DESIGN OF A SECOND LIFE PRODUCT FAMILY FROM THE PERSPECTIVE OF THE REMANUFACTURING AGENT

BY

NATHAN B. SCHROEDER

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Systems and Entrepreneurial Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2011

Urbana, Illinois

Adviser:

Assistant Professor Harrison M. Kim

Abstract

This thesis presents a method of solving a newly posed Second Life Product Family Design problem. This is unique in that the architecture of the product is not specified to be identical to one of the recaptured products, rather it is determined through optimization. The problem is framed using Conjoint Analysis and the Multi Nomial Logit Model, formatted with respect to components available for inclusion in the final products and then solved using an implementation of Genetic Algorithms. The solution method is also encapsulated in a software module which can be disseminated to industrial users without a background in optimization or familiarity with Genetic Algorithms.

A case study is performed to determine the effectiveness of the proposed solution method, and analyze the influences different market conditions and component similarities can have on the optimal design. It is concluded that the proposed method converges to an optimal Second Life Product Family Design.

Acknowledgments

I would like to acknowledge Prof. Harrison Kim, as well as Minjung Kwak, Shen Lu, and Conrad Tucker, without who's support and advice this work would not exist.

To Andrea, who makes life excellent.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Due to the ever-expanding focus on the human interaction with the environment, extra attention is being paid to what happens to products and materials after their typical consumer lifetime has expired. There are clear environmental and economic gains to be captured by recycling and reusing raw materials [11]. Aluminum, for example, can be recycled and reprocessed for 12 to 20 times less energy than mining bauxite and smelting raw aluminum [17]. These economic and environmental returns have long been known with regards to raw materials, and now companies are applying the same mentality and methodology to the recycling and remanufacture of consumer goods.

Remanufacturing involves the reuse of not only the raw materials, but also the value-added aspects that make the product valuable to the consumer. For example, a computer that is returned to a remanufacturing center might have it's motherboard taken out and used in the production of a refurbished product(given that the motherboard is functioning properly, of course). This process allows the capture of additional value that the recycling of materials alone would miss.

Today there exist companies who do just that; collect products at the end of their useful life, disassemble them into their individual components, and reassemble the working components to produce a product with value for sale to the refurbished market. One such example is Recellular. They are able to capture approximately 75,000 used cellular phones weekly, diverting them from the waste stream and utilizing the remaining value to produce a profit for the company [19].

The typical mode of business for remanufacturing operations such as Recellular is to disassemble the collected product, analyze and test the components for functionality, and recombine working components to make refurbished products that exactly resemble the returned product. By producing a product identical to the uptake product, the remanufacturer ensures that there is demand in the marketplace for the output product. This process is illustrated for the computer remanufacturing case in Figure 1.1.

In this work, we propose a method to optimally design a Second Life Product Family. This would allow the remanufacturer to forgo merely recreating the design of the recaptured product, and instead obtain the maximum amount of profit from their given recaptured inventory. The process is shown in Figure 1.2. There

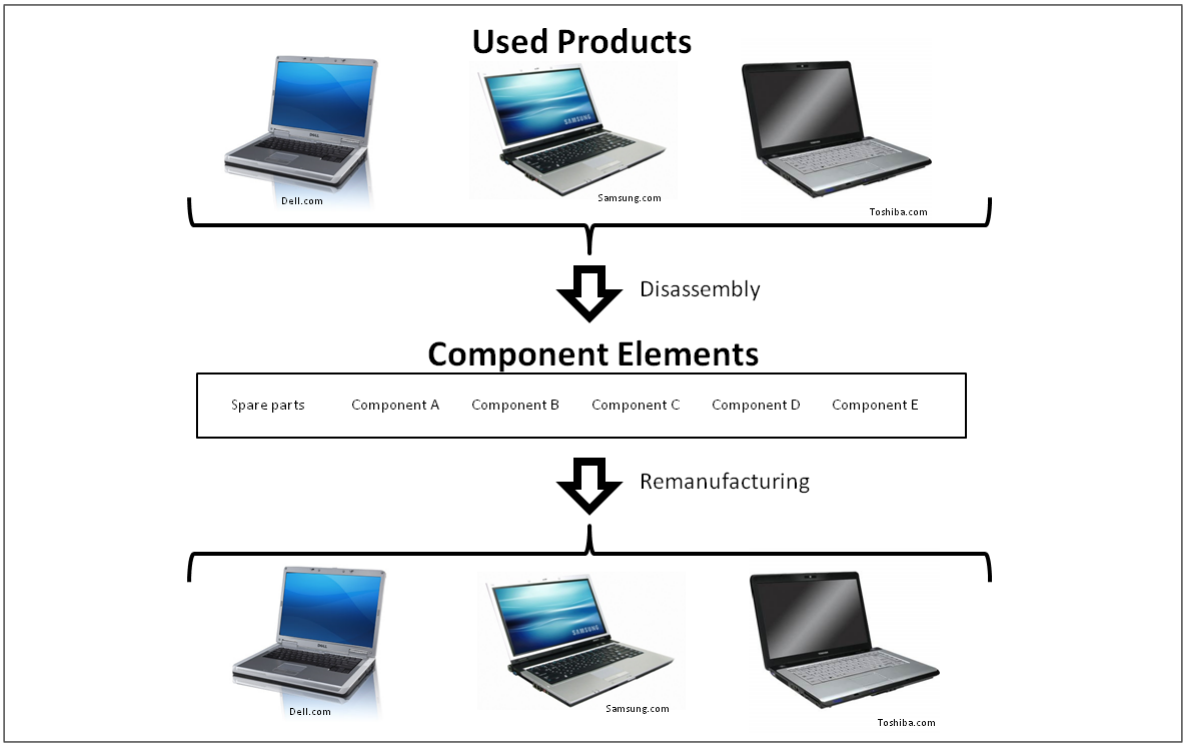

Figure 1.1: Overview of Current Remanufacturing Process

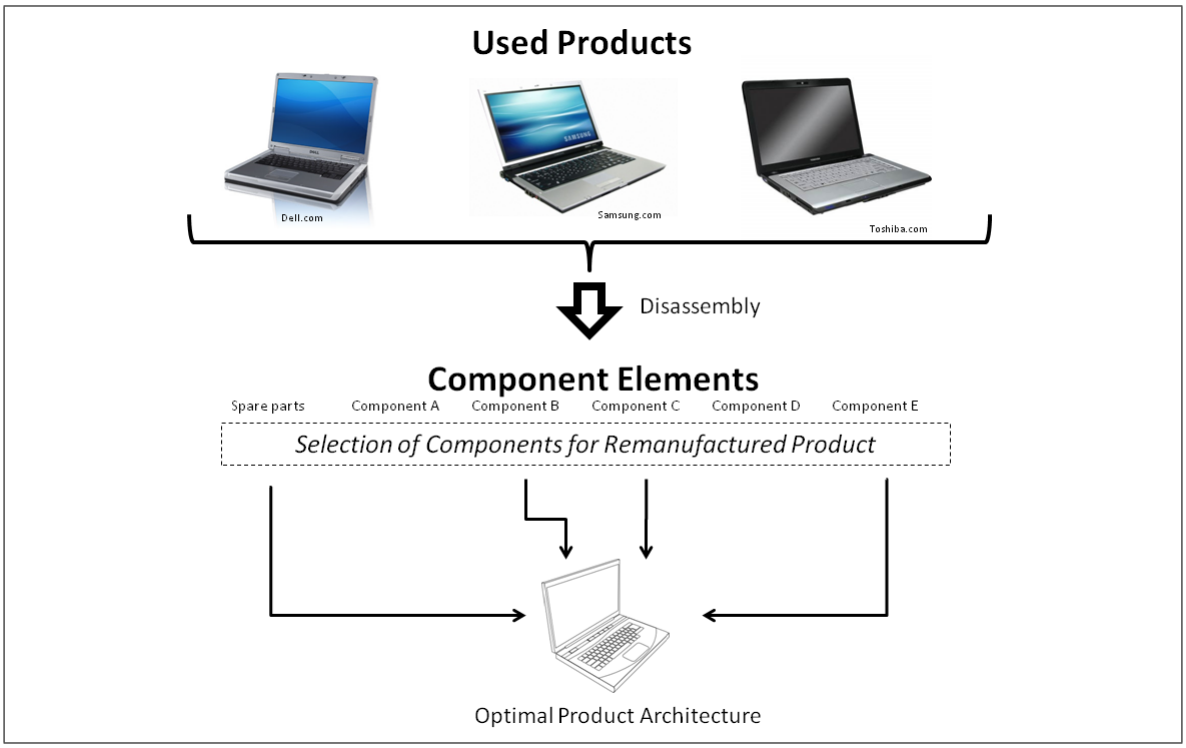

Figure 1.2: Overview of Proposed Remanufacturing Process

has been little work in the field of Second Life Product Family Design. In pursuing a novel optimal design for the second life product, the need arises for additional considerations about component compatibility. The remanufacturer must ensure that the components used in their product will function properly with one another, as the components could have originated in different host products. Component compatibility with regards to Second Life Product Design is also addressed in this work. In addition to the compatibility consideration, the proposed problem has no gradient information available for the solver. This leads to the implementation of Genetic Algorithm as the necessary solver for the problem.

This thesis will be organized as follows: First we will review the state of the art in Remanufacturing Design, and where research is concentrated in the remanufacturing process. We will focus on ways utility is measured with regard to products and components, and show that the Multi-Nomial Logit Model is ideal for Second Life Product Design. Next, the Second Life Product Family Design problem will be formulated, addressing the multiple product case and its affect on problem structure. After formulation and explanation of the problem, we will present an architecture of the problem whereby Genetic Algorithms can be utilized to streamline the problem implementation and sidestep issues regarding the Mixed Integer Non-Linear Programming (MINLP) implementation of the problem.

1.1 Contributions

The contributions of this work are twofold. First, the Second Life Product Design problem is described and is shown to have a representation whereby component utility can be considered independent of final product design, allowing for a description of product utility dependent on a linear combination of component utilities. This allows for the Second Life Product Design problem to be addressed. Prior to this work, the primary focus of decision making involved selection among the set of recaptured products. This inherently limits the design space to very few of the possibilities. This work expands the space to include all combinations of components, subject to compatibility restrictions.

A case study is performed, and the results are analyzed and shown to provide insight into the selection of included components and the relationship between selected components and the market preferences. The second deliverable is a solution platform, which is published to allow industry users a convenient interface to solve real-life Second Life Product Family Design problems, optimizing their company's profit.

Now a synopsis of the current research focuses in Remanufacturing Optimization will be described, and this problem will be found to fit in a niche that has not yet been investigated. The methods of data gathering and expression will also be discussed to provide a sound base on which to build the optimization procedure.

Chapter 2

Literature Review

In this section we will address the current state of art with regards to optimization in the remanufacturing context. We will discuss methods of product and component evaluation, such as Conjoint analysis and Discrete Choice Models, and will look at various methods in use to determine market share. We will then address methods for compatibility capture.

2.1 Optimization in Remanufacturing

Much work has been done in addressing the process requirements of the remanufacturing system. The production systems have been analyzed and designed in the context of remanufactured products [22, 23]. An essential component of remanufacturing is the collection of used parts, so recapture methodologies and models have been developed with regard to the reverse supply chain[9][20]. Optimal system design has also been performed from many perspectives to construct an optimal recovery network[25][10]. In addition to recovery network design, the inventory system impact has been a large focus in remanufacturing systems, as the inflow of used products is not under the direct control of the remanufacturer. This novelty is a problem not faced by typical manufacturing organizations, and thus robust inventory control systems are needed to balance the purchase of spare products with the recapture of used parts[12].

In addition to the remanufacturing supply chain and process analysis, there has been work done in the area of new product design for remanufacturability[26][5]. In this area, ease of disassembly and associated influences on the recaptured value are included in the initial product design in order to make the entire life-cycle more profitable or reduce environmental impact. However, the design process of the Second Life Product is distinct from the work that has been done, and has not been considered as an independent problem given a fixed set of recovered components. The description and solution of such a problem is the focus of this work.

2.2 Component Utility Calculation and Demand Expression

One of the contributions of this work is to develop a problem formulation which will search the entire space of possible product architectures to design the Second Life Product Family. In order to search this whole space, an expression of demand had to be used which was dependent on the component-level contributions. This is necessary because we are no longer only considering the performance of products already released to the market, where actual data can be recorded to describe demand of the product whole.

Typical methods of describing market share which were considered are the Discrete Choice Model and Conjoint Analysis[8]. Though some work has been done with the Discrete Choice Model in the remanufacturing space [24], Conjoint Analysis is found to be most suited to design problems, due to the ability to analyze the utility from the attribute level without extensive data collection[3]. Also advantageous is the ability to model information regarding product configurations not currently in the marketplace[16].

In addition to the measurement of utility, this problem involves the utility's effect on market size. The Multi Nomial Logit Model is a common framework from which market share can be computed based on the utilities of products available in the market[24]. This MNL model requires an assumption of Independence of Irrelevant Alternatives(IIA). IIA implies that, given the removal of one option from the market, all remaining choices would be impacted (ie no two choices are equal, commonly expressed by the Blue Bus vs. Red Bus example[15]). IIA will be assumed to hold, though in actual remanufacturing examples, this would need to be determined on a case by case basis.

2.3 Compatibility in Second Life Products

Because the Second Life Product Family Design problem has not been addressed as a component-based design previously, there is no work entailing compatibility in this specific area. However, the concept of compatibility is translatable from the new product design problem. Some areas where this has been considered are in optimal component sharing and security design for new products [1][2]. In the component sharing area, compatibility was analyzed to determine if modules could be formed to facilitate cross-product utilization, and thus savings to the company.

Now that the current research landscape has been defined and the tools necessary for the data analysis have been presented, the form of the Second Life Product Family design problem will be discussed, followed by a case study implementing the new framework.

Chapter 3

Optimal Design of a Second Life Product Family

Now that the basis for the Logit Models, Compatibility, and Compatibility is understood, we attempt to combine these three tools to accurately and completely design an optimal family of products in the context of a remanufacturing company. To do so, it must be determined which components to include in the given products in order to maximize company profit, as well as determine the quantity of each to produce and what price to sell them at in the marketplace. A complete description of the problem with constraints and assumptions follows.

3.1 Problem Framework

The Second Life Product Family Design Problem is intended to set out an optimal product family design based on 1) the remanufacturing company's current inventory of components and 2) the current marketplace. The inventory of the company is easily determined and is assumed to be given, but the dynamics of the marketplace into which the remanufactured product will be sold presents a challenge. This formulation of this problem was first proposed by Minjung Kwak in [14], but was never worked to a solution. We will now present the problem, and discuss how it captures the market dynamics as well as the impact a given component has on the final product offering. The parameters and variables needed are as follows:

- X_i : Quantity of Product i to Produce
- P_i : Price at which Product i is sold
- y_{ijk} : Indicator to include Part j, Variant k in Product i
- Q_{jk} : Quantity of Part j, Variant k collected by remanufacturer
- c^s_{jk} : Unit Cost of Spare Part j, Variant k
- r_{jk} : Unit Revenue from Selling Part j, Variant k
- c^p : Cost of Reprocessing/Cleaning/Assembling one Unit of Product

- W_{jk}^{c} : Utility of Part j, Variant k
- M : Market Size in Units
- U_l : Utility of Competitor Product l
- $\mathbf{T}$: Binary Compatibility Matrix
- β : Scaling Parameter of Conditional Multi Nomial Logit Choice Rule
- ρ : Discount Factor for Used Product
- W^p : Weight of Price in Utility Function

3.2 Mathematical Formulation

The mathematical expression of the Second Life Product Family Design problem[14] is as follows:

$$Max\ Profit = \sum_i (X_i * P_i) + \left[Q_{jk} - \sum_i (X_i * y_{ijk}) + Z_{jk} \right] * r_{jk} - Z_{jk} * C^s_{jk} - C^p * \sum_i (X_i) \tag{3.1}$$

Subject to:

$$X_i \leq D_i \quad \forall i \tag{3.2}$$

$$\textstyle\sum_k y_{ijk} = 1 \forall i, \quad \forall j \tag{3.3}$$

$$(y_{ijk})' * T * y_{ijk} = 0 \quad \forall i \tag{3.4}$$

$$y_{ijk}\ is\ binary \tag{3.5}$$

$$X_i \geq 0 \quad \forall i \tag{3.6}$$

$$P_i \geq 0 \quad \forall i \tag{3.7}$$

$$P_i \leq P^{max} \tag{3.8}$$

Where:

$$D_i = M * \left[\frac{e^{U_i}}{\sum_i e^{U_i} + \sum_l e^{U_l}} \right] \quad \forall i \tag{3.9}$$

$$Z_{jk} = max\left(\left[\textstyle\sum_i (X_i * y_{ijk}) - Q_{ij} \right], 0 \right) \quad \forall j, \ \forall k \tag{3.10}$$

$$U_i = \left[\textstyle\sum_j \left(\sum_k (W^c_{jk} * y_{ijk}) \right) + W^P * \left(\frac{P^{max} - P_i}{P^{max}} \right) \right] * \beta * \rho \quad \forall i \tag{3.11}$$

$$i = Product\ Index \qquad j = Component\ Type\ Index \qquad k = Component\ Variant\ Index$$

Under Assumptions:

$$c^s_{jk} > r_{jk} \quad \forall j, \ \forall k \tag{3.12}$$

$$\textstyle\sum_k y_{ijk} \leq 1 \quad \forall i, \ \forall j \tag{3.13}$$

3.2.1 Objective

As is shown in the framework in Equation 3.1, the objective of the problem is to maximize the profit of a remanufacturing company. This objective function consists of four main elements. These are:

1. The sale of finished products
2. The sale of excess components to the marketplace
3. The cost of purchasing the necessary spare components
4. The cost of testing/cleaning/assemblying the finished products

The calculation of the revenue from the finished products is straightforward, simply using $\sum_i(X_i * P_i)$, we are able to multiply the number of products sold (X_i) by the Price (P_i), and sum over all product offerings i. The computation of the sale of excess components is somewhat more complex. Here we use $[Q_{jk} - \sum_i(X_i * y_{ijk}) + Z_{jk}] * r_{jk}$, where Q_{jk} is the number of recaptured components of each Type and Variant, $\sum_i(X_i * y_{ijk})$ removes the components used in the production of the Second Life Products, and Z_{jk} is a vector defined as the Spare Parts Needed, as seen in Equation 3.10. By subtracting the components used while adding the Spare Parts Needed, we avoid having a negative quantity, which makes sense in that we can only sell as many parts as we collect less the parts we use, and no more. However, it is possible to use more parts than are collected by purchasing spares, which is captured by Z_{jk}. Once we have determined the total quantity of extra components ($Q_{jk} - \sum_i(X_i * y_{ijk}) + Z_{jk}$), we then multiply by r_{jk}, the price they can fetch in the marketplace, to yield the total revenue from sale of excess components.

The remainder of the objective function are the costs, namely the cost to purchase spare components and the unit cost of testing, cleaning, and assembly. The spare cost is defined by $Z_{jk} * C^s_{jk}$, with Z_{jk} denoting the quantity of Part j Variant k needed to complete the specified quantity of products, while C^s_{jk} is the price at which the components can be purchased from the marketplace. The form of Z_{jk} will be described in Section 3.2.3. The unit production cost is captured in $C^p * \sum_i(X_i)$, with C^p being the unit cost scalar.

3.2.2 Constraints

Now that the objective is defined, we will examine the constraints. The first constraint (Equation 3.2) restricts the Quantity of remanufactured product to be less than the Demand for that product. Demand is defined in Equation 3.9, and will be discussed further shortly. The second constraint (Equation 3.3) restricts the inclusion of one and only one of each Type j of component. This is done by allowing y_{ijk} to take values of 1 or 0, with 1 denoting inclusion of component Type j, Variant k in Product i. This is essential to ensure

that each product designed includes all necessary components, and also does not have duplicity of any given component. This constraint is important to note and will present other issues related to solving the problem later.

The constraint in Equation 3.4 restricts the product to using only components compatible with each other. T is defined as a binary matrix, with both the rows and column indices corresponding to individual components. Elements in T which correspond to compatible components are given a value of 0, while elements corresponding to an intersection of incompatible components are assigned a value of 1. In both T and y_{ijk}, the elements must be organized by Type (j) and Variant (k) in the same order. In this way, the matrix T can be pre-multiplied by vector y'_{ijk} and post-multiplied by y_{ijk}, and the result describes the number of incompatibilities in the product. Clearly, the number of incompatibilities must be restricted to be 0 in order to have a functional product.

The remainder of the constraints simply limit the range of the variables. Equation 3.5 restricts y_{ijk} to be binary. Equation 3.6 ensures that a non-negative number of products are produced, and Equations 3.7 and 3.8 ensure that the selling price of the Second Life products is non-negative and also below the critical price(P^{max}).

3.2.3 Definitions

Now to tackle the definitions of D_i, Z_{jk}, and U_i. Equation 3.9 shows the expression of demand for Product i (D_i). The demand is found by multiplying the total market size in units, represented by M, by the ratio of the exponential of utility of Product i relative to the total utility available in the marketplace, again in exponential form, described by $\sum_i e^{U_i} + \sum_l e^{U_l}$. This is the implementation of the Multi Nomial Logit model. In this expression, U_i represents the utility of product i offered by the remanufacturer and U_l represents the utility offered by competitor product l, where all competitor products are included in the set of l.

Z_{jk} is the description of the quantity of spare components needed to produce all i finished Products. Shown in Equation 3.10, the spares needed are determined by multiplying the quantity of Product i remanufactured(Q_i) by the inclusion parameter(y_{ijk}). The quantity available(Q_{jk}) is then subtracted from this value to compute a vector which holds the difference between quantity on hand and quantity needed. However, elements can be negative if the quantity on hand is larger than the quantity needed. Clearly, to use the spare vector, these values must be represented by zeros. Hence, the maximum between the aforementioned difference and 0 is taken to be the definition of Z_{jk}.

The final definition needed is that of U_i, which represents the utility of Product i. The calculation of the product utility is dependent on the components included and their respective contributions to customer

Table 3.1: Component Attribute Example

Attribute Level	Utility from Speed	Utility from Range
1	0.0	0.1
2	0.25	0.2
3	0.35	0.3
4	0.4	0.4

utility, as well as the utility available to the customer from having the opportunity to purchase the product below their critical price. The utility garnered from the components included in the Second Life product is characterized by $\sum_k (W^c_{jk} * y_{ijk})$, with W^c_{jk} representing the contribution of the component of Type j, Variant k. This form comes directly from Conjoint Analysis[8]. As in the calculation of the spare components, y_{ijk} is used as a multiplier to remove the effect of components not included in the product. The utility contribution of price level is then added to the components' influence through the term $W^P * \left(\frac{P^{max} - P_i}{P^{max}}\right)$. Here, the W^P coefficient is a weight parameter to balance the contributions of components and price, while P_i is the price at which the Product i will be sold and P^{max} is the critical price for the product.

To illustrate the calculation of product utility using the Conjoint Analysis framework, a brief example will be given. If a given electronics component is described using the range and speed, as in a wireless communications card, the Conjoint Analysis would be performed with regard to both range and speed as independent attributes. The attributes are assumed to not be correlated. Table 3.1 gives a sample table with utility values assigned to various attribute levels. If Component A has a Speed of rank two and a Range of rank three, its utility contribution would be $0.25 + 0.3 = 0.55$.

Thus we see how the results of Conjoint Analysis, which defines the attribute utilities, can be used to determine component utilities. Using this formulation with the Multi-Nomial Logit model, the share of the total market can then be determined, and thus the upper limit on the number of units to be sold. Again, if further clarification on the use of Conjoint Analysis is necessary, please refer to [8].

3.2.4 Assumptions

In order to properly frame the problem, some assumptions are necessary. These are listed at the bottom of the Mathematical Formulation, and will be discussed here. The first necessary assumption is Equation 3.12, which states that the cost of purchasing a spare part (c^s_{jk}) must be greater than the price at which that part can be sold to the market (r_{jk}). This assumption follows logically from market dynamics, and if it were not to hold, the maximal profit model would include purchasing parts at a given cost and selling those parts at a higher cost. Though a valid business model, this is not the purpose of the remanufacturer. Thus we restrict

the price and cost of components as shown in Equation 3.12.

Another necessary assumption is that found in Equation 3.13. This constraint maintains that there is not more than one component of a given Type j included in Product i. It is evident that this assumption is also in play in Equation 3.3, where a Product is restricted to have one and only one of a given component Type. This assumption relates to the architecture of the final product, and is made for simplicity's sake. Though it is possible that the inclusion of more than one of a given component type can be advantageous (as in the case of computer hard drives), this case will be left for future work. This assumption will become extremely significant once the problem is reworked for implementation by Genetic Algorithm in Section 3.4.

3.3 Difficulties with Traditional Solvers

In this section we briefly describe the attributes of the Second Life Product Family Design Problem, and discuss obstacles to methods of obtaining a solution. First, it is important to point out the characteristics of the Second Life Product Family Design problem. It is apparent from the Section 3.2 that we have three categories of variables. The first is the quantity produced of a given product, represented by Q_i. The second is the price at which a product is sold, given by P_i. The third is the inclusion vector, y_{ijk}, which is a binary set of variables with the same number of elements as distinct components that exist in the inventory. All three of these variables scale with the number of products offered. Though the first two scale at a one to one rate with respect to i, the third scales as the product of i and the number of possible components. Though not exponential, this high degree of linear scaling poses a problem for solvers, as combinatorial solvers must then deal with exponentially increasing solution spaces.

It is also difficult to implement a relaxation technique that will work for these inclusion-determining variables. This is because the variables are not ordinal, but rather reference utility, purchase price, and selling price values that are component dependent, and , aside from the utility values, have no discernable relation to other components. Due to this fact, a non-heuristic solution algorithm would have no gradient related information to guide the search process, thus requiring some sort of combinatorial search method. With variables of degree two and upwards of 300 variables possible in a moderately sized problem (three products with 100 possible components) yielding on the order of 10^{90} combinations, the combinatorial method quickly gets out of hand. Even assuming that Equation 3.3, the constraint requiring one and only one of each component type to be included, is met a priori, given ten component types with ten variants each, the search space still contains 10 billion possible combinations.

This is a case where heuristic algorithms can be put to good use. Genetic Algorithms seem particularly

suited for this problem, as the structure required for GA implementation will capture some of the constraints, and allow fewer calculations to be performed by the solver. This synergy will be further explained in Section 3.4.

In addition to the difficulty to traditional solvers arising from the variables and the inability to relax them, there is also a combinatorial-based constraint, namely the compatibility constraint. Again, relaxation of the y_{ijk} variables will not be helpful in determining an appropriate search direction with regards to the constraint satisfaction.

3.4 Formulation for Genetic Algorithms

As was illustrated in the previous section, this problem is inherently difficult for traditional solvers to handle. As such, we formulate it for use with a Genetic Algorithm, and thus avoid searching the entire combinatorial space and also capture some synergy between the GA format and the problem formulation itself.

Genetic Algorithms are based on implementation of a chromosome to represent variables in the problem[7]. It is this representation of the problem variables that we are able to exploit. Through definition of the chromosome, and specifying that each component type is represented by a single integer variable, we are able to avoid computing the constraint in Equation 3.3. This is because every integer value necessarily corresponds to one and only one Variant of a given Type. Through this careful implementation of the chromosome, we are able to avoid the y_{ijk} variables all together, and thus shrink the size of the problem significantly. Instead of the problem including all $\sum_j(max(k))$ binary indicator variables and j additional constraints, we capture the same information and restrictions by the use of j integer variables. The format of the Second Life Product Design chromosome is shown in Figure 3.1. The values in positions one and two are Quantity produced and Price, respectively. These are continuous variables with the upper limits specified by Market Size (M) and Critical Price (P^{max}). The remainder of the chromosome is composed of integer variables corresponding to a specific component. The format in Figure 3.1 shows the implementation of C_k^j, where the third term in the chromosome is the integer value representing the variant included of the first component type. The fourth term is an integer representing the second Type of component, and so on through all j Types. Through use of this format, the j index is stored in the location of C_k^j, and the value at that location is left to represent the index k. Again, it should be noted that because the location stores the value of j, there can only be a single component variant chosen for a given j, thus capturing the constraint expressed in Equation 3.3.

Of course, the same implementation could be done without the use of GA, by representing the component

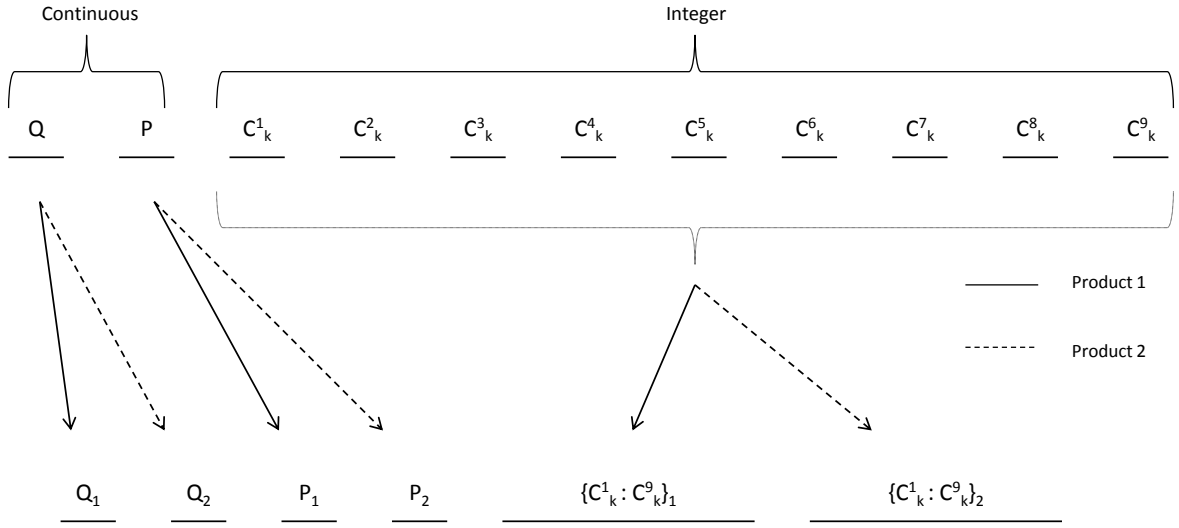

Figure 3.1: Chromosome for Single Product GA Implementation and Conversion to Multi-Product

indices using integer variables instead of sets of binary inclusion variables, but traditional solvers are still left (in our 10 Types with 10 Variants example) with 10 billion combinations of components. Relaxation of these variables still poses the same challenge as that of y_{ijk}, because the integer value has no bearing on the sell price, quantity captured, or purchase price associated with the component. Thus no gradient information, even through use of a relaxation scheme, is available to the solver.

Figure 3.1 also shows the expansion of the problem to include cases where multiple product architectures are produced, and so $i > 1$. The simplest case, $i = 2$, is shown, but the same principals apply to the $i = n$ product case. First the Q_i are listed in order of i, followed by P_i, again in order of i. After all quantities and prices are listed, the component indices are listed in the same fashion as the single product case, but grouped by product index i, again in ascending order of i. Other orderings of the chromosome could also be chosen without negatively impacting the outcome of the GA, but this grouping was chosen to facilitate the lookup code that must be used in the calculation of the objective and constraints in the produced software, which is described in the following section.

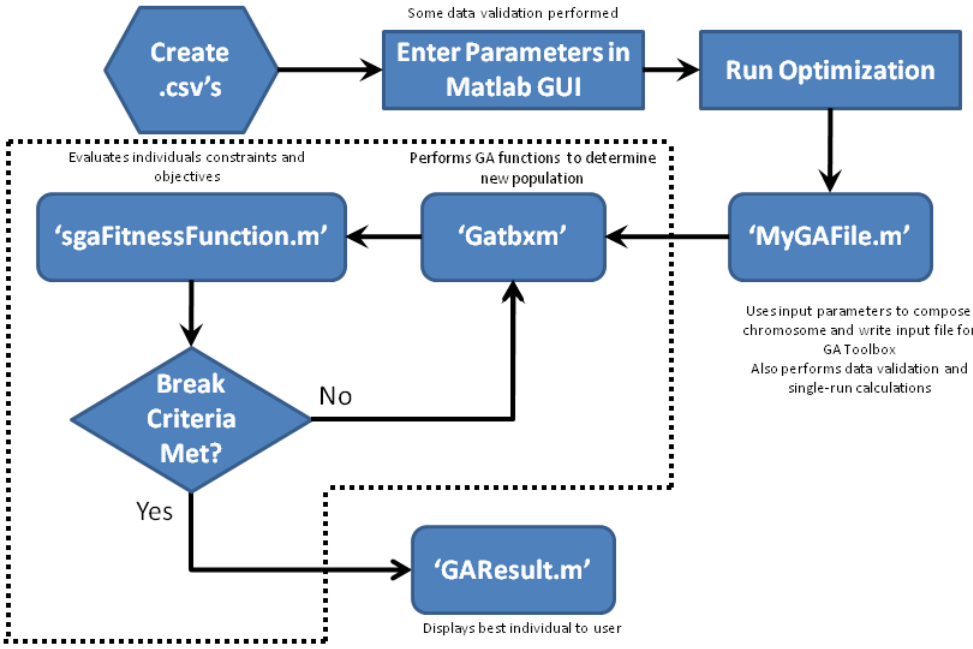

Figure 3.2: Software Architecture for Remanufacturing Second Life Product Family Design

3.5 Software Implementation

The second deliverable of this thesis, aside from the methodology to solve the Second Life Product Family Design problem, is the creation of a software program to make the developments in methodology accessible to industry. This involves creating a program to encapsulate the Genetic Algorithm; allowing us to automate the formulation of the chromosome, the objective function, and the constraint expressions. This program requires no working knowledge of GA to operate, with user input restricted to utility calculation parameters and two input matrices. The user is then able to run the optimization, and the output prices, quantities, and components are given in vector form. This will allow dissemination of this work to actual remanufacturers, enabling them to more efficiently produce value from their inputs, and hopefully increase the profitability (and thus induce growth) of the remanufacturing sector.

```
#GA type
SGA

# Number of Decision Variables
11

# Decision Variable Type
int 0 10000
double 0 500
int 1 8
int 1 3
int 1 6
int 1 6
int 1 6
int 1 3
int 1 3
int 1 3
int 1 3

# Objectives
1
Max

# Constraints
2
1
1

# General Parameters
100
30
0.9

# Niching
NoNiching

# Selection
TournamentWOR 2

# Crossover
.7
SBX 0.5 10

# Mutation
0.1
Polynomial 20

# Scaling Method
NoScaling

# Constraint Handling Method
Tournament

# Local Search Method
NoLocalSearch

# Stopping Criteria
0

# Load Population
0

# Save to File
1 EvaluatedGASolutions.txt
```

Figure 3.3: Control File for Input to GA Toolbox in MATLAB

3.5.1 Input Handling

The architecture of the provided software is shown in Figure 3.2. In the Figure, there are two types of input the user must provide, namely the .csv inputs required in the first block and the parameter inputs required in the MATLAB GUI. Once these inputs are made and the program is instructed to run, it calls a file named 'MyGAFile', which takes the user inputs, properly defines the chromosome, and then loads and validates the user .csv files. Once these steps are performed, a control file is written to feed to the GA toolbox. This toolbox, labeled 'GAtbxm', was produced in the IllIGAL lab at the University of Illinois[21]. The control file is what passes information to the GA toolbox, such as the size of the chromosome, the type and limits on the elements of the chromosome, the operators used during execution of the Genetic Algorithm, the direction of the objective, as well as where to find the evaluation of the objective and constraints. A sample control file is shown in Figure 3.3. The GA toolbox is where the optimization itself is performed, with the typical Genetic Algorithm operators of crossover and mutation controlling the evolution of the population. In this software, a penalty structure is used to compute constraint violation, where all constraints are equally weighted.

Table 3.2: Fixed and Adjustable Elements

Fixed Elements	Elements Determined by User Input
Number of Objectives	Number of Decision Variables
Objective Type	Type of Decision Variables
90% Replacement Rate	Upper Limits on Decision Variables
Weight of Constraints	Number of Constraints
No Niching	Population Size
Tournament Selection (WOR)	Number of Generations
Crossover Rate	
Polynomial Mutation Rate	
No Scaling	
Tournament Constraint Handling	
No Local Search	
Stop after Maximum Number of Generations	
Start with a Random Population	

As is evidenced in Figure 3.3, there are many specifications needed to use the GA Toolbox. Some of these are fixed for all user inputs, and some are dependent on the user specified parameters. The breakdown between elements fixed for any user input and those dependent on problem size and other inputs is shown in Table 3.2. It should be noted that the general form and function of the GA solver remains the same for any problem size input by the user. However, the definition of the chromosome is a function of the number of component Types and Variants available as well as the number of products, and the number of constraints needed in the problem is determined by the number of Products the user would like to have in the product family.

Population Size and Number of Generations are the only parameters relating to the GA solver that the user has direct control over, and this is simply to facilitate a change from initial testing of the problem, in which fewer calculations are necessary, to computation of a solution, where convergence of the solution would be required. To gain a deeper understanding of what the various settings mean (Tournament Selection without replacement 2, for example) and the complete list of available options for use in the GA Toolbox, please refer to [21].

After the user inputs have been uploaded and the control file written, it is then passed to the GA Toolbox, which generates the population of individuals and controls the evolution of the population according to the rules selected in the control file. Each individual is assigned an associated objective function value and constraint violations, which impact the probability that it will be able to pass it's information on to the next generation. This computation of Objective and Constraints is left to an additional file, 'sgaFitnessFunction'. Once the program has met the termination criteria, which in this case is a specified number of generations, it is terminated and the best individual is returned to the user via the 'GAresult' file.

3.5.2 Objective and Constraint Calculation

Let's take a closer look at the calculation of the Objective and Constraints and see how this is performed. 'sgaFitnessFunction' takes as its input the individual's chromosome provided by the GA Toolbox, and uses the Quantity, Price, and Component information to determine the associated Profit, as well as any incompatibilities that exist in the product. This is done by using the component indexes given in the chromosome to refer to the input matrix given by the user. The associated utility, sale price, and purchase price can then be used in calculation of the profit, as explained in Section 3.4.

After Objective calculation is complete, the same component indices are used to reference another matrix, the Compatibility matrix provided by the user. The intersection of all the components used in the product are checked to ensure values of 1 in the Compatibility matrix, meaning that the components will function with each other. If a 0 is found in the intersection of two components used in the same product, a compatibility counter is incremented. There is no numerical justification for the choice of value to assign to 'compatible' or 'incompatible', as the code simply searches for a match to a specific value in the Compatibility Matrix. After all component combinations that exist in the given product are checked, the value of the compatibility counter is equal to the number of incompatibilities in the product. This value is then multiplied by a weight factor to increase the impact of compatibility to the same magnitude as the demand constraint.

It is clear from Figure 3.4 that the Compatibility matrix will be symmetric about the diagonal. This is analogous to the symmetric property of equality, in that if Component A is compatible with Component B, then Component B must necessarily be compatible with Component A. It is also evident that a component does not have to be checked for compatibility against itself, which is represented by the shaded area in the Figure. These conditions lead to the conclusion that it is only necessary to check the upper triangular portion of the Compatibility matrix, less the block diagonal(as the components here are all of the same Type). After the Objective and Constraint values have been determined, the results are returned to the GA Toolbox, where they influence the next generation of individuals.

The distinct advantage of this software implementation of the problem is that it automatically constructs the necessary GA definitions based on user inputs, such as the structure of the chromosome and evaluation of the objective function and constraints. This allows industry users unfamiliar with GA to reap the benefits of optimal design in the Second Life Product Family Design space.

In the following chapter this software will be used to determine an optimal product architecture for a remanufacturing example. The problem formulation that was described previously will be used, and conclusions drawn to demonstrate the effectiveness of the software and the reliability of the solutions found.

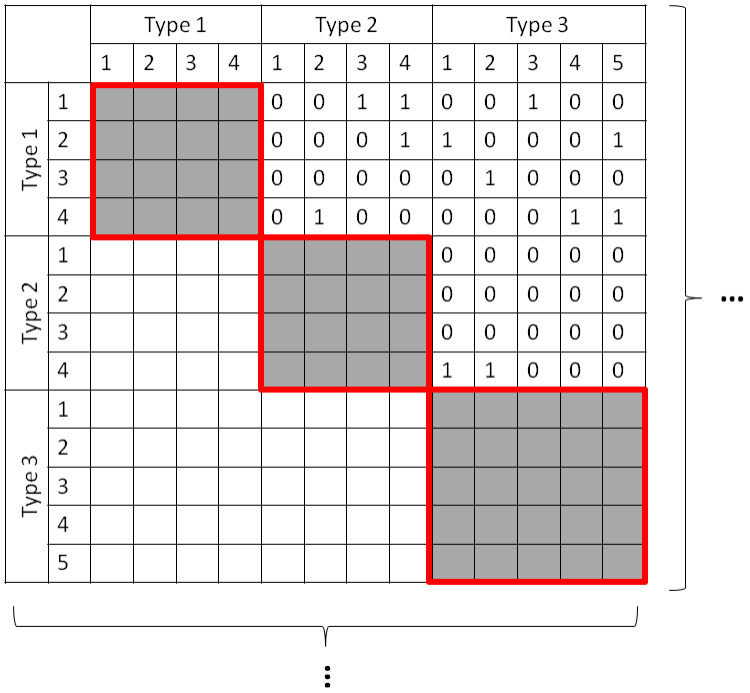

Figure 3.4: Sample Compatibility Matrix

Chapter 4

Second Life Produce Family Design Example - Computer Remanufacture

Now that the formulation of the problem and the adaptation to the Genetic Algorithm solution method have been proposed, we will use a case study to demonstrate the effectiveness of the solution method, as well as the software program. This case study will address a typical remanufacturing example, that of a computer remanufacturer. The remanufacturing company will be assumed to collect products composed of 41 distinct components of 9 differing Types. The case study will begin with the determination of utility values via conjoint analysis for all component Types and Variants. These values will then be combined with the Selling and Purchase prices for all components, as well as the Quantities obtained, and designated the Input Table. Compatibility must also be captured, as explained in Section 2.3, in what will be termed the Compatibility Matrix.

The objective of the case study will be to determine the optimal combination of components to include in a final product, as well as the price at which to sell the product and the quantity of products to produce. Variation in the solution values will be examined, and conclusions drawn relating the importance of component utility to the optimal product architecture. The multiple product case will also be considered, and it's affects on the resulting profit analyzed.

4.1 Necessary Assumptions

In this case study, we begin with an assumption of the number of components recovered from the marketplace. This is done by assuming a given number of returned computers of each generation, then describing each generation with both Budget and High value products. This produces eight distinct products that have been recovered. Each of these eight products is then described in terms of included components. See Table A.1 in the Appendix for further information regarding product architecture. We will assume 500 units collected of the oldest and newest generations, and a higher quantity of the two middle generations of returned product, with 750 of each being recaptured. This is logical, as there are likely fewer old units in circulation due to previous failure or disposal, and there are likely less newer units being returned, because these still provide

Table 4.1: Attribute Definitions and Levels

Component Type	Attribute	Attribute Level
Processor	1) Speed	1) 1GHz(1.0), 2GHz(1.1), 3GHz(1.2)
	2) Brand	2) Celeron(2.0), Pentium(2.1), Core2(2.2), Corei(2.3)
	3) Socket type	3) Socket I(3.0), Socket II(3.1)
	4) Power Consumption	4) 80-140W
Motherboard	1) Socket type for Processor	1) Socket I, Socket II
	2) Memory type	2) DDR, DDR2
	3) Graphic card type	3) Type I, Type II
	4) Hard drive connection type	4) Parallel ATA, Serial ATA
	5) Power Consumption	5) 50-150W
Hard Drive	1) Size	1) 80GB(1.0), 160GB(1.1), 250GB(1.2), 320GB(1.3), 500GB(1.4)
	2) Connection type	2) Parallel ATA, Serial ATA
	3) Power consumption	3) 15-30W
Memory	1) Memory size	1) 256MB(1.0), 512MB(1.1), 1GB(1.2), 2GB(1.3), 3GB(1.4), 4GB(1.5)
	2) Technology type	2) DDR, DDR2
	3) Power consumption	3) 15W/GB
Video Card	1) Memory size	1) 64MB(1.0), 128MB(1.1), 256MB(1.2), 512MB(1.3), 1GB(1.4)
	2) Card type	2) Type I, Type II
	3) Power consumption	3) 30-50W
Optical Drive	1) Technology type	1) CR-ROM, DVD-ROM, DVD-burner
	2) Power consumption	2) 20-30W
Operating System	1) Version	1) WindowsXP(1.0), WindowsVista(1.1), Windows7(1.2)
Case	1) Power capacity	1) 300W, 400W, 500W
Warranty	1) Warranty period	1) None(1.0), 1yr(1.1), 3yr(1.2)

a high degree of value to the customer.

Once the products are specified, we determine which components are present in each product, and also determine the probability that the component will be reusable(i.e. not damaged). The result from this first analysis, which is shown in Table A.1, is then used to provide the End-of-Life Part Assumptions, found in Table 4.1. In this representation, the identical components among the returned products have been combined to give the Quantity collected in terms of each specific component. For example, though multiple returned products use Motherboard 2, all of these motherboards are collected in the same bin, regardless of which parent it was collected from. The Price of each component was set in accordance with the level of technology. For this case study, prices were set by the experimenter, based loosely on market data [4], while maintaining the assumption that newer, faster, or larger component was always more expensive, but in an actual remanufacturer's case, this data should be readily available from the suppliers that are being used.

In addition to the quantity and purchase price of each component in inventory, the remanufacturer must determine the price at which each component can be sold individually (independent of a full product). For this study, the Sell price was assumed to be half of the purchase price. This value will then be assigned to r_{jk}, and will be used in the computation of the profit function.

Table 4.2: Assumptions on Market Preferences

Key attribute	Type	Critical	Ideal	W_{new}
Processor Brand	HIB	Celeron	Core i	0.057
Processor Speed	HIB	1GHz	3GHz	0.074
Memory size	HIB	512MB	4GB	0.189
Video Memory	HIB	128MB	1GB	0.000
Optical drive	HIB	CD-ROM	DVD-burner	0.189
Hard Drive size	HIB	80GB	500GB	0.057
Operating System	HIB	WindowsXP	Windows7	0.107
Warranty	HIB	None	3yr	0.074
Price ($)	LIB	High ($500)	Low (less than $100)	0.254

4.2 Utility Calculation

Now that the Sell price, Purchase price, and Quantity of the various components have been specified, it must be determined how much utility each component contributes to the product. This is done through use of Conjoint Analysis.

As explained in Section 2.2, Conjoint Analysis is a method of analyzing a customer base's choice preferences with regard to products available in the marketplace. This is done through strategically formulated surveys and other sampling techniques, which are designed to allow the data collector to determine the influence of individual product attributes on the final purchase decision[13]. It is assumed that a potential industrial user of this method will be able to perform, or obtain, a Conjoint Analysis in their market space.

However, for the case study, the Conjoint Analysis was constructed using artificial data, as this study is intended as a proof of concept. Table 4.2 shows the results of the Conjoint Analysis, given the assumptions in the intended market. The relationship among attributes is assumed to be linear, with equal spacing enforced between attributes. For example, the critical level for Processor Speed is 1GHz, which will be assigned a utility of 0, while the ideal level is 3GHz, which will receive a utility of 0.057. If there were only one other Processor Speed level between the critical and ideal levels, it would receive a utility value of 0.057/2 or 0.0285. This would hold for all values of Speed between the critical and ideal levels, not just the 2GHz level. In this way, we are able to account simply for categorical variables such as Processor Brand. The Type column denotes whether a higher specification is better(HIB) or a lower specification is better(LIB). From customer survey information (or in this case, assumptions made by the author) the ranking of the Brands can be made, and then assigned at equal intervals in a linear fashion between 0 and W_{new}. This process is performed for all attributes, and the resulting attribute level utilities are used to form W_{jk}^{c} by summing over the attributes in the given component. Table 4.1 Shows the attribute levels which were used in this case study. The data used to perform the Conjoint Analysis, such as the Taguchi Orthogonal Array with

artificially assigned product weights, are attached in the Appendix.

4.3 Inputs and Outputs

Now that all components have been defined, along with their respective sell prices (r_{jk}), purchase prices (c^s_{jk}), quantities (Q_{jk}), and utilities (W^c_{jk}), this data must be organized in a manner to be input to our solver. The format is specific, in order for the software to automatically determine the correct chromosome representation. Table 4.3 shows the proper organization of our case study data. The first column is designated as the component Type column, while the second column holds the indices for the Variants. The following columns store the Utility, Quantity, Purchase Price, and Sell Price, respectively.

It should be noted that there are zeros in Table 4.3. The zeros in the utility column with regard to Case imply that the customer does not value the choice of case in the decision of which computer to purchase. Obviously, even though no utility is assigned to the component, the final product must still include a case. There are also zeros that exist for other components in the quantity column. There are multiple reasons for this. A zero could be due to the component not being recoverable, such as warranties. A warranty will not be transferable to a remanufactured product, but the customer still values warranties (shown by the non-zero utility values), so it must be decided which warranty to offer. In the case of Video Card 1, there is a zero quantity because the component is new to the market, and no recovered computers contain that component. However, the remanufacturer could purchase the component as a spare and use it in the production of the remanufactured product, if the utility it contributed outweighed the spare purchase cost. This decision will be made automatically by the GA, so the part should be considered as a possible component.

In addition to the component information, we also need to input the Compatibility Matrix. The form is described in detail in Section 3.4. In Figure 4.1 we see a representation of the compatibilities considered in this case study. In this instance, it was only necessary to consider four interactions between components, as all the others are mutually compatible. The compatibility concerns in this specific problem arise from the interconnection between electronic components, as physical connectors and connection standards have changed in the course of component development. The actual Compatibility Matrix used in the case study is shown in Figure 4.2. Note once again that the 1's represent compatible components and the 0's indicate incompatibilities.

Now that the two necessary tables have been formed, the other problem parameters must be specified. Figure 4.3 shows the Inputs that must be provided by the user. The rest of the parameters and inputs are either hard-coded or dependent on the user parameters in some way. Please see Section 3.5.1 for a review of

Table 4.3: Conjoint Analysis - Example Attributes

Part j	Variant k	W_{jk}	Q_{jk}	Z_{jk}	r_{jk}
Processor	1	0.03700	450	25	12.5
	2	0.09300	450	30	15
	3	0.03700	700	30	15
	4	0.09300	700	50	25
	5	0.05600	700	40	20
	6	0.07500	1200	75	37.5
	7	0.11200	500	90	45
	8	0.13100	0	100	50
Motherboard	1	0.00000	800	35	17.5
	2	0.00000	3600	50	25
	3	0.00000	0	100	50
Hard Drive	1	0.00000	320	20	10
	2	0.01425	320	25	12.5
	3	0.01425	600	25	12.5
	4	0.02850	1200	28	14
	5	0.04275	1000	30	15
	6	0.05700	450	35	17.5
Memory	1	0.00000	450	8	4
	2	0.00000	1200	10	5
	3	0.04725	1500	15	7.5
	4	0.09450	1200	25	12.5
	5	0.14175	500	50	25
	6	0.18900	0	75	37.5
Video Card	1	0.00000	400	20	10
	2	0.00000	400	25	12.5
	3	0.00000	1300	30	15
	4	0.00000	1800	35	17.5
	5	0.00000	450	50	25
	6	0.00000	0	80	40
Optical Drive	1	0.09450	200	20	10
	2	0.18900	2800	25	12.5
	3	0.00000	0	15	7.5
Case	1	0.00000	750	35	17.5
	2	0.00000	2000	40	20
	3	0.00000	1500	45	22.5
OS	1	0.00000	0	10	5
	2	0.05350	0	20	10
	3	0.10700	0	30	15
Warranty	1	0.00000	0	0	0
	2	0.03700	0	15	7.5
	3	0.07400	0	50	25

	CPU	Mother-board	Hard Drive	Memory	Graphic Card	Optical Drive	Case	OS	Warranty
CPU		X							
Mother-board			X	X	X				
Hard Drive									
Memory									
Graphic Card									
Optical Drive									
Case									
OS									
Warranty									

Figure 4.1: Incompatibilities considered in Case Study

Part i		Processor								Motherboard			Hard Drive						Memory						Video Card						Optical Drive			Case			OS			Warranty		
	Variant j	1	2	3	4	5	6	7	8	1	2	3	1	2	3	4	5	6	1	2	3	4	5	6	1	2	3	4	5	6	1	2	3	1	2	3	1	2	3	1	2	3
Processor	1	1	1	1	1	1	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	2	1	1	1	1	1	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	3	1	1	1	1	1	1	1	1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	4	1	1	1	1	1	1	1	1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	5	1	1	1	1	1	1	1	1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	6	1	1	1	1	1	1	1	1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	7	1	1	1	1	1	1	1	1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	8	1	1	1	1	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Motherboard	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Hard Drive	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	4	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	5	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	6	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Memory	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	4	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	5	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	6	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Video Card	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	4	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	5	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	6	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Optical Drive	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Case	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
OS	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Warranty	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 4.2: Incompatibilities considered in Case Study

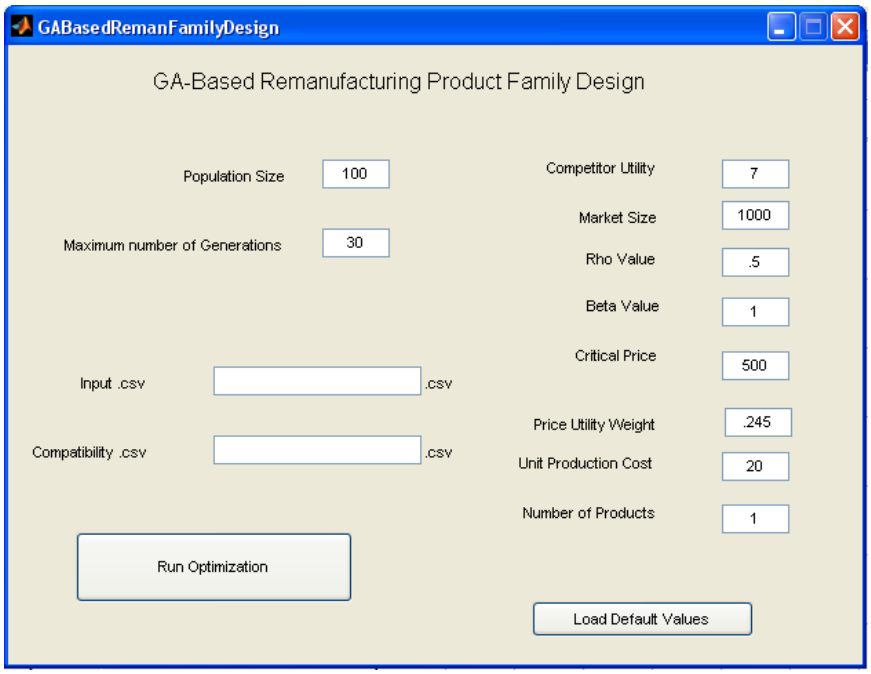

Figure 4.3: Graphic User Interface for Matlab Inputs

parameter dependence. The values shown in Figure 4.3 are the actual parameters used in this case study. Table 4.4 shows where the parameters originate from, as well as any assumptions made for the case study to determine the parameter value.

The first two parameters, Population Size and Number of Generations, are control parameters for GA implementation. These can be tuned to provide for better performance. A simple plot of the progress of the solver is shown in Figure 4.4. In this case, our objective and constraints are converged after approximately fifteen generations, so it is unnecessary to increase the number of generations beyond thirty. In the case of

Table 4.4: Case Study Parameters

Parameter Name	Symbol	Origin(Assumption)
Population Size	-	Standard GA Setting
Maximum Number of Generations	-	Standard GA Setting
Competitor Utility	$\sum_l e^{U_l}$	From Conjoint Analysis
Market Size	M	(Assumption)
Rho Value	ρ	Discount for Used Product (Assumption)
Beta Value	β	Market parameter (Assumption)
Critical Price	P^{max}	From Conjoint Analysis (Assumption)
Price Utility Weight	W^p	From Conjoint Analysis
Unit Production Cost	c^p	(Assumption)
Number of Products	i	User Decision

Table 4.5: Competitor Product Attributes

Attribute	Competitor 1	Competitor 2	Competitor 3	Competitor 4
Processor speed	2GHz	3GHz	3GHz	1GHz
Processor brand	Celeron	Pentium	Core2	Pentium
Memory size	2GB	3GB	4GB	1GB
Video RAM size	256MB	256MB	512MB	128MB
Optical Drive	DVD Burner	DVD Burner	DVD Burner	DVD-ROM
Hard Drive	320GB	500GB	500GB	80GB
Operating System	Windows7	Windows7	Windows7	WindowsXP
Warranty	1	1	1	3
Price	300	400	500	250
Condition	New	New	New	Used

Table 4.6: Competitor Utilities

Attribute	Competitor 1	Competitor 2	Competitor 3	Competitor 4
Processor speed	0.0287	0.0574	0.0574	0.0000
Processor brand	0.0000	0.0246	0.0492	0.0246
Memory size	0.1131	0.1508	0.1885	0.0754
Video RAM size	0.0000	0.0000	0.0000	0.0000
Optical Drive	0.1885	0.1885	0.1885	0.0943
Hard Drive	0.0459	0.0574	0.0574	0.0000
Operating System	0.1066	0.1066	0.1066	0.0000
Warranty	0.0369	0.0369	0.0369	0.0738
Price	0.1270	0.0635	0.0000	0.0000
Utility (U_l)	0.5996	0.6635	0.6844	0.2398
e^{U_l}	1.8214	1.9416	1.9826	1.2709

a larger problem, this might not be the case, and so these parameters are left to be editable by the user. Below the GA parameters in the Graphical User Interface there are two input locations for the names of the Input Table and Compatibility Matrix, which were explained previously.

The right column begins with the Competitor Utility. In order to calculate this value, it must be determined what competitors are in the marketplace and what components are used in their products. To represent an approximate marketplace, we have specified four competitors. The components chosen to represent each product are shown in Table 4.5. These specific combinations of components have been constructed so as to represent an entry-level product, a mid-range product, a high-end product , and a used product, corresponding to Competitors 1-4, respectively. The utility values calculated from these attribute combinations are shown in Table 4.6. It is the value of $\sum_l (e^{U_l})$ which is entered into the 'Competitor Utility' textbox.

The next user input is the Market Size. This value was assumed to be 10,000 units. The following four

parameters, Rho Value, Beta Value, Critical Price, and Price Utility Weight, all pertain to the construction of the Demand function from the MNL model. These values will be identical to those used in the pre-processing using Conjoint Analysis. The Rho Value is set at 0.5 because the remanufactured product requires a utility discount relative to the new product. This discount of 0.5 was also used in the calculation of Competitor 4's utility, as Competitor 4 is assumed to also be a remanufactured product. The next data entry is the Beta Value. We will assume $\beta = 1$. In an actual marketplace, this β value is likely too low, and it's effect on the optimal solution will be discussed in Section 4.4.

The Critical Price must also be entered. Again, this value was used in the Conjoint Analysis, and is equal to \$500. The Price Utility Weight is a result of the Conjoint Analysis, and in this case was found to be 0.245. Lastly, the user is asked to specify how many products will be included in the final Product Family. Cases of i= {1:5} will be tested.

Once the parameters and .csv names are input to the Graphical User Interface, all thats left for the user is to click the 'Run Optimization' button.

To understand exactly the steps that are taken during the solution process, all MATLAB code is included in the Appendix. A sample calculation would simply reiterate what has been described in Section 3.4, and take the reader through the code step by step. Thus, it will be left as an exercise for the interested reader.

4.4 Results and Analysis

The data returned from the solver will now be analyzed, first for the single product case, and then for the multi-product cases. In Table 4.7, the chromosomes of the best individual from five runs of the optimization procedure are shown, with the objective and constraint values shown to the right of the component indices. First, we note the objective values. There is some variation in the optimal value, but this is a result of using a heuristic optimization procedure without a local search method attached[6]. The standard deviation of these five runs is found to be \$6657, which comes to 0.68% of the average value. For a heuristic solver with no local search, this low standard deviation leads to the conclusion that our solution has essentially converged.

This conclusion can be supported by Figure 4.4. In this figure, the objective value of the best individual without constraint violation is plotted against the generation number. As is expected with GA, there are oscillations during the search procedure, but the algorithm converges and remains constant after approximately 30 generations. Both plots show results using identical input parameters. The differences are due to a combination of the random starting population used by the GA, as well as the random operators the GA

Table 4.7: Case Study Results - Single Product

Run #	Q	P	Component Type									Objective	Demand Constraint	Compatibility Constraint	∑Const.
			1	2	3	4	5	6	7	8	9				
1	1421	499.99	5	2	6	5	5	3	2	1	3	973668.60	0	0	0
2	1439	499.92	3	2	5	4	3	3	3	3	2	975422.96	0	0	0
3	1426	499.99	3	2	2	6	4	2	3	3	3	971646.54	0	0	0
4	1415	499.11	3	2	3	3	5	2	2	3	1	988355.92	0	0	0
5	1433	499.99	3	2	1	3	3	3	2	2	3	980096.85	0	0	0

uses to 'evolve' the population. Thus no two runs should progress in the same manner. However, both runs are shown to converge to the same value, indicating convergence to a globally optimal region. This is further demonstrated in Section A.1 in the Appendix, which includes results from multi-product cases. Again, the multiple runs are shown to converge to the save value.

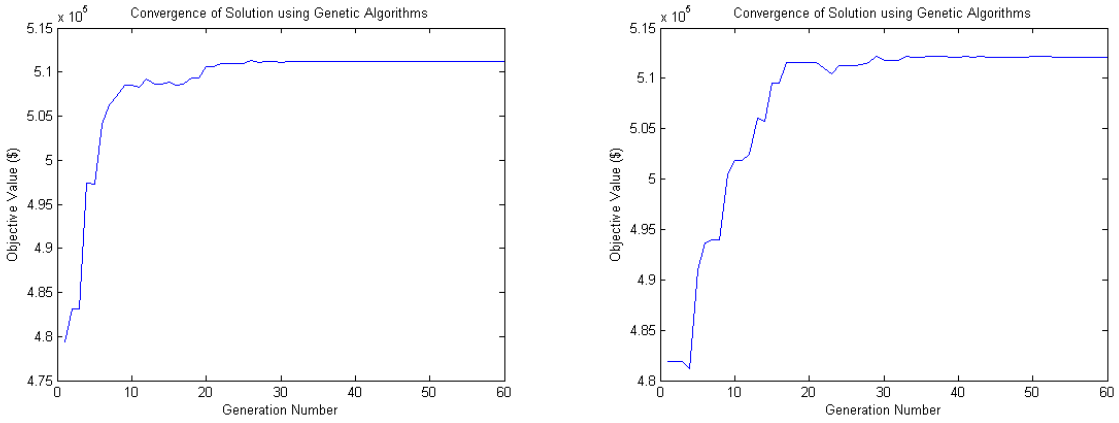

Figure 4.4: Objective Values vs. Generation Number - Single Product Case

Now that the we are confident in the results of the objective function, what information can we gather from the optimal combinations of components? Ideally, the solver would determine the same combination of components for every solution. However, this is clearly not the case. The only component to remain in all of the solutions is Component Type 2, Variant 2, which is Motherboard 2. This choice is not driven by the Variant's impact on product utility, as the motherboards were rated as non-differentiable by the Conjoint Analysis. Instead, this is driven by the quantity recaptured, as there are 3600 units of Motherboard 2 in inventory, compared to 800 and 0 of the other two Variants. The use of Motherboard 2 precludes any spare purchases, and thus is the lowest cost option for the final product.

In other component types, there are multiple components included in different solutions. There are a

Table 4.8: Case Study Results - Single Product Variable Limits

Run #	Quantity Predicted	Demand Limit	Price Predicted	Price Limit
1	1421	1421	499.99	500
2	1439	1442	499.92	500
3	1426	1442	499.99	500
4	1415	1420	499.11	500
5	1433	1433	499.99	500

multitude of reasons that multiple components could be considered in the near-optimal designs. One is an inverse relationship between two of the components parameters. For example, with regard to the three warranties, Warranty 3 has a higher utility value but also costs more to include while Warranty 1 has zero utility but costs nothing. These trade-offs could lead to the improvement in costs by including Warranty 1 to be offset by the decrease in demand.

In other cases, rather than an obvious choice of Variant to include, there are Variants which are missing from all of the proposed solutions. This is the case with both the Memory(Component Type 4) and Video Card (Component Type 5). In neither of these columns does Variant 1 or 2 show up in a solution. This is due to the compatibility constraint. These variants are not compatible with Motherboard 2, and thus including them would violate said constraint.

More dominant in the problem are the values for Quantity and Price. Table 4.8 shows both the values computed by the GA as well as the limits of the constraints with the component combinations from runs one through five. As is clear in Equation 3.9, the demand limit depends on both the utility of the components and the price. The price limit is set by the user in the P^{max} parameter. In both of these cases, the constraints are seen to be active. Again, due to the GA, we know that the solution values are not precisely at the local optimum point, but rather in a globally optimal region [6]. They are sufficiently close to the limits to allow for us to conclude both of these constraints are indeed active.

The variation in the components can be attributed to the degree of influence the utility values have on the market share. Recall that the value of β influences the degree to which customers base their purchase decision on the product utility[3]. With a β of zero, the utility of the product does not impact purchase decisions, while a β of 1 makes the impact of utility affect the market share, but only minimally. Higher β values increase the impact of utility further. In an actual marketplace the β will need to be estimated. During these runs, we were using a default value of $\beta = 1$. This value does not allow for much separation between high and low utility products. If a value of $\beta = 10$ were used instead, less variation in components should be seen. This is confirmed and illustrated in Table 4.9. In this Table, we see a maximum of two Variants included in the optimal product architecture. This demonstrates that the characteristics of the

Table 4.9: Case Study Results - Single Product Results with $\beta = 10$

Run #	Q	P	Component Type									Objective
			1	2	3	4	5	6	7	8	9	
1	99	426.94	4	2	6	6	6	3	3	3	3	509610
2	91	497.64	7	2	5	4	5	2	3	3	3	512150
3	94	484.76	7	2	5	5	6	3	2	3	3	511870
4	102	452.06	7	2	6	6	6	3	3	3	3	510910
5	83	496.98	4	2	6	6	6	3	3	3	3	511270

variant, namely the utility, become an important factor in the maximization of the profit function.

Also note that the quantity remanufactured decreases drastically, to approximately fourteen times fewer units . This is due to the effect of our utility discount for a used product. This model discounts half of the utility for a remanufactured product. Thus, given the greater influence of utility in the market demand estimation (due to the higher β value), the competitors with new products (Competitors 1, 2, and 3) are able to capture a much larger portion of the demand, leaving even the optimally designed remanufactured product with only 1% of the sample market. As is shown here, in an actual Second Life Product Family Design problem β would need to be known due to it's influence on market share, and hence the effect of component utility on optimal product design.

The combination of components included in the solutions presented in Table 4.9 are not similar to any of the eight architectures that were collected as used products. This clearly illustrates the additional value that this design process would provide to the remanufacturing company.

Now we address the multi-product design opportunity. Shown in Table 4.10 is a comparison between average profits using the default values given previously, with β changed to be equal to 10. The zero product case shown in Table 4.10 results from the sale of all recaptured components with no remanufacturing occurring. It is clearly visible in this table that additional products in the portfolio provide an incremental increase in profit. This is likely due to the use of additional inventory to capture additional market share, but this hypothesis will need to be established in future work.

Missing in this analysis is the additional cost for installing multiple (re)production lines. This would likely cause the incremental increase in profit from the addition of multiple products to be offset at some point by the incremental cost associated with the extra line. However, in this simplification, the additional revenue available from the introduction of a product family rather than a single product is clearly evident.

Table 4.10: Case Study Results - Multiple Product Results with $\beta = 10$

# Products	Average Objective over 5 runs($)
0	483925
1	511162
2	533318
3	555110
4	559325
5	566978

Chapter 5

Conclusions and Future Work

This work has illustrated the development and release of a software tool to aid in decision making with regard to the Second Life Product Family Design Problem. This problem was presented as a true design problem, determining the optimal product architecture, price, and quantity to provide the largest profit to a firm. This differs from typical approach to remanufacturing architecture selection, in that product architectures are considered that are not available in the current market space. Through careful formulation of the problem and the targeted use of Genetic Algorithms, **the entire feasible space was searched in the decision process and a globally optimal solution was obtained**.

In helping the decision maker to search the entire space defined by the returned components, the component-based design optimization allows the inclusion of spare parts not obtainable through collection. In our case study some of these cutting edge components were selected in the optimal product design (such as Component Type 4, Variant 6, which was 4GB Memory shown in Table 4.9). This could prove to be particularly advantageous in the electronics markets, where developments in a specific component contribute significantly to the utility of a product. If the problem were restricted to pre-defined product architectures, this opportunity for profit improvement would have been missed. **Utilization of novel components in the remanufactured product is a significant contribution of this work**.

The optimization procedure can also be performed over a multitude of products, allowing for the design of a portfolio of remanufactured products that optimizes company profit. This product family analysis considers interactions between products both in the marketplace and internal to the company, with interaction effects on inventory and spare part purchases accounted for. As was noted in Table 4.10, there are additional gains to be made by designing a product family rather than a single remanufactured product. Again, the previous conclusion about component inclusion in the design can be extended to the multiple product case.

5.1 Future Work

Though this case study was able to demonstrate the solving of the Second Life Product Family Design problem, the assumptions placed on the problem due to the artificial development of the market space, take-back information, and component utility severely hamper efforts to translate the characteristic findings into heuristics to guide actual remanufacturers in their decision making. One such characteristic is the relationship between β, which is a market parameter describing utility impact on consumer preferences, and the variability in the optimal components selected. Further work can be done in this area, using the proposed solution method and tools to solve current industry problems. Only from such 'real world' case studies will the full impact of the proposed solution method be seen. In this context, heuristics could also be developed to provide general guidelines in the Second Life Product Family Design space.

Further analysis can also be performed on the Product Family Design results, to determine the consistency of product niches covered. It would be of interest to note if there are heuristics governing the selection of primary, secondary, and tertiary product architectures, and how the architecture choice changes with the inclusion of more products in the family.

Appendix A

Case Study Data

A.1 Convergence Results

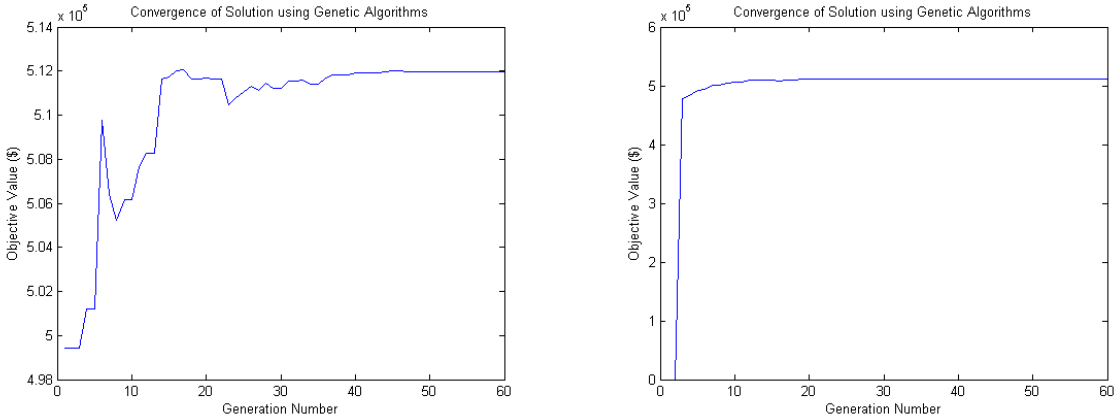

Figure A.1: Objective Values vs. Generation Number - Single Product Case - Runs 3 and 4

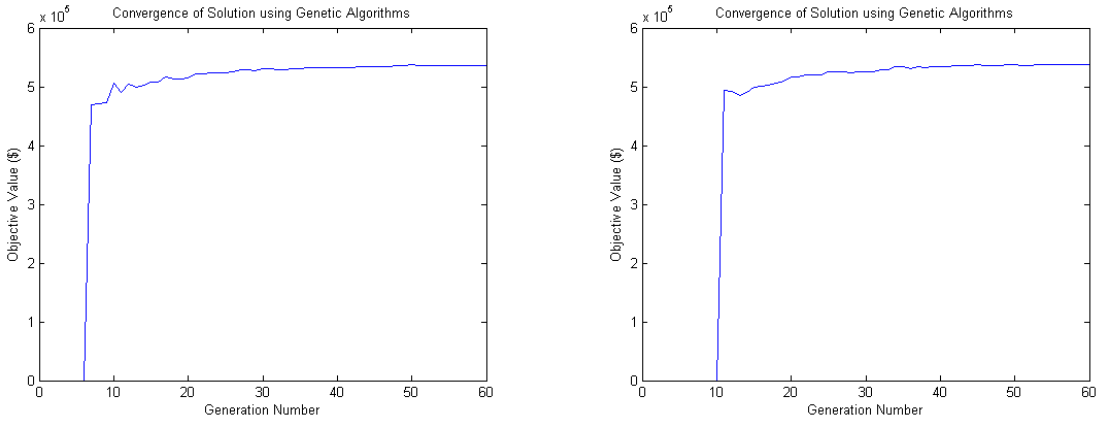

Figure A.2: Objective Values vs. Generation Number - Two Product Case - Runs 1 and 2

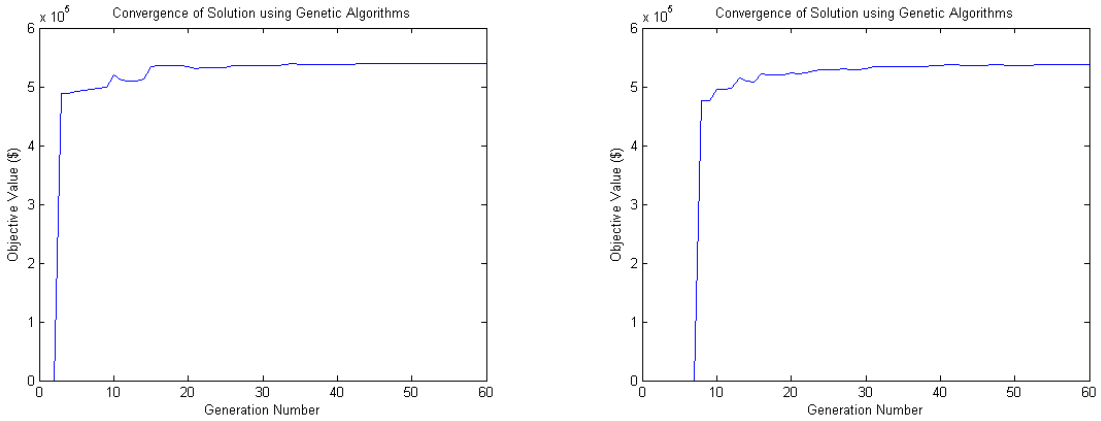

Figure A.3: Objective Values vs. Generation Number - Two Product Case - Runs 3 and 4

A.2 Data Inputs

Table A.1: Recaptured Product Assumptions

Part	Attribute	2003/Budget	2003/High	2006/Budget	2006/High	2007/Budget	2007/High	2008/Budget	2008/High	Present/Spare
	Num. Units Recaptured	500	500	750	750	750	750	500	500	-
Processor	Speed	2GHz	3GHz	2GHz	3GHz	2GHz	2GHz	2GHz	3GHz	3GHz
	Brand	Celeron	Pentium	Celeron	Pentium	Pentium	Core2	Core2	Core2	Corei
	Socket type	I	I	II	II	II	II	II	II	II
	Power consumption	80	100	100	120	100	120	120	140	120
	Lambda=0.018 [18]	7	7	4	4	3	3	2	2	-
	$Q_{functional}$	441	441	698	698	711	711	482	482	-
Motherboard	Socket type	I	I	II	II	II	II	II	II	-
	Memory type	DDR	DDR	DDR2	DDR2	DDR2	DDR2	DDR2	DDR2	-
	Graphic Card	I	I	II	II	II	II	II	II	-
	Hard drive connection type	PATA	PATA	SATA	SATA	SATA	SATA	SATA	SATA	-
	Power consumption	75	100	75	125	100	125	125	150	-
	Lambda=0.0302 [18]	7	7	4	4	3	3	2	2	-
	$Q_{functional}$	405	405	665	665	685	685	471	471	-
Hard drive	Size	80GB	160GB	160GB	250GB	250GB	320GB	320GB	500GB	-
	Connection type	PATA	PATA	SATA	SATA	SATA	SATA	SATA	SATA	-
	Power consumption	15	20	20	25	25	30	30	30	-
	Lambda=0.0633 [18]	7	7	4	4	3	3	2	2	-
	$Q_{functional}$	321	321	582	582	620	620	441	441	-
Memory	Memory size	256MB	512MB	512MB	1GB	1GB	2GB	2GB	3GB	4GB
	Technology type	DDR	DDR	DDR2	DDR2	DDR2	DDR2	DDR2	DDR2	DDR2
	Power consumption	4	8	8	15	15	30	30	45	60
	Lambda=0.0147 [18]	7	7	4	4	3	3	2	2	-
	$Q_{functional}$	451	451	707	707	718 718	486	486	-	
Graphic card	Memory	64MB	128MB	128MB	256MB	128MB	256MB	256MB	512MB	1GB
	Card type	I	I	II	II	II	II	II	II	II
	Power consumption	30	35	35	40	35	40	40	45	50
	Lambda=0.039 [18]	7	7	4	4	3	3	2	2	-
	$Q_{functional}$	381	381	642	642	667	667	462	462	-
Optical drive	Technology type	DVD-ROM	DVD-burner	DVD-burner	DVD-burner	DVD-burner	DVD-burner	DVD-burner	DVD-burner	CD-ROM
	Power consumption	20	30	30	30	30	30	30	30	20
	Lambda=0.1372 [18]	7	7	4	4	3	3	2	2	-
	$Q_{functional}$	191	191	433	433	497	497	380	380	-
Case	Power Capacity	300	300	400	400	400	500	500	500	-
	Lambda=0.0438 [18]	7	7	4	4	3	3	2	2	-
	$Q_{functional}$	368	368	629	629	658	658	458	458	-

Table A.2: End-of-Life Processor Assumptions

Speed	2GHz	3GHz	2GHz	3GHz	2GHz	2GHz	3GHz	3GHz
Brand	Celeron	Pentium	Celeron	Pentium	Pentium	Core2	Core2	Corei
Socket type	I	I	II	II	II	II	II	III
Power consumption	80	100	100	120	100	120	140	120
Price	25	30	30	50	40	75	90	100
Utility	0.037	0.093	0.037	0.093	0.056	0.075	0.112	0.131
Quantity	450	450	700	700	700	1200	500	Spare

Table A.3: End-of-Life Motherboard Assumptions

Socket type	I	II	III
Memory type	DDR	DDR2	DDR2
Graphic Card	I	II	II
Hard drive connection type	PATA	SATA	SATA
Power consumption	100	125	110
Price	35	50	100
Utility	-	-	-
Quantity	800	3600	Spare

Table A.4: End-of-Life Hard Drive Assumptions

Size	80GB	160GB	160GB	250GB	320GB	500GB
Connection type	PATA	PATA	SATA	SATA	SATA	SATA
Power consumption	15	20	20	25	30	30
Price	20	25	25	28	30	35
Utility	0	0.01425	0.01425	0.02850	0.04275	0.057
Quantity	320	320	600	1200	1000	450

Table A.5: End-of-Life Memory Assumptions

Memory Size	256MB	512MB	1GB	2GB	3GB	4GB
Technology type	DDR	DDR	DDR2	DDR2	DDR2	DDR2
Power consumption	4	8	15	30	45	60
Price	8	10	15	25	50	75
Utility	X	0	0.04725	0.0945	0.14175	0.189
Quantity	450	1200	1500	1200	500	Spare

Table A.6: End-of-Life Graphics Card Assumptions

Video Memory	64MB	128MB	128MB	256MB	512MB	1GB
Card type	I	I	II	II	II	II
Power consumption	30	35	35	40	45	50
Price	20	25	30	35	50	80
Utility	X	0	0	0	0	0
Quantity	400	400	1300	1800	450	Spare

Table A.7: End-of-Life Optical Drive Assumptions

Technology type	DVD-ROM	DVD-burner	CD-ROM
Power consumption	20	30	20
Price	20	25	15
Utility	0.0945	0.189	0
Quantity	200	2800	Spare

Table A.8: End-of-Life Case Assumptions

Power Capacity	300	400	500
Price	35	40	45
Utility	-	-	-
Quantity	750	2000	1500

Table A.9: End-of-Life Operating System Assumptions

Version	XP	Vista	7
Price	10	20	30
Utility	0	0.0535	0.107

Table A.10: End-of-Life Warranty Assumptions

Year	None	1	3
Price	0	15	50
Utility	0	0.037	0.074

Table A.11: Assumption on Market Preferences

Key attribute	Type	Critical	Ideal	W_j(new)
Processor Brand	HIB	Celeron	Corei	0.057
Processor Speed	HIB	1GHz	3GHz	0.074
Memory size	HIB	512MB	4GB	0.189
Video Memory	HIB	128MB	1GB	0.000
Optical drive	HIB	CD-ROM	DVD-burner	0.189
Hard Drive size	HIB	80GB	500GB	0.057
Operating System	HIB	WindowsXP	Windows7	0.107
Warranty	HIB	None	3yr	0.074
Price ($)	LIB	High ($500)	Low (less than $100)	0.254

Taguchi Orthogonal Array (9 factors*2 levels)										
Profile ID	**Processor Brand**	**Processor Speed**	**Memory size**	**Video Memory**	**Optical drive**	**Hard Drive size**	**Operating System**	**Warranty**	**Price ($)**	**Score (0-100)**
1	Celeron	1GHz	512MB	128MB	CD-ROM	80GB	WindowsXP	None	High ($500)	10
2	Celeron	1GHz	512MB	128MB	CD-ROM	500GB	Windows7	3yr	Low (less than $100)	60
3	Celeron	1GHz	4GB	1GB	DVD-burner	80GB	WindowsXP	None	Low (less than $100)	75
4	Celeron	3GHz	512MB	1GB	DVD-burner	80GB	Windows7	3yr	High ($500)	55
5	Celeron	3GHz	4GB	128MB	DVD-burner	500GB	WindowsXP	3yr	High ($500)	70
6	Celeron	3GHz	4GB	1GB	CD-ROM	500GB	Windows7	None	Low (less than $100)	80
7	Corei	1GHz	4GB	1GB	CD-ROM	80GB	Windows7	3yr	High ($500)	55
8	Corei	1GHz	4GB	128MB	DVD-burner	500GB	Windows7	None	High ($500)	70
9	Corei	1GHz	512MB	1GB	DVD-burner	500GB	WindowsXP	3yr	Low (less than $100)	75
10	Corei	3GHz	4GB	128MB	CD-ROM	80GB	WindowsXP	3yr	Low (less than $100)	75
11	Corei	3GHz	512MB	1GB	CD-ROM	500GB	WindowsXP	None	High ($500)	30
12	Corei	3GHz	512MB	128MB	DVD-burner	80GB	Windows7	None	Low (less than $100)	80

Figure A.4: Taguchi Orthogonal Array for Conjoint Analysis

Part *j*	Variant *k*	W_{jk}	Q_{jk}	Z_{jk}	r_{jk}
Processor	1	0.03700	450	25	12.5
	2	0.09300	450	30	15
	3	0.03700	700	30	15
	4	0.09300	700	50	25
	5	0.05600	700	40	20
	6	0.07500	1200	75	37.5
	7	0.11200	500	90	45
	8	0.13100	0	100	50
Motherboard	1	0.00000	800	35	17.5
	2	0.00000	3600	50	25
	3	0.00000	0	100	50
Hard Drive	1	0.00000	320	20	10
	2	0.01425	320	25	12.5
	3	0.01425	600	25	12.5
	4	0.02850	1200	28	14
	5	0.04275	1000	30	15
	6	0.05700	450	35	17.5
Memory	1	0.00000	450	8	4
	2	0.00000	1200	10	5
	3	0.04725	1500	15	7.5
	4	0.09450	1200	25	12.5
	5	0.14175	500	50	25
	6	0.18900	0	75	37.5
Video Card	1	0.00000	400	20	10
	2	0.00000	400	25	12.5
	3	0.00000	1300	30	15
	4	0.00000	1800	35	17.5
	5	0.00000	450	50	25
	6	0.00000	0	80	40
Optical Drive	1	0.09450	200	20	10
	2	0.18900	2800	25	12.5
	3	0.00000	0	15	7.5
Case	1	0.00000	750	35	17.5
	2	0.00000	2000	40	20
	3	0.00000	1500	45	22.5
OS	1	0.00000	0	10	5
	2	0.05350	0	20	10
	3	0.10700	0	30	15
Warranty	1	0.00000	0	0	0
	2	0.03700	0	15	7.5
	3	0.07400	0	50	25

Figure A.5: Component Inputs for Case Study

Part i		Processor								Motherboard			Hard Drive						Memory						Video Card						Optical Drive			Case			OS			Warranty		
	Variant j	1	2	3	4	5	6	7	8	1	2	3	1	2	3	4	5	6	1	2	3	4	5	6	1	2	3	4	5	6	1	2	3	1	2	3	1	2	3	1	2	3
Processor	1	1	1	1	1	1	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	2	1	1	1	1	1	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	3	1	1	1	1	1	1	1	1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	4	1	1	1	1	1	1	1	1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	5	1	1	1	1	1	1	1	1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	6	1	1	1	1	1	1	1	1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	7	1	1	1	1	1	1	1	1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	8	1	1	1	1	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Motherboard	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Hard Drive	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	4	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	5	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	6	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Memory	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	4	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	5	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	6	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Video Card	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	4	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	5	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	6	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Optical Drive	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Case	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
OS	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Warranty	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure A.6: Compatibility Matrix for Case Study

Appendix B

Genetic Algorithm Solver: Software Implementation

B.1 'MyGAFile.m'

```
%Control file tasks:
% 1  Read CSV and determine # of components and # choices/component
% 2  Delete the old input file and write a new one (same format as example
% file)
% 3  Delete the old objective file and write an new one
%  function MyGAFile(popsize_in,maxgen_in,computil_in,marketsize_in,rhovalue_in,kvalue_in,
pcritical_in,priceweight_in,unitproductioncost_in,product_number_in,datafile_in,
datafile2_in)

global popsize; %population size for runs
global maxgen; %maximum # of generations
global GAdatatable; % name of array with input data
global tablesize; %# of components(*all types) included in problem
global computil; %value of competition's utility
global marketsize; %size of the total market
global rhovalue; %value of discount factor for utility calculation
global kvalue; %value of k for utility calculation
global pcritical; %critival value of price
global genelength; %# of components in a given type
global genenumber; %# of total types of components (# parts in finished product)
global priceweight; %value from conjoint analysis determining influence of price on utility
global allcomponentssold; %this will store the $ we can make if we just sell all components at
    their market values
global unitproductioncost; %this represents the cost to refurb/clean/assemble a finished product.
    the value is independent of the components used in the product
global com_matrix_used; %this stores the compatability matrix (1's and 0's) 1=compatible  0=incompatible
global product_number;%this stores the # of products to be manufactured
%
% popsize = popsize_in;
```

```
% maxgen=maxgen_in;
% computil=computil_in;
% marketsize=marketsize_in;
% rhovalue=rhovalue_in;
% kvalue=kvalue_in;
% pcritical=pcritical_in;
% priceweight=priceweight_in;
% unitproductioncost=unitproductioncost_in;
% product_number=product_number_in;
% datafile=strcat(datafile_in,'.csv');
% datafile2=strcat(datafile2_in,'.csv');

%crossover rate value for sensitivity analysis
global sens_parama;  %this is a  sensitivity parameter you can use during sensitivity analysis with
    GAsensitivity.m
%mutation rate for sensitivity analysis
global sens_paramb;  %this is a  sensitivity parameter you can use during sensitivity analysis with
    GAsensitivity.m




%for now we will assume a constant value for 'priceweight'
priceweight =.254;
%
%
% %this section is used when commenting out the user inputs for debugging
% %purposes
popsize=100;
maxgen=60;
% computil = 7;
computil = 832.61;
marketsize=10000;
rhovalue=.5;
unitproductioncost=20;
datafile  = 'Optimization_Table1.csv';
product_number=2;

%
```

```
% %k is set in this case, could also ask for user input value
kvalue=10;
% %
% % %pcritical is set in this case, could also ask for user input
pcritical=500;


%load the datatable from the file name and determine the size
GAdatatable = csvread(datafile,1,1 );
tablesize=size(GAdatatable);


%now need to determine max value for components(#of different types per
%component) and total # of components using only datasheet

i=1;
j=1;
k=0;
genelength=0;
genenumber=0; % number of total types of components
while( k < tablesize(1))

if(i==GAdatatable((k+1),1))
    i=i+1;
    k=k+1;
else
    genelength(j)=GAdatatable((k),1); % max value of gene i ( # of possible components of type i)
    genenumber=genenumber+1;
    i=1;
    j=j+1;
end

end
genelength(j)=GAdatatable((tablesize(1)),1);%this line is required because the loop skips the last element
    in the table
genenumber=genenumber + 1;%see above comment



%now we need to load the compatibility .csv just as we loaded the
```

```
%component/utility/recapture .csv
%this .csv should be binary, with 1's representing compatible components
%and 0's representing incompatibilities.  The lower triangular area should
%be all 1's, and 0's should be confined to the upper triangular area.  The
%order of the components MUST be the same as that in the .csv for
%component/utility/recapture

com_matrix=csvread('GAcompatability.csv',2,2);



% com_matrix=csvread('GAcompatibility.csv',2,2);
com_matrix_size=size(com_matrix);

 %could have extra columns of zeros from .csv storage, so need to chop them
 %off
 com_matrix_used=com_matrix(:,1:com_matrix_size(1));

 if tablesize(1)~=com_matrix_size(1)
%      error('Utility .csv is not the same size as Comparability .csv!');
%      return;
 end



%create an input file to feed to GA toolbox
%you must maintain the line ordering as given in the documentation for GA
%toolbox

 fid=fopen('GAproductinput','wt'); %opens input file and returns file id; also clears file
 %fprintf(fid,'this is the text to print \n'); %this line prints out a single line of code into the file
    corresponding to fid

 fprintf(fid,'#GA type \n');
 fprintf(fid,'SGA \n');
 fprintf(fid,' \n');

 fprintf(fid,'# Number of Decision Variables \n');
 numvars=(2+genenumber)*product_number; % included variables are Qreman, Price, and 'all components'
    times '# products')
 fprintf(fid,'%i \n',numvars);
 fprintf(fid,' \n');
```

```
fprintf(fid,'# Decision Variable Type \n');
for (j=1:product_number)%cycles to list Qreman for multiple products
   fprintf(fid,'int 0 %i \n', marketsize); %this is for Qreman: first # is min, second # is max
end
for (j=1:product_number)%cycles to list Price for multiple products
   fprintf(fid,'double 0 %i \n', pcritical); %this is for Price: first # is min, second is max
end
for (j=1:product_number)%cycles to list chromosome for multiple products
for (i=1:genenumber) %now we need to list 'int' for all the components
fprintf(fid,'int 1 %i \n', genelength(i));%first # is min value, second is max
end
end
fprintf(fid,' \n');

fprintf(fid,'# Objectives \n'); %currently we will only be using a single objective function
fprintf(fid,'1 \n');
fprintf(fid,'Max \n'); %objective is to maximize profit
fprintf(fid,' \n');


%first constraint is compatibility constraint
%second constraint is Qreman<=Demand for product 1
%third constraint is Qreman<=Demand for product 2
fprintf(fid,'# Constraints \n');
numconstraints=product_number+1;
fprintf(fid,'%i \n',numconstraints); %single constraint is Qreman<=Demand
fprintf(fid,'1 \n'); %weight of constraint for compatibility
for i=1:product_number
    fprintf(fid,'1 \n'); %weight of constraint for Qreman(i)<=Demand
end
fprintf(fid,' \n');

fprintf(fid,'# General Parameters \n'); %will likely have to make these user-defined
fprintf(fid,'%i \n',popsize); %population size
fprintf(fid,'%i \n',maxgen); %maximum # of generations
fprintf(fid,'0.9 \n');
fprintf(fid,' \n');

fprintf(fid,'# Niching \n');
```

```
 fprintf(fid,'NoNiching \n');
 fprintf(fid,' \n');

 fprintf(fid,'# Selection \n'); %determine a selection procedure
 fprintf(fid,'TournamentWOR 2 \n');
 fprintf(fid,' \n');

 fprintf(fid,'# Crossover \n');
 fprintf(fid,'.9 \n');
%  fprintf(fid,'%f \n',sens_parama);
 fprintf(fid,'SBX 0.5 10 \n');
 fprintf(fid,' \n');

 fprintf(fid,'# Mutation \n');
 fprintf(fid,'0.1 \n');
%  fprintf(fid,'%f \n',sens_paramb);
 fprintf(fid,'Polynomial 20 \n');
 fprintf(fid,' \n');

 fprintf(fid,'# Scaling Method \n');
 fprintf(fid,'NoScaling \n');
 fprintf(fid,' \n');

 fprintf(fid,'# Constraint Handling Method \n');
 fprintf(fid,'Tournament \n');
 fprintf(fid,' \n');

 fprintf(fid,'# Local Search Method \n');
 fprintf(fid,'NoLocalSearch \n');
 fprintf(fid,' \n');

 fprintf(fid,'# Stopping Criteria \n');
 fprintf(fid,'0 \n'); %already have max generation stopping criteria set in "general parameters"
 fprintf(fid,' \n');

 fprintf(fid,'# Load Population \n');
 fprintf(fid,'0 \n'); %will start with a randomly generated population
 fprintf(fid,' \n');

 fprintf(fid,'# Save to File \n');
```

```
 fprintf(fid,'1 EvaluatedGASolutions.txt \n');
 fclose(fid);



 %compute the value for 'allcomponentssold'
 %this value represents the money made if all recaptured components were
 %sold to the market place.
 x=0;
 for i=1:tablesize(1)
     x=x+GAdatatable(i,3)*GAdatatable(i,6);
 end
 allcomponentssold=x;
%  Call the GAToolbox solver

GAtbxm('GAproductinput');

GAresult
```

For an example output of 'MyGAFile', see Figure 3.3.

B.2 sgaFitnessFunction

```
%Code to evaluate the Objective and Constraints for the GA-based
%remanufacturing problem
%
% Called by the toolbox developed by Kumara Sastry in the IllIGAL at UIUC in 2007.
%
%
function objConst = sgaFitnessFunction(decVars)
x = decVars;



% x(1) through x(productnumber) is Qreman
% x(productnumber+1) through x(productnumber*2) is Price
% x(productnumber*2+1+genelength*(i-1)) through
% x(productnumber*2+1+genelength*i) are indexes for components for product
% i and so on through i=productnumber

global genenumber;
```

```
global genelength;
global GAdatatable;
global computil;
global rhovalue;
global kvalue;
global pcritical;
global priceweight;
global marketsize;
global allcomponentssold;
global unitproductioncost;
global com_matrix_used;%this is a binary matrix, with 0's representing the incompatibilities and 1's
    representing compatible components; this is determined in MyGAFile.m so that we have to
    perform data validation on it only once.
global product_number;

%first calculate the utility of our product using the data table utility
%values.  to do this, we need to find the utility values corresponding to
%the various components represented in our chromosome
tablesize=size(GAdatatable);
quant_needed=zeros(tablesize(1),1);%sets up a vector with one column and the same number of rows
    as GAdatatable
for j=1:product_number
    index=1;%designation of row in table
    index2=0;%helps designate where the next loop should start(moves index to next component set)
    for (i=1:genenumber)%runs the same number of times as there are component types
        while(GAdatatable(index,1)~=x(i+product_number*2+(j-1)*genelength))
        %iterates 'index' until a match is found
            index=index+1;
            if(index>max(genenumber))
                break;
            end
        end
        utility(j,i)=GAdatatable(index,2);%good %stores the utility value which corresponds to the chosen
            component in utility(i)
        quant_needed(index,1)=quant_needed(index,1)+x(j);%the quantity used of the component is the quantity
            needed for the other product(s) plus the quantity of this product being made

        index2=index2+genelength(i);%finds the last row of the current component type
        index=index2+1;%finds the first row of the next component type
    end
```

```
end

%need to determine how many of each component were used in our
%remanufactured product(ie subject to amount available and <= qreman.

quantused=min(quant_needed,GAdatatable(:,3));

%now the 'utility' vector holds the utility values of all i components used
%in this individual, so we can then compute the utility of the product

prices=x((1+product_number):(product_number*2));%creates a vector to hold just the prices
quantities=x(1:product_number);%creates a vector to hold just the quantities
productutility=kvalue*rhovalue*(sum(utility')+priceweight*(1-
(prices/pcritical)));

%calculate vectors for spare components needed and excess components for
%resale
s=-1*(quant_needed-GAdatatable(:,3));% creates a vector 's' with (-) elements for spares needed and (+)
    elements for excess parts
sparecomponentsneeded=-(s-abs(s))/2;%eliminates all positive values from 's' and then turns entire
    vector positive

%from product utility, competitor utility, and market size we can now compute the demand

SPARECOST=sum(sparecomponentsneeded.*GAdatatable(:,5));
objConst(1)=  sum(x(1:product_number).*x((product_number+1):product_number*2)) + allcomponentssold -
    sum(quantused.*GAdatatable(:,6)) - sum(sparecomponentsneeded.*GAdatatable(:,5)) -
    unitproductioncost*sum(x(1:product_number));%evaluation of the objective
% x(1)*x(2)  :  the money made from selling x(1) units at x(2) price

% + allcomponentssold  :  the money made by selling all components
% recovered at their respective spare values

%- sum(quantused.*sparesell)  :  the money that is removed from
%'allcomponentssold' because these components are used in the manufacture
%of our product

%- sparecomponentsneeded.*sparepurchase  :  the money needed to purchase
%the spare components at the 'sparepurchase' prices.
```

```
%- unitproductioncost*x(1)  :   the cost to refurb/clean/assemble a finished product times the total
    number of products produced


%this constraint requires that the price be less than pcritical.  This can
%be removed if higher values of price are acceptable
% if x(2)>pcritical
%     objConst(3)=x(2)-pcritical;
% %     objConst(3)=0;
% else
%     objConst(3)=0;
% end

%The next section is used to determine if the components are compatible
%with one another.  If they are not compatible, the constraint is a linear
%function of the number of incompatibilities.

penaltymultiplier=2000; %penalty added for each incompatibility found
% penaltymultiplier=0;
incompatibility=0;%will end up being the # of incompatibilities
for k=1:product_number%cycles through all products in family
        for i=1:genenumber%cycles through all genes(all components used)
            if i > 1%ensure that the row referenced in the comparability matrix corresponds to the
                    proper component
                row=(x(2*product_number+i+(genenumber*(k-1)))+
                sum(genelength(1:i-1)));
            else
                row=(x(2*product_number+i)+(genenumber*(k-1)));%in the case of the first component, just
                    use the component index as the row index
            end

            %inner loop to check all components are compatible with component i

            for j=i:genenumber%again, cycles through all components used

                if j>1%ensure that the column referenced in the comparability matrix corresponds to the
                    proper component
                    col=(x(2*product_number+j+(genenumber*(k-1)))+
                    sum(genelength(1:j-1)));
                else
```

```
                col=(x(2*product_number+j)+(genenumber*(k-1)));%in the case of the first component, just
                    use the component index as the column index
            end

            %now that the correct index(row and column) is determined for the components used,
            %check if the components are compatible
            if com_matrix_used(row,col) <1
                incompatibility=incompatibility+1;%if the components are not compatible, add 1 to the
                    variable 'incompatibility'
            end
        end
    %end the inner loop
    end
end
objConst(2)=penaltymultiplier*incompatibility;%the compatibility constraint is determined by multiplying the
    number of incompatibilities with the assigned penalty value


for i=1:product_number
    demand(i)=marketsize*(exp(productutility(i))/(sum(exp(productutility))+
    computil));

    constraint = x(i) - demand(i)  ; %Qreman must be less than the demand
    objConst(2+i)=constraint*(constraint > 0);% only have a value if const1>0, else the value will be 0


end
```

B.3 GAresult

```
%file to extract the optimal solution from the 'EvaluatedGASolutions.txt'
%file

function best_individual=GAresult;

global genenumber;
global product_number;

es=load('EvaluatedGASolutions.txt');%loads the saved individuals with their objectives and constraint
    violations from most recent GA run
```

```
essize=size(es);%used to determine how many products were in the last run
obj_col=2*product_number+product_number*genenumber+1;
const_col=2*product_number+product_number*genenumber+3+product_number;

%this loop sets objective values to zero if there was any constraint
%violation
for i=1:essize(1)
    if es(i,const_col)>0
        es(i,obj_col)=0;
    end
end

[a,b]=max(es(:,obj_col));%finds location of best remaining individual
best_individual=es(b,:)%returns chromosome of best individual
objective_value=es(b,obj_col)
```

B.4 Sensitivity Analysis

A sensitivity analysis was performed for the crossover and mutation rates to determine the optimal parameter settings for the Second Life Product Design problem. The data from the case study was used as inputs, and the results are shown in Figure B.1.

From the Figure it is evident that the mutation rate should be 0.1 and the crossover rate should be 0.9. Once these values were determined, they were then hard-coded into the software.

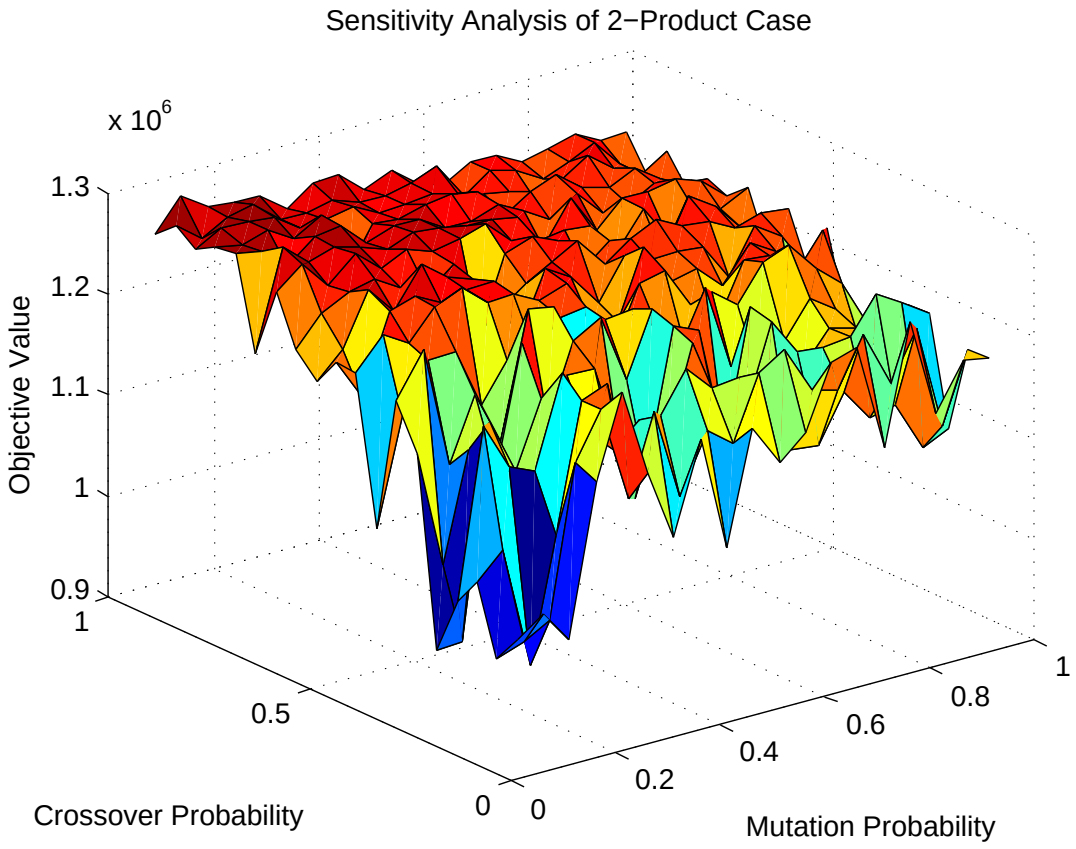

Figure B.1: Sensitivity Analysis of Mutation and Crossover using Case Study Inputs

References

[1] A. J. R. ARCINIEGAS, *Incorporating new considerations for platform selection in product family design: Connectivity and security*, master's thesis, University of Illinois at Urbana-Champaign, 2010.

[2] A. R. Arciniegas and H. M. Kim, *Optimal component sharing in a product family by simultaneous consideration of minimum description length and impact metric*, Engineering Optimization, 43 (2010), pp. 175–192.

[3] C. Breidert, M. Hahsler, and T. Reutterer, *A review of methods for measuring willingness-to-pay.*

[4] CNET.com, 2011.

[5] A. Desai and A. Mital, *Evaluation of disassemblability to enable design for disassembly in mass production*, International Journal of Industrial Ergonomics, 32 (2003), pp. 265 – 281.

[6] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley Professional, New York, NY, first ed., 1989.

[7] ——, *The Design of Innovation*, Kluwer Academic Publishers, norwell, Massachusetts, first ed., 2002.

[8] P. E. Green, J. D. Carroll, and S. M. Goldberg, *A general approach to product design optimization via conjoint analysis*, The Journal of Marketing, 45 (1981), pp. pp. 17–37.

[9] A. Gungor and S. M. Gupta, *Issues in environmentally conscious manufacturing and product recovery: a survey*, Computers & Industrial Engineering, 36 (1999), pp. 811 – 853.

[10] E. Iakovou, N. Moussiopoulos, A. Xanthopoulos, C. Achillas, N. Michailidis, M. Chatzipanagioti, C. Koroneos, K.-D. Bouzakis, and V. Kikis, *A methodological framework for end-of-life management of electronic products*, Resources, Conservation and Recycling, 53 (2009), pp. 329 – 339.

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

[12] K. Inderfurth, A. G. de Kok, and S. D. P. Flapper, *Product recovery in stochastic remanufacturing systems with multiple reuse options*, European Journal of Operational Research, 133 (2001), pp. 130 – 152.

[13] J. Jiao and Y. Zhang, *Product portfolio planning with customer-engineering interaction*, IIE Transactions, 37 (2005), pp. 801–814.

[14] M. Kwak, *Novel description of second life product design problem*, Tech. Report UILU-ENG-2011-2101, University of Illinois, 2011.

[15] G. Lilien, A. Rangaswamy, and A. D. Bruyn, *Conjoint analysis: Marketing engineering technical note*, 2007.

[16] G. Lilien, A. Rangaswamy, and A. D. Bruyn, *Principles of Marketing Engineering*, Trafford Publishing, New York, NY, second ed., 2007.

[17] O. D. of Natural Resources, *Aluminum recycling*, 2011.

[18] L. Painton and J. Campbell, *Genetic algorithms in optimization of system reliability*, Reliability, IEEE Transactions on, 44 (1995), pp. 172 –178.

[19] Recellular, *Environmental statement*, 2011.

[20] M. I. G. Salema, A. P. B. Pvoa, and A. Q. Novais, *Design and planning of supply chains with reverse flows*, in European Symposium on Computer-Aided Process Engineering-15, 38th European Symposium of the Working Party on Computer Aided Process Engineering, L. Puigjaner and A. Espua, eds., vol. 20 of Computer Aided Chemical Engineering, Elsevier, 2005, pp. 1075 – 1080.

[21] K. Sastry, *Single and Multiobjective Genetic Algorithm Toolbox for Matlab in C++*, Illinois Genetic Algorithms Laboratory at the University of Illinois Urbana-Champaign, 117 Transportation Building 104 S. Mathews Avenue Urbana, IL 61801, June 2007.

[22] G. Souza and M. Ketzenberg, *Two-stage make-to-order remanufacturing with service-level constraints*, International Journal of Production Research, 40 (2002), pp. 477 – 493.

[23] G. Souza, M. Ketzenberg, and V. Guide, *Capacitated remanufacturing with service level constraints*, Production and Operations Management, 11 (2002), pp. 231 – 248.

[24] R. Wojanowski, V. Verter, and T. Boyaci, *Retail-collection network design under deposit-refund*, Computers and Operations Research, 34 (2007), pp. 324 – 345. Reverse Logistics.

[25] A. Xanthopoulos and E. Iakovou, *On the optimal design of the disassembly and recovery processes*, Waste Management, 29 (2009), pp. 1702 – 1711. First international conference on environmental management, engineering, planning and economics.

[26] P. Zwolinski, M.-A. Lopez-Ontiveros, and D. Brissaud, *Integrated design of remanufacturable products based on product profiles*, Journal of Cleaner Production, 14 (2006), pp. 1333 – 1345. EcoDesign: What's happening?