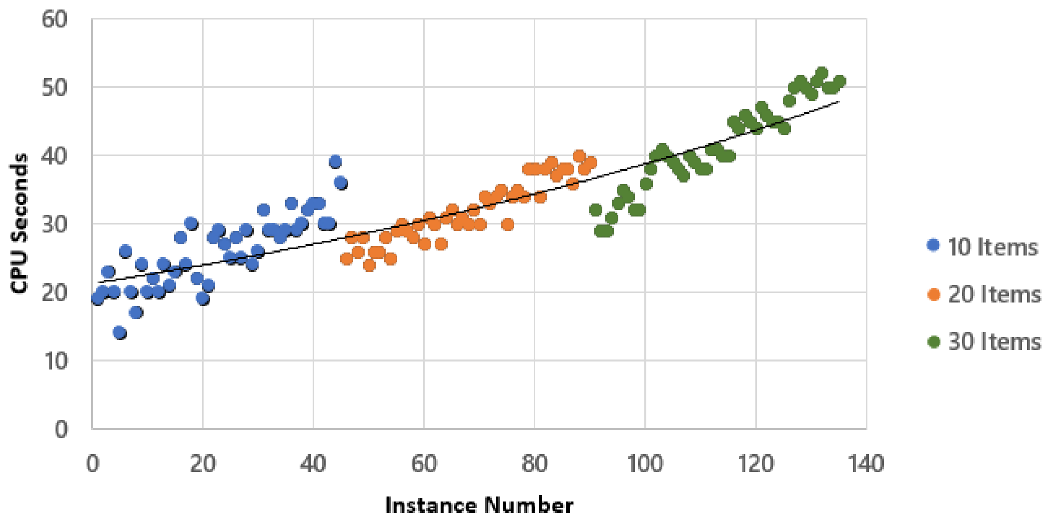


Figure 5.6: Scatter Plot on CPU Seconds of CPLEX

Above figure shows an uphill pattern which indicates a positive nonlinear relationship between number of items and CPU seconds of CPLEX. As the test instances changes on a number of item design basis (increasing number of items), the CPU seconds of CPLEX tend to increase. Note that, mathematical model execution time reaches up to almost 60 sec as the problem gets more complex in terms of design parameters described at the beginning of this chapter. On the other hand, the proposed solution approach solves the problem under 0.1 sec for all cases which gives us the information that in terms of execution time, proposed solution approach is a faster alternative.

Chapter 6

Conclusion and Future Study

In this study, we considered the problem of determining the disassembly schedule of used/EOL products subject to capacity restrictions, while satisfying the demand of their parts/components over a given multi-period planning horizon. The case of single product without parts commonality product structure is focused for the objective of maximizing revenue. To represent the problem mathematically, a MIP model with profit-based objective function and capacity constraints was developed. The proposed MIP model considers setup, operation, purchasing, inventory holding and penalty costs while allowing demand for both subassemblies and parts/components. Then, besides the mathematical model, in order to solve large-scale problems in reasonable computation times, a heuristic solution is suggested which provides a scheduling mechanism for disassembly. It should be emphasized that, the main contribution of this study is to integrate this extended MIP model whose objective is to maximize revenue, with a heuristic approach on multi-period capacitated disassembly scheduling. In order to see the proposed heuristic algorithm's effectiveness, we constructed a numerical study on both solution approaches. To generate random instances, also a product tree generator is developed which specifies number of items, number of children and number of level parameters which helped us practically generate test-bed problems. To the best of author's knowledge, this thesis is first to specify d-BOM level depth and maximum number of children as a design parameter with a heuristic algorithm approach considering capacity restriction and multi-period planning horizon. Computational experiments carried out on randomly generated test instances and the experimentation results showed that the proposed heuristic algorithm gave near optimal solutions within a very short amount of computation time.

As future research directions, this thesis can be extended in several ways. First, more test instances can be included in terms of both number of experimentations and number of design parameters. Doing more experimentations is also needed for learning more about the problem dynamics and evaluating current solution approaches. Second, since we considered the most basic form of the disassembly scheduling problem that is, single product without parts commonality, an extension with more general capacitated problems those with multiple product and parts commonality versions of the problem is an important further research topic that will result in more useful insights. Finally, it would be interesting to consider uncertain demand or yield in order to investigate the stochastic environment in disassembly domain.

Appendices

Appendix A

CPLEX Code

```
/*********************************************
 * OPL 12.7.1.0 Model
 *********************************************/

//sets and indices
int NbItems = ...;//item number
range item = 1..NbItems;
int NbPeriods = ...;//period number
range period = 1..NbPeriods;

//parameters

int yieldratio[item][item]=...; //yield ratio
int ParentOf[item]=...; //parent of the item
execute {
 for(var i=1; i<=NbItems;i++)
   for(var j=1;j<=NbItems;j++){ if (yieldratio[j][i]>0)
                                  ParentOf[j]=i;
        }
}

int Gozinto[item]=...;
//Gozinto=1 means it is a parent item, except root item
    (1st item) rest are intermediate items.
Gozinto=0 states leaf items.
execute {
      for(var i=1; i<=NbItems;i++)
                     for(var j=1;j<=NbItems;j++){
                                    Gozinto[i]=Gozinto[
                                       i]+yieldratio[j
                                       ][i];
                                        }
        for (var i=1; i<=NbItems;i++)
                if (Gozinto[i]>0)
                 Gozinto[i]=1;
                }

float Demand[item][period] = ...;//demand of item i in
   period t
float SellingPrice[item]=...;//selling price of item i
float PenaltyCost[item]=...;//penalty cost of not
   satisfying one unit of item i
float OperationCost[item]=...;//disassembly operation
   cost of item i
float InventoryCost[item]=...;//inventory holding cost
   of item i
float PurchasingCost[period]=...;//purchasing cost of
   EOL item
float SetupCost[item]=...;//setup cost of item i
float FillRate[item][period]=...;//fill rate of item i
   in period t
float Capacity[item]=...;//capacity restriction of item
    i
float Inventory0=...;//initial inventory = 0

int M=9999999999; //a very big number

//decision variables

dvar int+ Dis[item][period]; //number of item i
   disassembled in period t
dvar int+ Inv[item][0..NbPeriods]; //number of item i
   in inventory at the end of period t
dvar int+ Sold[item][period]; //number of sold item in
   period t
//dvar int+ Uns[item][period];//amount of unsatisfied
   demand of item i in period t
dvar boolean Stp[item][period];//if setup occurs for
   item i in period t 1, otherwise 0
dvar int+ NbPurchased[period]; //number of EOL item
   purchased OR scheduled receipts

//objective function: Maximize Revenue

maximize sum (i in 2..NbItems, t in period) (
   SellingPrice[i]*Sold[i][t]) -
                 sum (t in period) (PurchasingCost[t]*
                    NbPurchased[t]) -
                 sum (i in item: Gozinto[i]>0, t in
                    period) (SetupCost[i]*Stp[i][t]) -
                 sum (i in item: Gozinto[i]>0, t in
                    period)  (OperationCost[i]*Dis[i][t
                    ]) -
                 sum (i in item, t in period)  (
                    InventoryCost[i]*Inv[i][t]) -
                 sum (i in 2..NbItems, t in period)  (
                    PenaltyCost[i]*(Demand[i][t]-Sold[i
                    ][t] ))
;

subject to {

 forall(i in item ) //initial inventory constraint:
    initialinventory_const:
     Inv[i][0] == Inventory0;

forall( t in period)
//inventory balance constraints
        inventoryroot_const:
      Inv[1][t] == Inv[1][t-1] + NbPurchased[t]  - Dis
         [1][t]; // inventory balance constraint for
         root items

  forall(t in period, i in 2..NbItems: Gozinto[i]==1)
    inventoryinter_const:
      Inv[i][t] == Inv[i][t-1] + yieldratio[i][ParentOf
         [i]]*Dis[ParentOf[i]][t] - Dis[i][t] - Sold[i
         ][t]; // inventory balance constraint for
         intermediate item
     //IN(i,t) =IN(i,t-1) + sum(parents for (yieldratio
        (i,parent)>0 and ord(i)<>ord(parent)),
        yieldratio(i,parent)*X(parent,t)) - sum(j,X(i,t
        )) - S(i,t) ;

  forall( t in period, i in item: Gozinto[i]==0)
    inventoryleaf_const:
      Inv[i][t] == Inv[i][t-1] + yieldratio[i][ParentOf
         [i]]*Dis[ParentOf[i]][t] - Sold[i][t]; //
         inventory balance constraint for leaf items

  forall (t in period, i in item: Gozinto[i]>0) //setup
      constraint (no disassembly no setup)
    setup_const:
     Dis[i][t]<=M*Stp[i][t];

  forall (t in period, i in item: Gozinto[i]>0) //
     workload capacity constraint
    capacity_const:
     Dis[i][t]<=Capacity[i];

  forall (t in period, i in 2..NbItems) //sold item
     constraint
    solditem_const:
     FillRate[i][t]*Demand[i][t]<=Sold[i][t]<=Demand[i
        ][t];

  forall (i in item)  //unsatisfied demand constraint
    unsatisfieddemand_const:
     Uns[i][t]==Demand[i][t]-Sold[i][t];

 }

/*********************************************
 * OPL 12.7.1.0 Data
 *********************************************/
NbPeriods = 3;
NbItems = 10;


Gozinto = [0 0 0 0 0 0 0 0 0 0];
ParentOf= [0 0 0 0 0 0 0 0 0 0];

SheetConnection my_sheet("mydata.xlsx");

Inventory0 from SheetRead(my_sheet, "inventory0");
yieldratio from SheetRead(my_sheet,"yieldratio");
Demand from SheetRead(my_sheet,"demand");
SellingPrice from SheetRead(my_sheet, "sellingprice");
PenaltyCost from SheetRead(my_sheet, "penaltycost");
OperationCost from SheetRead(my_sheet, "operationcost")
   ;
InventoryCost from SheetRead(my_sheet, "inventorycost")
   ;
SetupCost from SheetRead(my_sheet,"setupcost");
FillRate from SheetRead(my_sheet,"fillrate");
Capacity from SheetRead(my_sheet,"capacity");
PurchasingCost from SheetRead(my_sheet, "purchasingcost
   ");
```

Appendix B

C++ Code

```
#include <ilcplex/ilocplex.h>
#include <stdlib.h>
#include <conio.h>
#include <fstream>
#include <algorithm>
#include <functional>
#include <queue>
#include <vector>
#include <time.h>
#include <math.h>
#include <stdio.h>
#include <tchar.h>

using namespace std;
typedef IloArray<IloNumArray> NumMatrix;
typedef IloArray<IloNumVarArray> NumVarMatrix;

int main() {
        printf("hello");
        char item[] = {
                'a','b','c','d','e','f','g','h','i','j'
        };

        int i, j, n;
        n = 9; // item - 1
        int a, k, m, b;
        a = 0;
        int asd = 0;
        float hold1[3][6];
        float info[10][6];
        int invenint[3];
        for (k = 0; k < 3; k++) //k period.
        {
          for (j = n; j >= 0; j--)
                 {
           for (i = 0; i <= n; i++)
                         {
if (yieldratio[j][i] != 0) {
printf("This␣is␣item␣%c␣,␣%d␣penaltycost␣%d␣inventory␣
   cost␣%d␣with␣demand␣of␣%d␣at␣period␣%d\n", item[i],
   penalty[i], inventory[i], yieldratio[j][i], demand[i
   ][k], k + 1);
a = a + 1;
printf("Fillrate␣of␣%c␣is␣%f\n", item[i], fillrate[i][k
   ]);
float* var1 = info[a - 1];
var1[0] = i;
var1[1] = penalty[i];
var1[2] = inventory[i];
var1[3] = yieldratio[j][i];
var1[4] = demand[i][k];
var1[5] = fillrate[i][k];

printf("%f,␣%f,␣%f,␣%f,␣%f\n", var1[0], var1[1], var1
   [2], var1[3], var1[4]);
}
if (a != 0)
{
printf("%c␣has␣%d␣children\n", item[j], a);

    for (b = 0; b < a; b++)
                 {
float* var2 = info[b];
printf("%f␣,␣%f,␣%f,␣%f,␣%f␣%f\n", var2[0], var2[1],
   var2[2], var2[3], var2[4], var2[5]);
hold1[b][0] = var2[0];
hold1[b][1] = var2[1]; //penalty
hold1[b][2] = var2[2]; //inventorycost
hold1[b][3] = var2[3]; //yieldratio
hold1[b][4] = var2[4]; //demand
hold1[b][5] = var2[5]; //fillrate
invenint[b] = var2[0];

                               };

IloEnv env1;
//try {
IloExpr obj(env1);
IloNumVarArray saless = IloNumVarArray(env1, a, 0,
   IloInfinity, IloNumVar::Int);
//IloNumVarArray furthdiss = IloNumVarArray(env1, a, 0,
    IloInfinity, IloNumVar::Float);
IloNumVarArray assignedinv = IloNumVarArray(env1, a, 0,
    IloInfinity, IloNumVar::Int);
IloNumVar xd = IloNumVar(env1, 0, IloInfinity,
   IloNumVar::Int);
IloModel model1(env1);
   for (IloInt i = 0; i < a; i++)
                                {
//obj += (hold1[i][1] * ((xd*hold1[i][3]) - (saless[i]
   + furthdiss[i]))) + (hold1[i][2] * ((xd*hold1[i][3])
    - (saless[i] + furthdiss[i])));
//obj += (hold1[i][1] * (hold1[i][4] - saless[i])) + (
   hold1[i][2] * ((xd*hold1[i][3]) - (saless[i] +
   furthdiss[i])));
obj += (hold1[i][1] * (hold1[i][4] - saless[i])) + (
   hold1[i][2] * assignedinv[i]);
}
        for (IloInt i = 0; i < a; i++)
                                {
model1.add((hold1[i][1] * (hold1[i][4] - saless[i])) >=
    0);
                                }
        for (IloInt i = 0; i < a; i++)
                                {
invenint[i] = asd;
model1.add((hold1[i][2] * ((xd*hold1[i][3]) - (saless[i
   ] + furtherdiss[asd][k]))) >= 0);
                                }
        for (IloInt i = 0; i < a; i++)
                                {
model1.add(hold1[i][5] * hold1[i][4] <= saless[i] <=
   hold1[i][4]);
                                }
model1.add(xd <= capacity[j]);
        for (IloInt i = 0; i < a; i++)
                                {
invenint[i] = asd;
model1.add(xd*hold1[i][3] >= saless[i] + furtherdiss[
   asd][k]);
                                }
        for (IloInt i = 0; i < a; i++)
                                {
int totalinvcostt = static_cast<int>(hold1[i][0]);
invenint[i] = asd;
model1.add(assignedinv[i] == (xd * hold1[i][3]) +
   totalinventory[totalinvcostt][k] - (saless[i] +
   furtherdiss[asd][k]));
                                }
```

```
model1.add(IloMinimize(env1, obj));
IloCplex solver(model1);
solver.solve();
cout << "\n\n";
cout << "the␣objective␣is␣" << solver.getObjValue() <<
   "␣and␣" << solver.getValue(xd) << endl;
        for (i = 0; i < a; i++)
                                {
cout << "sales␣" << solver.getValue(saless[i]) << "␣
   inventory␣" << solver.getValue(assignedinv[i]) <<
   endl;
                                }

assignedinventory[j][k + 1] = solver.getValue(xd);
operationcost[j][k] = operation[j] * assignedinventory[
   j][k + 1];
        for (i = 0; i<a; i++)
                                {
int totalinvcostt = static_cast<int>(hold1[i][0]);
totalinventory[totalinvcostt][k + 1] = solver.getValue(
   assignedinv[i]);
cout << solver.getValue(assignedinv[i]) << "inventory␣"
    << endl;
var3[totalinvcostt][k] = furtherdiss[totalinvcostt][k]
   + solver.getValue(saless[i]);
soldd[totalinvcostt][k] = solver.getValue(saless[i]);
                if (solver.getValue(saless[i] - hold1[i
                   ][4] < 0))
                                {
unsatisfieddemand[totalinvcostt][k] = hold1[i][4] -
   solver.getValue(saless[i]);
                                }
                                }
furtherdiss[j][k] = solver.getValue(xd);
cout << assignedinventory[j][k + 1] << "␣of␣" << item[j
   ];
//}
//catch (IloException& e) {
//      cerr << e << endl;
//}
env1.end(); // free all memory

/*IloNumVar xd = IloNumVar(env1, 0, IloInfinity);
IloNumVar sg = IloNumVar(env1, 0, IloInfinity);
IloObjective obj(env1, for (n = 0, n < a, n++) {
float* object = info[n];
object[1]*(totalinventory[n][k]+(xd*object[3])-(sg+,
   IloObjective::Minimize);
IloNumVar*/
                        }
                        a = 0;
                }
        }
/*IloEnv env;
IloModel model(env);

IloNumVar x1 = IloNumVar(env, 0, IloInfinity);
IloNumVar x2 = IloNumVar(env, 0, IloInfinity);

IloRange r1(env, 10, x1 + x2, 10);

IloObjective obj(env, (1 * x1) + (4 * x2), IloObjective
   ::Maximize);

//sum_to_one.setLinearCoef(x1, 15);
//sum_to_one.setLinearCoef(x2, 15);
//obj.setLinearCoef(x1, 2);
//obj.setLinearCoef(x2, 1);
model.add(r1);
model.add(obj);
//model.add(sum_to_one);

IloCplex solver(model);

solver.solve();
cout << "\n\n";
cout << solver.getObjValue() << "\n";*/
ofstream outfile;
outfile.open("C:/Users/Cemre/Desktop/result.csv");
//outfile << "Total Endperiod Inventory, " << endl;
for (i = 0; i<10; i++)
        {
// print first column's element
outfile << totalinventory[i][0];

// print remaining columns
        for (j = 1; j<k + 1; j++)
                {
                        outfile << ",␣" <<
                           totalinventory[i][j];
                }

// print newline between rows
                outfile << endl;
        }
        //outfile << "Assigned inventory, ";
        outfile << endl;
        for (i = 0; i<10; i++)
        {
// print first column's element
                outfile << assignedinventory[i][1];
// print remaining columns
                for (j = 1; j<k; j++)
                {
                        outfile << ",␣" <<
                           assignedinventory[i][j + 1];
                }
// print newline between rows
                outfile << endl;
        }
        outfile << endl;
        for (i = 0; i<10; i++)
        {
// print first column's element
                outfile << var3[i][0];
// print remaining columns
                for (j = 1; j<k; j++)
                {
                        outfile << ",␣" << var3[i][j];
                }
// print newline between rows
                outfile << endl;
        }
        outfile << endl;
        for (i = 0; i<10; i++)
        {
// print first column's element
                outfile << soldd[i][0];
// print remaining columns
                for (j = 1; j<k; j++)
                {
                        outfile << ",␣" << soldd[i][j];
                }
// print newline between rows
                outfile << endl;
        }
        cout << "ceil␣of␣0.3␣is␣" << ceil(0.00000003 /
           99999) << endl;
        int solddtotalcost1 = 0;
        int purchaseddtotalcost1 = 0;
        int setuptotalcost1 = 0;
        int inventorytotalcost1 = 0;
        int penaltytotalcost = 0;
        int operationtotalcost1 = 0;
        for (int j = 0; j < 3; j++) // sales
        {
                for (int i = 0; i < 10; i++)
                {
                        solddtotalcost1 =
                           solddtotalcost1 + (selling[i
                           ] * soldd[i][j]);
                }
        }

        /*for (int j = 0; j < 3; j++)
        {
purchaseddtotalcost1 = purchaseddtotalcost1 + (
   purchasing[j] * assignedinventory[0][j + 1]);
        }*/

        for (int j = 0; j < 3; j++)
        {
                for (int i = 0; i < 10; i++)
                {
                        float yardimci =
                           assignedinventory[i][j + 1];
                        setuptotalcost1 =
                           setuptotalcost1 + (ceil(
                           yardimci / 999999) * setup[i
                           ]);
                        //cout << ceil(
                           assignedinventory[i][j +
                           1]/999) << endl;
                }
        }

        for (int j = 0; j < 3; j++)
        {
                for (int i = 0; i < 10; i++)
                {
                        inventorytotalcost1 =
                           inventorytotalcost1 + (
                           inventory[i] *
                           totalinventory[i][j + 1]);
                }
        }

        for (int j = 0; j < 3; j++)
        {
                for (int i = 0; i < 10; i++)
                {
                        penaltytotalcost =
                           penaltytotalcost + (penalty[
                           i] * unsatisfieddemand[i][j
                           ]);
                }
        }

        for (int j = 0; j < 3; j++)
        {
                for (int i = 0; i < 10; i++)
                {
                        operationtotalcost1 =
                           operationtotalcost1 +
                           operationcost[i][j];
                }
        }

        cout << "Total␣Cost␣is␣" << solddtotalcost1 -
           purchaseddtotalcost1 - setuptotalcost1 -
           inventorytotalcost1 - penaltytotalcost -
           operationtotalcost1;

        cin.get();

}
```

Appendix C

VBA Code

```
Sub RNDBOM()
delete
Dim maxchildren As String
maxchildren = InputBox(Enter, "Enter␣maximum␣number␣of␣
   children␣per␣item", 3)
Dim randommmax As Integer
randommmax = maxchildren
maxchildren = Int((randommmax - 0 + 1) * Rnd + 1) '
   random␣generated␣maximum
Dim␣totalcount␣As␣Integer
totalcount␣=␣0
Dim␣a␣As␣Integer
Dim␣randomm␣As␣Integer
a␣=␣0
Dim␣maxrand␣As␣Integer␣'max random number
Dim LRandomNumber, LRandomNumber2 As Integer
LRandomNumber = InputBox("Number␣of␣items", "Item#",
   10) 'LRandomNumber␣␣'Int((10 - 5 + 1) * Rnd + 5)
LRandomNumber2 = LRandomNumber
ReDim arr(LRandomNumber, LRandomNumber2)
Dim yield As Integer
yield = InputBox(yield, "Enter␣Max␣Yield␣Ratio")

Dim counter As Integer
counter = 0
Dim countt As Integer
For i = 1 To LRandomNumber
countt = 0
    For j = 1 To LRandomNumber2

        If maxchildren < totalcount + 1 Or j = 1 Then
        Cells(i + 1, j + 1).Value = 0
        GoTo NextIteration
        Else
        LRandomNumber3 = Int((yield - 0) * Rnd + 1)
        If i + 1 = j Then
          arr(i, j) = LRandomNumber3
            Cells(i + 1, j + 1).Value = LRandomNumber3
        End If


        If a < counter And i <> 1 Then
            arr(i, j) = 0
            Cells(i + 1, j + 1).Value = 0
            a = a + 1
        Else
            arr(i, j) = LRandomNumber3
            Cells(i + 1, j + 1).Value = LRandomNumber3
            totalcount = totalcount + 1
            countt = countt + 1
        End If
        End If
NextIteration:
    Next j
    randomm = Int((randommmax - 0 + 1) * Rnd + 1)
    counter = counter + totalcount
    maxchildren = randomm + maxchildren
Next i
Call levels
End Sub

Sub delete()
Range("b2:ae31").ClearContents
End Sub


Sub levels()
'Find␣the␣last␣used␣row␣in␣a␣Column:␣column␣A␣in␣this␣
   example
Dim␣countt␣As␣Integer
countt␣=␣0
Dim␣var123␣As␣Integer
var123␣=␣0
Dim␣kant␣As␣Boolean
kant␣=␣False
Dim␣holdd␣As␣Integer
holdd␣=␣0
Dim␣level␣As␣Integer
level␣=␣1
Dim␣kitle␣As␣Integer
kitle␣=␣0
```

```
␣␣␣␣Dim␣LastRow␣As␣Long
␣␣␣␣With␣ActiveSheet
␣␣␣␣␣␣␣␣LastRow␣=␣.Cells(.Rows.Count,␣"B").End(xlUp).
   Row
␣␣␣␣End␣With
'Find the last used column in a Row: row 1 in this
   example
    Dim LastCol As Integer
    With ActiveSheet
        LastCol = .Cells(2, .Columns.Count).End(
           xlToLeft).Column
    End With
    For i = 1 To LastRow
    holdd = 0
        For j = 1 To LastCol
            If Cells(i + 1, j + 1) <> 0 And kant =
               False Then
            countt = countt + 1
            Else
            holdd = holdd + Cells(i + 1, j + 1)
            End If
            If j = LastRow And Cells(i + 1, j).Value <>
                0 Then
                kitle = level
            End If
            If j = LastRow And kant = False Then
            level = level + 1
            End If
'MsgBox(LastRow&"␣"&␣j)
Next␣j
If␣holdd␣<>␣0␣Then
var123␣=␣var123␣+␣1
End␣If
If␣var123␣=␣countt␣Then
kant␣=␣False
countt␣=␣0
var123␣=␣0
End␣If
If␣countt␣<>␣0␣Then
kant␣=␣True
End␣If
Next␣i
Range("AK1").Value␣=␣kitle
'Range("AL1").Value = level

End Sub
```

Chapter 7

Bibliography

[1] Barba-Gutiérrez, Y., Adenso-Díaz, B. and Gupta, S. M. (2008). *Lot sizing in reverse MRP for scheduling disassembly.* International Journal of Production Economics, Vol.111(2): 741-751.

[2] Barba-Gutiérrez, Y. and Adenso-Díaz, B. (2009). *Reverse MRP under uncertain and imprecise demand.* International Journal of Advanced Manufacturing Technology, Vol.40(3-4): 413-424.

[3] Bok, C., Nilson, J., Masui, K., Suzuki , K., Rose, C., and Lee, B. H. (1998). *An international comparison of product end-of-life scenarios and legislation for consumer electronics.* Electronics and the Environment, 1998. ISEE- 1998. Proceedings of the 1998 IEEE International Symposium 19-24.

[4] Gao, N. and Chen, W. (2008). *A genetic algorithm for disassembly scheduling with assembly product structure.* Proceedings of 2008 IEEE International Conference on Service Operations and Logistics, and Informatics, Vol.2: 2238-2243.

[5] Gokgur, B., Gokce, M.A and Ozpeynirci, S. (2015). *Large-scale disassembly operations planning with parallel resources.* International Journal of Advanced Manufacturing Technology, Vol.81(5-8): 1195-1214.

[6] Gungor, A., and Gupta, S. M. (1999). *Issues in environmentally conscious manufacturing and product recovery: A survey.* Proceedings of the 1998 IEEE International Symposium on Electronics and the Environment, Vol.36(4): 811-853.

[7] Gupta, S. M. and Taleb, K. N. (1994). *Scheduling Disassembly.* International Journal of Production Research, Vol.8: 1857-1866.

[8] Hines, W.W., Montgomery, D. C., Goldsman, D. M. and Borror, C. M. (2003). *Probability and Statistics in Engineering*, 4th edn, United States of America, John Wiley and Sons, Inc.

[9] Hyong-Bae, J., Jun-Gyu, K., Hwa-Joong, K. and Dong-Ho, L. (2006). *A Two-Stage Heuristic for Disassembly Scheduling with Capacity Constraints.* International Journal of Management Science, Vol.12(1): 715-722.

[10] Ilgin, M. A and Gupta, S. M. (2010). *Environmentally conscious manufacturing and product recovery (ECMPRO): A review of the state of the art.* Journal of Environmental Management, Vol.91(3): 563–591.

[11] Imtanavanich, P. and Gupta, S. M. (2004). *A Multi-criteria Decision Making Approach for Disassembly-to-order Systems.* Proceedings of the SPIE International Conference on Environmentally Conscious Manufacturing IV,

Vol.11(2): 147-162.

[12] Ji, X., Zhang, Z., Huang, S. and Li, L. (2016). *Capacitated disassembly scheduling with parts commonality and start-up cost and its industrial application.* International Journal of Production Research, Vol.54(4): 1225-1243.

[13] Jovane, F., Alting, L., Armillotta, A., Eversheim, W., Feldmann, K., Seliger, G. and Roth, N. (1993). *A Key Issue in Product Life Cycle: Disassembly.* CIRP Annals - Manufacturing Technology, Vol.42(2): 651-658.

[14] Kang, K-W., Doh, H-H., Park, J-H. and Lee, D-H. (2012). *Disassembly leveling and lot sizing for multiple product types: a basic model and its extension.* International Journal of Advanced Manufacturing Technology, Vol.82(9-12): 1463-1473.

[15] Kim, D-H. and Lee, D-H. (2011). *A heuristic for multi-period disassembly leveling and scheduling.* International Symposium on System Integration, 762-767.

[16] Kim, D-H., Doh, H-H. and Lee, D-H. (2016). *Multi-period disassembly leveling and lot-sizing for multiple product types with parts commonality.* Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture

[17] Kim, H-J., Lee, D-H., Xirouchakis, P. and Züst, R. (2003). *Disassembly Scheduling with Multiple Product Types.* CIRP Annals - Manufacturing Technology, Vol.52(1): 403-406.

[18] Kim, H-J., Lee, D-H. and Xirouchakis, P. (2005). *An Optimal Algorithm for Disassembly Scheduling with Assembly Product Structure*. Annual Conference on Artificial Intelligence: Advances in Artificial Intelligence, 235-248.

[19] Kim, H-J., Lee, D-H. and Xirouchakis, P. (2006a). *Two-phase heuristic for disassembly scheduling with multiple product types and parts commonality*. The Journal of the Operational Research Society, Vol.44(1): 195-212.

[20] Kim, H-J., Lee, D-H. and Xirouchakis, P. (2006b). *A Lagrangean Heuristic Algorithm for Disassembly Scheduling with Capacity Constraints*. The Journal of the Operational Research Society, Vol.57(10): 1231-1240.

[21] Kim, H-J., Lee, D.-H., Xirouchakis, P. and Kwon, O. K. (2009). *A Branch and Bound Algorithm for Disassembly Scheduling with Assembly Product Structure*. The Journal of the Operational Research Society, Vol.60(3): 419-430.

[22] Kim, H-J., Lee, D-H. and Xirouchakis, P. 2010. *Disassembly scheduling: literature review and future research directions*. International Journal of Production Research, Vol.45(18-19): 4465-4484.

[23] Kim, H-J. and Xirouchakis, P. (2010). *Capacitated disassembly scheduling with random demand*. International Journal of Production Research, Vol.48(23): 7177-7194.

[24] Kim, J-G., Jeon, H-B., Kim, H-J, Lee, D-H. and Xirouchakis, P. 2006. *Disassembly scheduling with capacity constraints: minimizing the number of prod-*

ucts disassembled. Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture, Vol.220(9): 1473-1481.

[25] Kongar, E. and Gupta S.M. (2006). *Disassembly to order system under uncertainty.* Omega, Vol.34(6): 550-561.

[26] Lambert, A. and Gupta S.M. (2002). *Demand-driven disassembly optimization for electronic products.* Journal of Electronics Manufacturing, Vol.11(2): 121-135.

[27] Langella, I. M. (2007). *Heuristics for demand-driven disassembly planning.* Computers and Operations Research, Vol.34(2): 552-577.

[28] Lee, D-H., Kang, J-G. and Xirouchakis, P. (2001). *Disassembly planning and scheduling: review and further research, Proceedings of the Institution of Mechanical Engineers.* Part B: Journal of Engineering Manufacture, Vol.215(5): 695-709.

[29] Lee, D-H., Xirouchakis, P. and Zust, R. (2002). *Disassembly scheduling with capacity constraints.* CIRP Annals-Manufacturing Technology, Vol.51(1): 387-390.

[30] Lee, D-H., Kim, H-J., Choi, G. and Xirouchakis, P.(2004). *Disassembly scheduling : integer programming models.* Proceedings of the Institution of Mechanical Engineers, Part B: Engineering Manufacture, Vol.218: 1357-1372.

[31] Lee, D-H. and Xirouchakis, P. (2004). *A two-stage heuristic for disassembly scheduling with assembly product structure.* Journal of the Operational

Research Society, Part B: Engineering Manufacture, Vol.55(3): 287-297.

[32] Lee, D-H. (2005). *Disassembly Scheduling for Products with Assembly Structure.* Management Science and Financial Engineering, Vol.11(1): 63-78.

[33] Neundorf, K-P., Lee, D-H., Dimitris, K. and Xirouchakis, P. (2001). *Disassembly Scheduling with Parts Commonality Using Petri Nets with Timestamps.* Fundamenta Informaticae, Vol.47(3-4): 295-306.

[34] Prakash, P. K. S., Ceglarek, D. and Tiwari, M. K. (2011). *Constraint-based simulated annealing (CBSA) approach to solve the disassembly scheduling problem.* International Journal of Advanced Manufacturing Technology, Vol.60(9-12): 1125-1137.

[35] Sasikumar, P. and Kannan, G. (2008). *Issues in reverse supply chains, part I: end-of-life product recovery and inventory management – an overview.* International Journal of Sustainable Engineering, Vol.1(3): 154-172.

[36] Sung, J. and Jeong, B. (2014). *A heuristic for disassembly planning in remanufacturing system.* The Scientific World Journal, Vol.2014: 1-10.

[37] Taleb, K. N., Gupta, S. M. and Brennan, L. (1997). *Disassembly of complex product structures with parts and materials commonality.* Production Planning and Control: The Management of Operations, Vol.8(3): 255-269.

[38] Taleb, K.N. and Gupta, S.N. (1997). *Disassembly of multiple product structures.* Computers and Industrial Engineering, Vol.32(4): 949-961.

[39] Tang, Y., Zhou, M., Zussman, E. and Caudill, R. (2002). *Disassembly modeling, planning, and application.* Journal of Manufacturing Systems, Vol.21(3): 200-217.

[40] Vongbunyong, S. and Chen, W. H. (2015). Disassembly Automation, ed., *General Disassembly Process.* Switzerland, 9 -25.