The Pennsylvania State University

The Graduate School

College of Engineering

AUTOMATED CONCEPT GENERATION USING BRANCHED FUNCTIONAL MODELS

A Thesis in

Mechanical Engineering

by

Abhinav K. Choudhary

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

August 2010

The thesis of Abhinav K. Choudhary was reviewed and approved* by the following:

Cari B. Arnold
Assistant Professor of Mechanical Engineering
Thesis Advisor

Timothy Simpson
Professor of Mechanical and Industrial Engineering
Thesis Reader

Karen A. Thole
Professor of Mechanical Engineering
Head of the Department of Mechanical Engineering

*Signatures are on file in the Graduate School

ABSTRACT

This thesis discusses a new concept generation technique that improves upon a previous automated concept generation theory and algorithm developed by Bryant, *et al*. at the University of Missouri – Rolla. The previous automated concept generation algorithm utilizes the design knowledge present in a repository to produce an array of partial concept solutions. While the previous algorithm is capable of handling branched functional models, it does not efficiently remove all of the infeasible partial solutions to leave only whole concepts in the final results. A matrix-based algorithm is presented in this thesis that utilizes the result from the previous concept generation algorithm and solves for complete solutions of branched concepts. The proposed algorithm eliminates incomplete and infeasible concepts or components from the results and generates a set of full solutions for further analysis by a designer. The details of the algorithm are described in this thesis, and a peanut-sheller example is used to illustrate the effective use of the algorithm for producing branched concept variants.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to sincerely thank my advisor at the Pennsylvania State University, Dr. Cari Bryant Arnold, for granting me the opportunity to pursue this research. She is a very patient advisor who strives to see her students continue to develop their skills and succeeded in life. I owe her a debt of gratitude.

I would also like to thank Dr. Timothy Simpson for reviewing my thesis and giving me valuable feedback. His suggestions helped me to enhance the value of my work.

This research was supported by the National Science Foundation under Grant No. 0742693. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

Chapter 1

Introduction

Concept generation is one of the most important phases in the product design process. It is a stage in the product design process where designers create ideas that are converted into physical configurations to meet the design specifications. The process of generating creative concepts for any technological product requires integration of information from different knowledge domains and, is a time-and experience-intensive task for designers. With increasing complexity of the design problems and increasing demand to bring products quickly to market, designers need effective concept design tools to quickly generate and evaluate concepts during conceptual design. Techniques like brainstorming, sketching, patent search, reverse engineering, etc. may be used for concept generation, but they can be very laborious and time-consuming tasks. These techniques are limited by a designer's experience, and they may not explore solutions that seem unrelated but are analogous to generate the best results.

There are very few options available for supporting designers during the concept design phase, and many formal concept generation techniques are yet to be realized as computational algorithms. Although there are tools available that focus on design stages such as gathering customer inputs [1-3] and later stages of design embodiment or detail design like defining geometry of concept, kinematic analysis, and performance prediction of concepts [4-7], very few computational tools have been investigated and developed to support conceptual design. Product conceptualization is responsible for more than 70% of the total cost incurred during product life cycle [8] and hence even the best designing optimization and manufacturing cannot perfect a poor conceptual design. However recently, there have been attempts to automate conceptual design tools to help guide designers toward the best solutions by building on existing design experience.

This thesis proposes a new automated concept generation technique that improves upon the existing automated concept generation theory and algorithm developed by Bryant, *et al.* [9] at University of Missouri – Rolla (UMR). Chapter 2 presents a summary of relevant research into automated concept generation along with the details of the existing concept generation algorithm developed by Bryant, *et.al.* [9] and the design tools related to it. Chapter 3 outlines the limitations of the existing concept generation algorithm and discusses the details of a new algorithm that was developed to address these limitations. Chapter 4 presents a case study of a peanut-sheller design problem, and discussion of the results of the problem to illustrate the effectiveness of the proposed algorithm. The final Chapter 5 concludes the presented work and discusses the avenues for future work that can be built on the research presented.

Chapter 2

Background

The review of the literature first focuses on systematic approaches for conceptual design and then on various techniques and algorithms that support computational concept generation. Different techniques for concept generation have been outlined, followed by a review of the concept generation algorithm developed by Bryant, *et al.* [9] at the University of Missouri – Rolla (UMR).

2.1 Conceptual Design - Tools and Techniques

The complexity of conceptual design poses a major challenge for automated of concept generation. The concept generation process is not very well understood and the information available is often qualitative in nature [10]. Nevertheless, recent times have seen an increase in the amount of research in the area of conceptual design, and systematic approaches to conceptual design have emerged [11-13]. These concept design techniques have been refined to serve as a launch pad for automation of conceptual design.

Different techniques have been developed based on functional decomposition and partial solution manipulation by Pahl and Beitz [11] and Hubka [14]. The general approach taken in these techniques to solve a problem is to break down the design problem into smaller manageable sub-problems and develop an overall design based on solutions for these sub-problems. Manual concept generation methods like 6-3-5 method [15], brainstorming, and C-sketch [16] can assist in exploration of design spaces beyond a designer's experience. "Catalog design" approaches can be used to browse catalogues of physical components that may match the required design

specifications, but they restrict novelty in design as data in design catalogs are usually subsets of previously designed systems [17]. The theory of inventive problem solving or TRIZ [18] (as known by the Russian acronym) uses 'Design by Analogy' technique to solve for concepts. This technique provides a tabulated representation of large number of solution principles that have been extracted from existing patents. Cognitive models for concept design, aimed at studying iteration of mental processes that occur in designer's mind when performing conceptual design tasks [19-21], have been explored recently, but computational algorithms based on them are yet to be developed. Other manual conventional techniques like morphological matrix [22] and chi-matrix [23] methods are also available for designers, but non-computational tools like these produce abstract concept descriptions instead of concrete concept variants.

Computational tools that support the design process do exist, but they are mainly developed for assisting designers during the initial information gathering stages [1-3], creating function structures [24], or in later stages of design [4-7] where the concept has been finalized. Though recently, researchers have tried to put more effort towards developing computational tools for automated concept generation [10, 25-27]. Computational algorithms like the A-design approach to conceptual design, which was developed by Campbell, *et al.* [25], can produce multiple concepts using design agents. Co-evolutionary approaches to concept generation using genetic algorithm and genetic programming has been investigated by Li and Jin [10,27]. Other automated concept generation algorithms using Greedy Search techniques, Dynamic programming, Hidden Markov Models, and Viterbi algorithms that use archived design knowledge [26] have also been developed to formalize concept generation.

One such tool, which uses archived design knowledge to generate concept variants, was developed Bryant, *et al.* [9] at the University of Missouri-Rolla. The tool uses a matrix-based algorithm to generate concept solutions. The following sections provide a brief overview of the current version of the concept generation tool and the design tools related to it.

2.2 Review of Design Tools for Concept Generation Algorithm

This section describes the design tools that have been used to automate the conceptual phase of design processes. Details of the *Functional Basis* and *Component Basis* (standard design languages for product information and design knowledge collection), and a web-based repository to store design knowledge are also described. Lastly, the details of the concept generation algorithm developed by Bryant, *et al.* [9] are outlined.

2.2.1 Functional Basis

Evolving strategies and emerging methodologies in the field of conceptual design has spurred research to automate the concept design process. As part of this effort, some researchers have focused on creating standardize languages (vocabularies or taxonomies) for product function information and design knowledge collection in order to streamline the approaches to functional modeling [28-34]. This research lead to the formulation of a design language known as the *Functional Basis* [34], which has emerged as a widely used design language to capture and represent design knowledge [35]. Researched in collaboration with the National Institute of Standards and Technology (NIST), the Functional Basis is intended to span the entire mechanical design space without repetition [9,35]. The artifact functions in the Functional Basis are categorized as a set of function-flow terms.

The function terms are described using verbs and, the function set is classified into eight primary class categories [34]. Each category has further divisions in secondary and tertiary classes. The primary classes give a more "high-level" or generalized definition of the function while the lower classes give more "low-level" or more explicit definitions of the function.

The flow set in the Functional Basis enables the designer to give input and output flows related to each function and are described using nouns. The flow set is divided into three categories designated the primary classes, i.e., material, energy, and signal. The secondary and tertiary classes have 20 and 44 noun terms respectively to describe the flow more explicitly. Figure 2-1 shows the primary and secondary classes for function set and flow set of terms in the Functional Basis.

Functional Basis Reconciled Function Set	Primary Class	Secondary Class
	Branch	Separate
		Distribute
	Channel	Import
		Export
		Transfer
		Guide
	Connect	Couple
		Mix
	Control Magnitude	Actuate
		Regulate
		Change
		Stop
	Convert	Convert
	Provision	Store
		Supply
	Signal	Sense
		Indicate
		Process
	Support	Stabilize
		Secure
		Position

Functional Basis Reconciled Flow Set	Primary Class	Secondary Class
	Material	Human
		Gas
		Liquid
		Solid
		Plasma
		Mixture
	Signal	Status
		Control
	Energy	Human
		Acoustic
		Chemical
		Electrical
		Electromagnetic
		Hydraulic
		Magnetic
		Mechanical
		Pneumatic
		Radioactive/Nuclear
		Thermal

Figure 2-1: *Primary and secondary terms for function and flow classes under functional basis [34].*

2.2.2 Component Basis

The component basis provides a consistent means of representing component design knowledge that captures the specific relationships between functional needs and components that are used to fulfill them [36,37]. The component basis enhances the usefulness of component information by grouping similar product artifacts into related classes [36,37]. Groupings of

similar components enable a more generalized representation of concept results and eliminate redundancies in computations. Once the abstract concept variants are selected using the generalized component basis names, individual artifacts classified under chosen component basis names can be further investigated to generate specific concept ideas.

The component basis uses function terms from the Functional Basis as a guide for the classification scheme for the components. The function terms in the Functional Basis become higher-level categories, for example, the component 'divider' falls under the category of separators, and the highest level category 'branchers'. Both categories 'separators' and 'branchers' are adapted from secondary and primary classes in the Functional Basis. Using this ontology allows for well-defined grouping of artifacts, maintaining matrices of manageable sizes, and eliminating artifact redundancies that may not be immediately evident due to the variations in user-dependent naming of artifacts.

2.2.3 Functional Models

Functional models for any product can be generated using the function-flow terms in the Functional Basis. The derivation of a functional model has been described in the works of Stone and Wood [38] and Kurfman, *et al.* [39]. A black box model is generated initially, based on over-all customer needs and product function requirements which include various energy, material, and signal flows. Then, a detailed functional model is created by identifying sub-functions that operate on the flows listed in the black box model [35].

Functional models generated using the Functional Basis have inherent advantages like repeatability, ease of storing data, increased clarity in design problem, and increased scope in search for solutions [40]. Functional models can be used to capture design knowledge about

existing products for inclusion into the design repository [11]. An example of a functional model for an insulated cup is shown below in Figure 2-2.

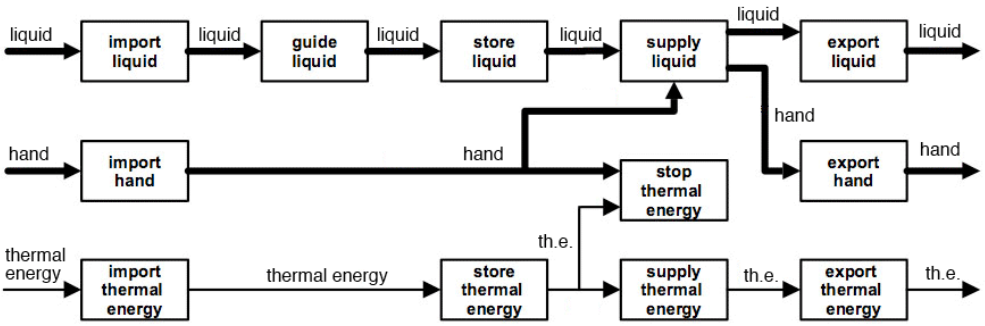


Figure 2-2: *Example of a functional model for an insulated cup.*

2.2.4 Design Repository

The Design Engineering Lab at Oregon State University, and in collaboration with the University of Texas at Austin, Pennsylvania State University, Bucknell University, and Virginia Polytechnic Institute and State University, has developed and populated a web-based design repository for describing and recording product design data [41,42]. This web-repository can be accessed, either as a guest or as a registered user, at the URL:

http://function2.mime.oregonstate.edu:8080/view/index.jsp

The web-based repository allows the user to browse and search artifacts, generate design tools, and view a dictionary of function-flow terms. Currently, the repository is populated with 146 products and 5710 artifacts, mainly from the electromechanical domain, and the knowledge contained in the repository is steadily increasing. The design repository contains a database of various artifacts, along with their attribute information such as functionality, physical parameters, manufacturing processes, failure data, and component connectivity [9]. Quantity, color, material, and dimensions fields further describe the artifact in more detail. Part family name and part

number is given to each artifact to catalog similar artifacts. The web-based repository uses component basis to class to classify the product artifacts.

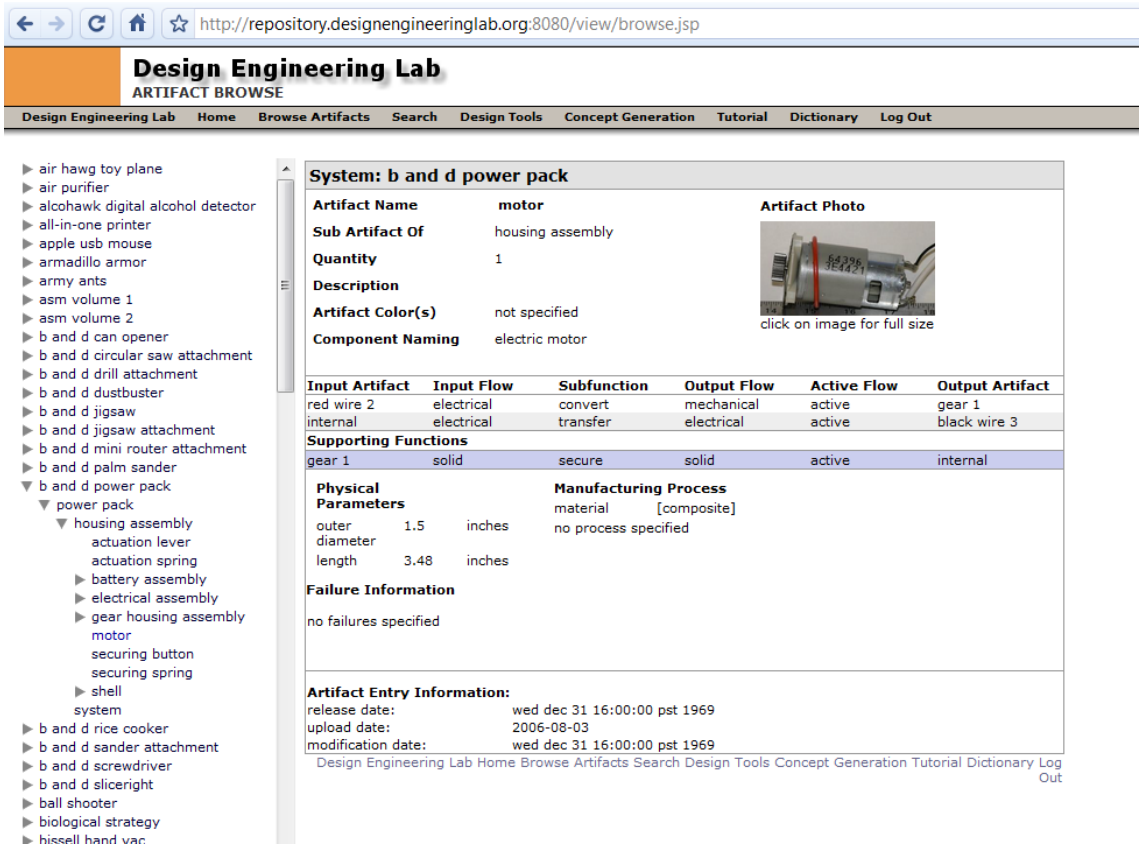


Figure 2-3: *The design repository web interface (GUI)*

One way to utilize the design repository is as a tool for new product design or product redesign [26]. The component basis enables the design repository to be used to generate design tools like the function-component matrix (FCM) and design structure matrix (DSM). The purpose of a FCM is to represent the components that solve each of the listed sub-functions. The FCM is represented in a matrix form where the functions are represented in rows and components are represented in columns. The value of every matrix element represents the number of times the

function is solved by the component, as per the design data stored in the web-repository. The DSM represents the component-component compatibility in the existing consumer products [26] and is a symmetric matrix. A sample of FCM and DSM are shown in Figure 2-4 and Figure 2-5.

FUNCTION / COMPONENT	Agitator	Airfoil	Axle	Ball	Battery	Bearing	Belt
Export solid	0	0	4	0	0	3	2
Import solid	3	0	3	0	0	4	2
Secure solid	1	0	2	0	0	3	0
Transmit mechanical energy	0	0	4	0	0	0	0

***Figure 2-4** : Example of function-component matrix (FCM) for a functional model [26].*

COMPONENT / COMPONENT	Agitator	Airfoil	Axle	Ball	Battery	Bearing	Belt
Agitator	1	0	1	0	0	1	1
Airfoil	0	1	0	0	0	0	0
Axle	1	0	1	0	0	1	1
Ball	0	0	0	1	0	0	0
Battery	0	0	0	0	1	0	0
Bearing	1	0	1	0	0	1	0
Belt	1	0	1	0	0	0	1

***Figure 2-5:** Example of design structure matrix (DSM) [26].*

FCM and DSM enable the function-component relationships and component-component connectivity to be represented in matrix form. The concept generation algorithm developed by

Bryant, *et al*. [9] reads in these matrices to generate and filter the conceptual design variants. The next section gives a brief overview of the concept generator tool developed by Bryant, *et al*. [9].

2.3 Review of Concept Generation Algorithm

The algorithm developed by Bryant, *et al*. [9] utilizes the Functional Basis to link the component functionality with component compatibility, and generates concepts in a brute force manner. The FCM and DSM are created from the web-based repository to describe the function-component relationships and component-component compatibility of the existing product data. Because it is possible for the automated concept generator to create large numbers of design solutions that cannot be reasonably investigated by a designer, data such as failure mode and frequency of occurrence can help to rank and limit the design solutions. Ranking and limiting design solutions helps designers identify the best possible concepts for their desired application. The functions comprising the proposed product's functional model are mapped to the list of components according to component compatibility [9]. The tree of possible component chain is then pruned off by eliminating unworkable component connections according to component-component compatibility. Figure 2-6 briefly illustrates the theory behind the concept generator.

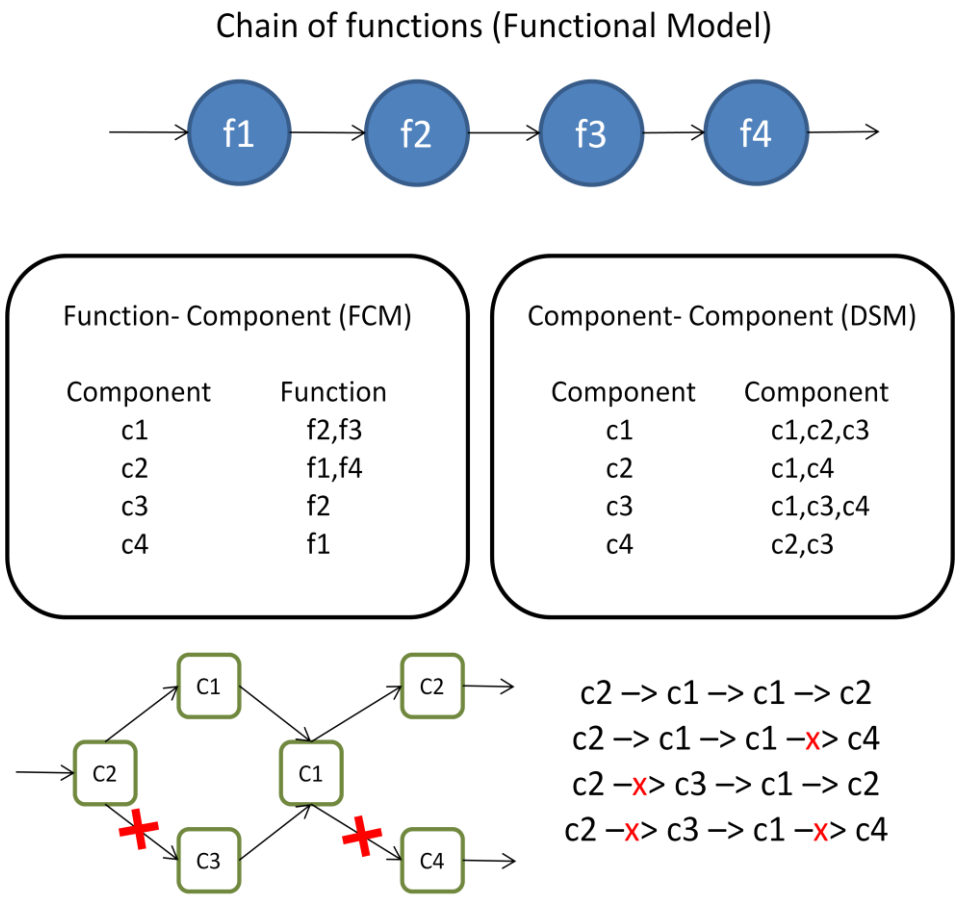


Figure 2-6: *Schematic of the theory behind concept generation algorithm [26].*

The first step requires a designer to develop a functional model for the proposed product. Using the Functional Basis, the functional model is created in a graphical block diagram form. The block diagram is then converted into matrix form, which describes the connections between function chains and sub-functions.

The next step is to establish relationships between the functions and corresponding components that can solve the functions. Information outlining the functionality of each component can be found in the database of the design repository. This information can be used to generate FCM of individual products or groups of products. A zero in the FCM indicates that the

component is not able to solve the corresponding function based on data available in the repository, and a non-zero entry indicates that the component from the column containing the cell can solve the function from the row containing the cell [35]. Once the functional model and FCM are generated, this information is used to create the unfiltered set of design solutions. Matrix multiplication is used to identify all component parings that will solve each function pair defined by the function chain. This gives a list or "a tree" of all the theoretically possible component chain variations that will solve the given function chain.

Finally, the DSM is used to prune the tree of results from the previous step. The component connections between each function pair are checked for compatibility, and incompatible component connections are eliminated. Thus the design knowledge contained in the repository can be used to filter out potentially inadequate design solutions and reduce the set to a manageable size. Once the design solutions are filtered, they can be presented to designers for further analysis [35].

2.3.1 Automation of Concept Generator

Using the algorithm shown in Figure 2-7, a Java-based program was created for the concept generation tool. The user interface for the concept generator tool is shown in Figure 2-6. The tool prompts the user for the location of the FCM and DSM data files generated from the web-based design repository. Once the data files are provided, the user enters the number of function chains and selects the inputs, outputs, and sub-functions in each flow chain. The ranked concepts can be generated by selecting the "Create Concepts!" button.

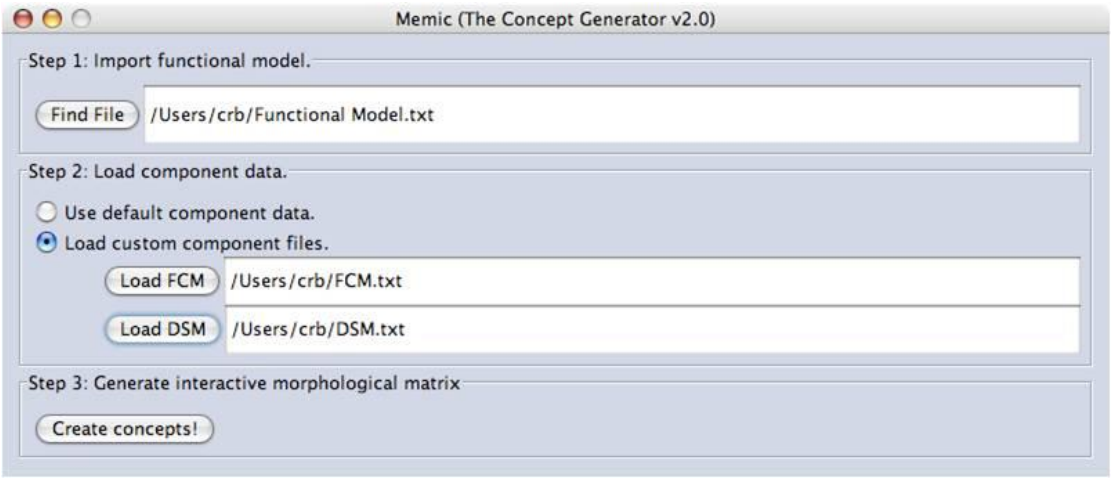


Figure 2-7: *Automated concept generator tool (GUI)*

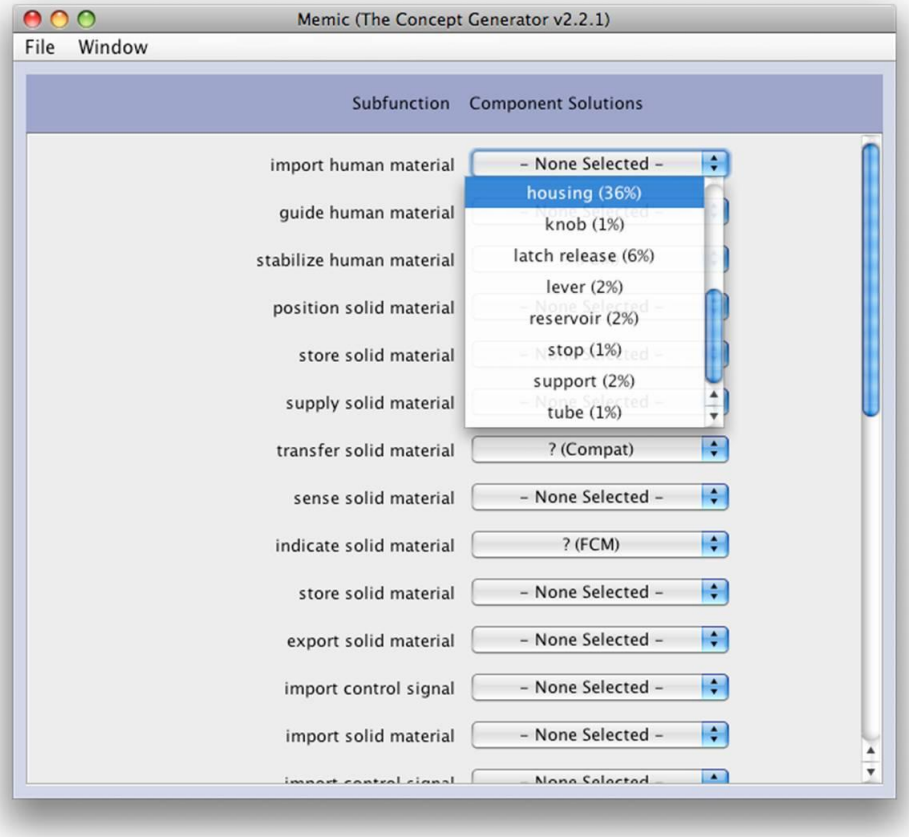


Figure 2-8: *A sample of results generated by the concept generator*

Figure 2-8 shows an example of results generated by the automated concept generator. The component solutions to the sub-functions are listed in a drop-down menu beside each sub-function. A grayed out component in the drop-down menu is incompatible with the other component solutions that have already been selected. The user can also choose components according to the rankings indicated next to each component solution. The number in the parenthesis shown in Figure 2-7 indicates the percentage of total instances the component has solved the indicated sub-function.

Once the ranked concepts are created, the user can save the results and browse through the various concepts variants. The results from the concept generator may also be used as inputs for other non-computational tools for further development or refinement by the designer, to meet the design requirements.

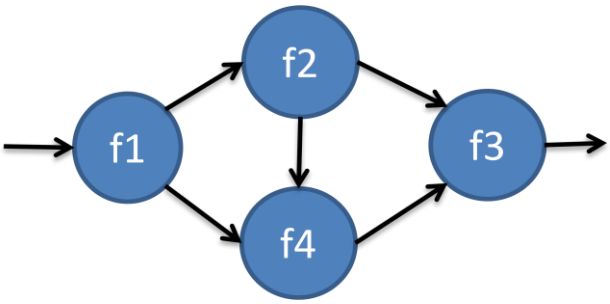
The algorithm developed by Bryant, *et a*l. [9] is capable of generating concepts for straight chain functional models as illustrated in Figure 2-5. generate concepts. limitations of the existing algorithm and the

2.3.2 Limitations of the Existing Concept Generation Algorithm

One of the primary issues of the algorithm developed by Bryant, *et al*. is that it does not eliminate infeasible component solutions to all adjacent function pairs from results when a branched functional model is used. The following section discusses an example of branched functional model to demonstrate this generation of an incomplete conceptual solution. The results for a branched functional model are used to explain the concept of infeasible solutions within the existing approach and to demonstrate the proposed algorithm to eliminate incomplete solutions in branched functional models.

2.4 Branched Functional Models

Figure 2-9 illustrates an example of a branched functional model along with its adjacency matrix. The FCM and DSM used to compute the component solutions to the branched functional model are shown in the Figure 2-10. The results with the component solutions to the function pairings (partial solutions), in the form of result matrix as shown in Figure 2-11, are generated using the existing concept generation algorithm developed by Bryant, *et al.* [9]. Note that the result matrix in Figure 2-11 is generated using a non-symmetric DSM as shown in Figure 2-10 to efficiently illustrate the proposed algorithm using small matrices. A symmetric DSM is still capable of generating the same infeasible component pairings that are discussed in Chapter 3.



Branched Functional Model

		Function			
		f1	f2	f3	f4
Function	f1	0	1	0	1
	f2	0	0	1	1
	f3	0	0	0	0
	f4	0	0	1	0

Adjacency matrix

Figure 2-9: *Example of a branched functional model with its adjacency matrix*

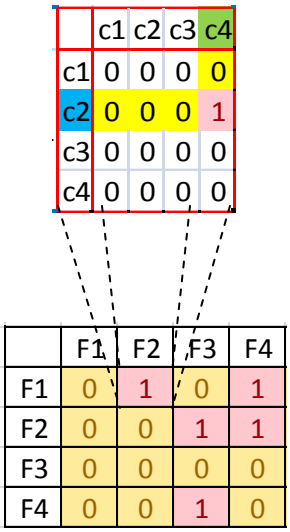
		Component			
		c1	c2	c3	c4
Function	f1	1	1	0	0
	f2	0	0	1	1
	f3	0	1	0	1
	f4	1	1	1	0

FCM

		Component			
		c1	c2	c3	c4
Component	c1	1	0	0	0
	c2	0	1	0	1
	c3	0	1	0	1
	c4	0	1	1	0

DSM

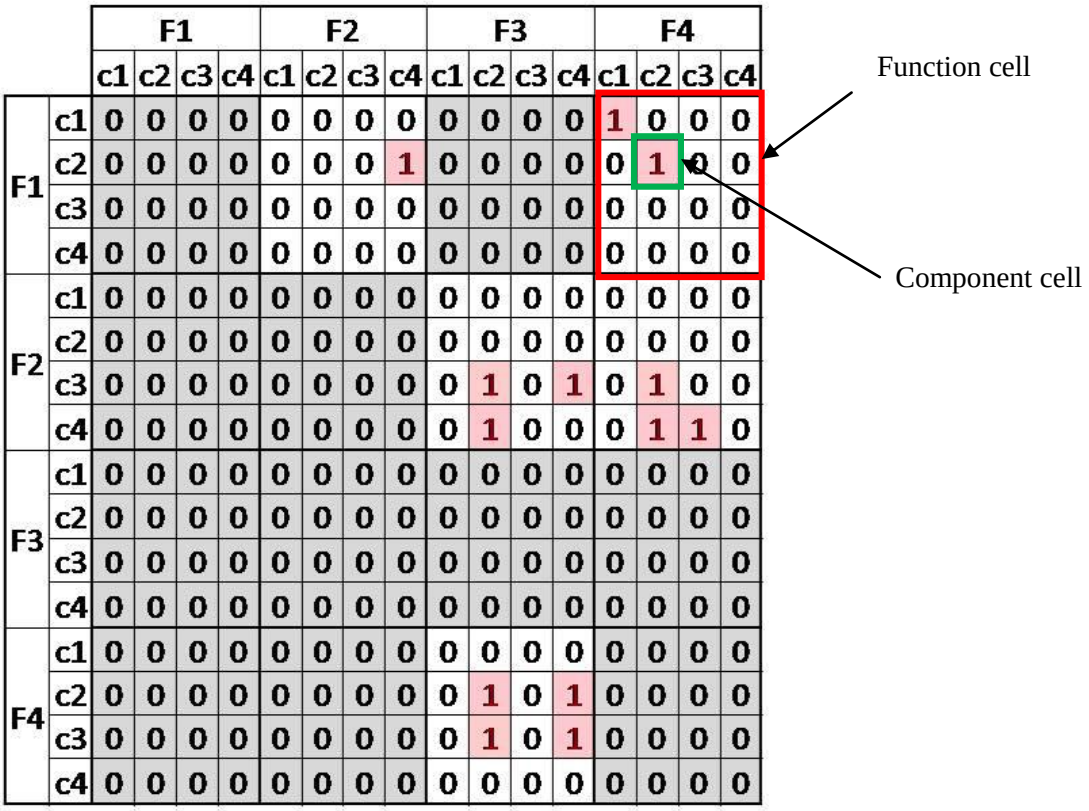
Figure 2-10: *FCM and DSM used to generate result matrix.*



	c1	c2	c3	c4
c1	0	0	0	0
c2	0	0	0	1
c3	0	0	0	0
c4	0	0	0	0

	F1	F2	F3	F4
F1	0	1	0	1
F2	0	0	1	1
F3	0	0	0	0
F4	0	0	1	0

Figure 2-11: *Filtered component adjacency matrix is embedded into the function adjacency matrix.*



		F1				F2				F3				F4			
		c1	c2	c3	c4	c1	c2	c3	c4	c1	c2	c3	c4	c1	c2	c3	c4
F1	c1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
	c2	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0
	c3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	c4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
F2	c1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	c2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	c3	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	0
	c4	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	0
F3	c1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	c2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	c3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	c4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
F4	c1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	c2	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0
	c3	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0
	c4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 2-12: *Result matrix describing pair-wise solutions for each adjacent function set.*

The result matrix can be described using the function cells and component cells. The component cells are contained within the component connectivity matrix, and each connectivity matrix is further embedded within a function cell F_lF_m of the adjacency matrix as shown in Figure 2-11. In Figure 2-11, function cell is represented by F_1F_2 in adjacency matrix contains its component connectivity matrix where every non-zero cell value represents the possible connection between the components that can solve the function pairs F_1F_2. For example, in Figure 2-11, c_2c_4 cell contains a non-zero value implying that the components c_2 and c_4 can solve functions F_1 and F_2 respectively and are also compatible with each other. The corresponding result matrix obtained after embedding all the filtered component connectivity matrices into the adjacency matrix is shown in Figure 2-12. To explain the pertinent information in the result matrix the cells that contained zero values in the original function adjacency matrix have been grayed out in order to avoid confusion.

Hence, a function cell represents the connected function pairs according to the adjacency matrix and the functional model. A component cell is represented by c_ic_j and describes the connectivity between the i^{th} component and the j^{th} component in a component connectivity matrix. A non-zero value for any c_ic_j represents a possible connection between the corresponding components c_i and c_j. Each function cell in the result matrix has its specific component connectivity matrix that is filtered from the DSM using the algorithm developed by Bryant, *et al.* [9].

The existing method to obtain the complete concept solutions to branched functional models from the result matrix is to divide the branched functional model into a number of unbranched function chains, and analyzing the component solutions for each unbranched chain in the result matrix separately. The branched functional model in Figure 2-9 is divided into three unbranched function chains as shown in Figure 2-13. Each function pair is considered only once to avoid redundancy in analyzing the component solutions. The component solutions for each

unbranched function chain can be obtained from the result matrix by tracing the paths of the component connections for the un-branched function chain. The solutions for each function chain are tabulated in Table 2-1graphically represented in Figure 2-14.

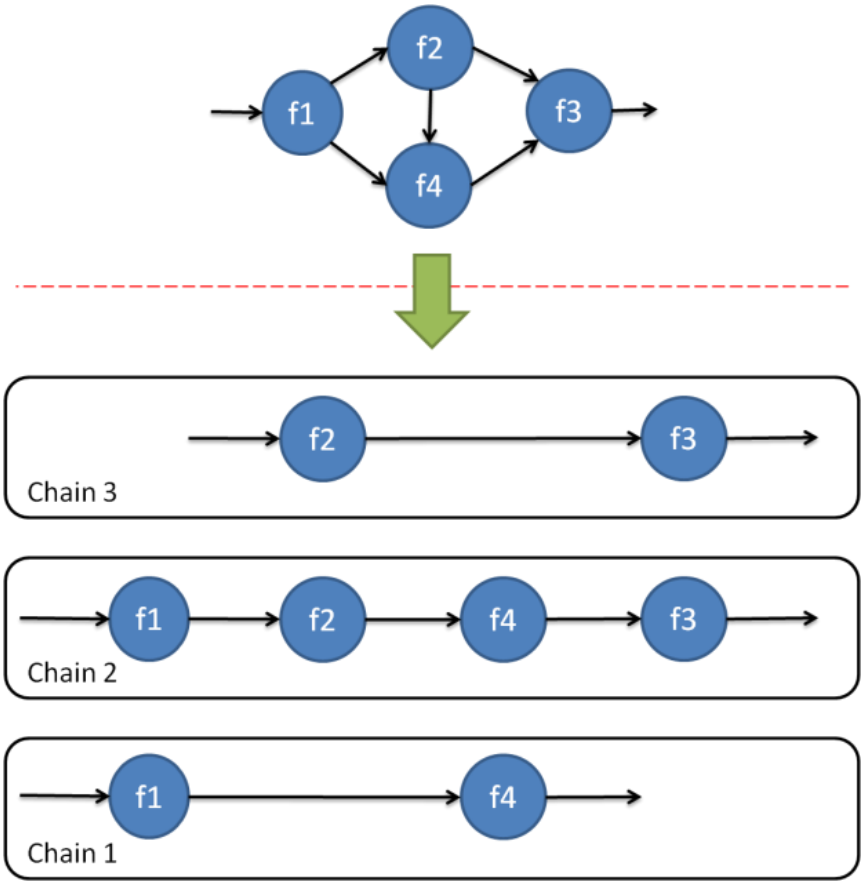


Figure 2-13: *Branched functional model divided into unbranched function chains.*

Table 2-1: *Component solutions to the unbranched function chains obtained from result matrix.*

Function	*Chain 1*	*Chain 2*	*Chain 3*
F_1	c_1, c_2	c_2	-
F_2	-	c_4	c_3, c_4
F_3	-	c_2, c_4	c_2, c_4
F_4	c_1, c_2	c_2, c_3	-

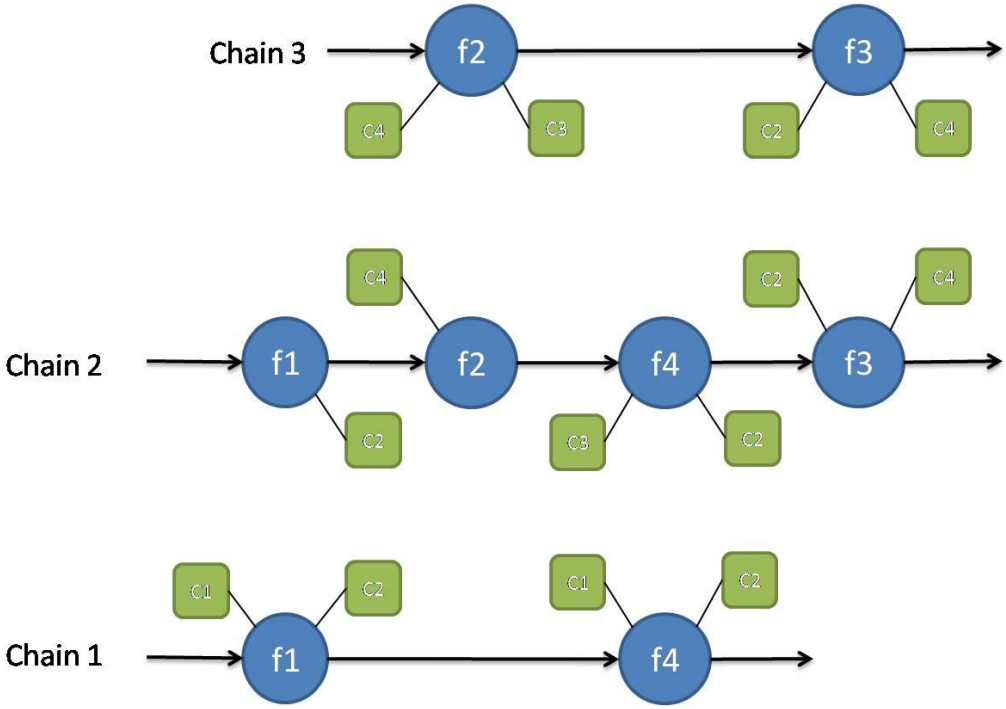


***Figure 2-14**: Graphical representation of component solutions to unbranched function chains.*

Once the results for every individual function chain are known, the function chains can be reassembled to produce complete concept variants for the product to be designed. The function chains can be combined by overlapping the components that are repeated at the branching function in the functional model as shown in the Figure 2-15. For example, the branching function $\boldsymbol{F_2}$ is solved by components $\boldsymbol{c_4}$ and $\boldsymbol{c_3}$ in *chain 3*, but it is solved by just component $\boldsymbol{c_4}$ in *chain 2*. Hence $\boldsymbol{c_4}$ is a *repeated component* in *chain 2* and *chain 3*, and the branched function chain can be reconnected at $\boldsymbol{F_2}$ by overlapping the component $\boldsymbol{c_4}$, as represented by the dotted lines. Such overlapping components that solve the branching functions are termed *feasible components*, and the components that solve branching functions but do not overlap in the function chains are termed *infeasible components.* Component $\boldsymbol{c_3}$ for function $\boldsymbol{F_2}$, is an *infeasible component* as it appears in *chain 3* only, and not in *chain 2*. Hence *chain 2* and *chain 3* cannot be

combined at F_2 using component c_3. These infeasible components have to be eliminated from the result matrix.

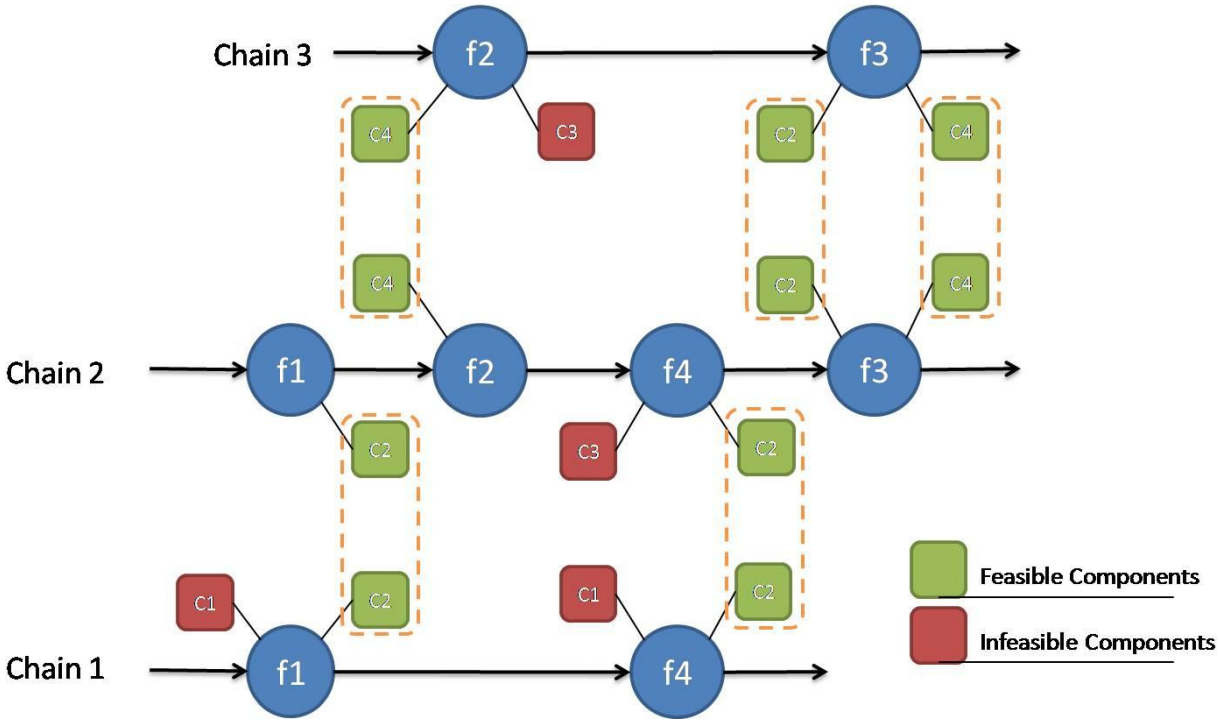


Figure 2-15: *Chains are combined by overlapping the repeated components at the branching functions in the function chains.*

Figure 2-16 shows the component connections for the combined function chains after overlapping the repeating components. Although the component c_4 at F_3 is a repeating component (as seen in Figure 2-14), c_4 at F_3 has to be eliminated, as it is not compatible with the components that solve function F_2, as required by the functional model. Hence, c_4 at F_3 is also an *infeasible component*.

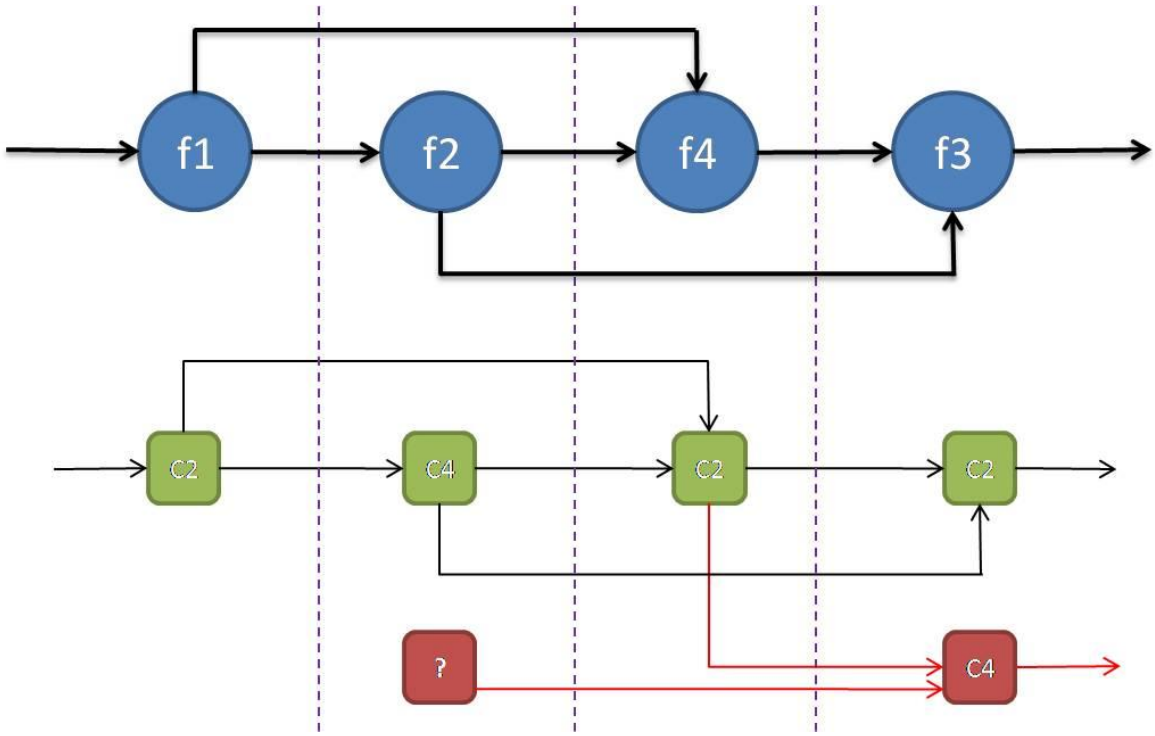


Figure 2-16: *Component connections for combined function chains after overlapping the repeating components.*

This process of dividing the branched functional model, identifying the infeasible component solutions, and recombining the feasible component chain can prove to be a very time- and labor-intensive task for the designer, particularly for more complex branched functional models. Hence, keeping in mind that the concept generator has to simplify the concept generation task and present designers with complete feasible concepts, a new algorithm is proposed that can eliminate infeasible component solutions from the results of concept generator and presents only the feasible component solutions for the branched functional models. The next chapter discusses the approach and the details of the proposed algorithm.

Chapter 3

Automated Concept Generation Technique for Branched Functional Models

This chapter discusses the details of the proposed algorithm to eliminate the infeasible component solutions. The infeasible components are classified into four different categories, i.e., infeasible components of the *Type A, Type B, Type C,* and *Type D*. The approach to eliminate each type of infeasible component is discussed followed by the details of the algorithm.

3.1 Algorithm Approach

This section discusses approach of the proposed algorithm to eliminate the infeasible solutions. The infeasible component connections are eliminated in two iterative steps. The first step eliminates infeasible components classified as *Type A* and *Type B,* and the second step eliminates infeasible components classified as *Type C* and *Type D.* The classification of the infeasible components and the terminology related to them are discusses in the following sections.

Infeasible component Type A are those components that do not satisfy the component compatibility at the branching functions when viewed via the output side of the result matrix. Similarly, *Infeasible component Type B* are those components that do not satisfy the component compatibility at the branching functions when viewed via the input side of the result matrix. The input and output side of the result matrix are shown in the Figure 3-1. Every component at the output side has the ‘connection links’ going out of the components and every component at the input side has “connection links” coming out of the component.

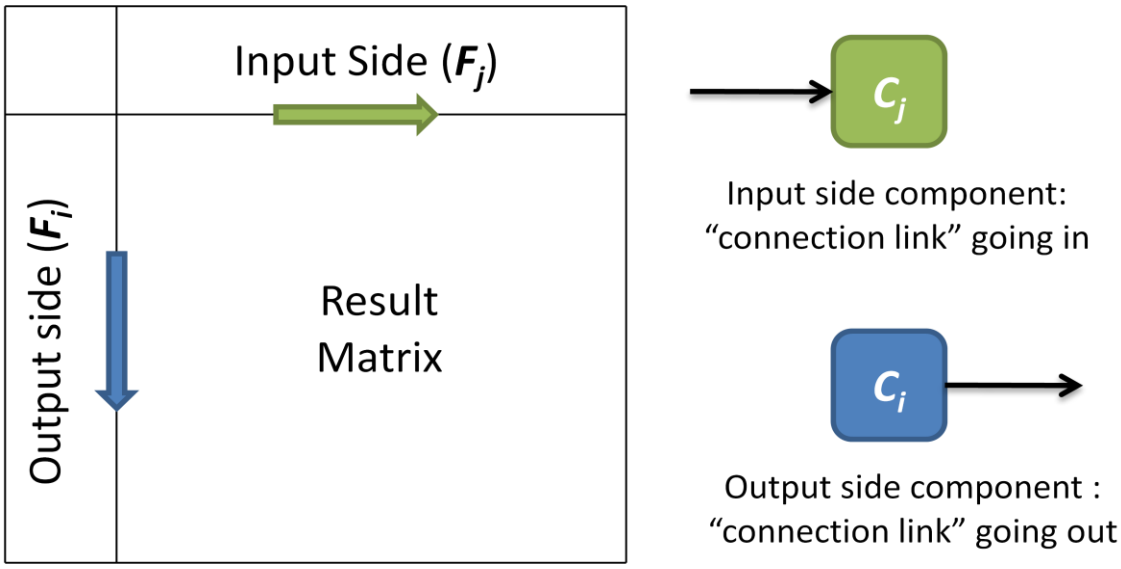


Figure 3-1: *Illustration of the input and output sides of a result matrix.*

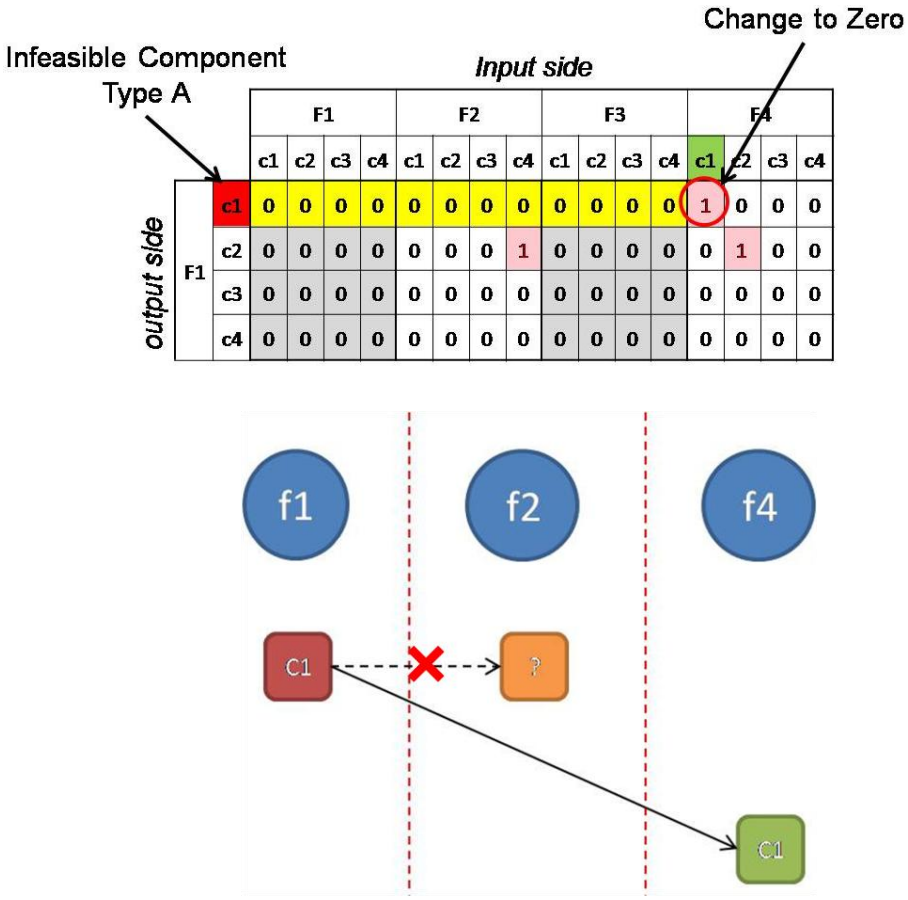


Figure 3-2: *Elimination of infeasible component of Type A.*

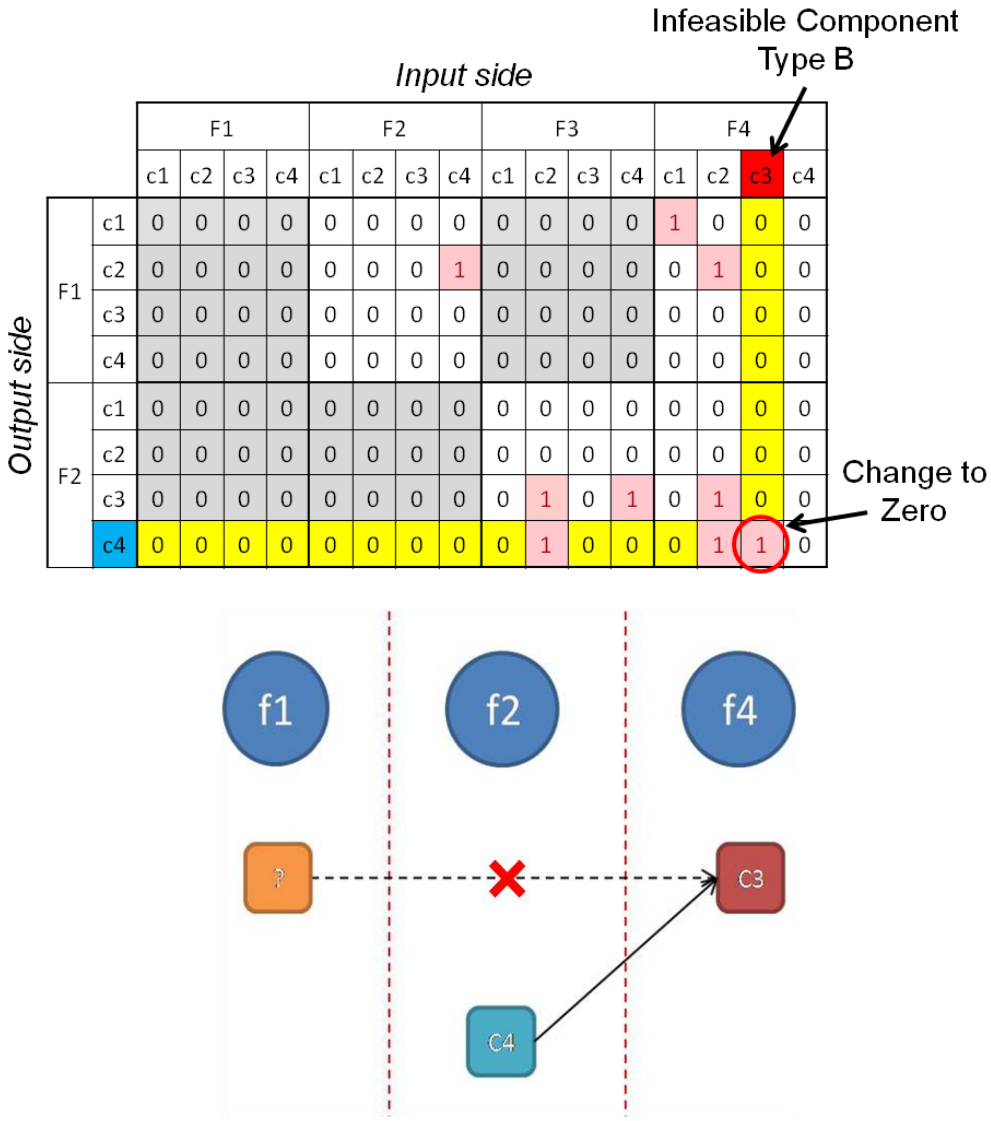


Figure 3-3: *Elimination of infeasible component of Type B.*

In Figure 3-2, the branching function is F_1 and the adjacent functions are F_2 and F_4. Component c_1 is an infeasible component. Although c_1 can solve function F_1, it is compatible only with component c_1 that can solve function F_4 and not with any component that solves F_2 and therefore does not satisfy the compatibility conditions as required by the functional model. Thus here, component c_1 is an infeasible component of Type A and is eliminated by changing any non-zero values to zero along the row for component c_1 in function F_1.The infeasible components of

Type B are eliminated in the similar manner along the columns of the result matrix as shown in the Figure 3-3.

Infeasible components Type C and Type D are those components that have missing input or output connection links in the result matrix. Figure 3-4 illustrates the infeasible components of Type C and Type D.

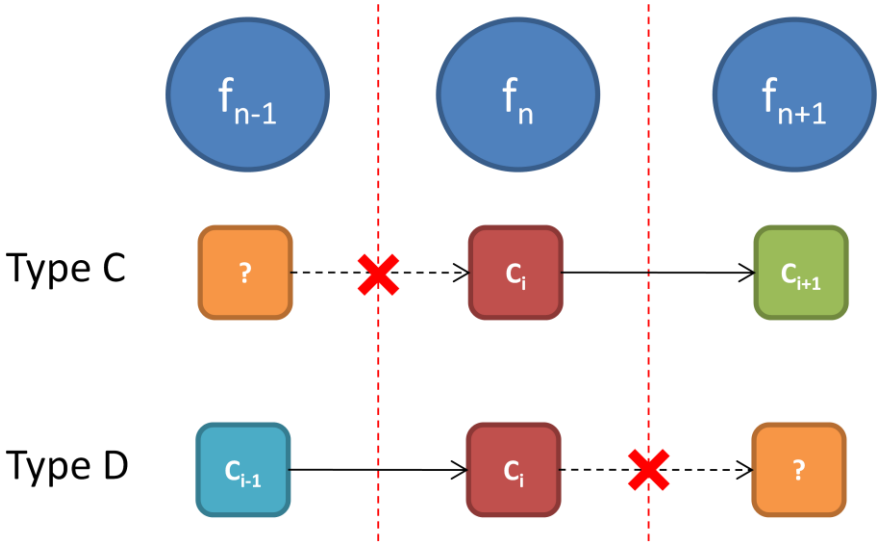


Figure 3-4: *Illustration of infeasible component of Type C and Type D.*

For the result matrix to have no infeasible components of Type C and Type D, the result matrix should have the same set of intermediate components choices (components other than the first and last component in the component chain) for the functions, when functions are viewed from input or output side of the result matrix. The intermediate components that are present only at one side of the result matrix are components with just an *input connection link* or an *output connection link*. The rows/columns in the result matrix containing such components are designated as an *incompatible row/column*. These *incompatible rows/columns* have to be eliminated to ensure that there are no *infeasible components of Type C and Type D*. Figure 3-5 illustrates the elimination of an incompatible row containing an infeasible component of Type C.

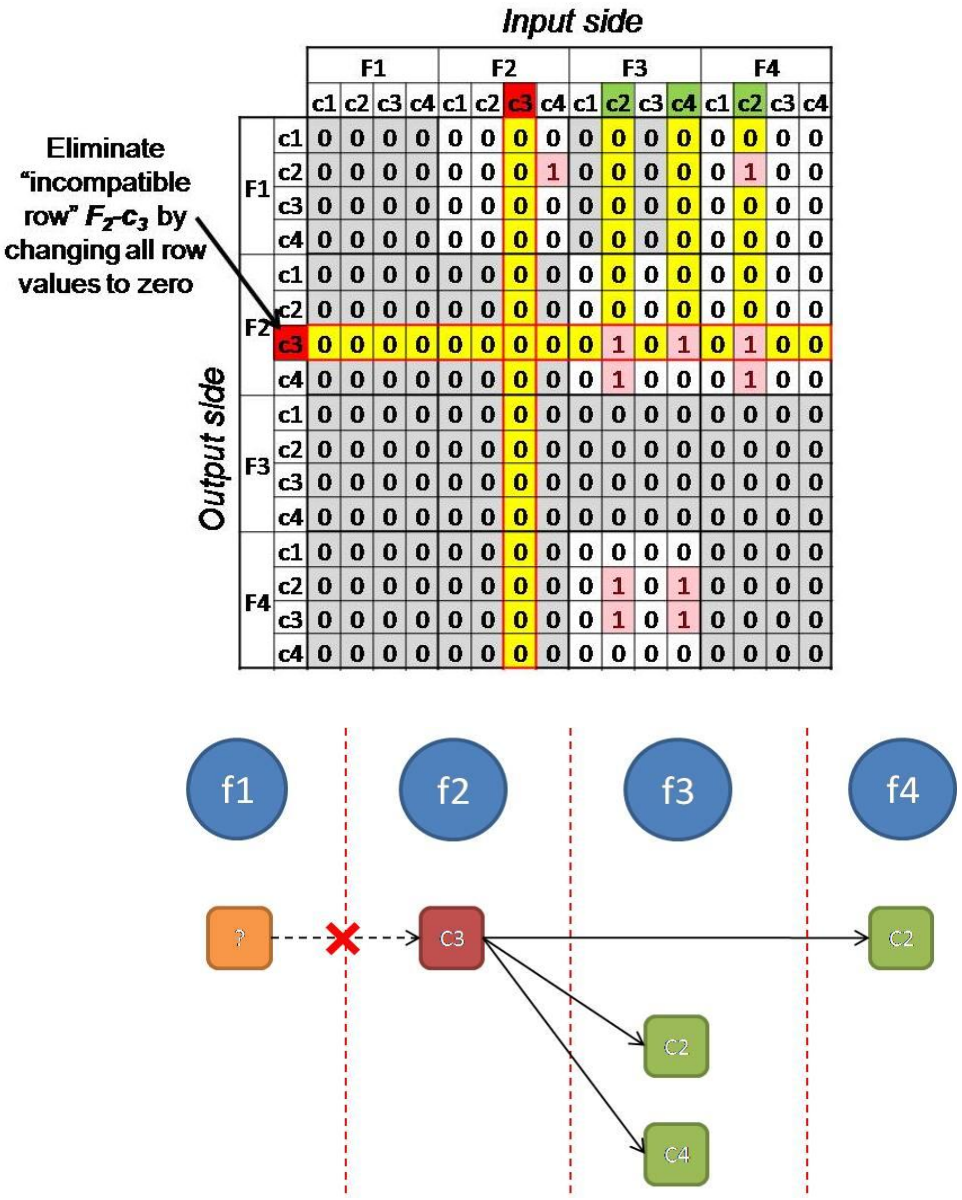


Figure 3-5: *Elimination of infeasible component of Type C.*

In Figure 3-5, the output side of F_2 has components c_3, but the input side of the F_2 does not have component c_3. Hence component c_3 has to be eliminated from the list of components that solve F_2 as using c_3 for F_2 will not all satisfy compatibility with other components of adjacent functions. The component c_3 is eliminated by changing all the non-zero cells in the F_2-c_3 row to

zero. A similar process is followed to eliminate infeasible components of Type D by changing the incompatible columns cells to zero in the result matrix.

3.2 Algorithm Details

This section outlines the various details of the matrix operations carried out on the result matrix to generate complete concepts. The first step is to eliminate the infeasible components of Type A and Type B. To check for infeasible components of the Type A and Type B, two new matrices are created: ***row_sort*** and ***col_sort*** matrix. Figure 3-6 illustrates the operations for creating the ***row_sort***. The data in every function cell is consolidated by using "OR" operation over every row of the function cells to create a consolidated ***row_matrix***. Every row in the ***row_matrix*** is compared with the corresponding rows in the adjacency matrix to verify that there are component choices for every adjacent branching function. A similar operation is carried out column-wise for every function cell in the result matrix to create consolidated ***col_matrix*** and obtain the ***col_sort***. Matrix multiplication of ***row_sort*** and ***col_sort*** gives the ***eliminator_matrix***. The element-wise multiplication of the result matrix and the ***eliminator_martix*** eliminates the infeasible components of Type A and Type B from the result matrix. Figure 3-6, 3-7, and 3-8 explain the operations in detail.

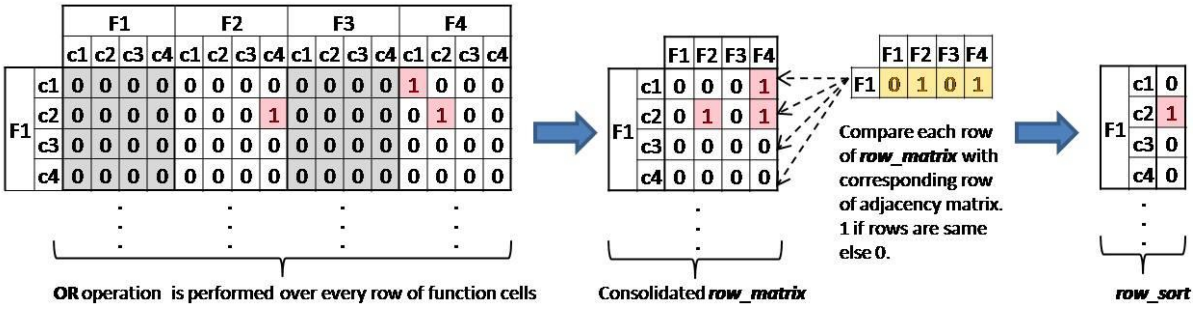


Figure 3-6: *Operations to create row_sort matix.*

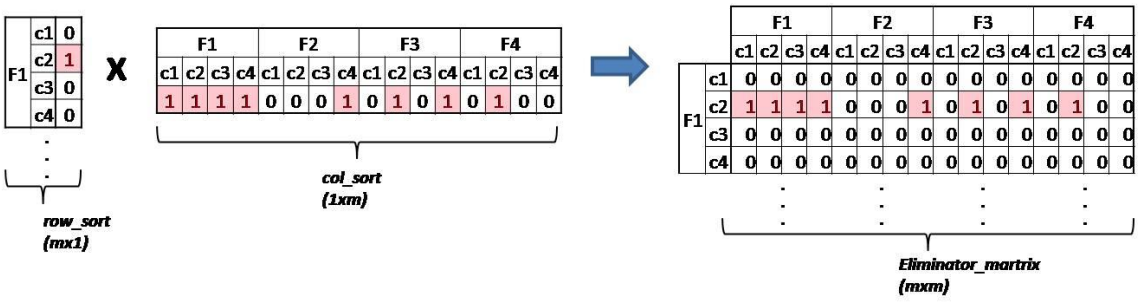


Figure 3-7: *Operations to create Eliminator_matrix.*

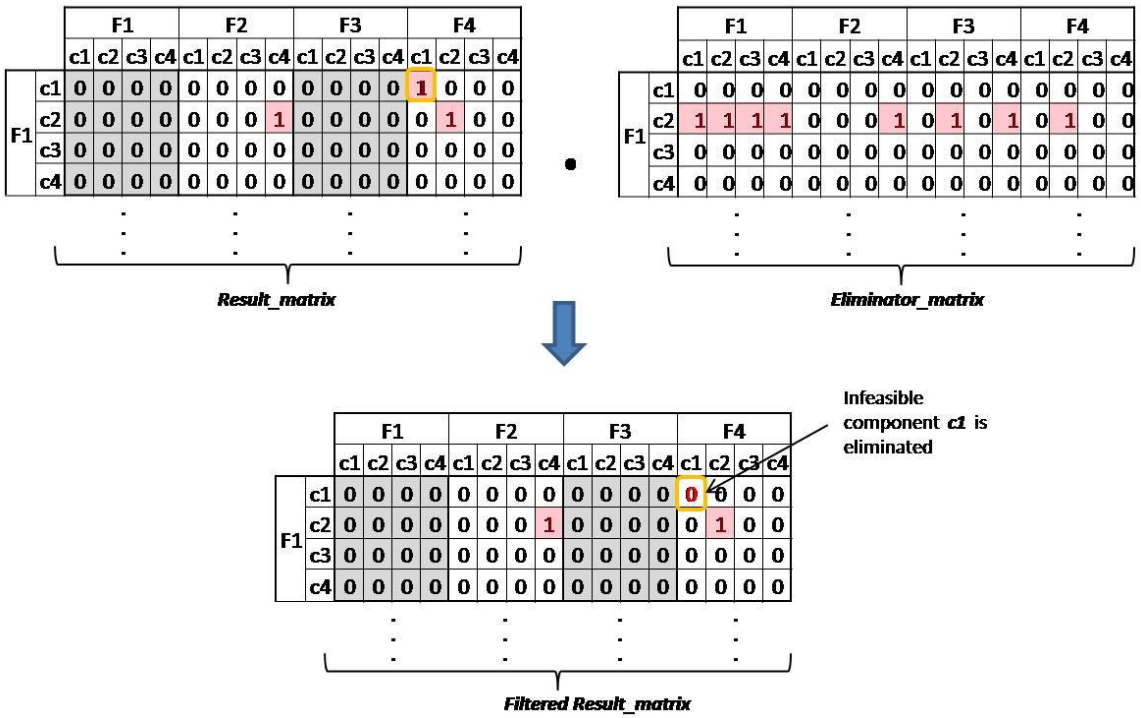


Figure 3-8: *Operations to filter the feasible components in the result matrix.*

Once the filtered result matrix is obtained, the next step is to eliminate infeasible components of Type C and Type D. Two new matrices, ***row_vect*** and ***col_vect***, are created by using an "OR" operation over the rows and columns of the filtered result matrix respectively. The ***row_vect*** and ***col_vect*** matrices represent the components that are active at the "input side" and "output side" of the filtered result matrix. Both the "input side" and the "output side" of the result matrix should have the same active components for a given function to meet the component

compatibility requirements. A non-zero value in ***row_vect*** and ***col_vect*** adjacent to a component represents that the component is active in the filtered result matrix. Figure 3-9 shows an example of a ***row_vect***. The ***col_vect*** matrix is created in a similar manner.

The next step is the element-wise multiplication of ***row_vect*** and the transpose of ***col_vect*** to create ***sig_vect***. All values corresponding to the functional model (global) input functions (the first function that is performed on the "flow") and functional model (global) output functions (the last function that is performed on the "flow") are substituted as "1". The ***sig_vect*** is then superimposed on the every column of the result matrix (i.e., element-wise multiplication of every column of result matrix with the ***sig_vect***) to eliminate the incompatible rows or columns. Figure 3-10 and 3-11 illustrates the process to create the ***sig_vect*** and eliminate incompatible rows/columns.

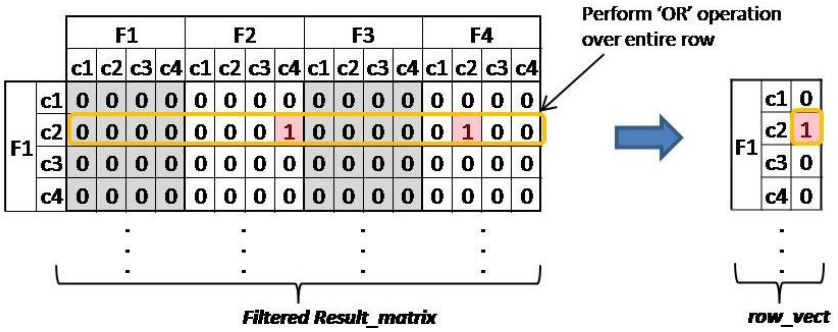


Figure 3-9: *Illustration of operations to create row_vect.*

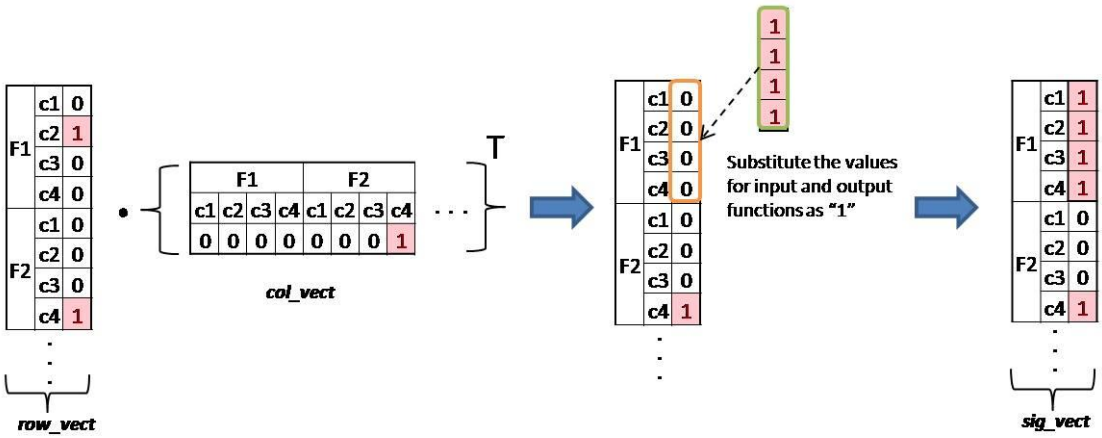


Figure 3-10: *Illustration of operations to create sig_vect.*

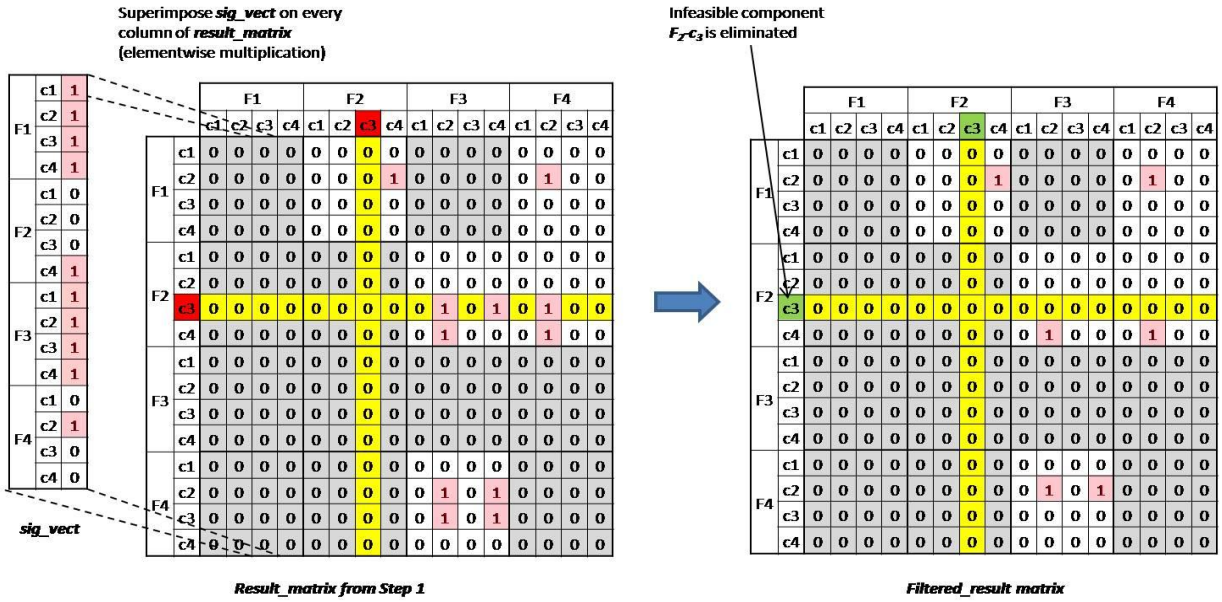


Figure 3-11: *Illustration of operations to eliminate the incompatible rows/columns using sig_vect.*

These steps should be iterated as eliminating infeasible components of Type A and Type B may generate new infeasible components of Type C and Type D, and vice-versa. The iteration is carried out until there are no more infeasible components or incompatible rows/columns to eliminate when generating the final result matrix. Once no more elimination is possible, tracing every "path" of the connections will give the complete set of *complete* component solutions variants that solve the branched functional model.

Chapter 4

Case Study Example

A case study example of a branched chain functional model for a peanut-sheller is used to demonstrate the capability of the proposed algorithm to solve real-world problems of concept generation. The example helps to demonstrate the effectiveness of the described methodology to aid the designer during the conceptual design.

In many countries like Haiti and western African countries, peanuts are a staple crop. Most farmers shell their peanuts by hand, which is an inefficient and laborious process. The objective of the case study was to come up with concepts for peanut-shellers that are low-cost and easy to manufacture with materials readily available in target communities. This design problem, originally posed on a design website for underserved communities known as ThinkCycle, was taken from previously published research by Linsey, *et al*. [43]. Figure 4-1 shows a high-level functional model for a peanut-sheller. The function adjacency matrix was created from this model as shown in Figure 4-2 and served as an input for the proposed concept generator. Note that the peanut-sheller has 2 branches in its functional model. The sub-function "separate solid material" has two input flows: 1) solid material (or the whole peanuts that would be separated into shells and nuts), and 2) mechanical energy, which is used up by the system to separate shells from the nuts. There are two outputs from the "separate solid material" sub-function. Both outputs are connected to "channel solid material" to move the shells away from the nuts.

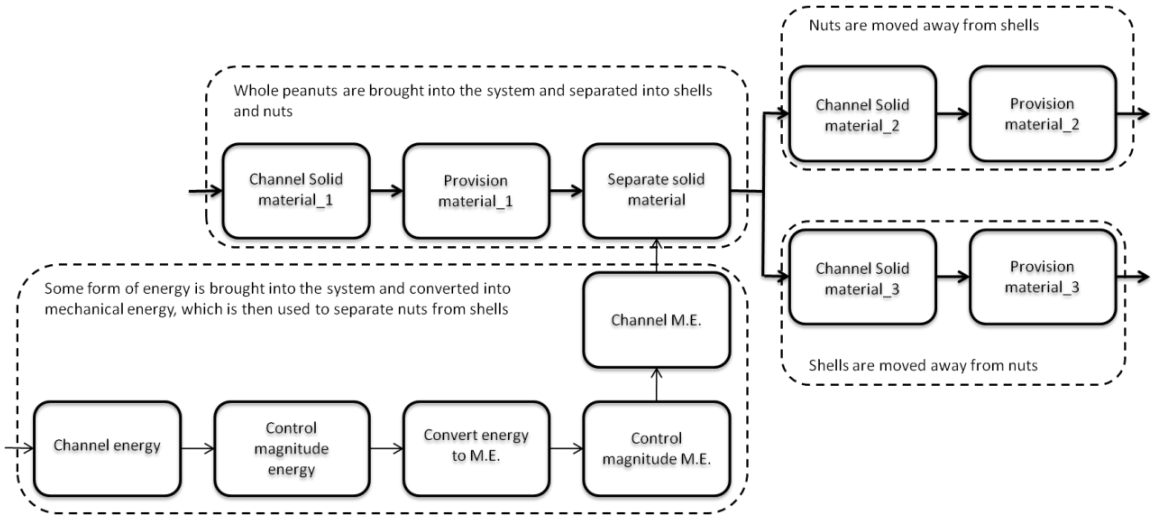


Figure 4-1: *The functional model used to abstractly define the device to shell peanuts [43].*

FUNCTION / FUNCTION	channel solid_1	provision solid_1	separate solid	channel solid_2	channel solid_3	provision solid_2	provision solid_3	channel energy	control magnitude energy	convert energy	control magnitude ME	channel ME
channel solid_1	0	1	0	0	0	0	0	0	0	0	0	0
provision solid_1	0	0	1	0	0	0	0	0	0	0	0	0
separate solid	0	0	0	1	1	0	0	0	0	0	0	0
channel solid_2	0	0	0	0	0	1	0	0	0	0	0	0
channel solid_3	0	0	0	0	0	0	1	0	0	0	0	0
provision solid_2	0	0	0	0	0	0	0	0	0	0	0	0
provision solid_3	0	0	0	0	0	0	0	0	0	0	0	0
channel energy	0	0	0	0	0	0	0	0	1	0	0	0
control magnitude energy	0	0	0	0	0	0	0	0	0	1	0	0
convert energy to ME	0	0	0	0	0	0	0	0	0	0	1	0
control magnitude ME	0	0	0	0	0	0	0	0	0	0	0	1
channel ME	0	0	1	0	0	0	0	0	0	0	0	0

Figure 4-2: *Function adjacency matrix developed from the functional mode of the peanut-sheller.*

A Matlab-based code was created for the proposed concept generator algorithm. A function-component matrix (FCM) and design structure matrix (DSM) were generated from a list of 133 components from the web-based design repository. These files are in Excel format (tab-delimited) and were loaded into the concept generator as input files. The data from the FCM and DSM files were processed to a create result matrix using the existing concept generator algorithm. The result matrix was refined using the proposed concept generator algorithm to filter

infeasible component solutions to function pairs and keep the feasible ones to create complete concepts.

Total solutions to function pairs: 9633

FUNCTION / FUNCTION	channel solid _1	provision solid_1	separate solid	channel solid_2	channel solid _3	provision solid_2	provision solid_3	channel energy	control magnitude energy	convert energy	control magnitude ME	channel ME
channel solid _1	0	949	0	0	0	0	0	0	0	0	0	0
provision solid_1	0	0	200	0	0	0	0	0	0	0	0	0
separate solid	0	0	0	572	572	0	0	0	0	0	0	0
channel solid_2	0	0	0	0	0	949	0	0	0	0	0	0
channel solid _3	0	0	0	0	0	0	949	0	0	0	0	0
provision solid_2	0	0	0	0	0	0	0	0	0	0	0	0
provision solid_3	0	0	0	0	0	0	0	0	0	0	0	0
channel energy	0	0	0	0	0	0	0	0	1640	0	0	0
control magnitude energy	0	0	0	0	0	0	0	0	0	1053	0	0
convert energy	0	0	0	0	0	0	0	0	0	0	922	0
control magnitude ME	0	0	0	0	0	0	0	0	0	0	0	1324
channel ME	0	0	503	0	0	0	0	0	0	0	0	0

Figure 4-3: *Component solutions to function pairs from existing algorithm.*

Filtered solutions to function pairs: 9600

FUNCTION / FUNCTION	channel solid _1	provision solid_1	separate solid	channel solid_2	channel solid _3	provision solid_2	provision solid_3	channel energy	control magnitude energy	convert energy	control magnitude ME	channel ME
channel solid _1	0	949	0	0	0	0	0	0	0	0	0	0
provision solid_1	0	0	200	0	0	0	0	0	0	0	0	0
separate solid	0	0	0	572	572	0	0	0	0	0	0	0
channel solid_2	0	0	0	0	0	934	0	0	0	0	0	0
channel solid _3	0	0	0	0	0	0	934	0	0	0	0	0
provision solid_2	0	0	0	0	0	0	0	0	0	0	0	0
provision solid_3	0	0	0	0	0	0	0	0	0	0	0	0
channel energy	0	0	0	0	0	0	0	0	1640	0	0	0
control magnitude energy	0	0	0	0	0	0	0	0	0	1052	0	0
convert energy	0	0	0	0	0	0	0	0	0	0	921	0
control magnitude ME	0	0	0	0	0	0	0	0	0	0	0	1323
channel ME	0	0	503	0	0	0	0	0	0	0	0	0

Figure 4-4: *Component solutions to function pairs filtered using proposed algorithm.*

The results from the existing concept generator and proposed concept generator were compared to check the number of partial solutions eliminated from the result matrix. Figure 4-3 and Figure 4-4 show the results of the partial solutions from existing and proposed algorithm. The existing algorithm generated a total of 9633 partial solutions for all of the sub-functions in the functional model, inclusive of feasible and infeasible component solutions. The proposed algorithm reduced the set of solutions to a total of 9600, eliminating 33 infeasible component solutions from the results. The remaining 9600 solutions could be aggregated together to generate complete concept solutions to function pairs for the peanut–sheller device problem. The results were generated using a computer with specifications listed in Table 4-1. The proposed algorithm

took an additional time of 3.39 seconds over the existing algorithm to filter the results as shown in Figure 4-5, making the total computation time of 4.05 seconds. Based on that estimate, it takes 0.1 seconds to filter each infeasible solution.

Table 4-1: *Specifications of the computer used for generating results from existing and proposed algorithms.*

Processor	AMD Turion RM-70 @ 2.00 Ghz
RAM	3 Gb @ 800 Mhz FSB

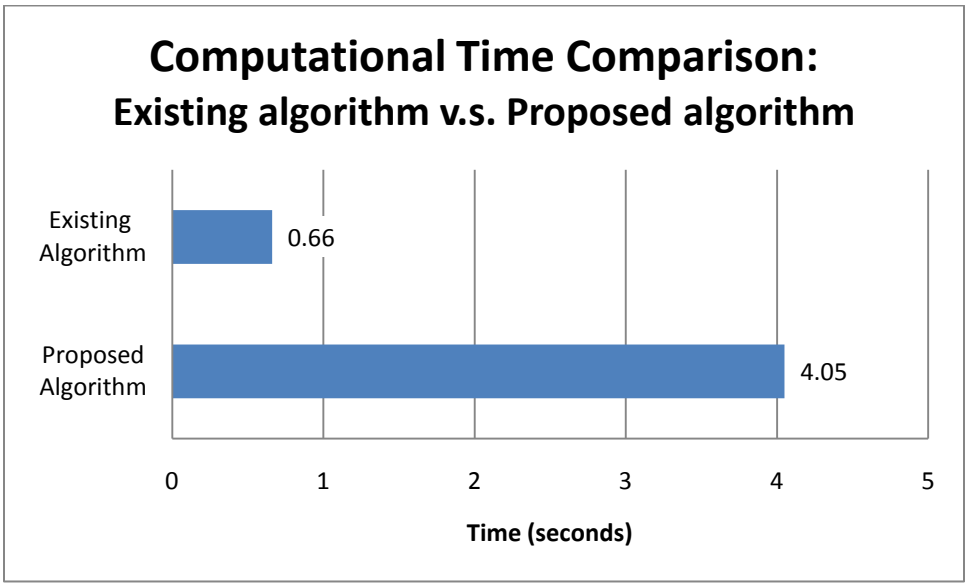


Figure 4-5: *Computational time comparison of existing algorithm and proposed algorithm.*

Comparison of the results in Figure 4-3 and Figure 4-4 shows that the result matrix from the existing algorithm did not contain any infeasible components of Type A and Type B as the component solutions for the branching function (‘separate solid’) and its adjacent functions show no change in the number of component solutions after filtering the results using the proposed algorithm. Infeasible components of Type C and Type D were eliminated from the function pairs

that are marked in red in Figure 4-4. The list of eliminated component solutions to function pairs is provided in the Table 4-2.

Table 4-2: *List of infeasible component solutions eliminated from the result matrix for the peanut-sheller example using the proposed algorithm.*

FUNCTION OUT (F_i)	FUNCTION IN (F_j)	Component Out (c_i)	Component in (c_j)
channel solid _2	provision solid _2	airfoil generator electric plate magnitude controller divider divider electric plate pressure vessel pneumatic piston divider pressure vessel airfoil electric plate pneumatic piston pneumatic piston	airfoil Battery circuit board fastener flywheel housing housing housing lever positioner pressure vessel spring spring tube valve
channel solid _3	provision solid _3	airfoil generator electric plate magnitude controller divider divider electric plate pressure vessel pneumatic piston divider pressure vessel airfoil electric plate pneumatic piston pneumatic piston	airfoil battery circuit board fastener flywheel housing housing housing lever positioner pressure vessel spring spring tube valve
control magnitude energy	convert energy	burner	burner
convert energy	control magnitude ME	burner	burner
control magnitude ME	channel ME	supporter	supporter

An example of a solution generated from the proposed algorithm is illustrated in Figure 4-6. Sketching was employed by a team of freshman-level undergraduate students as a final step to create a drawing of the selected concept variant, and the result is shown in Figure 4-7. Using component basis naming terms [36,37] and pictures from the web-based repository [41,42] of specific artifacts as guides, specific embodiments of the conceptual design were generated for the peanut-sheller device by sketching various configurations of the returned component basis artifact names.

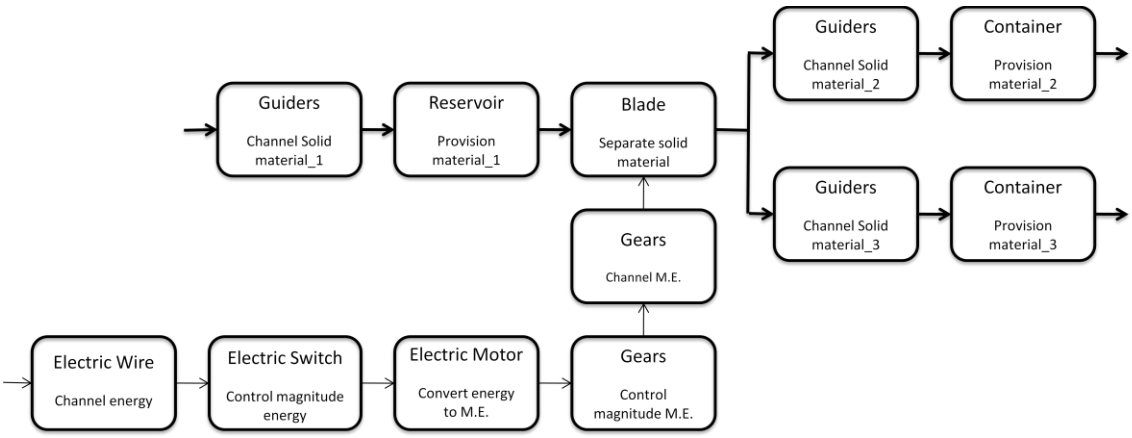


Figure 4-6: *Component solutions for peanut-sheller device generated by the proposed algorithm.*

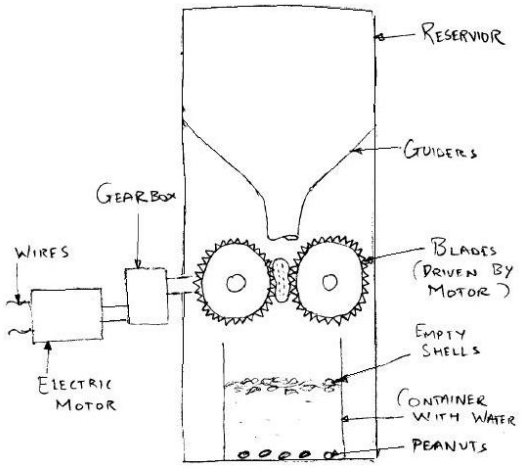


Figure 4-7*: Conceptual sketch of a peanut-sheller device generated by the proposed algorithm.*

Chapter 5

Conclusion and Future Work

This thesis presents a technique to eliminate infeasible solutions to function pairings from an existing concept generation algorithm. The proposed method is quick (about 0.1 seconds to filter out each infeasible solution) and does not require effort from the designer to divide the functional model into single non branching chains, discard the infeasible component solutions, and aggregate the non- branching chains from the results to get the concept solutions. A case study example of concept generation for a peanut-sheller device is also presented in this thesis to demonstrate the effectiveness of the proposed algorithm. The algorithm eliminated a total of 33 infeasible solutions to the function pairs for the peanut-sheller example. Although the percentage of infeasible solutions eliminated is low compared to the actual number of concepts generated, using more explicit functions in the functional model may enable more infeasible component solutions to be filtered from the result matrix. An example and a sketch of a concept generated from the proposed algorithm are also presented.

Future improvements in the current concept generation software could be to include the code of the presented approach for complete concept generation. It is important to note that the output of the concept generator algorithm can lead to thousands of viable concept solutions, depending on the size of the input functional model, FCM, and DSM. This makes ranking concepts necessary to highlight a more manageable set of the best concepts. A better technique to rank the complete concept solutions needs to be addressed in order to present the designer with a set of manageable *good* concept designs. In the future, attributes like reliability, cost, manufacturability and efficiency may be used to identify promising concept variants from the complete design space.

Biblography

[1] Woodehouse, A., Grierson, H., Ion, W.J., Juster, N., Lynn, A. and Stone, A.L., 2004, "Tikiwiki: A Tool to Support Engineering Design Students in Concept Generation," *International Engineering and Product Design Education Conference*, Delft, the Netherlands, September 2-3 2004.

[2] Prasad, B., 1998, "Review of QFD and Related Deployment Techniques," *Journal of Manufacturing Systems*, 17(3): 221–235.

[3] Feng, C.X., Li, P.G., and Liang, M., 2001, "Fuzzy Mapping of Requirements onto Functions in Detail Design," *Computer-Aided Design*, 33: 425–437.

[4] Simpson, T.W., Bauer, M.D., Allen, J.K., and Mistree, F., 1995, "Implementation of DFA in Conceptual and Embodiment Design Using Decision Support Problems," *Proceedings of the 1995 ASME Design Engineering Technical Conferences*, DE,Vol. 82, pp. 119-126.

[5] Thornton, A.C. and Johnson, A.L., 1996, "CADET: A Software Support Tool for Constraint Processes in Embodiment Design," *Research in Engineering in Design*, 8(1): 1–13.

[6] Ishii, K., Adler, R., and Barkan, P., 1988, "Application of Design Compatibility Analysis to Simultaneous Engineering," *Artificial Intelligence for Engineering Design and Manufacture (AIEDAM)*, 2(1): 53–65.

[7] Fox, E.P., 1994, "The Pratt and Whitney Probabilistic Design System," *35th AIAA/ ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, AIAA-94-1442-CP, April.

[8] Yan, W., Chen, C.H., Shieh M.D., 2006, "Product concept generation and selection using sorting technique and fuzzy c-means algorithm", *Computers & Industrial Engineering*, 50(3): 273-285

[9] Bryant, C.R., McAdams, D.A., Stone, R.B, Kurtoglu T., and Campbell C.I., 2005, "A Computational Technique for Concept Generation", *Proceedings of DETC05*, DETC2005-85323, Long Beach, CA.

[10] Li, W. and Jin, Y., 2005," Automated Concept Generation: A Co-Evolutionary Approach", *Proceedings of IDETC/CIE2005*, DETC2005-85125, Long Beach, California, USA.

[11] Pahl, G. and Beitz, W., 1988, "*Engineering Design: A Systematic Approach*", Springer-Verlag.

[12] Ulrich, K. and Eppinger, S., 1995, *Product Design and Development*, McGraw-Hill.

[13] Otto, K. and Wood, K., 2001, *Product Design: Techniques in Reverse Engineering and New Product Development*, Prentice- Hall.

[14] Hubka, V. and Eder, W.E., 1984, *Theory of Technical Systems, Springer-Verlag*, Berlin.

[15] Rohrbach, B., 1969. "Kreativ nach Regeln – Methode 635, eine Neue Technik zum Lösen von Problemen," *Absatzwirtschaft*, 12: 73–75.

[16] Shah, J. J., 1998, "Experimental Investigation of Progressive Idea Generation Techniques in Engineering Design," *Proceedings of the DETC'98,* 1998 ASME Design Engineering Technical Conferences, DETC98/DTM-5676, Atlanta, GA.

[17] McAdams, D. and Wood, K., 2000. "Quantitative Measures for Design By Analogy," DETC2000/DTM-14562, *Proceedings of DETC2000*, Balitmore, MD.

[18] Altshuller, G., 1984, *Creativity As An Exact Science,* Gorden and Breach, Luxembourg.

[19] Benami, O. and Jin, Y., 2000, "An e-documenting approach to conceptual design", *Proceedings of ASME 2000 Design Theory and Methodology Conference*, DETC2000/CIE-14649, Baltimore, MA.

[20] Cross, N., Christianns, H. and Dorst, K., 1997, *Analysing Design Activity*, John wiley & Sons, New York, NY.

[21] Chuslip, P. and Jin, Y., 2004 "Cognitive Modeling of Iteration in Conceptual Design," *Proceedings of ASME 2004 Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, DETC2004-57521, Salt Lake City, UT.

[22] Zwicky, F., 1969, *Discovery, Invention, Research - Through the Morphological Approach,* The Macmillian Company, Toronto.

[23] Strawbridge, Z., McAdams, D. A. and Stone, R. B., 2002, "A Computational Approach to Conceptual Design," *Proceedings of DETC2002*, DETC02/DTM-34001, Montreal, Canada.

[24] Sridharan P., Campbell M.I., 2005, "A Study on the Grammatical Construction of Function Structures," *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing*, 19(3): 139–160.

[25] Campbell, M.I, Cagan, J., and Kotovsky, K., 2000, "Agent-Based Synthesis of Electromechanical Design Configurations*", Journal of Mechanical Design*, 122(1): 61 – 69.

[26] Vucovich, J., Bhardwaj, N, Hoi-Hei (Terence) Ho, Ramakrishna, M., Thakur, M., and Stone, R., "Concept Generation Algorithms for Repository-Based Early Design", *Proceedings of IDETC/CIE2006*, DETC2006-99466, Philadelphia, PA.

[27] Jin, Y and Li, W., 2007, "Design Concept Generation: A Hierarchical Coevolutionary Approach", *Journal of Mechanical Design*, 129(10): 1012 – 1022.

[28] Stone, R. and Wood, K., 1999, "Development of a Functional Basis for Design," *Proceedings of DETC99*, DETC99/ DTM-8765, Las Vegas, NV.

[29] Hundal, M., 1990, "A Systematic Method for Developing Function Structures, Solutions and Concept Variants," *Mechanism and Machine Theory*, 25(3): 243-256.

[30] Koch, P., Peplinski, J., Allen, J. and Mistree, F., 1994, "A Method for Design Using Available Assets: Identifying a Feasible System Configuration," *Behavioral Science*, 30: 229-250.

[31] Malmqvist, J., Axelsson, R., and Johansson, M., 1996, "A Comparative Analysis of the Theory of Inventive Problem Solving and the Systematic Approach of Pahl and Beitz," *Proceedings of the 1996 ASME Design Engineering Technical Conferences*, 96-DETC/DTM-1529, Irvine, CA.

[32] Murdock, J., Szykman, S. and Sriram, R., 1997, "An Information Modeling Framework to Support Design Databases and Repositories," *Proceedings of DETC'97*, DETC97/DFM- 4373, Sacramento, CA.

[33] Szykman, S., Racz, J., and Sriram, R., 1999, "The Representation of Function in Computer-Based Design," *Proceedings of DETC99*, DETC99/DTM-8742, Las Vegas, NV.

[34] Hirtz, J., Stone, R., McAdams, D., Szykman, S. and Wood, K., 2002, "A Functional Basis for Engineering Design: Reconciling and Evolving Previous Efforts," *Research in Engineering Design*, 13(2): 65-82.

[35] Bryant, C.R, Bohm, M., Stone, R.B., and McAdams, D.A., 2007, "An Interactive Morphological Matrix Computational Design Tool: A Hybrid of Two Methods", *Proceedings of IDETC/CIE2007*, DETC2007-35583, Las Vegas, Nevada, USA.

[36] Kurtoglu, T., Campbell, M.I., Bryant, C.R., Stone, R.B., McAdams, D.A., 2005, "Deriving a Component Basis for Computational Functional Synthesis," *Proceedings of*

International Conference on Engineering Design, ICED05, August 15-18, Melbourne, Australia.

[37] Kurtoglu, T., Campbell, M.I., Bryant, C.R., Stone, R.B., 2009, "A Component Taxonomy as a Framework for Computational Design Synthesis," *Journal of Computer and Information Science in Engineering*, 9(10): 01107.1 – 01107.10.

[38] Stone, R. and Wood, K., 2000, "Development of a Functional Basis for Design," *Journal of Mechanical Design*, 122(4): 359–370.

[39] Kurfman, M.A., Stone, R.B., Rajan, J.R., Wood, K.L., 2001, "Functional Modeling Experimental Studies," *Proceedings of ASME DETC and CIE Conferences,* DETC2001/DTM-21709, Pittsburgh, PA.

[40] Bohm, M. and Stone, R., 2003, "Refining Design Repositories: Creating a Usable Framework with XML Data Representation," *Proceedings of the 2003 NSF Grantees Conference*, Birmingham, AL.

[41] Bohm, M., and Stone, R., 2004, "Representing Functionality to Support Reuse: Conceptual and Supporting Functions," *Proceedings of DETC'04*, DETC2004-57693, Salt Lake City, UT.

[42] Bohm, M., Stone, R. and Szykman, S., 2005, "Enhancing Virtual Product Representations for Advanced Design Repository Systems," *Journal of Computer Information Science in Engineering*, 5(4): 360–372.

[43] Linsey, J.S., Green, M.G., Murphy, J.T., and Wood, K.L., 2005, "Collaborating To Success: An Experimental Study of Group Idea Generation Techniques" *Proceedings of the 2005 ASME International Design Engineering and Technical Conferences*", DETC2005-85351, Long Beach, CA.

Appendix

A Matlab code for eliminating infeasible solutions

```
%% Concept generator by Abhinav Choudhary
%% akc151@psu.edu
clc
clear
%% Read the FFM, CCM, FCM from excel file
% intialize the FFM
[FFM,FFM_mat] = xlsread('testdata.xlsx', 'FFM');
% initialize the FCM
[FCM,FCM_mat]= xlsread('testdata.xlsx', 'FCM');
% intitalize the CCM
[CCM,CCM_mat] = xlsread('testdata.xlsx', 'CCM');

%% Algorithm Dr. Bryant et al. to create initial result matrix

%Filter the component solutions for each function pair using Dr.
Bryant's
%algorithm
for i = 1:length(FFM)
    for j= 1:length(FFM)
        if FFM(i,j) == 1
            MM{i,j} = (transpose(FCM(i,:))*FCM(j,:)).*CCM;
        else
            MM{i,j} = zeros(length(CCM),length(CCM));
        end
    end
end
MM_cari = MM;

% Create Initial Result matrix by joining all the MM_cari cells
XY = [];
for i=1:length(MM_cari)
    for j = 1:length(MM_cari)
        XY = [XY MM_cari{i,j}];
    end
    XY2{i,:} = XY;
    XY = [];
end

MM_mat = [];
for i=1:length(XY2)
    MM_mat = [MM_mat;XY2{i,1}];
end

%MM_final_cari is the intial result matrix
MM_final_cari = MM_mat;
```

```
%% PROPOSED ALGORITHM
% Set flag for looping algorithm
flag = 0;
MM_final = zeros(length(FFM)*length(CCM),length(FFM)*length(CCM));

%% Start the loop
while flag == 0

    %% Elimination of Infeasible components of Type A and Type B

    for i = 1:length(FFM)
        for j = 1:length(FFM)
            if FFM(i,j)==1
                for k=1:length(CCM)
                    if sum(MM{i,j}(k,:))> 0
                        row_sort{i,j}(k,1)=1;
                    else
                        row_sort{i,j}(k,1)=0;
                    end
                    if sum(MM{i,j}(:,k))> 0
                        col_sort{i,j}(1,k)=1;
                    else
                        col_sort{i,j}(1,k)=0;
                    end
                end
            else
                row_sort{i,j}= zeros(length(CCM),1);
                col_sort{i,j}=zeros(1,length(CCM));
            end
        end
    end
    CM = [];
    for i=1:length(row_sort)
        for j = 1:length(row_sort)
            CM = [CM row_sort{i,j}];
        end
        CM_rowsum{i,:} = [CM];
        CM = [];
    end

    CM = [];
    for i=1:length(col_sort)
        for j = 1:length(col_sort)
            CM = [CM;col_sort{j,i}];
        end
        CM_colsum{:,i} = [CM];
        CM = [];
    end

    row_selector=[];
    for i = 1:length(CM_rowsum)
        for j = 1:length(CCM)
            if FFM(i,:) == CM_rowsum{i,1}(j,:)
```

```
            A{i,:}(j,:)=1;
        else
            A{i,:}(j,:)=0;
        end
    end
    row_selector = [row_selector; A{i,:}];
end

col_selector=[];
for i = 1:length(CM_colsum)
    for j = 1:length(CCM)
        if FFM(:,i) == CM_colsum{1,i}(:,j)
            A1{:,i}(:,j)=1;
        else
            A1{:,i}(:,j)=0;
        end
    end
    col_selector = [col_selector A1{:,i}];
end

for i = 1:length(A)
    for j = 1:length(A1)
        eliminator{i,j} = A{i,:}*A1{:,j};
        MM{i,j} = MM{i,j}.*eliminator{i,j};
    end
end


% Note: CM_rowsum is 1xn cell & CM_colsum is nx1 cell
% Note: row_selector is 1xn matrix & col_selector is nx1 martix
% A & A1 are dummy variables

%% Elimination of Infeasible components of Type C and Type D

for i = 1:length(FFM)
    for j = 1:length(FFM)
        if FFM(i,j)==1
            for k=1:length(CCM)
                if sum(MM{i,j}(k,:))> 0
                    row_sort_1{i,j}(k,1)=1;
                else
                    row_sort_1{i,j}(k,1)=0;
                end
                if sum(MM{i,j}(:,k))> 0
                    col_sort_1{i,j}(1,k)=1;
                else
                    col_sort_1{i,j}(1,k)=0;
                end
            end
        else
            row_sort_1{i,j}= zeros(length(CCM),1);
            col_sort_1{i,j}=zeros(1,length(CCM));
        end
    end
```

```
end

CM = [];
for i=1:length(row_sort_1)
    for j = 1:length(row_sort_1)
        CM = [CM row_sort_1{i,j}];
    end
    CM_rowsum_1{i,:} = [CM];
    CM = [];
end

CM = [];
for i=1:length(col_sort_1)
    for j = 1:length(col_sort_1)
        CM = [CM;col_sort_1{j,i}];
    end
    CM_colsum_1{:,i} = [CM];
    CM = [];
end

for i = 1:length(CM_rowsum_1)
    for k=1:length(CCM)
        if sum(CM_rowsum_1{i,1}(k,:))> 0
            row_vector{i,1}(k,1)=1;
        else
            row_vector{i,1}(k,1)=0;
        end
    end
end

for i = 1:length(CM_colsum_1)
    for k=1:length(CCM)
        if sum(CM_colsum_1{1,i}(:,k))> 0
            col_vector{1,i}(1,k)=1;
        else
            col_vector{1,i}(1,k)=0;
        end
    end
end

for i =1:length(row_vector)
    sig_vector{i,1} = row_vector{i,1}.*transpose(col_vector{1,i});
end


for i = 1:length(FFM)
    if FFM(:,i) == zeros(length(FFM),1)
        sig_vector{i,1}=ones(length(CCM),1);
    end
end

for i = 1:length(MM)
    for j = 1:length(MM)
```

```
            for k = 1:length(CCM)
                MM{i,j}(k,:) = MM{i,j}(k,:).*sig_vector{i,:}(k,:);
            end
        end
    end

    non_solvable_function = [];
    for i = 1:length(MM)
        for j = 1:length(MM)
            if FFM(i,j) ==1
                if MM{i,j} == zeros(length(CCM),length(CCM));
                    flag =2;
                    non_solvable_function = [i j];
                end
            end
        end
    end

    %% Create final result matrix by joining all the MM cells
    XY = [];
    for i=1:length(MM)
        for j = 1:length(MM)
            XY = [XY MM{i,j}];
        end
        XY2{i,:} = XY;
        XY = [];
    end

    MM_mat = [];
    for i=1:length(XY2)
        MM_mat = [MM_mat;XY2{i,1}];
    end

    if MM_final == MM_mat
        flag = 1;
    else
        MM_final = MM_mat;
    end
end
%% Write the final matrix results into excel file

% Write the text data (names of functions and components)
k=2;
for i = 2:length(FFM_mat)
    for j = 2:length(CCM_mat)
        RM{1,k} = FFM_mat{1,i};
        RM{2,k} = CCM_mat{1,j};
        k=k+1;
    end
end
shift = [{''};{''}];
RM = [shift RM];
RM{1,2}='';
RM{2,2}='';
```

```
RM_row_write = xlswrite('Results.xlsx', RM, 'Results_final');
RM_col_write = xlswrite('Results.xlsx', transpose(RM),
'Results_final');

RM_ini_row_write = xlswrite('Results.xlsx', RM, 'Results_initial');
RM_ini_col_write = xlswrite('Results.xlsx', transpose(RM),
'Results_initial');

% Write the initial and final result matrices (numeric data)
MM_cari_write = xlswrite('Results.xlsx', MM_final_cari,
'Results_initial','C3');
MM_final_write = xlswrite('Results.xlsx', MM_final,
'Results_final','C3');
```