

Carnegie Mellon University
CARNEGIE INSTITUTE OF TECHNOLOGY

THESIS

Submitted in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy

TITLE The A-Design Invention Machine:
 A Means of Automating and Investigating Conceptual Design

PRESENTED BY Matthew Ira Campbell

ACCEPTED BY The Department of Mechanical Engineering

_____ Major Professor _____ Date

_____ Major Professor _____ Date

_____ Department Head _____ Date

APPROVED BY THE COLLEGE COUNCIL

_____ Dean _____ Date

Carnegie Mellon University
CARNEGIE INSTITUTE OF TECHNOLOGY

THE A-DESIGN INVENTION MACHINE:
A MEANS OF AUTOMATING AND INVESTIGATING CONCEPTUAL DESIGN

A Dissertation Submitted to the Carnegie Institute of Technology
in Partial Fulfillment of the Requirements for the degree of
Doctor of Philosophy in Mechanical Engineering

by

Matthew Ira Campbell

July 2000

THE A-DESIGN INVENTION MACHINE:
A MEANS OF AUTOMATING AND INVESTIGATING CONCEPTUAL DESIGN

Copyright © 2000

Matthew Ira Campbell

Acknowledgments

There are so many people to acknowledge in the development of this research. First and foremost, I would like to thank my thesis advisors. Prof. Jon Cagan has been a constant inspiration to me in the past nine years of our acquaintance. Jon introduced me to design research and allowed me to develop ideas on my own while working on real engineering issues. Without his encouragement, vision, attention to quality, and spirit of collaboration, this work would never have developed as it did. Also, thanks to Prof. Ken Kotovsky who has been much more than a co-advisor to me. He introduced me to a wide range of different research ideas, and ways of thinking. Because of their support, I am sure to remember these graduate years as an inspiring, and enjoyable challenge. I would also like to thank Prof. Gary Fedder, Prof. Tom Stahovich, and Prof. Kenji Shimada, who have over the years discussed with me, challenged me, and guided me to become a better researcher. I am pleased that they make up my thesis committee.

My fellow peers at Carnegie Mellon University have provided so much support, inspiration, and encouragement for this work. I only hope that I have provided the same for their endeavors. Special thanks to my fellow coworkers in the Computational Design Lab: Manish Agarwal, Shashvat Prakash, Su Yin, Jay McCormack, and Saurabh

Deshpande. Also, special thanks to Jarrod Moss who was much more than an undergraduate helper, and to Vishal Gupta for assistance with MEMS simulations.

I also owe a debt of gratitude to many other important people in my life. Special thanks to my parents and family members who supported me emotionally, and spiritually throughout these years at Carnegie Mellon. Also, my dear friend, Melissa Ippolito, whose kindness, sensitivity, and insight kept me going. Thanks to Tom Keating for helping me sharpen my writing skills and enjoy the challenges of the English language. Numerous friends and fellow musicians have been ever so supportive and have helped me see things in perspective over the past five years. Special thanks to Bob Altman, Pete Cramer, Dave Weinstein, Ray Chick, Christine Carnevali, Megan Williams, Heidi Kohne, and Don Roehlich.

Finally, I would like to acknowledge the financial support of the National Science Foundation under grants EID-9256665 and DDM-9258090, and the Defense Advanced Research Projects Agency under agreement number F30602-96-2-0304¹.

¹ The research effort was partially sponsored by the Defense Advanced Research Projects Agency (DARPA) and Rome Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-96-2-0304. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA, Rome Laboratory, or the U.S. Government.

Abstract

The A-Design approach to automated design has been developed for a general class of conceptual design problems. The methodology is founded on the notion that engineering design occurs in interaction with an ever-changing environment and therefore computer tools developed to aid the design process should be adaptive to these changes. A-Design invents solutions to open-ended design problems through the interactions of a multitude of agents folded into a stochastic iterative process capable of adapting to changes in user preference.

The motivation for A-Design is to integrate qualities of the human design process into a computational algorithm. This has been accomplished by creating four subsystems that each embody a different characteristic of human conceptual design: an open-ended representation of the design problem, an adaptability to changes in problem specifications, a collaborative involvement of different ideas and preferences, and an iterative yet guided search for successful solutions.

A design problem is presented to A-Design by a description of the desired functionality in the form of system inputs and outputs, a set of objectives to be optimized, and a library of electromechanical components. Two test problems (an electromechanical weighing machine and a MEMS accelerometer) test A-Design's ability to invent novel

configurations. Several experimental results validate the development of A-Design as a successful model of human design process characteristics. The results show that A-Design is independently capable of invention, has the ability to aid the human designer in developing new conceptual designs, and provides an experimental framework to model how to achieve conceptual design as well as how humans might achieve conceptual design.

Table of Contents

Acknowledgments.....	ii
Abstract.....	iv
Table of Contents	vi
List of Figures.....	ix
List of Tables.....	xiii
Chapter 1 Introduction.....	1
1.1 MOTIVATION.....	3
1.2 CONCEPTUAL DESIGN AS SEARCH.....	5
1.2.1 Representation.....	7
1.2.2 Generation.....	9
1.2.3 Evaluation.....	10
1.2.4 Guidance.....	10
1.3 OVERVIEW	11
1.4 ORGANIZATION	13
1.5 THESIS STATEMENT.....	14
Chapter 2 Iterative Search Process	15
2.1 RELATED WORK.....	16
2.2 THE A-DESIGN ITERATIVE SEARCH PROCESS.....	19
2.3 ELECTROMECHANICAL CONFIGURATION A-DESIGN	21
2.4 PURPOSE BEHIND ITERATIVE STRATEGY	25
Chapter 3 Design Selection.....	28
3.1 NOMENCLATURE AND ASSUMPTIONS	29
3.2 RELATED WORK.....	33
3.3 DIVISION OF DESIGN SOLUTIONS	34
3.4 ADAPTING TO USER PREFERENCE	40
Chapter 4 Design Representation	44
4.1 RELATED WORK.....	46
4.2 BASIC COMPONENTS	48

4.2.1	<i>Functional Parameter (FP)</i>	49
4.2.2	<i>Embodiment (EB)</i>	52
4.3	UPDATING DESIGNS: HANDLING INCOMPLETE CONFIGURATIONS.....	55
4.4	EXTRACTING EQUATIONS.....	59
4.5	CATALOG OF COMPONENTS.....	61
4.6	SYSTEM CONFIGURATION	63
4.7	COMPARISON WITH PREVIOUS METHODS.....	63
Chapter 5	Agents Architecture	68
5.1	RELATED WORK.....	69
5.2	THE A-DESIGN AGENT.....	70
5.2.1	<i>Definition of Agent</i>	72
5.2.2	<i>Utility agent model</i>	75
5.3	MAKER-AGENTS	76
5.3.1	<i>Configuration-Agents (C-Agents)</i>	76
5.3.2	<i>Instantiation-Agents (I-Agents)</i>	82
5.4	MODIFICATION-AGENTS	83
5.4.1	<i>Fragmentation-agents (F-Agents)</i>	83
5.5	COLLABORATION AMONG AGENTS	84
Chapter 6	The Manager-Agent: Guidance for Iterative Search.....	86
6.1	RELATED WORK.....	87
6.2	BASIC OPERATION.....	89
6.3	FINDING TRENDS.....	91
6.4	STOCHASTIC EXECUTION OF AGENTS.....	94
Chapter 7	Preliminary Test Problems	98
7.1	MANHATTAN TRANSFER	98
7.2	NUMERICAL OPTIMIZATION	101
Chapter 8	Electromechanical Test Cases.....	104
8.1	WEIGHING MACHINE.....	104
8.1.1	<i>Results Under the Influence of TODO and TABOO Learning</i>	107
8.1.2	<i>Discussion of Weighing Machine Results</i>	109
8.2	MEMS ACCELEROMETER.....	113
8.2.1	<i>Description of design problem</i>	114
8.2.2	<i>Evaluation</i>	117
8.2.3	<i>Results</i>	119
8.2.4	<i>Discussion of Accelerometer Results</i>	124
8.3	COMPARISON OF TWO DESIGN PROBLEMS	128
Chapter 9	Experimental Results.....	131
9.1	SIGNIFICANCE OF RANDOM STARTING POINTS.....	131
9.2	ADAPTABILITY EXPERIMENTS.....	133
9.2.1	<i>Good Set and Pareto Set Isolation</i>	134
9.2.2	<i>Changing User Preference</i>	136

9.2.3	<i>Adding and Removing Objectives</i>	139
9.3	LEARNING EXPERIMENTS	142
Chapter 10	Conclusions	147
10.1	SUMMARY	147
10.2	DISCUSSION.....	149
10.2.1	<i>Automating Conceptual Design: Theoretical Claims of A-Design</i>	150
10.2.1.1	Iteratively-Guided Search Process	150
10.2.1.2	Adaptive Design Selection	152
10.2.1.3	Open-Ended Representation.....	153
10.2.1.4	Collaborative Agents.....	154
10.2.2	<i>Investigating Conceptual Design</i>	154
10.2.3	<i>Uniting Automated Design and Automated Analysis</i>	155
Chapter 11	Coda	159
11.1	CONTRIBUTIONS.....	159
11.2	FUTURE WORK.....	160
11.3	CONCLUDING REMARKS.....	162
References	163
Appendix: Implementation	171
A.	GENERAL A-DESIGN PROCEDURE	171
B.	DIRECTORY STRUCTURE.....	174
C.	INPUT FILES.....	176
D.	SYNOPSIS OF PROGRAM FILES	186
E.	OUTPUT FILES.....	188

List of Figures

Figure 1.1: (a) Represents the space of possible body weight scales, (b) represents this space in context with the larger class of weighing machines.....	6
Figure 1.2: (a) Weighing machine design problem. Design an electromechanical system that converts a specimen's weight (Input) to a specified dial displacement (Output). (b) This representation of function isolates a section of the search space.	8
Figure 1.3: View of search space as an intermediate step between the process of creating design states and evaluating the utility of such states.	9
Figure 1.4: Generalized view of the four challenges present in modeling conceptual design as search and how the four subsystems of A-Design overcome these challenges with human design characteristics.....	12
Figure 2.1: General flowchart of A-Design process.	19
Figure 2.2: Detailed flowchart for electromechanical configuration design.....	22
Figure 3.1: An example of various terms used throughout this chapter.	32
Figure 3.2: a) Two-dimensional plot of designs depicting Pareto-optimal set for two objectives, b) the performance profile method of comparing of designs states.....	35
Figure 3.3: Updating Pareto surface while considering user preference.....	37
Figure 3.4: Bulb of Good populations move with user preference.	38
Figure 3.5: A display of the dialog that occurs between the user and the system in adapting to user preference.	41
Figure 3.6: Determining the new weights is a process of solving the linear system of equations that is created by the cardinal ranking of various Pareto design states.....	43

Figure 4.1: The development of this representation is related to the contributions of various previous research projects.	47
Figure 4.2: The contents of the FP structure.	49
Figure 4.3: Weighing machine with input and outputs described as Functional Parameters.	52
Figure 4.4: The contents of the EB structure: (a) shows the general description of the slots of the EB and b) shows an example of a gear EB.	54
Figure 4.5: Pseudo-code for updating partial design state.	57
Figure 4.6: a) Partial design state with update FPs, b) Design is completed with addition of spring.	58
Figure 4.7: Gear Embodiment with sample of gear component file: gear.comps.	62
Figure 4.8: The contents of the SC structure.	63
Figure 4.9: Views of the conceptual design process : a) the view established in Welch and Ulrich, b) the A-Design view.	65
Figure 5.1: The interaction of the agents occurs with the designs and the user.	72
Figure 5.2: An agent is defined by its sensors, effectors and judgments.	73
Figure 5.3: Utility agent tests all possible actions before performing the action with the highest utility.	75
Figure 5.4: The C-agent's perception of an incomplete design state.	77
Figure 5.5: Pseudo-code for Configuration-agent.	80
Figure 6.1: Pseudo-code for Manager-agent.	90
Figure 6.2: An example of detecting TODO and TABOO trends from a set of 10 designs (A through J). (a) The approximation of the user's utility allows designs to be collapsed a single metric. (b) The sorted list is then divided into elements for TODO and TABOO comparison. (c) The top five TODO designs are intersected exhaustively for all possible combinations.	92
Figure 6.3: Manager-agent proportions probabilities based on evaluation function.	95

Figure 7.1: A solution to the Manhattan Transfer problem generated by the A-Design process.....	100
Figure 7.2: The two mathematical functions (a) f_1 and (b) f_2 used to test the optimizing power of A-Design.....	102
Figure 8.1: Description of weighing machine design problem as posed to A-Design....	105
Figure 8.2: Three different alternatives created by the A-Design process. Design (a) is found by an equal preference for the four design objectives, whereas designs (b) and (c) are found by placing more importance on minimizing input displacement.	106
Figure 8.3: Two additional weighing machines found under the influence of TODO and TABOO learning.....	108
Figure 8.4: Performance plot of five designs from Figures 8.2 and 8.3.....	109
Figure 8.5: There is some variability for the position of ports on an EB. For the gear, this variability is the variable, ϕ , which relates the port A of the shaft hole to port B, the gear teeth interface point. This point can be anywhere on the radius of the gear... ..	112
Figure 8.6: Configuration view of ADXL accelerometer.....	115
Figure 8.7: Description of accelerometer design problem as posed to A-Design.....	117
Figure 8.8: The additional computational demands of the evaluation process requires that external code be developed which is capable of further executing the SABER dynamic simulation.....	118
Figure 8.9: Accelerometer A is created by the A-Design process with no learning.....	121
Figure 8.10: Accelerometer B is created by the A-Design process under the presence of TODO and TABOO learning.....	122
Figure 8.11: Accelerometer C is created by initiating the TODO list with a U-Spring subsystem.....	123
Figure 8.12: ADXL-style accelerometer optimized using approach from Mukherjee et al. (1999).....	125
Figure 8.13: Performance plot of the three A-Design accelerometers and the ADXL-style accelerometer according to (a) SPECTRE analysis done by hand, and (b) heuristic based automated analysis.....	126

Figure 9.1: Comparison of random vs. set initial design populations a) best design at each iteration for 20 runs, b) average of best design from 20 runs.	132
Figure 9.2: Comparison of best designs from Pareto only, Good only and both.	134
Figure 9.3: Adaptability of A-Design: a) the change in preferences is viewed on two different plots to see the effect, b) flowchart of adaptability experiment, c) best designs viewed by W1, d) best designs viewed by W2.	137
Figure 9.4: The addition and removal of an objective to further test the adaptability of A-Design.....	140
Figure 9.5: A comparison of TODO, TABOO and no learning.....	143
Figure 9.6: A comparison of three learning techniques that involve a combination of TODO and TABOO.	144
Figure 9.7: Performance plots for learning techniques shown in Figure 8.5 and Figure 8.6.	146

List of Tables

Table 4.1: Through and Across Variables for Each Domain	51
Table 4.2: Current Embodiments implemented in the system	62
Table 5.1: Different preferences that exist in the current set of C-, I- and F-agents.....	78
Table 10.1: Derivation and innovations of the A-Design theory and the electromechanical implementation.....	148

Chapter 1

Introduction

In the latter half of the twentieth century, the computer's development into a powerful analysis tool revolutionized engineering practice and education. Computational tools in engineering provide new ways of visualizing systems (Computer-Aided Design), analyzing components (finite-element analysis) and even automating the manufacture of devices (Computer Numerical Control Machining). Perhaps the computer's biggest impact on engineering is the ability to design and test artifacts on a computational testbed prior to constructing prototypes. Such computational analysis can reduce design costs and cycle times by predicting difficulties early in the design process. While numerous computational applications have helped the engineering design process through automating complex analyses, few applications attend to the actual design synthesis.

Beyond the development of 'computer as analyzer', lies a lesser-known and more challenging development of 'computer as designer'. While the computer has a definite advantage over humans in calculation speed and accuracy, many believe it lacks the faculties to make informed and intuitive decisions, and thus is incapable of embodying the creative process needed for true design and invention. However, this perspective has

slowly changed with advances in design automation. With the establishment of state-space search (originally formulated as a cognitive model of human problem solving by Newell and Simon, 1972; Simon, 1969), computational design has progressed from solving well-behaved mathematical problems towards addressing more ambitious engineering design problems. The slower development of computational design can be attributed to three challenges unique to design:

- First, characteristics of the human design process (hereafter referred to as “human design”) have yet to be realized in an implemented process. While research in automated design involves static problems, real design is not static. Human creativity is capable of developing novel artifacts through adapting to difficulties and challenging past conventions. This dissertation’s main goal is to establish a new theory for automated computational design that incorporates characteristics of human conceptual designing, thereby broadening the applicability of computers in engineering.
- Second, unlike analysis, which has a history of mathematical formalism at its foundation, engineering design has yet to be studied with the same rigor. Studies in cognitive psychology may provide an understanding of the underlying principles inherent to conceptual design. Such studies would provide the understanding needed to formalize design as a computational endeavor. One of this dissertation’s intentions is to demonstrate the need for more cognitive science studies of the design process to increase our understanding of both human and computational design.

- Third, design requires the comparison of various alternatives. Such comparison is best performed by evaluating solutions on a common metric. Depending on the complexity of the design problem, this comparison may require detailed analysis to occur as a subset of design. The challenge lies in developing a computational search process to automatically invoke analysis both reliably and efficiently. However due to the numerous design alternatives being compared, automated analysis must be structured to allow for a quick evaluation of alternatives. As a result of these time constraints, the computational design researcher may need to develop heuristics around complex analysis to address the challenge of both searching for and automatically analyzing designs in real time.

This dissertation establishes a new theory for computational design known as A-Design. A-Design brings together various innovations in design theory and automation to address the early phase of conceptualization that occurs only when a design need has been established. This early design phase is currently the product of human creativity and has yet to be realized in a computational system. This research makes several strides to decoding this part of the design process by developing a computational system capable of invention for a range of design problems.

1.1 MOTIVATION

Traditionally, computational design tools have been employed downstream in the design process to improve features in an existing design. Usually, this occurs as an optimization of variables already defined within a design. However, many conceptual design decisions have yet to benefit from computational design aids prior to establishing a design concept. In this early design phase, the engineering designer is faced with a

variety of difficult decisions. While the purpose of the designed artifact is often well understood, goals and trade-offs among the goals are rather transitory. Due to new technologies and changes in marketing demands, the engineering designer needs to not only build and test new prototypes but also to reevaluate the problem specifications in an iterative procedure of adapting and fine-tuning.

Within this preliminary stage of design, computational aids could assist the designer in reducing the space of possible solutions and in establishing the specifics of a design artifact. It is difficult to envision the basis for such a computational tool, since current computational systems naturally operate on well-defined structures following well-defined algorithms, thereby making them inappropriate for the open-ended and unstructured nature of conceptual design. Furthermore, unstructured conceptual design has a much larger design space compared to the restricted space of solutions addressed by optimization techniques. Whereas optimization requires variables to be defined prior to search, conceptual design is not limited by the number of variables or set design configurations.

The design system presented in this dissertation addresses this formidable and often less investigated issue of understanding and formalizing the early conceptual phase of design. This new conceptual design theory investigates basic characteristics of human conceptual design and seeks to establish these traits in a computational system. The motivation for developing this technique is threefold: (1) to further the capacity of computation by creating an invention machine that automatically creates solutions to open-ended design problems, (2) to create a basis for future design tools that can assist

the engineering designer in the early conceptual stage of designing and (3) to model aspect of the human conceptual design process to learn more about how human design is accomplished.

1.2 CONCEPTUAL DESIGN AS SEARCH

Throughout this dissertation, the view of conceptual design is likened to the concept of search. We envision a space of design instances whereby each state within the space is a solution to a common design problem. For example, Figure 1.1a introduces the body weight scale example as the basic design problem used throughout this dissertation for both describing the methodology and for running experiments. This figure represents the space of possible solutions to the problem of measuring a person's weight. Within this space, a variety of different designs achieve this design purpose including upright beam scales, dial scales and digital scales. However, because this space of solutions is delimited only by the problem description, it also contains all future solutions to the design problem that have yet to be realized. These future designs are likely to include novel solutions that combine previous technologies in unique ways as well as designs utilizing new technologies.

Within such spaces, instances can be organized such that solutions with similar configurations are close in proximity. Therefore, divisions into different design families can be accomplished, such as dial scales and digital scales as seen in Figure 1.1a. The principle for this visualization is that designs that require little modification to transform them from one state to another are closer to each other than designs that require larger modification. Therefore, to move about this space of solutions, one makes

transformations to designs to arrive at neighboring solutions. Through numerous modifications, one can visit a wide variety of possible configurations. Because the space is infinite in its organization and includes past, present and future design states, this “searching” through the space becomes analogous to “creating,” “designing,” or “inventing” in real design problems. If this space is describable to a computational

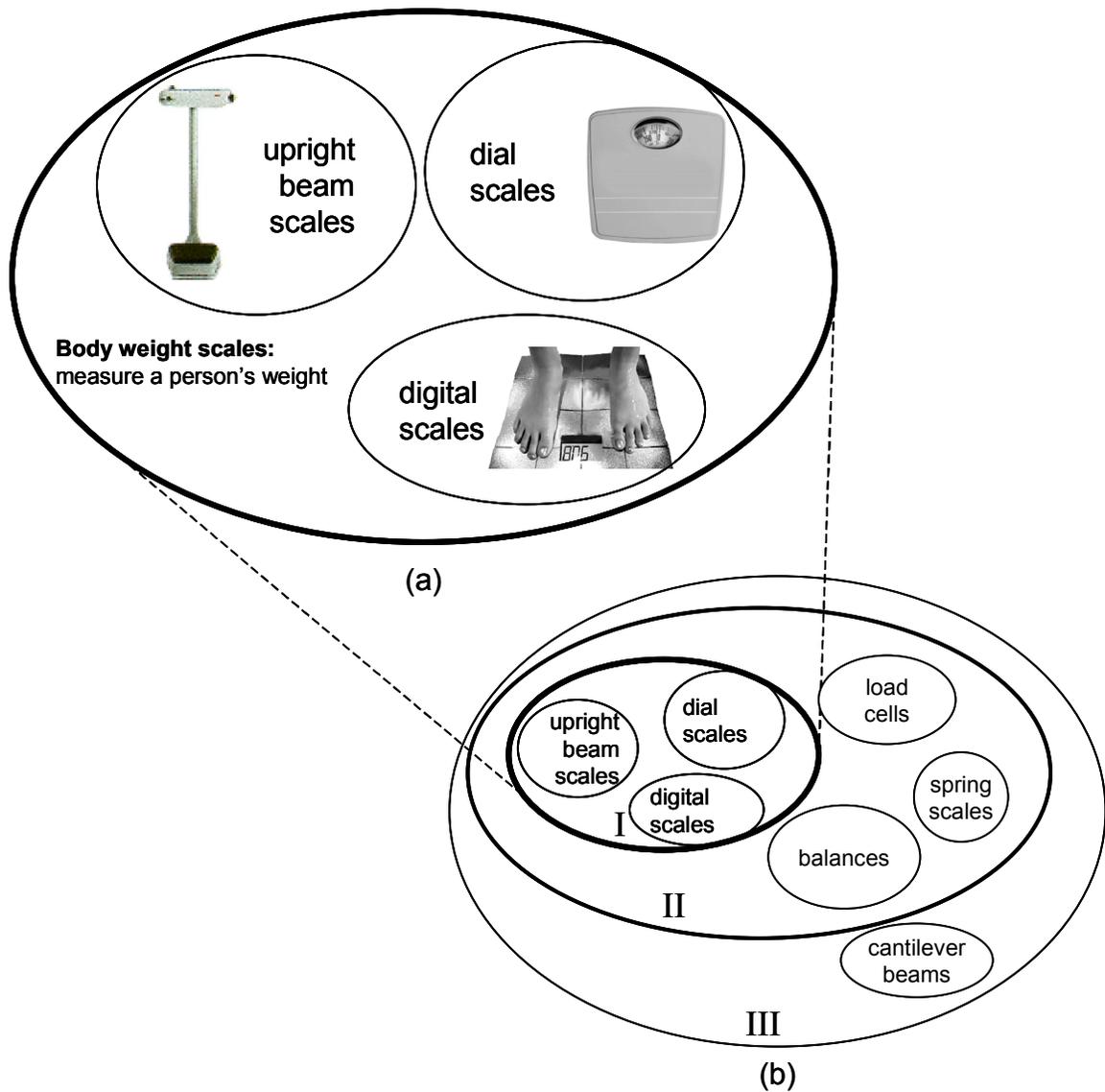


Figure 1.1: (a) Represents the space of possible body weight scales, (b) represents this space in context with the larger class of weighing machines.

system, then the challenge is to effectively find in this set the solution that best meets the demands of the design problem. However, no search process can completely capture the conceptual design spaces. There are four challenges to the development of making conceptual design a computational search process: representation, generation, evaluation and guidance. A discussion of each challenge follows.

1.2.1 Representation

The space of possible body weight scales shown in Figure 1.1a is a subset of a larger space of solutions. As seen in Figure 1.1b, the space of body weight scales (I) is a subset of the larger space (II) of weighing machines representing devices that measure the weight of an unspecified object. This space includes other families of weighing machines including load cells, spring scales and balances. This could be further extended to other supersets such as the space illustrated by III in Figure 1.1b. This space includes all devices that provide a proportional displacement as a result of a supplied downward force, such as a cantilever beam.

The purpose of this illustration is to show that this view of the search space, while somewhat arbitrary in definition and division, provides insight into the challenge of representing design concepts. In order to demarcate different types of configurations, styles and behaviors, a method of describing these concepts in a formal manner must be established. While it might be easy to delineate various design functions with natural language, such as “body weight scales include all designs that are used to measure a person’s weight”, it is not clear how to best define this for a computational system. The method used in this dissertation is one of representing functionality by describing the input and output behaviors of a design. For example in Figure 1.2a, the weighing

machine design problem is formally specified as a transformation of an input downward force to an output rotational displacement.

By mapping this representation of desired functionality onto the previously shown search spaces, only a partial set of the larger space of solutions is captured (see Figure 1.2b). The output of a dial in Figure 1.2a limits the space of possible solutions to only the set of body weight scales that use a dial at the output as well as including non-body weight scales that might fit this input and output behavior. While this representation does not provide a perfect fit to the design problem of body weight scales, it does provide a general syntax for describing design function. A variety of different design problems can be posed in this description of input and output behaviors. With the exception of the endpoints of a design, this representation method does not constrain the configuration details, thereby allowing a wealth of possible design configurations to exist in solving

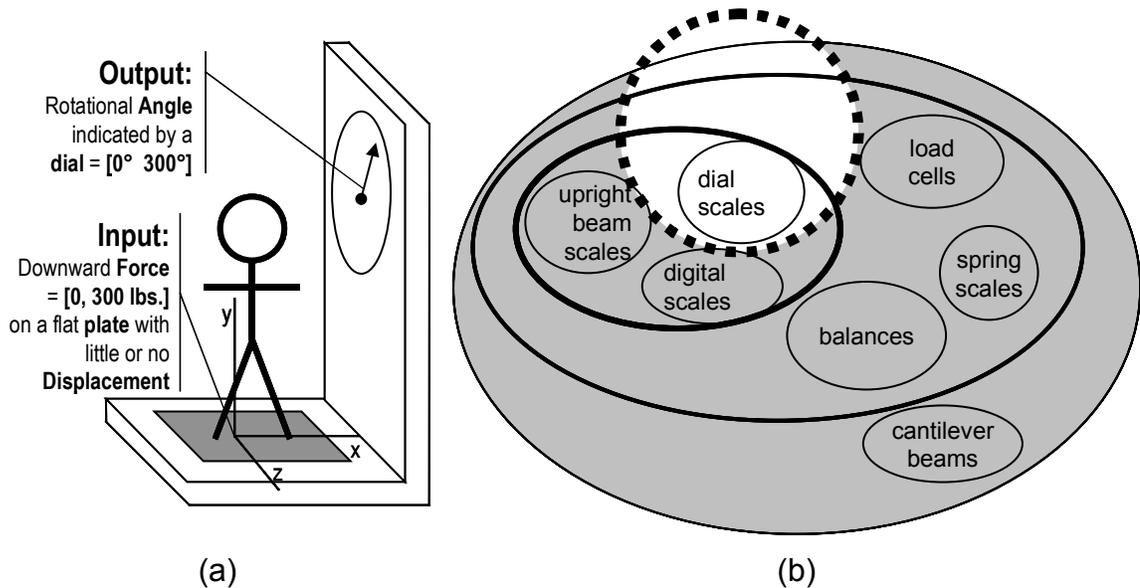


Figure 1.2: (a) Weighing machine design problem. Design an electromechanical system that converts a specimen's weight (Input) to a specified dial displacement (Output). (b) This representation of function isolates a section of the search space.

this design problem.

1.2.2 Generation

The second challenge in reducing conceptual design to search is the method for generating design concepts from a design problem description. The top of Figure 1.3 depicts the process of generating design states. The process starts with a “seed” or description of the design problem (as shown by the bolded circle at the top of the figure). All design alternatives are constructed in stages progressing from problem description to complete design instance. The result of the construction is a point in the search space shown in the center of Figure 1.3. This generation of design solutions is not a well-defined task. Ideally, the generation method should be capable of creating the wide diversity of solutions that are demarcated by the representation and should provide the means to move between design configurations in the search space.

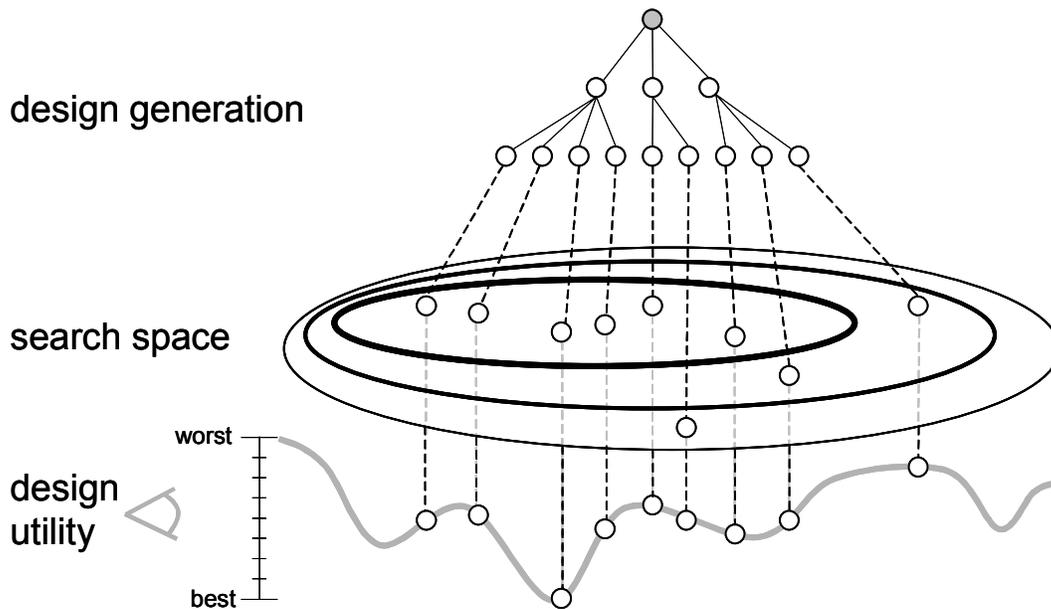


Figure 1.3: View of search space as an intermediate step between the process of creating design states and evaluating the utility of such states.

1.2.3 Evaluation

In order to direct the search process a computational system must have some knowledge of how to compare design states. While design is in essence an open-ended problem with “no right answer”, definite distinctions can be made between good and bad solutions. A metric may be constructed to articulate these distinctions as an overall value for individual design states. A numerical value, no matter how approximate, is determined for each unique design in the search process, as seen in Figure 1.3. In the bottom of the figure, the utility of each design is visualized as a numeric value on a surface of “evaluated” design states.

This utility represents an aggregation of all the attributes that characterize a design including performance metrics such as efficiency, maximum speed and power handling; market-driven metrics such as cost, durability and repairability; and consumer perceptions of the design state such as aesthetics and user-friendliness. The reduction of these many diverse attributes to a single value is further complicated by market changes, changes in one’s perception of design worth, and fluctuations due to the aggregation of many people’s view of the design. While an approximate value for such a metric can be made to effectively guide the search process, the means of actually ascertaining the user’s utility function is complex enough to be a research topic within itself.

1.2.4 Guidance

An exhaustive search for the best design is not possible due to the infinite size and complexity of the conceptual design search space. Fortunately, numerous optimization techniques have been developed which can provide a starting point in guiding the search for successful solutions. By using the utility function as the basis for comparison,

techniques can be developed to efficiently find successful solutions without a complete search of the space. However, in this work the use of the word ‘optimization’ is used with caution as this often implies the existence of some global optimum of a static objective function. The space in conceptual design envelops new technologies and processes, and the utility function is always under constant change. Therefore, unlike optimization, there is no optimal point; no utopian solution exists in these conceptual design spaces. To better understand this nihilistic statement, imagine the conceptual design search space for automobiles. This includes a space of all designs created in the last one hundred years with no clear indication of a true optimal solution. New technologies, market demands and consumer perspectives have greatly affected and will continue to affect automotive design.

Therefore while optimization is a good starting point, one must realize that in true design nothing is fixed. The search space and the utility function change constantly, thus the search for the best design is never ending. The real design process is marked by a series a “successful” design states. These successful designs, although prone to revision in the future, emerge as a result of the design process converging on solutions that appear to satisfy the current utility function well enough to diminish the need for further search.²

1.3 OVERVIEW

Thus far, four main theoretical challenges have been presented in developing an automated approach to conceptual design: **representation**, **generation**, **evaluation** and **guidance**. In overcoming these challenges A-Design looks for direction in the only

² This brief description of search spaces and utility functions is a synopsis of many diverse research efforts. While this information is gathered from many sources, it is rooted the seminal contributions of Jones, 1976; Simon, 1969; Keeney and Raiffa, 1976.

currently successful conceptual design machine: the human mind. In so doing, each of the four challenges is identified with a particular human trait that makes people capable designers. Figure 1.4 graphically depicts these areas and the human characteristics that A-Design integrates. As a result of this integration, A-Design can be viewed as having four subsystems: 1) an open-ended formulation for **representing** design states, 2) an agent

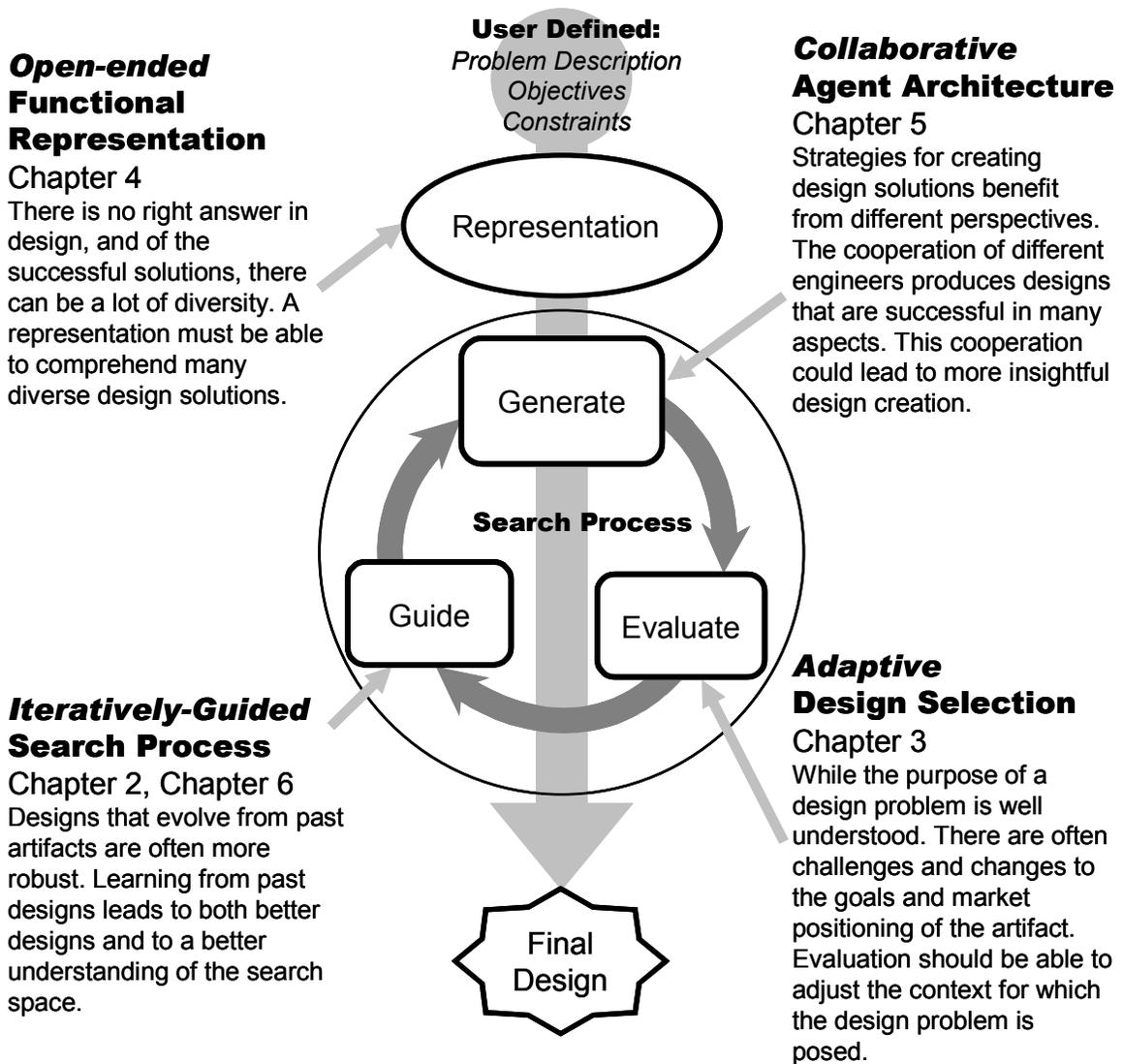


Figure 1.4: Generalized view of the four challenges present in modeling conceptual design as search and how the four subsystems of A-Design overcome these challenges with human design characteristics.

architecture that, through the collaboration of various agent-types is responsible for **generating** candidate solutions, 3) a scheme for **evaluating** multi-objective decisions to allow for an adaptive approach to meeting user preference, and 4) an iterative algorithm for **guiding** basic design concepts to successful design solutions.

Originally, A-Design was conceived as a combination of agent collaboration and adaptive design selection. However, throughout the course of bringing these two areas together, a stochastically guided iterative process and open-ended functional representation were brought in to address design problems at a more conceptual level than previously investigated by design automation.

1.4 ORGANIZATION

In this dissertation a full description of the A-Design system is presented. The early chapters present the fundamental elements of the theory and the later chapters validate the various facets of the theory through test problems and experiments.

As seen in Figure 1.4, the next five chapters are devoted to the subsystems of A-Design. Each of these chapters presents the purpose, related research projects, and details of the operations of each subsystem. Chapter 2 outlines the iterative procedure of A-Design. The description of this procedure includes a brief overview of the remaining subsystems, and thus provides an introduction to the flow of information between these subsystems. Chapter 3 then presents the method for choosing which design solutions best meet a user's utility function. Next, the method for representing designs is described in Chapter 4, followed by the method for generating and modifying designs through agent

collaboration in Chapter 5. Finally, Chapter 6 explains how learning is used to guide the iterative search for new designs.

The second half of the dissertation begins at Chapter 7 with the introduction of some preliminary test results. Two electromechanical design problems are then described in Chapter 8: the design of a weighing machine and the design of a Micro-Electromechanical accelerometer. Next, Chapter 9 validates parts of the A-Design methodology that are not easily observed in the examples of Chapter 7 and 8.

Chapter 10 provides a summary of A-Design and discussion of the theoretical claims it makes. Also, this chapter addresses conceptual design research challenges that emerge from the development of the A-Design methodology. Chapter 11 then closes this dissertation with a list of contributions and possible areas of future work.

1.5 THESIS STATEMENT

A conceptual design theory combining the adaptive, open-ended, collaborative and iteratively guided characteristics of human design automatically creates design concepts to meet a user's needs and respond to changes in those needs.

Chapter 2

Iterative Search

Process

Main Entry: **it·er·a·tive**³
Pronunciation: 'i-t&- "rA-tiv, -r&-
Function: *adjective*
Date: 15th century
: involving repetition: as **a** : expressing repetition of a verbal action **b** : relating to iteration of an operation or procedure
- **it·er·a·tive·ly** *adverb*

While many impressive innovations in computer science seem to mimic human behavior, the underlying operations of human intelligence and artificial intelligence are often quite different. The speed at which the computer performs routine mathematical tasks provides a basis for complex problem solving that rivals the capabilities of human problem solving (as seen in the accomplishments of IBM's Deep Blue chess playing program (Campbell, 1999)). As a substitute for complex human thought, artificial intelligence can take advantage of the rapid yet routine behavior of computational systems to iterate through many possible solutions and present the best solution in a seemingly intelligent leap of decision-making (Pylyshyn, 1992).

³ This definition and the definitions beginning the next four chapters are courtesy of Merriam-Webster's Online Collegiate Dictionary (<http://m-w.com>)

Likewise, the iterative subsystem of A-Design takes advantage of computational speed to create and search numerous design states. However, this use of the word iterative is a simplification of what is implied when human design is described as iterative. Through the evaluation of previous designs, the human designer is informed of deficiencies, failure modes and constraint violations that were not apparent in the original creation. Iterating in design allows the human designer to acquire knowledge and progressively focus on identifying and overcoming the difficulties of a specific design problem. This *iteratively-guided* characteristic of human design is the basis for this A-Design subsystem. As the process iterates, A-Design becomes more efficient at making decisions, guided by knowledge acquired during the creation of previous alternatives.

In this chapter, the *iterative* procedure of A-Design is presented to provide the framework for explaining the remaining subsystems. After the other subsystems are described, Chapter 6 presents the *guided* part of this subsystem. The subsystem embodying the *iteratively-guided* nature of human designing is separated into these two parts. The *iterative* part captures the cyclic nature of design, while the *guided* part of the subsystem learns from past designs to direct future design activity.

2.1 RELATED WORK

Optimization techniques attempt to find the best numerical value, be it maximum or minimum, of a mathematical expression of one or more variables. A suite of different algorithms exists for the range of different forms of an objective function that exist including unconstrained, constrained, discrete, or multi-modal functions. The reason for studying optimization in the context of conceptual design is to hopefully gain insight into how A-Design can efficiently search for successful designs. It is assumed that conceptual

design cannot be described by simple mathematical expressions and that the best solution, as determined by a user's utility, presents itself in a highly nonlinear, discontinuous and multi-modal fashion. Stochastic optimization techniques such as simulated annealing (Kirkpatrick et al., 1983), Tabu search (Glover, 1989), and extended pattern search (Yin and Cagan, 2000) have been able to address these complex search spaces with some success. These approaches make informed decisions about how best to search by performing numerous perturbations to solutions while evaluating the objective function with each change. Subsequent iterations of the search process are guided by the statistics gathered from previous iterations, thereby allowing the optimization to overcome the challenges of complex search spaces.

The field of evolutionary computation is based on the principles of natural evolution. Within this field, genetic algorithms provide a powerful means to solve complex problems and handle complex search spaces (see overview in Mitchell, 1996; Goldberg, 1989). As a design automation technique, genetic algorithms have been utilized to solve a number of engineering design problems (e.g., Queipo et al., 1994; Brown and Hwang, 1993; Gage and Kroo, 1995). Genetic algorithms compare, propagate and modify many design states simultaneously to produce an efficient and parallel search for successful designs. Oftentimes genetic algorithms are used to find solutions in highly constrained situations where the user is more concerned about finding a feasible solution than an optimal one. Genetic algorithms synthesize solutions by modifying the configurations of past alternatives in ways similar to those occurring in nature, through genetic crossover and mutation.

Traditionally, stochastic optimization and genetic algorithms have only been able to address search spaces containing a set number of variables. There have been some exceptions to this; algorithms such as genetic programming (Koza, 1992), messy GA's (Goldberg, et al., 1993), shape annealing (Shea et al., 1997) and recursive annealing (Schmidt and Cagan, 1995) have furthered the capacity of optimization as a design automation tool by providing a richer means of representing design states. The general progression has been to extend the capabilities of optimization from addressing well-behaved static variable mathematical functions to handling more elaborate engineering systems. A-Design epitomizes this progression by including the notion of an open-ended design formulation. In addition, it also differs most from other approaches because it maintains and evolves both populations of designs and populations of the design creators, the agents.

In order to navigate the complex search spaces and avoid becoming trapped in local optima, both stochastic optimization and genetic algorithms rely heavily on random moves and statistical behavior. As a result, these algorithms need to perform many iterations to arrive at good results. Optimization of this sort can be costly especially when the generation and evaluation of each alternative is time-consuming. In engineering applications, evaluation times might prove to be a critical factor due to the complex and time-consuming analysis. Modification of designs in A-Design is performed by highly goal-directed agents invoked by a stochastic process rather than by a purely random process. The directness allows for more efficient search by preventing unfocused design creation.

2.2 THE A-DESIGN ITERATIVE SEARCH PROCESS

The iterative subsystem outlined in this chapter includes the step-by-step procedure of the A-Design process. The remaining subsystems are described briefly here, as their operations are an intrinsic part of the overall framework. At first, a general description of the A-Design procedure is described followed by a presentation of the A-Design specifics for the weighing machine design problem.

Figure 2.1 shows a general flowchart of the A-Design process, which can be viewed as an instantiation of the search process cycle from Figure 1.4. In this figure, the gray boxes indicate tasks involving agents. These agents are strategies that interact to solve these design tasks (see Section 5.2.1 for definition). Each of these three boxes contains

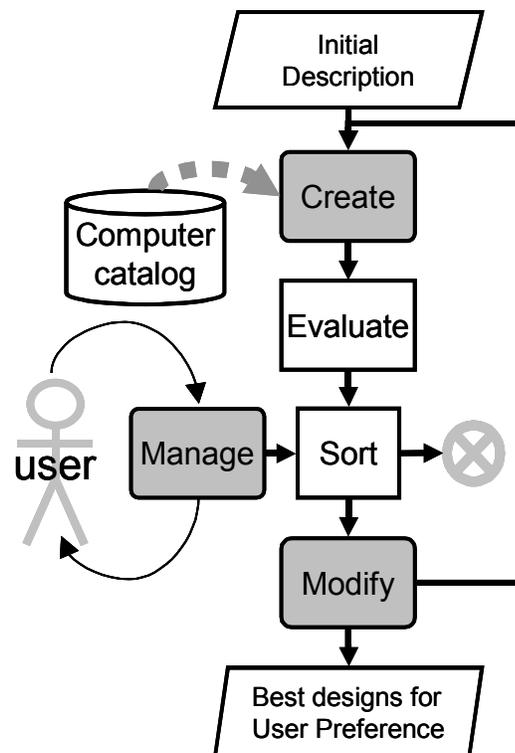


Figure 2.1: General flowchart of A-Design process.

the interaction of numerous agents and is described in more detail in Chapter 5 and 6.

At the top of the figure, the process is initiated with some seed specified by the user that encapsulates the description of the design problem. The creation of design alternatives is accomplished by a set of interacting agents, known as Maker-agents that work directly with these input specifications to produce a population of design alternatives. Each of these alternatives results from the contributions of several agents. The Maker-agents are responsible for producing new candidate solutions from the original description of the design problem and for revising designs returned from previous iterations.

Then, the candidate solutions created by the Maker-agents are evaluated. This evaluation can include simple objectives that are easily calculable such as the overall mass of a design as well as other objectives that require external computational analysis. This evaluation process, which occurs independent of agents, can be viewed as the engineering analysis that happens as a subset of the larger engineering design task.

Next, design alternatives are examined in a process involving the user, the Manager-agent and the adaptive design selection mechanism described in Chapter 3 (Design Selection). The Manager-agent finds the design solution that best meets the approximation of the user's utility function and presents it to the user. If the user wishes, he/she can initiate a dialog with the Manager-agent to adjust this preference. As a result of this dialog and the evaluation of the designs, the Manager-agent gathers data on which agents perform to the user's specifications and which design elements are useful in meeting the user's specifications. Then, the sorting mechanism described in Chapter 3

determines which designs are to be saved to the next iteration, and which designs are discarded to make room for new design concepts.

The designs that are saved for modification are then passed to the next phase of the process. Interacting Modification-agents choose designs from the preserved set of best solutions and attempt to refine them based on the quality of their evaluation. Agents in this category add, remove, or otherwise alter elements of design alternatives to create new states that are returned to the process by way of the Maker-agents.

After the modification of designs, the process repeats, evolving design populations and agent populations. As the process unfolds, design states cycle through the exchange between Maker-agents and Modification-agents until the system converges or resource and time constraints require the acceptance of the current best design.

2.3 ELECTROMECHANICAL CONFIGURATION A-DESIGN

As an instantiation of the general flowchart of A-Design shown in Figure 2.1, the electromechanical configuration A-Design system used throughout this dissertation can be detailed further. Figure 2.2 shows more clearly how the four subsystems of A-Design (the agents, the iterative process, the design representation and the design selection) interact. At the top of the figure, the initial description of the design problem for electromechanical design is supplied as a list of input and output behaviors. These input and output descriptions are accepted by the first set of Maker-agents known as Configuration-agents (C-agents). These agents build design configurations by connecting abstract components drawn from a computer catalog. The designs that result from the C-agent operations are a connection of components that lack concrete values for the

parameters within the system. They are abstract functional descriptions of possible complete design configurations, and are described further in Chapter 4 (Design Representation).

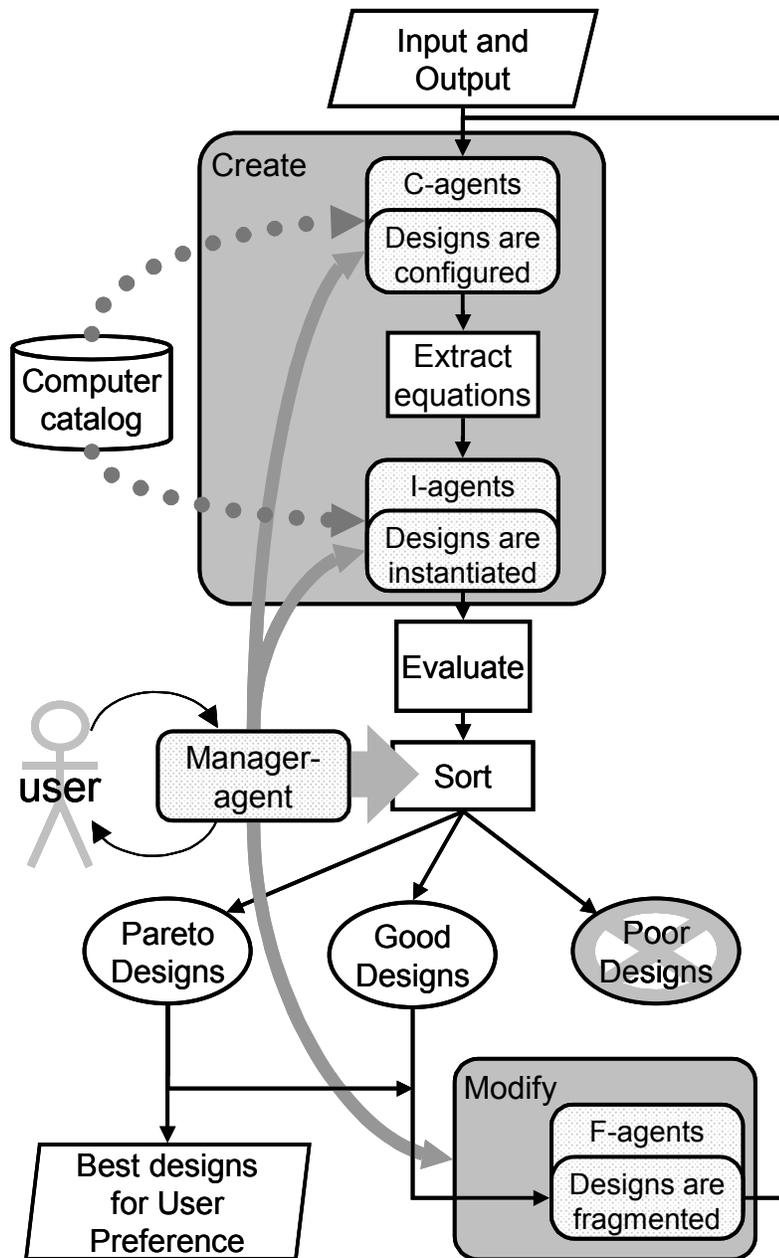


Figure 2.2: Detailed flowchart for electromechanical configuration design.

From these functional design alternatives, behavioral equations are constructed that describe the functional relationships of the inputs and outputs of the problem description. By referencing these equations, Instantiation-agents (I-agents) choose actual components from the computer catalog to instantiate the conceptual components chosen by the C-agents. The details of the agent strategies for constructing configurations are further described in Chapter 5. After outfitting designs with real components, the process is now able to evaluate the alternatives created in this Maker-agent phase.

The evaluation stage of the process includes objectives that the user has specified as the important criteria successful designs. For electromechanical configuration design, these evaluations can rely on the aforementioned behavioral equations (as in the test case shown in Section 8.1), or external computational analysis (as in the test case shown in Section 8.2).

Next, the sorting of designs divides the candidates into three populations, Pareto-optimal, Good and Poor, via the design selection process described in the next chapter. The Pareto-optimal solutions are first delineated as the candidates from this set that best meet a range of rational utility functions. As a result, the designs selected as Pareto-optimal solutions exhibit a range of strengths in the various objectives of the design problem. These designs are saved to the next iteration where they are compared with new

and modified alternatives. Good designs are non-Pareto designs that contain beneficial characteristics under the current user utility function as determined by the Manager-agent's interaction with the user (see Section 3.4). The populations of Good and Pareto designs are passed to the modification phase of the process in hopes of improving these solutions in the next iteration. The remaining alternatives, Poor designs, are discarded to make room for new design concepts since they have the least potential to improve future designs.

The modification phase of the process is done through a fragmentation of designs by the Fragmentation-agents (F-agents). The basic strategy of these agents is to improve design states by removing components from solutions that appear to be reducing their overall worth. As a result of this fragmentation, designs are then reconstructed in the following iteration by the Maker-agents. In this manner, the combination of F-agents and C-agents together improve designs through their knowledge-driven deconstruction and reconstruction. For example, a somewhat heavy design can be identified by a Fragmentation-agent. This agent removes the design's more massive components and passes it back to the C-agents so that it might be rebuilt with lighter components in the following iteration.

As the iterations ensue, the process generates numerous design configurations and moves towards solutions that best meet the user's specifications. The Manager-agent also tracks successful and unsuccessful trends in the design process to discover the characteristics of successful designs that can be learned from past iterations. This

tracking provides the iterative guidance to make the process more efficient over the iterations and is discussed in detail in Chapter 6.⁴

2.4 PURPOSE BEHIND ITERATIVE STRATEGY

The iterative approach described here is similar to that used in genetic algorithms and other stochastic optimization techniques, which take advantage of the number-crunching abilities of computers to search as many solutions as possible. However such approaches are rarely capable of operating beyond the optimization of a well-defined set of variables set prior to execution. This predefining of set variables is counter to our goal in developing an automated conceptual design process.

Other than stochastic optimization, a handful of knowledge-driven or rule-based techniques exist for solving ill-defined conceptual design problems (e.g. Brown and Chandrasekaran, 1986 and Navinchandra, et al., 1991). However these approaches often do not take advantage of searching through numerous alternatives as is done in stochastic search strategies. This lack of exploration can be seen as particularly inadequate for conceptual design, since the search spaces are even larger than those addressed in parameter optimization. While knowledge-based techniques allow for ill-defined, or open-ended design formulations, they do not explore the vast space of possible solutions as is done by stochastic strategies. To resolve this, the A-Design iterative process described here combines aspects of both stochastic optimization and knowledge-based design strategies. This is accomplished by creating knowledge-driven strategies in agents interacting under a stochastically guided iterative algorithm. This combination searches

⁴ A pseudo-code description of the operations of this iterative process can be found in Section A of the Appendix. In addition, Figures 4.5, 5.5, and 6.1 show pseudo-code for agent operations and design construction.

the design space for solutions that appear to be optimal given the space and time resources allotted.

The A-Design approach to simultaneously managing multiple design solutions is similar to the genetic algorithm approach. By comparing the population of design alternatives, the best ones are selected to propagate to the next iteration while the remaining ones are discarded to make room for new solutions. In genetic algorithms, this selection pressure or “survival of the fittest” is the primary motivating factor for finding successful designs. A-Design’s selection pressure conservatively eliminates only the designs that would never be desired under any user preference. The user’s preference is also used to guide the process towards solutions that best exemplify the desired trade-offs by propagating a higher concentration of designs that best meet the current user preference.

In addition to the selection pressure mechanism, the design process must have a method for generating new or improved solutions. The traditional approach in genetic algorithms is to have a random process of crossover and mutation - two mechanisms for producing diversity from a limited population of specimens. The randomness of these processes, while not guaranteeing an improvement over the current set, provides a method for the effective search of complex search spaces. A-Design searches for improved designs through intelligent modification of past alternatives and through feedback provided to agents. Agents do not perform simple mutation and crossover operations as in genetic algorithms, but instead intelligently fragment or chunk designs based on the evaluations of the design.

Having many knowledge-driven agents with different strategies responsible for the same task, as opposed to a single knowledge-driven strategy, generates a greater variety of possible alternatives. The system gains robustness through the collaboration of these various agents, which makes the system flexible to changes in the user's preference. These agents are analogous to individual specialists within a design firm. As with a company, there is usually not a single optimal design, but rather designs evolve and as they do so the company selects designs at certain stages and markets them as products. Furthermore, a company can rethink their products and improve upon them to better adapt to changes in market conditions, or to adopt new emerging technologies. This dynamic nature of true engineering design is the philosophy behind making A-Design both iterative and adaptive. The iterative process creates an opportunity for adapting both design specification and goals.

Chapter 3

Design Selection

Main Entry: **adap·tive**

Pronunciation: &- 'dap-tiv, a-

Function: *adjective*

Date: 1824

: showing or having a capacity for or tendency toward adjusting to environmental conditions

- **ad·ap·tiv·i·ty** /"a-"dap- 'ti-v&-tE/ *noun*

- **adap·tive·ly** *adverb*

The second subsystem is the *adaptive* selection of design solutions. Reformulating design specifications and goals is a common occurrence in conceptual design. The design problem can be restructured based on market demands or on experience gained in the design process. This restructuring is a mix of compromising and understanding the trade-offs of competing attributes. The decision-making in design is quite similar to the research known as multi-attribute utility theory first explored in depth by Keeney and Raiffa (1976). For a conceptual design process, a formal understanding of this decision-making can lead to the construction of a utility function that encapsulates a designer's preferences in guiding design automation.

Decisions in conceptual design can have large impact downstream in the design process. Sometimes, these early decisions produce unexpected and undesirable effects in

the attributes when the artifact is fully realized at the end of the design cycle. The engineering designer must predict how early decisions effect a product's attributes. For example, money is often an important attribute and it contrasts with performance attributes such as efficiency and power consumption. Ideally, after some design progress, the designer becomes aware of how to manage the trade-offs, as well as determine what are the key challenges in meeting design goals. As the designer acquires the new information on the design process or as the goals for the design problem change, a parallel change to the utility function occurs.

Design is the art of making products for a changing world. The creation of new products is an ever-adapting and interactive process of integrating new information, new technologies and new biases from the marketplace. This chapter lays out the method by which A-Design is able to adapt to these types of changes, thereby reinforcing the claim that A-Design is a unique conceptual design theory. Previous computational design aids operate on well-behaved or static problems where there is no means for user interaction or interjection of new knowledge and preferences. The adaptive method described here provides an in-depth search of the design space, retains flexibility in the design process, and allows for robust selection of designs regardless of the number of objectives.

3.1 NOMENCLATURE AND ASSUMPTIONS

Because the work related to this subsystem deals with such a broad range of research areas including both mathematical and economic studies of decision-making, the terminology used throughout the remainder of the dissertation is clarified in this discussion.

First, the computational A-Design process interacts with a *user* or engineering *designer*. The use of the words *user* and *designer* are used interchangeably to describe the person that communicates with A-Design throughout the conceptual search of new designs. In this dissertation, the *user* does not refer to the person interacting with the designed artifact, as in the term *end-user*. Several times we refer to the *end-user*, or equivalently the *consumer* as the one providing the target market that may indirectly influence the actions of the *user* or *designer*.

The definitions of design *goals*, *attributes* and *objectives* are slightly different as defined in Keeney and Raiffa, and these subtle differences are honored here. In short, an *attribute* is merely a characteristic of an artifact – a placeholder for a specific value either qualitative or quantitative. An *objective* is a statement (usually a mathematical expression) of an *attribute* that is to be maximized or minimized as much as possible, for example, maximize efficiency, or minimize cost. A *goal* poses an *attribute* within the context of a constraint to be met, for example, reduce weight to less than 3.5 pounds. It is often the case that design problems are overwhelmed by goals such as this and, according to the Simon (1986), these goals are the real crux of design and decision-making. His belief is that human decision-making is based on “satisficing” instead of optimizing, where satisficing is the act of improving *attributes* to meet or possibly exceed predefined acceptable values. However, due to mathematical formalisms, the A-Design design process is focused mainly on *objectives*. This is due to the fact that in search, every design state must be comparable to other design states in the space. By imposing a metric afforded by the objective model used in optimization, an algorithm can be constructed to determine design worth on a continuous scale. The *goals* of a design process are, in a

sense, Boolean in nature; they are either met or not met. One way to fulfill design *goals* in the computational process is to superimpose the continuity that *objectives* demonstrate as a measure of how well *goals* are met. The *goal* is then posed as an *objective* that through minimization of that *objective* leads to a fulfillment of the *goal* statement.

From this distinction of *attributes*, *objectives* and *goals*, one must also note the subtle difference in the research areas of multi-attribute decision-making and multi-objective optimization. The multi-attribute decision-making theory established by Keeney and Raiffa does not focus on the challenges encountered in the search for ideal attribute values, but with the formal structure required for understanding the choices made among established design solutions. This is opposed to multi-objective optimization, which focuses on the automated search for solutions that best meet more than one minimization or maximization statement.

In both multi-objective optimization and multi-attribute utility theory, one is faced with the difficult task of formalizing the perceptions of how attributes affect the overall worth of a design. This brings us to our next set of ambiguous terms: *value* and *utility*. Both *value* and *utility* represent the quality of an alternative as determined by some complex user perception. The main difference is that *value* is only used when certainty is guaranteed, while *utility* contains a degree of uncertainty about the actual *value* of an attribute. Therefore, the basic question is whether this perception held by the *designer* is a certain quantity or an approximation subject to change. This debate as applied to engineering design has supporters on both sides (see Thurston, 1999; and Scott, 1996), however approximating the *user's utility* of a design is favored to the *user's value* of a

design in this dissertation. It is this author's opinion that while decisions within the design process lead to specific and definite changes in the attributes, the context in which these attributes are judged by the user can be transient and subject to uncertainty.

This chapter describes how the A-Design system approximates the form of this *user utility* function by combining the *objectives* in a mathematical formula. The general term for how these objectives relate is known as the *preference* the user has for the objectives. The form for such a utility function can be quite difficult to approximate, as one needs to understand interdependencies among the attributes. In this research, the most basic form of a linearly weighted sum is used to approximate the utility function. The coefficients or *weightings* for the various attributes provide one simple instantiation of the user *preference*. Alternate forms of the utility function can be readily incorporated into the A-Design framework. Figure 3.1 provides a summary of some of these terms.

The final terminology issue concerns the use of *Pareto-optimality*. In this chapter, the principle behind *Pareto-optimality* is introduced as a means of separating the design states. It is sometimes labeled the *non-dominated set* or the *efficient frontier* in optimization to better delineate its use from its formal definition in decision-making.

f_i	<i>attribute</i> of a design (such as cost)
$\min f_i$	<i>objective</i> of the design process (such as minimize cost)
$f_i < \$C$	<i>goal</i> of the design process (reduce cost to under \$C)
$u_j = \sum_i w_i f_{ji}$	<i>utility function</i> , u_j , the worth of design j as approximated by a <i>preference</i> for the various attributes, f_{ji} , by linearly weighted sum. Each attribute has a corresponding weight, w_i .

Figure 3.1: An example of various terms used throughout this chapter.

However, within the majority of related work in multi-objective optimization the term *Pareto-optimal* set is used interchangeably with the *non-dominated set*. In this dissertation, the use of *Pareto-optimality* is used to label those solutions that currently represent the Pareto-optimal set of designs despite the fact that in future iterations new solutions might cancel out previously designated Pareto-optimal designs. This distinction will be discussed in more detail in Section 3.3.

3.2 RELATED WORK

Decision-making about more than one criterion was formally addressed by Keeney and Raiffa (1976). This seminal work laid out the details for future decision-making research and provided the foundation for multi-objective optimization. As an important step in formulating their theories, Pareto-optimality has been cited as a basic divisor between good and bad decisions. The original work of the Italian economist Vilfredo Pareto has had many diverse contributions to our understanding of economics and decision-making, and is extrapolated for use in optimization in Balachandran and Gero (1984) and in Eschenauer et al. (1990). Many algorithms, especially genetic algorithms, have incorporated this notion of comparing designs (see overview in Fonseca and Fleming, 1995). Interestingly enough, the combination of Pareto-optimality and agents has also been explored by Petrie et al. (1995) where existing software tools are controlled by a single governing agent that makes decisions to keep or discard designs based on Pareto-optimality.

In addition to Pareto-optimality's use in multi-objective optimization, other related research projects have addressed the multi-attribute problem of engineering design. Most notably, Thurston (1991) and D'Ambrosio and Birmingham (1995) have applied multi-

attribute utility theory to engineering design problems. Dong and Agogino (1995) developed the concept of spectral optimization to approximate Pareto-optimal design choices with specific application to catalog selection. The work presented here also uses Pareto-optimality as the basis for dividing designs, but makes further provisions to allow for flexibility in focusing on changing user preferences.

3.3 DIVISION OF DESIGN SOLUTIONS

At each iteration in the process, the algorithm contains a population of design candidates. From this population, it is necessary to determine which alternatives are most useful for advancing towards successful designs and which alternatives have little or no design worth. With this distinction, designs that offer little worth to the process are removed to free up space for new design states so that the search process can constantly move towards improved designs. In order to decide which designs to save and which to discard, A-Design employs a unique strategy whereby candidates are divided into three separate populations labeled “Poor” designs, “Good” designs and “Pareto” designs as described below.

The idea of Pareto-optimality provides the basis for mathematically determining which designs are clearly better than others without simplifying the objectives to a single scalar through an a priori determined utility function. Pareto-optimality guarantees that the best design for any utility function will be found in a set of designs called the Pareto-optimal set. The simple definition of Pareto-optimality is that solutions are Pareto-optimal if no attribute can be improved without compromising other attributes. In the scope of comparing design solutions, this means that a design is Pareto-optimal if no other designs exist that offer an improvement in one or more attributes without

compromising other attributes. Mathematically, this is defined in (Eschenauer et al., 1990) as:

$x^* \in X$ is a Pareto-optimal point if and only if

there is no other vector $x \in X$ such that

$$f_j(x) \leq f_j(x^*) \quad \forall j \in \{1, \dots, m\} \quad (1)$$

and

$$f_j(x) < f_j(x^*) \quad \exists j \in \{1, \dots, m\} .$$

In this equation, f is an attribute, m is the total number of attributes and X is the set of all design states. By requiring all objectives to be optimized through minimization, no other design should exist that contains more minimal values for the total of the attributes present. This can also be visually determined through plotting alternatives on a graph with the axes representing different objectives as in Figure 3.2a. If only two objectives

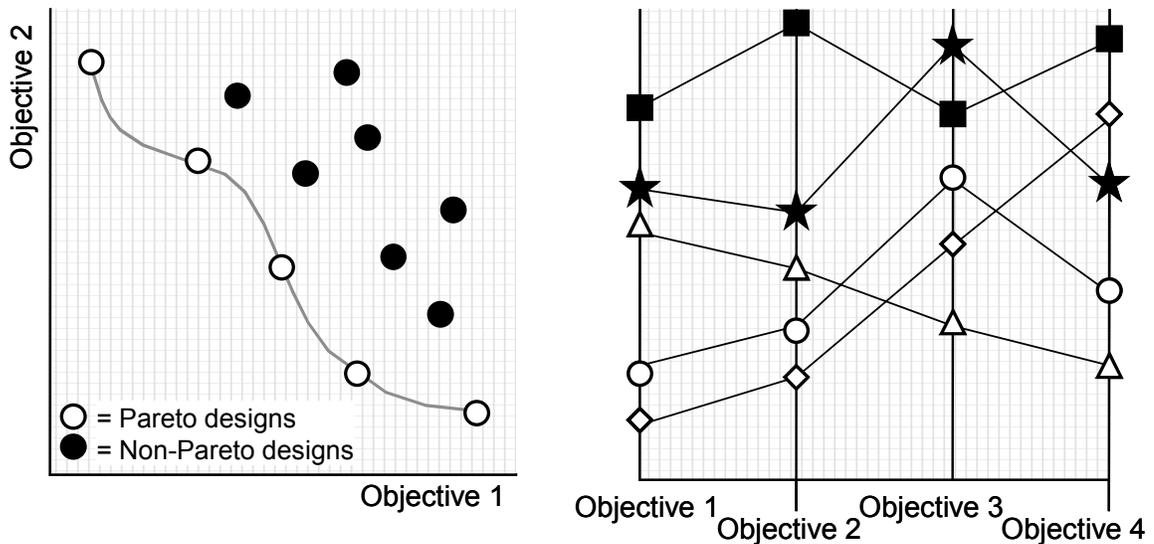


Figure 3.2: a) Two-dimensional plot of designs depicting Pareto-optimal set for two objectives, b) the performance profile method of comparing of designs states.

are to be optimized, one can view design alternatives on a two-dimensional grid and the Pareto-optimal designs can easily be determined. These designs form a “front” of solutions that are closest to the coordinate axes. One can also extend this visualization tenuously to three objectives but difficulties obviously exist for depiction of higher dimensions with this method. A visualization for higher dimensionality that is favored in this dissertation is the performance profile originally used in Keeney and Raiffa shown in Figure 3.2b. This plot lists attributes along the x-axis and represents a single design state as a link connecting attribute values. Here, again if a minimization of all objectives is imposed, one can find how various design states trade-off with other designs. When two links cross in Figure 3.2b neither solution dominates the other. Since Pareto-optimal solutions are those that are not dominated by other designs, all Pareto-optimal designs intersect with all other Pareto-optimal solutions. In this figure, the open shapes represent Pareto-optimal solutions, while the filled shapes represent non-Pareto solutions.

As mentioned in Section 3.1, this division of designs into Pareto and non-Pareto sets is valid only for the current set of designs. The true Pareto-optimal solutions, that is, those designs that fulfill the condition stated in Equation 1 for all designs in the search space, are not attainable in conceptual design. The search spaces for conceptual design contain an infinite number of solutions as described in Section 1.2, and as a result, one can only approximate the true Pareto-optimal set of solutions. Therefore, the set of designs labeled Pareto-optimal in this research is valid for only the set of visited design states and not for the entire space of solutions. This set of designs is updated throughout the iterations of the process, as can be seen in Figure 3.3. The essence of the A-Design search process is to continually update the Pareto front to constantly find better solutions.

As new design states are visited, previously believed Pareto-optimal designs are removed from the set. This constantly updated front of Pareto designs is a conservative preservation of designs depicting a diversity of relative strengths in the objectives. This diversity accommodates the changes in the user utility function that can occur in the course of the A-Design process. In Figure 3.3, the user utility function or preference for objectives identifies a member of the Pareto set that is most successful in meeting the design problem specifications. As the search process ensues, the best design available to the user can change to any member of the current Pareto-optimal set.

In addition to storing design diversity, A-Design also focuses on improving designs to best meet the current user preference. Outside of Pareto-optimal designs, the system further divides solutions into Good and Poor designs. Some designs, while not Pareto-optimal, might better meet user preference than some of the outlying Pareto-optimal alternatives that have been preserved for extreme changes in user preference. These

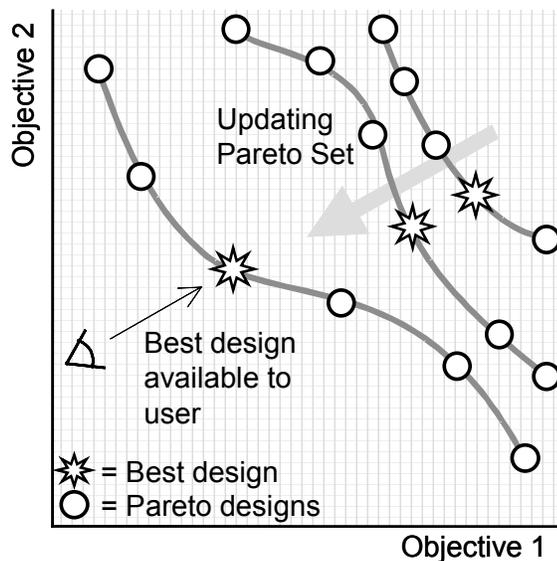


Figure 3.3: Updating Pareto surface while considering user preference.

preferred but non-Pareto designs comprise the Good population visualized as a set of solutions that are located within a given radius of the intersection of current user preference and the Pareto set (Figure 3.4). In general, the Good designs are the best designs (top 50% in the current implementation) of the non-Pareto designs that meet the current approximation of the user's utility function. If the user's preference should change then the location of the Good population also changes. Through preserving the Good designs, new design states are constructed in the modification phase to hopefully lead to improved or possibly Pareto-optimal solutions in future iterations. By preserving this set of Good designs along with Pareto designs, A-Design contains both a focus on the current user preference as well as design diversity. The remaining designs that do not fall into the Good population are labeled as Poor designs. These designs are discarded by the

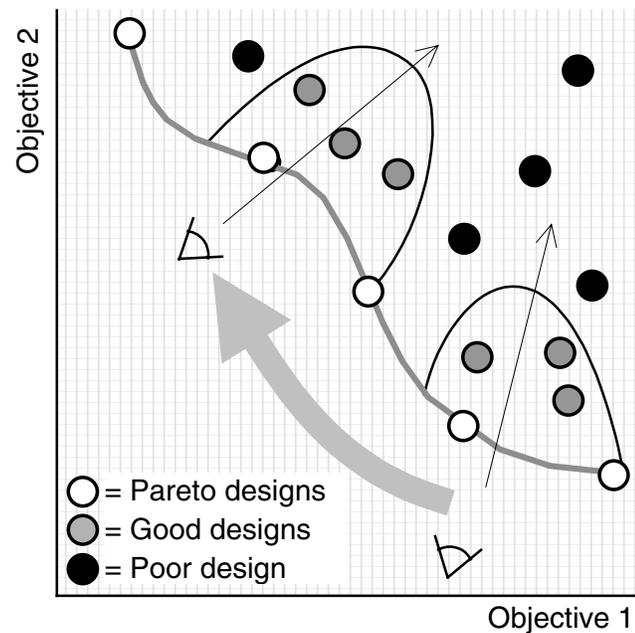


Figure 3.4: Bulb of Good populations move with user preference.

system to make room for new design states that are built from modifications of Good and Pareto solutions in subsequent iterations.

The multi-objective selection described here is capable of handling large numbers of objectives in a robust manner, as well as tailoring designs for specific designer needs or consumer markets. Imagine a situation in which weight and cost are two competing objectives in a conceptual design problem. A-Design might be asked to conceptualize designs with more importance given to minimizing cost than minimizing weight. The process concentrates its effort on this user preference by preserving Good designs in the region of this preference. Furthermore by maintaining the complete Pareto-optimal set, the system can accommodate changes if the preference should shift towards minimizing weight. The diversity of designs stored in the Pareto-optimal set is similar to the maintenance of recessive information in biological systems where unused characteristics are preserved to allow for rapid transition in response to environmental changes.

Another important feature of the division of designs is to track the agents that create them. By increasing the prevalence of agents that create Pareto and Good designs, future design activity will ideally produce more designs like the current Pareto and Good set. Conversely, agents that produce Poor designs are penalized in order to eliminate design activity that leads to inferior design states. Manager-agents control this feedback mechanism by keeping track of the agent contributions and modifying the Maker-agents and Modification-agents accordingly (this is described in detail in Section 6.4). This division of designs and the feedback it provides to the agents allows A-Design to *adapt* to changes that occur in the conceptual design process.

3.4 ADAPTING TO USER PREFERENCE

The previous section discussed how designs are divided into separate populations so that A-Design can change the focus of the search process to bring about the adaptive nature of conceptual design. This section explores how that change is impressed upon the system. The basic method of effecting the direction of the A-Design search process is to interact with the Manager-agent (as described further in Chapter 6). In each iteration, prior to design selection, the Manager-agent displays various statistics on the process and prompts the user for input. If the user decides not to interact, the process continues. If the user sees that a change is required, he/she can temporarily halt the process and enter into a dialog with the Manager-agent. The Manager-agent then prompts the user with questions about the current set of designs to determine what changes can be made to adapt to the user's needs.

Figure 3.5 shows a typical dialog from the weighing machine test problem that includes four objectives to be minimized: cost, weight, dial error, and input displacement. In this example, the process is initiated with linear weights on the objectives of "1 10 1,000,000 1,000,000." At each iteration, data on the current state of the process are presented to the user (population sizes, the amount of change in the Pareto set, etc.), as well as the current best design for the user preference. In this figure, the user hits return at the end of the sixth iteration initiating a brief dialog with the Manager-agent.

Having presented the attribute values of the best design at this juncture of the process, the Manager-agent asks the user to rate a set of three random pareto-optimal designs on a scale of one to ten. In order to provide a reference for the user, the current best solution is assigned a value of 5 so that the user may rank the remaining three designs in proportion

<pre> Iteration 6. Population = 44. Pareto has 13 members. Pareto changed by 7. The ave. Pareto values are (46.295383 0.570433 7.8543525 0.0028573587). The best Pareto values are (18.5 0.14699998 0.0012533892 9.648292e-4). Good has 16 members. The radius is 47.80917. Poor has 15 members. Todo has 0 members. Taboo has 0 members. Weights are (1.0 10.0 1000000.0 1000000.0) Entering evaluate...done Pareto last changed 0 iterations ago. Top Design last changed 5 iterations ago. Currently the best design for your preference is: COST = 72.84 MASS = 0.23563002 DIAL-ERROR = 0.0012533892 INPUT-DX = 0.002038508 ----- Talk to M-agent (hit return) ? 1: COST = 26.43 MASS = 0.175 DIAL-ERROR = 0.066433564 INPUT-DX = 9.648292e-4 ----- 2: COST = 84.13 MASS = 2.9145 DIAL-ERROR = 4.7503047 INPUT-DX = 9.648292e-4 ----- 3: COST = 85.13999 MASS = 2.232 DIAL-ERROR = 6.769965 INPUT-DX = 9.648292e-4 ----- Given that the best design presented is 5, how would you rate each of these designs(1-10) ? 1: 10 2: 2 3: 2 Is this design as good as the best shown above [y/n]? COST = 20.99 MASS = 0.164 DIAL-ERROR = 0.31395644 INPUT-DX = 0.002038508 ----- Y Interpolate or Extrapolate new preference[i/e] Is your preference between these two designs or more along the lines of the second? i new weights = 23.53806 28.02538 500023.03 500000.0 The design process is 0.15 complete. </pre>	<pre> Iteration 7. Population = 46. Pareto has 17 members. Pareto changed by 4. The ave. Pareto values are (45.26059 0.52912116 6.0937095 0.003011668) The best Pareto values are (18.11 0.14699998 2.5082377e-5 9.648292e-4). Good has 15 members. The radius is 47.212936. Poor has 14 members. Todo has 0 members. Taboo has 0 members. Weights are (23.5 28.0 500023.0 500000.0) Entering evaluate...done Pareto last changed 0 iterations ago. Top Design last changed 5 iterations ago. Currently the best design for your preference is: COST = 22.3 MASS = 0.169 DIAL-ERROR = 0.31395644 INPUT-DX = 0.002038508 ----- Talk to M-agent (hit return) ? Iteration 8. Population = 49. Pareto has 22 members. Pareto changed by 9. The ave. Pareto values are (48.8 0.46536332 2.2392752 0.002516745) The best Pareto values are (18.11 0.14199999 2.5082377e-5 9.648292e-4). Good has 14 members. The radius is 48.007374. Poor has 13 members. Todo has 0 members. Taboo has 0 members. Weights are (23.5 28.0 500023.0 500000.0) Entering evaluate...done Pareto last changed 0 iterations ago. Top Design last changed 1 iteration ago. Currently the best design for your preference is: COST = 19.71 MASS = 0.172 DIAL-ERROR = 0.31395644 INPUT-DX = 0.002038508 ----- Talk to M-agent (hit return) ? </pre>
--	---

Figure 3.5: A display of the dialog that occurs between the user and the system in adapting to user preference.

to this value. This ranking supplies the system with data used to approximate a new user's utility function. We are somewhat reassured that the user can accurately estimate meaningful values for his/her utility for such designs due to experiments in human perception. Most notably, Stevens (1975) has shown that people are capable of accurate prediction through experiments that correlate perception and cardinal rankings in a technique known as magnitude estimation.

As a result of this dialog, the Manager-agent determines a new form of the utility function in an approach similar to the learning of "static evaluators" through linear regression (Abramson, 1990). From these cardinal rankings, the Manager-agent performs a best fit to the system of linear equations that result from the weighted sum of objectives. Figure 3.6 shows how these equations are constructed for the dialog in Figure 3.5. Based on this best fit, the new best Pareto design is presented to the user. To further refine the accuracy of the newly adapted weights, the Manager-agent asks the user if the trend towards this new preference is severe enough to warrant further extrapolation of the preferences or to establish some middling between the current and prior preferences. In other words, this dialog tries to determine if the user preference is actually between the new best design and the prior best design or tending more towards the new design. In the dialog of Figure 3.5, the user opts for an interpolation of the new and prior preferences, and as a result a new approximation for the weights is established, in this example the new weights change from "1 10 1,000,000 1,000,000" to "23.5 28.0 500,023.0 500,000.0".

The design selection process described above now has a means to determine where the Good population of designs should lie – at the intersection of this new preference and the Pareto surface. Further, the result of changing this preference leads the A-Design process and the Manager-agent to present new solutions in the subsequent iterations of Figure 3.5.

While this demonstration shows the adaptive nature of the A-Design system, further experimental results address this matter in Chapter 9. The methodology underlying the Manager-agent’s behavior is based on several distinct research areas including machine learning, magnitude estimation and utility theory. The use of the above dialog mechanism raises many interesting research questions about the dialog between man and machine, the ability to correctly estimate design worth and the means of approximating utility.

Given: $d0$ (the best design dialog of Table 3.1)
 has attribute values of $f_{01} = 72.84$
 $f_{02} = 0.23563002$
 $f_{03} = 0.0012533892$
 $f_{04} = 0.002038508$
 and has a ranking $r_0 = 5$

Find: new values for w_1, w_2, w_3, w_4 .

- Manager presents three new designs.
- Users ranks these 10, 2, 2
 - $r_1 = 10, r_2 = 2, r_3 = 2$.
- Establish as a system of linear equations:

$$w_1 \bullet f_{01} + w_2 \bullet f_{02} + w_3 \bullet f_{03} + w_4 \bullet f_{04} = r_0$$

$$w_1 \bullet f_{11} + w_2 \bullet f_{12} + w_3 \bullet f_{13} + w_4 \bullet f_{14} = r_1$$

$$w_1 \bullet f_{21} + w_2 \bullet f_{22} + w_3 \bullet f_{23} + w_4 \bullet f_{24} = r_2$$

$$w_1 \bullet f_{31} + w_2 \bullet f_{32} + w_3 \bullet f_{33} + w_4 \bullet f_{34} = r_3.$$
- Solve for w_i .
- Interpolate/Extrapolate with previous w_i .

Figure 3.6: Determining the new weights is a process of solving the linear system of equations that is created by the cardinal ranking of various Pareto design states.

Chapter 4

Design

Representation

Main Entry: **open-end-ed**
Pronunciation: "O-p&n- 'en-d&d
Function: *adjective*
Date: 1825
: not rigorously fixed: as **a** : adaptable to the developing needs of a situation **b** :
permitting or designed to permit spontaneous and unguided responses
- **open-end-ed-ness** *noun*

This chapter focuses on the innovations of A-Design that provide a foundation for representing design concepts. As can be seen from Figure 1.4, this part of the automated conceptual design process exists outside of the search mechanism. The reason for the isolation is to show that the representation is defined prior to search as the foundation upon which designs are constructed.

It is necessary in any computational process to establish a formal language of the artifact being manipulated, created, or optimized. Traditionally, in optimization, the representation is a fixed set of variables representing different physical dimensions or parameters of a design. While a set number of variables provide a concrete method for

computational search, it provides too much constraint for conceptual design. The representations within many search techniques are constrained by the generation techniques intrinsic to the process. For example, in the case of genetic algorithms, a vector formulation used to define the design space is often expanded to a bit string in order to represent a genotype encoding. This encoding provides the crossover and mutation operators a means of manipulating design states to search the space of possible solutions.

In order to avoid the constraints imposed by the search process, we develop a description that is most natural to modeling the actual design space as opposed to one prescribed by the search mechanism. The freedom of constructing a representation independent of the search process allows one to include variable length vectors or complex data structures to attain a greater variety of design solutions. The drawback of having a more general design description is the need for more involved techniques for generating designs. This is where the agents are used to act as a buffer between the iterative process and the representation. The development of agents along with the development of the representation allows A-Design to be tailored to a general body of conceptual problems.

This research has concentrated on a specific electromechanical problem space, but there is no reason that other representations and agents for manipulating such representations cannot be developed within A-Design. Therefore, while this chapter lays out the important contribution of this work to a specific domain, one contribution is implicitly stated by the fact that the agents act as a buffer between the iterative search

process and the design representation. This important feature enables A-Design to address large and ill-defined search spaces characteristic of conceptual design.

As will be seen in this chapter, the electromechanical representation provides a language for understanding components and how they connect. No restrictions are placed on the form of the design, thereby accomplishing an unstructured or open-ended space of possible design solutions. The search space is defined as open-ended in that no common structure of the solution is presumed prior to design.

4.1 RELATED WORK

Several research areas are centered around implementing an understanding of how the physical world functions. On the most general level qualitative physics research (see overview in Forbus, 1988) has provided several computational strategies for symbolically or qualitatively modeling the physical environment. Specifically for mechanical systems, many widely varying approaches have been explored: through linguistic approaches such as (Stone and Wood, 1999); through causal models (Nayak, 1992), through configuration space models (Stahovich, et al, 1998; Chakrabarti and Bligh, 1996), or geometric algebras (Palmer and Shapiro, 1993).

In addition to understanding how the physical world behaves, we are also concerned with how to synthesize solutions to design problems. Several research projects have dealt with the automated synthesis problem for electromechanical configuration design. The methods for synthesizing topologies are as diverse as the representation methods shown above. Approaches to synthesis include case-based reasoning (Navinchandra, et al., 1991), constraint programming (Subramanian and Wang, 1995), and qualitative symbolic

algebra (Williams, 1990) to name a few. The most historically significant of these include several approaches applying expert system formulations to specific design problems. The paper roller system established in Mittal et al. (1985) and the R1/XCON system (McDermott, 1982, 1993) have yielded some interesting and useful results for industrial applications.

In the larger context of representing and synthesizing electromechanical function, a small number of projects have sought to both establish a generic scheme for electromechanical design and remove limitations on design topologies. These few research projects have influenced the functional representation of A-Design. As can be

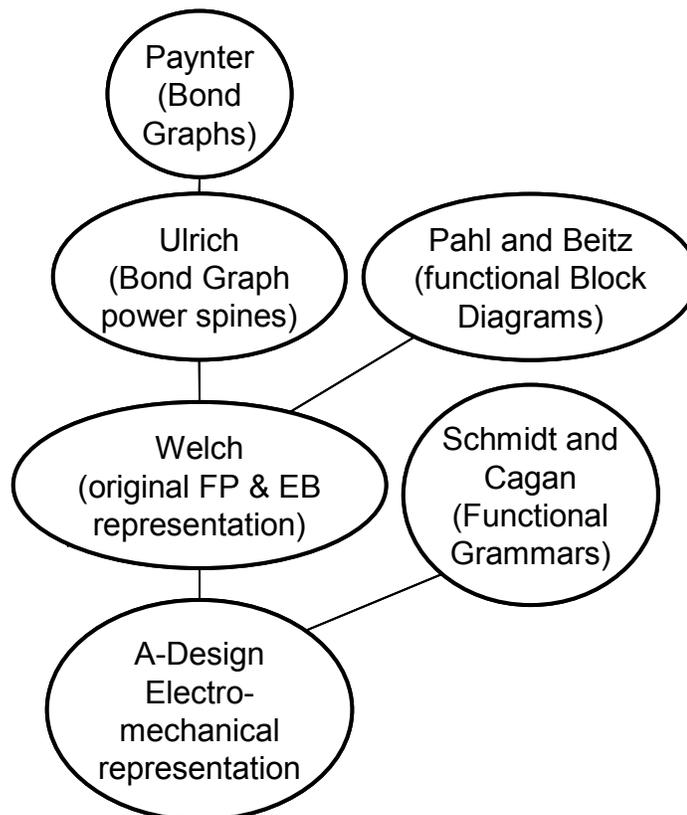


Figure 4.1: The development of this representation is related to the contributions of various previous research projects.

seen in Figure 4.1, the A-Design representation approach builds primarily on the work of Welch (Welch, 1992; Welch and Dixon, 1994). Furthermore, the innovations of Welch are the result of combining the research endeavors of Ulrich (1989, Ulrich and Seering, 1989) with Pahl and Beitz (1988). The work of Ulrich was first to realize the design potential of bond graphs, while Pahl and Beitz established a formalism of unique qualitative characteristics to define various facets of engineering design. The bond graph research was conceived by Paynter (1961) as a method of modeling various domains by a fundamental language of dynamic operators. While bond graphs originated as an analysis method, various other projects (Bracewell and Sharpe, 1996; Finger and Rinderle, 1989) have since realized the potential of the bond graph formalism as a foundation for design synthesis.

Function grammars are an emerging concept in design synthesis (see Schmidt, 1995; Fu, et al., 1993; and Pinilla, et al, 1989). These techniques create a formal language for generating and updating designs. Through the development of production rules, complex designs can be constructed from a simple initial specification, or “seed”. The artifacts of function grammars are often graphs (nodes and arcs) and rules formulated to add, remove, or modify elements of a graph. In the A-Design representation, a function grammar formalism is combined with the representation of Welch to handle a wide variety of design configurations.

4.2 BASIC COMPONENTS

The representation developed in A-Design is based on the description of individual components and the points of connectivity between the components. By describing points of connectivity within a design, a formal syntax is established to model the input-output

functionality of interacting components. The network of transformations resulting from the interactions of many components yields the overall functionality of a design. In this section, the two fundamental structures of the representation are introduced - the Embodiment and the Functional Parameter - as extensions of structures developed by Welch. In short, the Embodiment is the computational structure describing components, while the Functional Parameter describes the interaction between components.

4.2.1 Functional Parameter (FP)

At the interface between components, a structure referred to as the Functional Parameter (FP) provides qualitative and quantitative descriptions of electromechanical behavior. Figure 4.2 shows the contents of the Functional Parameter structure and the possible values variables can have. The first four slots in this structure represent the state variables at a given point in a design. The general terms, **through** and **across**, are used to represent important dynamic variables in a variety of electromechanical domains. Concepts like force, torque, current and flow rate are categorized as through variables

FP - Functional Parameter:	
Through	{number, range, <i>bound</i> , <i>unbound</i> }
Across-integral	{number, range, <i>bound</i> , <i>unbound</i> }
Across-none	{number, range, <i>bound</i> , <i>unbound</i> }
Across-differential	{number, range, <i>bound</i> , <i>unbound</i> }
Class	{ <i>power</i> , <i>signal</i> , <i>material</i> }
Domain	{ <i>translational</i> , <i>rotational</i> , <i>electrical</i> , <i>hydraulic</i> }
Coordinate	{[4 x 4] transformation matrix}
Interface	{standard size, e.g. 9/16" <i>bolt</i> }
Direction	{ <i>source</i> , <i>sink</i> }.

Figure 4.2: The contents of the FP structure.

because they are understood as passing “through” elements in the system. This is opposed to across variables such as velocity, voltage and pressure, which operate according to their difference “across” an element. Often, through and across variables also have a physical meaning when integrated or differentiated with respect to time. For example, velocity is integrated to find displacement and differentiated to find acceleration. Since these variables have physical meaning in many domains (i.e. electrical, mechanical, or hydraulic), they provide a systematic framework for reasoning about functionality as well as modeling unfamiliar domains.

The values for **through** and **across** slots can be either quantitative or qualitative. Each of these four slots can have a numeric value, a numeric range, the *bound* label, or the *unbound* label. The *bound* and *unbound* labels are used to qualify the behavior of a Functional Parameter before a design is fully realized. To classify a variable as *bound* means that the true numeric value, while unknown, will eventually converge to a single value. *Unbound* variables describe a diverging or unconstrained numeric value. Throughout the construction of a design, the *bound* and *unbound* values act as placeholders to inform agents of the behavior at points in an incomplete design. In addition, a *goal* prefix can be added to the through and across slots to differentiate an actual value from a desired value specified as part of the design description. The importance of these values in the context of design construction is detailed in Section 4.3

The **domain** variable in the FP structure describes the domain of the Functional Parameter. In the implementation, electromechanical design contains four domains: translational, rotational, electrical and hydraulic. Table 4.1 shows the physical

Table 4.1: Through and Across Variables for Each Domain

	Translational	Rotational	Electrical	Hydraulic
Through Variable	Force (f [Newtons])	Torque (T [N-m])	Current (I [amps])	flow rate (m [kg/s])
Across Variable	Velocity (v [m/s])	angular speed (Ω [rad./s])	Voltage (v [volts])	Pressure (P [N/m ²])
Through \propto d(Across)/dt	mass	Rotational inertia	capacitor	reservoir
Through \propto Across	Damper friction	Damper friction	resistor	Valve viscous drag
Through \propto \int (Across) dt	spring	Rotational spring	inductor coil	long piping

manifestation of the through and across variables for each domain. The table also depicts components within each domain that establish a proportional relationship between the through and across variables. Often domains have physical components for each transformation of the through variable to the across variables as can be seen by the components listed in the table.

Along with the **through**, **across** and **domain** variables, the FP structure includes a few more descriptors of the connections between components. The **class** characteristic derived from Pahl and Beitz provides a description of what is physically transferred through the connection. In this work and in the previous work of Welch, electromechanical designs are all described by the same class value (**class** = power). However, the hope is that other than describing electromechanical energy, one could also describe the flow of information (**class** = signal), or possibly even flow of a material (**class** = material). The Functional Parameter includes slots for describing the location of a connection point in space by storing the position in a **coordinate** transformation matrix. A description of the connection's **interface** with other components is described as well as the **direction** of flow at the interface.

As mentioned earlier, the basic manner of describing functionality to A-Design is a statement of the desired set of input and output behaviors. The Functional Parameter structure provides this method of describing design problems. In Figure 4.3, the weighing machine design problem that was previously specified by an input and output is now more formally represented by an FP_{input} and FP_{output} . Initially these FPs are specified by the user as the seed for the design problem. Specifications on the through and across variables in the figure have *goal* prefixes to denote their values are desired by the user and not actual values derived from the construction process.

4.2.2 Embodiment (EB)

Besides describing connecting points, the representation also describes components by a common structure known as the Embodiment (EB). The structure models

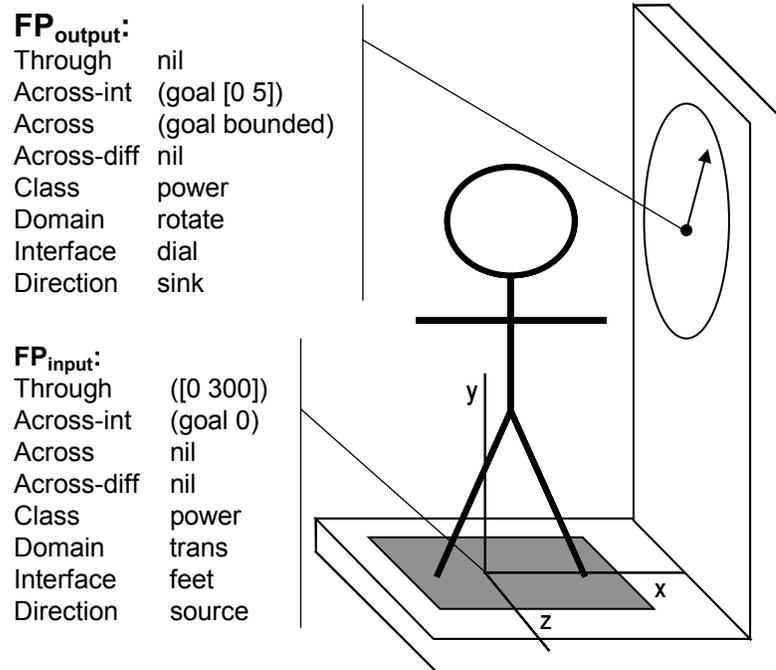


Figure 4.3: Weighing machine with input and outputs described as Functional Parameters.

components as ‘black boxes’ where the internal functions of a component are only evident in how the inputs and outputs of the component relate to one another. These descriptions are used to model how a class of components operate, not the specifics of a particular component. For example, a spring Embodiment has equations and constraints describing the general behavior of springs, but does not specify numerical values for variables such as spring stiffness.

This structure builds on the EB structure of Welch by including the following extensions: n-port components, non-linear transformations, and more thorough constraints on the connectivity with other components. However, the key innovation of the EB structure within A-Design is how it is used within the creation of new designs. The Configuration-agents (described in Section 5.3.1) attach Embodiments together to form an abstract configuration, and then these designs are instantiated with real components by the Instantiation-agents. This two level design generation method allows A-Design to explore design on two levels: determining the best topological connection of components, and determining the best components for such a configuration.

Figure 4.4 shows the makeup of the EB structure and an example of a gear Embodiment. The **variables** slot lists the names of the parameters of the abstract component, which are instantiated with actual values at the instantiation level. The **constraint parameters** (CPs) establish how components are constrained at their outputs thus preventing infeasible connections with other components in the system. A constraint parameter has the same variables as the Functional Parameter and can thus prevent improper matching of domain, interface, direction of flow, etc. The magnitude-change

(**MG-Change**) slot includes code describing the transformation between the state variables into and out of ports on the components. These include such formulas as “ $F=k \cdot x$ ” for springs, or possibly more complicated non-linear functions such as $F = K(x) \cdot x$. The MG-change slot contains the dynamic model of the Embodiment and is similar to how bond graph information is incorporated in previous work. Finally, the position-change (**PO-change**) slot lists transformation matrices for converting the coordinates of one port to another, and is thus used to evaluate the **coordinate** slot in the FPs connected to an Embodiment.

<u>EB - Embodiment:</u>	<u>EB - Gear:</u>
variables names of the dimensions or parameters of a component.	variables D _{gear} , D _{shaft} , Pitch
CPs constraint parameters - constrain which FPs can be connected.	CPs 1: class=power domain=rotation interface=shaft-hole(D _{shaft}) 2: class=power domain=translation interface=teeth(Pitch)
MG-Change magnitude change functions relating the through and across variables at each port to those of other ports.	MG-Change through1 = (through2 * D _{gear})/2 across-int1 = (across-int2 * 2)/D _{gear} across1 = (across-int2 * 2)/D _{gear} across-diff1 = (across-diff2 * 2)/D _{gear} through2 = (through1 * 2)/D _{gear} across-int2 = (across-int1 * D _{gear})/2 across2 = (across-int1 * D _{gear})/2 across-diff2 = (across-diff1 * D _{gear})/2
PO-Change position change matrix relating position of each port to the position of other ports.	PO-Change $\begin{bmatrix} 0 & \cos \phi & -\sin \phi & 0 \\ -1 & 0 & 0 & 0 \\ 0 & \sin \phi & \cos \phi & D_{\text{gear}} / 2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

(a)

(b)

Figure 4.4: The contents of the EB structure: (a) shows the general description of the slots of the EB and b) shows an example of a gear EB.

4.3 UPDATING DESIGNS: HANDLING INCOMPLETE CONFIGURATIONS

The A-Design representation of Functional Parameters and Embodiments includes a framework for constructing designs one Embodiment at a time. This allows for an interactive construction of designs where agents contribute to designs based on how previously agent contribution effect a design. When a Configuration-agent connects a new Embodiment to a partial design, the FPs throughout the system are updated. The notion of updating functionality in a design is based on Schmidt and Cagan (1998) where designs are created through a function grammar. The innovation in this research is to combine function grammars with the Functional Parameters of Welch to allow for a more general component connectivity and a more complete component description than previously achieved.

The difficulty in this approach is understanding how a design behaves even when it is not fully connected. In the previous work, no technique was developed to handle incomplete designs. Instead, design alternatives were conceived through an immediate expansion of the system's input and output behavior. As a result, only simple design states were constructed. The framework here for allowing agents to interact with partial design states allows for an infinite number of possible design configurations. Agents are not constrained to add components in series, parallel or any particular order, but instead follow their individual preferences for design. These agent strategies are addressed in detail in Chapter 5.

The weighing machine problem description shown in Figure 4.3 specifies the input and output FPs that describe the functionality of a device. At the output, the functionality of dial displacement is classified as a goal range of 0 to 5 in the across-integrated slot.

This range represents an angle of displacement since the integrated across variable in the rotational domain is an angle. The goal slots in the input and output FPs direct agents to achieve the desired functionality of a weighing machine. At the input FP, the domain variable is translational and therefore specifying the across-integrated slot to {goal 0} relates to zero displacement at this point. Also, the through variable has a range of {[0 300]} specifying it as an input force from zero to 300 pounds. Note that in the output's across variable slot the user has specified the values as {goal *bound*} since damped motion is also a desired requirement in the weighing machine design problem.

C-agents begin attaching various EBs to fulfill the *goal* slots in the original FP_{input} and FP_{output} . With the addition of a new EB into a configuration, the adjacent FPs need to be updated. The update mechanism follows the pseudo-code in Figure 4.5. This update assigns values to the slots of the FPs connecting to an EB and performs a recursive update of all the **through** and **across** variables in the design configuration. This recursive update changes *bound* and *unbound* flags in the **through** and **across** slots and checks to see if *goal* slots are met. The following example further explains how the recursive update mechanism and how complete configurations are assembled from incomplete configurations.

Figure 4.6a shows a partial design state with Functional Parameters represented as ovals and Embodiments as rectangles. The configuration presently contains seven EBs as a result of previous C-agent contributions. The FP_{input} and FP_{output} have changed slightly from those shown in Figure 4.3. With the introduction of every new EB to a design, an update mechanism makes changes to both the FPs adjacent to the new Embodiment and

possibly to FPs throughout the configuration. In the incomplete configuration of Figure 4.6, the path connecting input to output leads to a change throughout the design. The through variable range in the input of [0 300] causes the through slots in all FPs in the system to update from *unbound*, which is the default value, to *bound*. Furthermore, due to friction in the system, notably in the bearing, the across variable in the output FP specifying damped motion in the dial has changed from {*goal bound*} to *bound* from

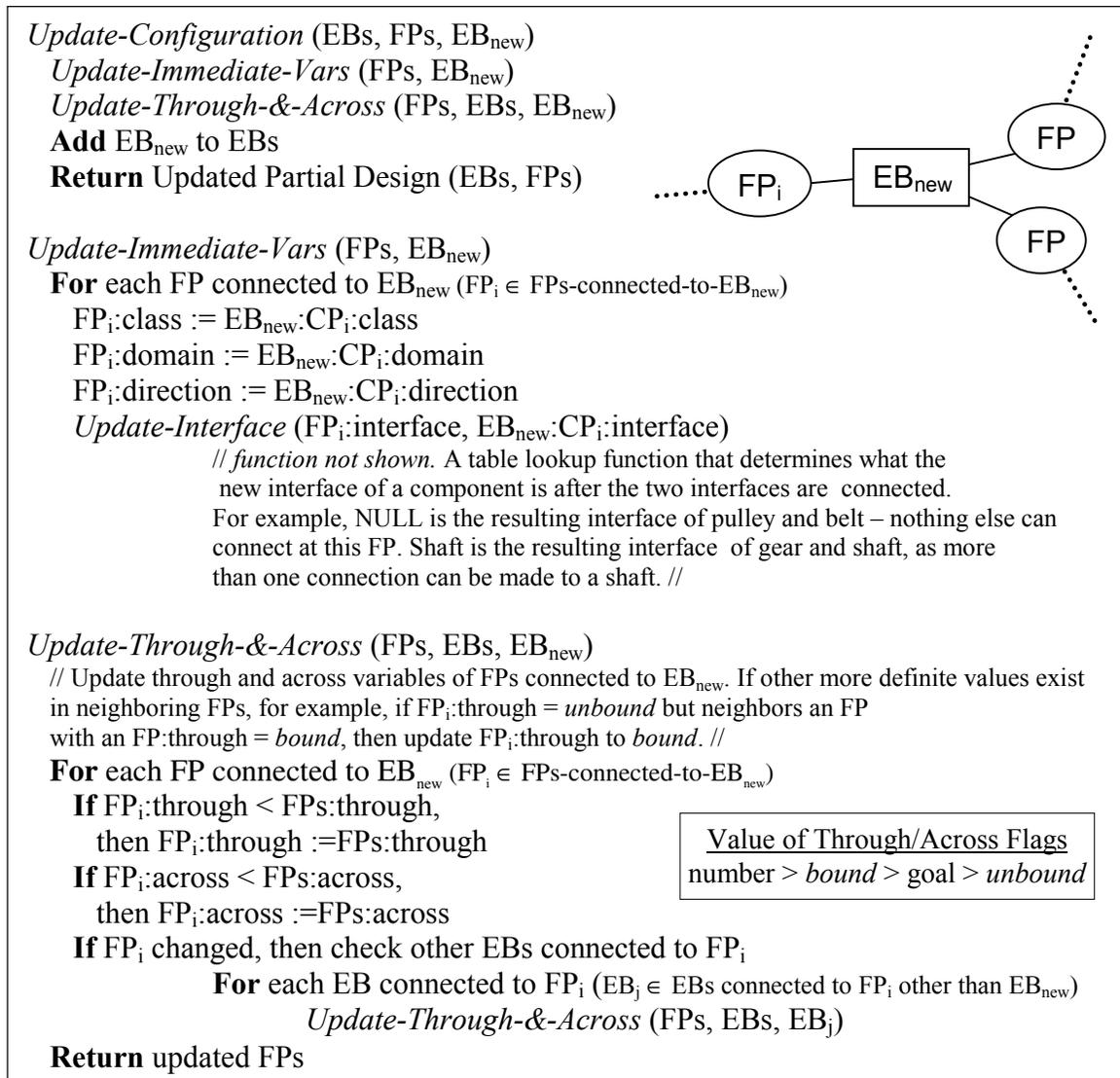


Figure 4.5: Pseudo-code for updating partial design state.

Figure 4.3 to Figure 4.6a. One of the three goal specifications is now fulfilled by the removal of this goal flag.

Two goal statements are still present in Figure 4.6a's description of FP_{input} and FP_{output} ($\{goal [0 5]\}$ and $\{goal 0\}$). When a new spring EB is added to the design between the lever and rack and ground, as seen in Figure 4.6b, a new design configuration is achieved. Information from this new connection must be propagated throughout the

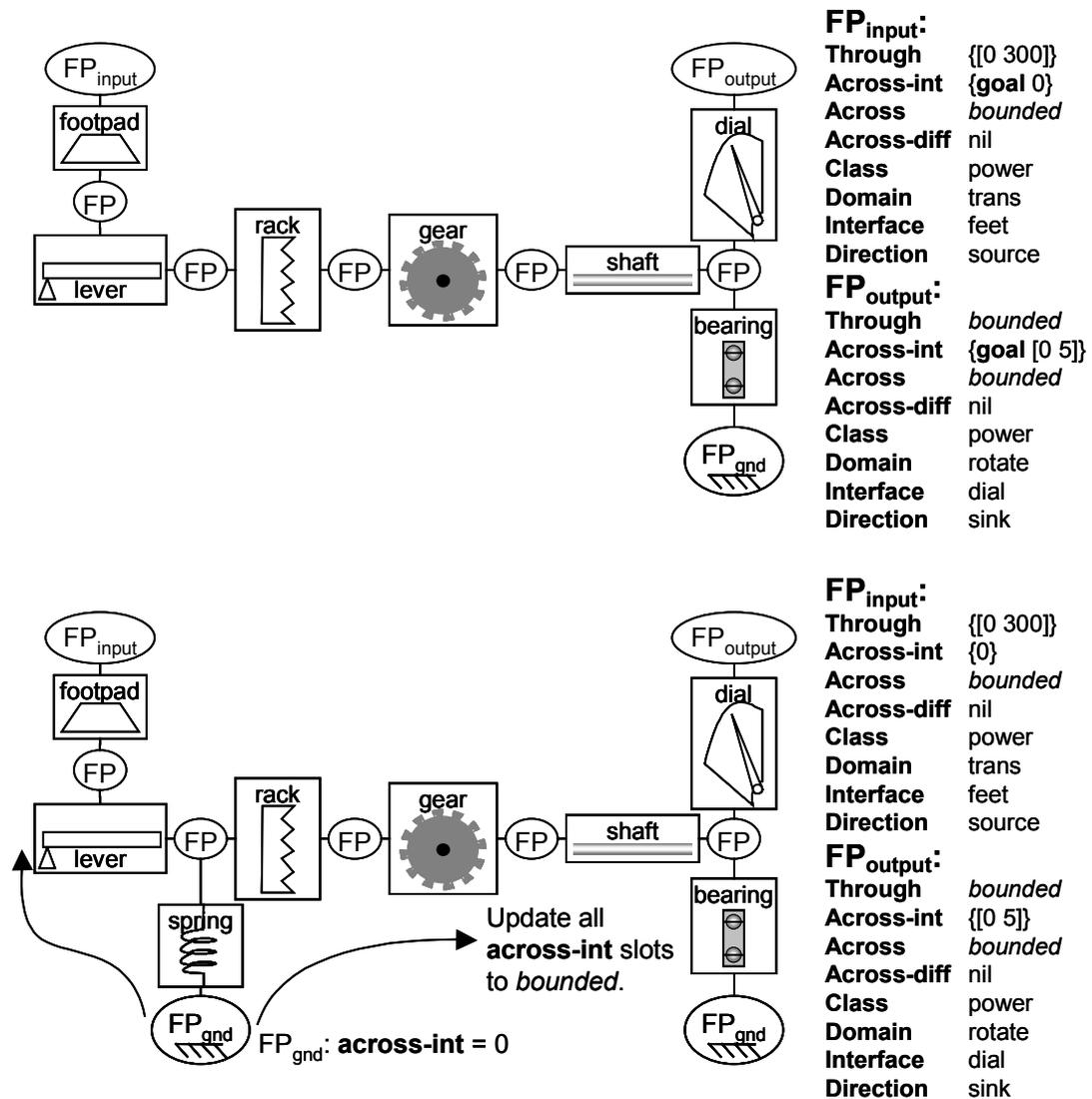


Figure 4.6: a) Partial design state with update FPs, b) Design is completed with addition of spring.

design. The recursion in the update mechanism transfers knowledge from the ground FP on the spring throughout the design to note that **across-integrated** is now *bound*. This bound variable is propagated throughout the design. The remaining goal ranges ($\{goal [0\ 5]\}$ and $\{goal\ 0\}$) represent bound values, and thus are fulfilled by this propagation. As a result, both *goal* flags are removed, thereby completing this design configuration.

As shown by this example, the update mechanism is quite intricate. The propagation of variable values allows the system to monitor the progress of the design from initial specifications to completed design. This occurs mainly as a result of propagating qualitative values such as *bound* and *unbound* to keep track of the effects of components as they are introduced into designs. These labels are also used by the agents to identify the best way to add new components to a design.

4.4 EXTRACTING EQUATIONS

After a design configuration is completed, symbolic equations are extracted to determine the analytical effect that the configuration of Embodiments has on the behavior of the design. A behavioral equation is constructed for each goal variable in the original FP_{input} and FP_{output} . For example, the equations for the *goal* dial angle (θ_{dial} is $FP_{output}:across-int = \{goal [0\ 5]\}$) and *goal* input displacement (x_{input} is $FP_{input}:across-int = \{goal\ 0\}$) from Figure 4.6b are found to be:

$$\theta_{dial} = \frac{1}{r_{gear} k_{spring}} \frac{d_1}{(d_1 + d_2)} F_{weight} \quad (2)$$

$$x_{input} = \frac{1}{k_{spring}} \frac{d_1^2}{(d_1 + d_2)^2} F_{weight} \quad (3)$$

where $\theta_{dial} = [0 \ 5]$, $x_{input} = 0$ and $F_{weight} = [0 \ 300]$.

These equations are formed by extracting key information about the connections in the system: r_{gear} is the radius for the gear EB, k_{spring} is the stiffness term from the spring EB and d_1 and d_2 are length terms extracted from the second class lever EB. These equations are determined by a computational recursive algorithm similar to the update mechanism for **through** and **across** variables shown in the previous section. The equation extractor constructs symbolic equations for the **through** and **across** variables similar to how bond graph systems derive behavior. The technique starts at each goal value in a design and works back to given data, such as the input weight ($F_{weight} = [0 \ 300]$) or zero values in the ground FPs. The equation extractor retrieves information stored in the Embodiments' magnitude-change (**MG-change**) functions, which describe how the Embodiments interact with neighboring Functional Parameters, and combines these interactions to form complete equations. By performing a depth first search through the graph of connecting FP nodes, the extractor determines **across** and **through** variable transformations for series and parallel connections. As a result, the process is capable of finding equations for a wide variety of design configurations including all series and parallel branchings, and non-linear transformations between Functional Parameters.

The equations extracted for each design have two purposes later in the process. First, the equations are referenced by the Instantiation-agents in their choosing actual components for the Embodiments in a design (furthered explained in Section 5.3.2). Second, the equations are used to determine the values for performance-based objectives in the design problem. In the previous chapter, the weighing machine problem was shown with four objectives, two of which are direct results of the equations shown here: minimize dial error and minimize input displacement. In Equations 2 and 3, the values on the left side of the equations are specified by the user ($\theta_{\text{dial}} = [0 \ 5]$ and $x_{\text{input}} = 0$). The right sides of the equations are determined for each design in the search process using this equation extraction. The values for the objectives are determined by the amount of mismatch in these equations. The closer that the left hand side of Equations 2 and 3 are to the behavior predicted on the right hand side, the better the system conforms with the user specifications.

4.5 CATALOG OF COMPONENTS

As seen in the flowchart in Figure 2.2, design states are first configured by the Configuration-agents, then equations are extracted, and finally, designs are instantiated with real components. At first, designs are configured using the Embodiment information from the catalog, so that specific component values do not create unnecessary constraints for the configuration. The catalog of components thus maintains both the abstract Embodiment data and data on real components. For electromechanical design, the catalog contains the 32 Embodiments shown in Table 4.2.

Table 4.2: Current Embodiments implemented in the system

Battery	Lever (class 3)	Resistor	Solenoid
Cable	Motor	Rotational Bearing	Spring
Capacitor	Pipe	Rotational Damper	Sprocket
Gear	Piston	Rotational valve	Stopper
Electrical valve	Potentiometer	Switch	Transistor
Inductor coil	Pulley	Tank	Translational Bearing
Lever (class 1)	Rack	Torsional Spring	Translational Damper
Lever (class 2)	Relay	Shaft	Worm gear

The main tasks of the Instantiation-agents to be discussed further in Chapter 5 is to lookup each Embodiment in a configuration and find a suitable component to fulfill that Embodiment. The library is a directory of files that contains these abstract and concrete component descriptions. There are over 300 components in all for instantiating the 32 Embodiments. Figure 4.7 shows how a gear is represented in the file “gear.comps”. The figure shows actual gear components, and their values for diameter, teeth pitch, cost, etc.

```

Components    ::: Gear.comps
               ::: This file contains information on gear components.
               ::: It contains two lists: one instantiating the variables
               ::: in the gear EB, and another with evaluable criteria
               ::: Components have the format:
               ::: (name      (Dia [m], shaft-Dia [m], teeth, Pitch [teeth/in.])
               :::          (cost [$], mass [kg.], efficiency))
               ::: The following gears are from the Nordex catalog p. 286
               (LAS-F7-28 (11.1e-3 6.35e-3 28 64) (5.75 5e-3 0.98))
               (LAS-F7-36 (14.2e-3 6.35e-3 36 64) (6.29 10e-3 0.98))
               (LAS-F7-46 (18.3e-3 6.35e-3 46 64) (6.66 14e-3 0.98))
               (LAS-F7-60 (23.8e-3 6.35e-3 60 64) (7.54 65e-3 0.98))
               (LAS-F7-75 (29.8e-3 6.35e-3 75 64) (8.18 17e-3 0.98))
               (LAS-F7-90 (35.7e-3 6.35e-3 90 64) (9.40 22e-3 0.98))
               (LAS-F7-104 (41.3e-3 6.35e-3 104 64) (10.26 25e-3 0.98))
               (LAS-F7-128 (50.8e-3 6.35e-3 128 64) (12.03 33e-3 0.98))
               (LAS-F7-168 (66.7e-3 6.35e-3 168 64) (15.69 35e-3 0.98))
               (LAS-F7-208 (82.6e-3 6.35e-3 208 64) (21.99 45e-3 0.98))
               (LAS-F7-248 (98.4e-3 6.35e-3 248 64) (24.74 50e-3 0.98))
    
```

Figure 4.7: Gear Embodiment with sample of gear component file: gear.comps.

4.6 SYSTEM CONFIGURATION

A completed design state enters the evaluation phase of the process with a topological connection of Embodiments and Functional Parameters, a list of components instantiating these Embodiments, and a list of all the agents that contribute to the design. Upon evaluation, the design also contains values for all the objectives specified in the design problem. This total set makes up the System Configuration (SC) structure as seen in Figure 4.8. These SCs are the representations of the design states that are transported throughout the system among the various agents and other functional modules.

4.7 COMPARISON WITH PREVIOUS METHODS

Of the four subsystems of A-Design, the functional representation shown here is most

SC – System Configuration:	
graph	List of all the FPs in the design. Each is indexed to the one or more EBs that it connects.
EBs	List of Embodiment names that make up a design. (gear dial spring rack ...)
Comps	List of components that makes a one-to-one with the EB list.
C-agents	List of C-agents – also makes a one-to-one match with EB list.
I-agents	List of I-agents – makes a one-to-one match with Comps list.
F-agents	List of F-agents that have fragmented the design in past iterations
Evaluations	List of attribute values for the design problem. e.g. for attributes (cost weight efficiency).

Figure 4.8: The contents of the SC structure.

similar to related work. The terminology in the A-Design representation borrows directly from the research of Welch, and the method for updating designs is akin to that of Schmidt and Cagan. Just as human design is unstructured or open-ended in the development of design concepts, the A-Design approach and the related approaches do not restrict the design solutions to a set topology. These representations are structured to address a wide variety of solutions for fulfilling a given conceptual design problem. However, by building on these previous techniques, the A-Design representation exhibits several unique innovations over the previous work.

The A-Design representation does not confine the complexity of design configurations. Any number of series and parallel connections can be made to instantiate a design, as opposed to the previous work of Welch and Ulrich, where series and parallel connections were limited. Their approaches involved creating configurations through a bond graph formalism described as a power spine. Similar to A-Design, the power spine fulfilled the description of a specific input/output behavior of the design problem. Configurations, or power spines, were created from a tree expansion starting with the most simple design state possible (a direct connection from input to output). Levels of the tree elaborate upon this simple design, but are bound by simple modification rules (or rewrite rules in Ulrich). The resulting designs from these techniques are a direct connection from input to output with components either in series or parallel with this direct connection.

A-Design's ability to comprehend incomplete designs allows for a much broader space of possible designs. The Functional Parameter formalism along with the function

grammar update mechanism allows the construction of designs to occur without constraint. By tracking the goals of the design problem, agents are given the freedom to connect components anywhere in a configuration as opposed to only along a direct path from input to output.

The A-Design representation also provides a more complete representation of design states than the previous work. Welch and Ulrich saw the interesting design activity occurring at a phenomenological level. In their model, this level is a step before Embodiment design. Figure 4.9a shows the view of conceptual design developed by Welch. This approach did not instantiate components in designs, since it stopped at the Embodiment level. The approaches of Welch and Ulrich first build dynamic models with a bond graph at the phenomenological level, then embody these bond graphs with component models.

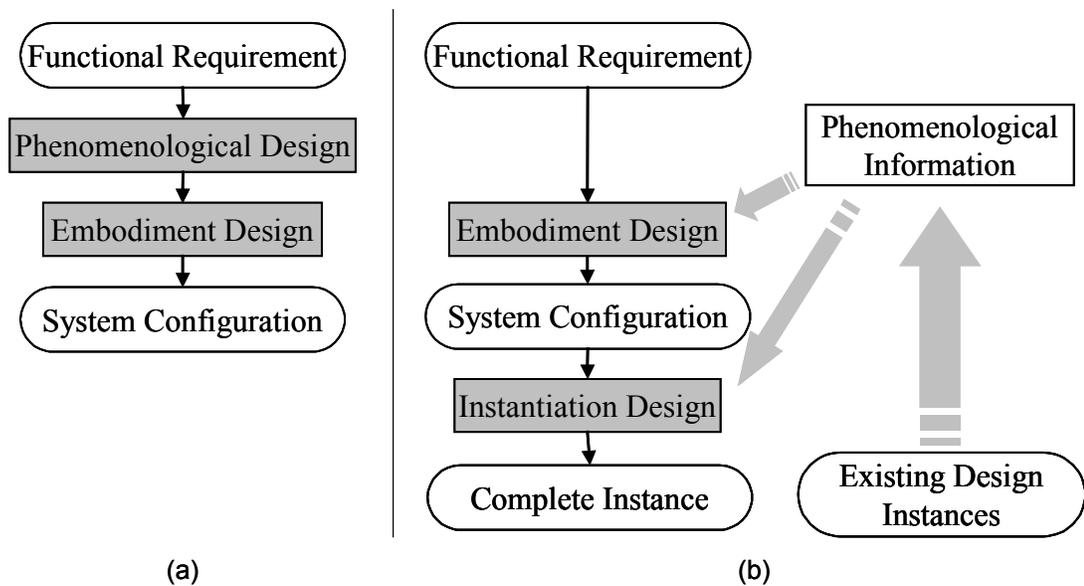


Figure 4.9: Views of the conceptual design process : a) the view established in Welch and Ulrich, b) the A-Design view.

In the A-Design model, design occurs at both the Embodiment level and the Instantiation level and not a Phenomenological level. The viewpoint taken in A-Design is that human design occurs with real components in mind and not phenomena like Hooke's Law or Kirchoff's Law as is suggested by these predecessors. Figure 4.9b shows the A-Design view of conceptual design. In this figure, a store of phenomenological information can be referenced to guide the design process at the Embodiment level, but the Embodiment level is not a result of prior design stages. Observations made from real design processes suggest that people think on a component level rather than a behavior level when solving design problems. For example, a designer is more likely to say, "We should put a spring in the design", rather than "We should use Hooke's Law to solve this design functionality". It is possible that a store of phenomenological data is used in making design decisions, but contrary to previous approaches, this data is auxiliary to the design process.

Of course, that is not to say that this information is not essential to design. The learning that guides the construction of new alternatives is an invaluable source of design knowledge, especially if such knowledge is distilled to basic principles or phenomena. In the A-Design model of Figure 4.9b, the storing of past design similarities is available to agents in the construction of designs at both the Embodiment level and the Instantiation level. In Chapter 6, the method for creating this store of information from past design instances is described.

The construction of the Welch and Ulrich model might result from the traditional use of bond graphs. Bond graphs were created to model actual dynamic systems to determine

underlying phenomena (i.e. to extract phenomenological behavior from design instances as in Figure 4.9b). These previous techniques have added additional concepts to bond graphs to allow them to design as well as analyze. As a result, the conceptual design process in Figure 4.9a is founded on and confined to the bond graphs analysis approach, as opposed to corresponding with the actual human design process.

In addition, problems occurred in Welch and in Ulrich after conceiving phenomenological design concepts. Fulfilling the “power spine” phenomena with a design at the Embodiment level created some problems. This realization from phenomena to embodiment occurred by hand in Ulrich and computationally in Welch. The idealized bond graph elements could not easily be matched to embodiments without introducing new behaviors intrinsic to real components. In addition, there was a degree of variability in the configurations that resulted from the bond graph “power spines”. To handle this, Welch’s system develops several possible configurations for a single bond graph representation. The realization from bond graph to physical configuration in Ulrich occurred with some creative license; Welch referred to Ulrich’s design configurations as a “quantum leap”. Because this realization from phenomena to design configuration is so problematic and unnatural to design, the A-Design approach constructs designs by directly configuring Embodiments and components. The iteration of the search process then provides for the discovery of successful designs and useful phenomenological information extracted from such designs.

Chapter 5

Agents Architecture

Main Entry: **col·lab·o·rate**

Pronunciation: k&- 'lɑ-b&- "rAt

Function: *intransitive verb*

Inflected Form(s): **-rat·ed; -rat·ing**

Etymology: Late Latin *collaboratus*, past participle of *collaborare* to labor together, from Latin *com-* + *laborare* to labor

Date: 1871

1 : to work jointly with others or together especially in an intellectual endeavor

2 : to cooperate with an agency or instrumentality with which one is not immediately connected

- **col·lab·o·ra·tion** /- "lɑ-b&- 'rA-sh&n/ *noun*

- **col·lab·o·ra·tive** /- 'lɑ-b&- "rA-tiv, -b (&-) r&-/ *adjective or noun*

The agent subsystem is the heart of the A-Design theory. A population of diverse agents provides a way of combining stochastic search algorithms and knowledge-driven methods. The *collaborative* exchange of designs among various agents achieves a sense of parallel execution. This interaction produces a more robust process capable of solving the conceptual design problem with more variety. Furthermore, the combination of agents and stochastic search allows one to tailor the representation to best embody the complexity of the design space. In order to bridge the search process and the functional representation, the agents are constructed in response to the specifications of these subsystems. This allows the representation to be developed without restriction.

Agents follow knowledge-driven strategies in adding or subtracting elements to designs. These strategies follow various preferences agents have for understanding the functional representation, and for solving the goal specifications of the input and output Functional Parameters. Because these knowledge-driven, goal-directed agents are deterministic, A-Design stochastically chooses agents to prevent stagnation at local minima. In addition, the agents are augmented by a learning strategy to hone their understanding of how to best overcome the unique challenges specific to each conceptual design problem.

5.1 RELATED WORK

The concept of cooperating agents has had a growing acceptance as a method for complex computation beginning with Artificial Life research (Langton, 1988; Holland, 1992). Artificial Life, or A-Life, seeks to understand how natural systems perform complex tasks as a result of interacting naïve agents. The highly distributed and parallel emergent behavior from these simple interactions leads to a robust and efficient process.

Such multi-agent systems have previously been applied in a number of engineering design applications (see overview by Lander, 1997). The approach in many of these engineering applications is to use agents to handle the pre- and post-processing of various computational analysis tools such as spreadsheets or CAD systems. By communicating through a common framework, the agents, which, act as “experts” in representing the results produced by their encapsulated applications (see example in Goldstein, 1994). Currently, these approaches offer an unobtrusive means to communicate between computer tools used in concurrent engineering design. Other research projects in this area

are incorporating reasoning and learning (Grecu and Brown, 1996) into agents in order to more closely simulate strategies used by human designers.

Asynchronous Teams, or A-Teams, (Talukdar, 1996) is a computational design methodology that combines design utilities such as optimization techniques with autonomous agents. These agents perform computations independently from other agents and contribute their results in a parallel and distributed fashion. It is believed that combining programs in a collaborative yet anarchistic manner will produce better results than if programs were executed individually. Talukdar (1993) describes this as synergy: “When the effectiveness of cooperation is so great that a super-object is, in some sense, greater than the sum of its parts, the cooperation is synergistic.” The philosophy of the agent-based approach of A-Design is similar to that of A-Teams. Note that A-Teams have also explored the use of a multi-objective selection of designs as seen in Murthy (1992) and Quadrel et al. (1993).

5.2 THE A-DESIGN AGENT

The creation of any one design in A-Design is due to collaboration among several different agents. These agents contain knowledge of how to design based on their individual strategies and preferences. They are constructed to understand the representation of a design state and contribute in a manner that leads to successful solutions. The strategies used by these agents are based on deterministic algorithms, which are triggered by the head Manager-agent in the current implementation. A simple hierarchy exists where the Manager-agent invokes the operations of the other agent classes: the Maker- and Modification-agents. Within the Maker- and Modification-agent classes, subclasses can exist as well, and within these subclasses different agent types are

expanded to populations of agents. For example, the electromechanical A-Design approach has two subclasses of Maker-agents, Configuration-agents (C-agents) and Instantiation-agents (I-agents); there are 48 different C-agent types and 18 different I-agent types; and of the 48 C-agent types, there are four agents per agent-type.

The Maker-agents have two responsibilities: create design alternatives based on the problem description and re-build designs returned by the Modification-agents. Construction of a complete design state is accomplished by several Maker-agents adding their distinctive parts to make a complete design alternative. The Modification-agents are active at the end of the evaluation phase of the process (see Figure 2.2) to take design states from the Pareto and Good populations and improve them based on how they were evaluated. These agents allow the process to move from current design states to possibly better ones and thus produce a better approximation of the true Pareto-optimal front.

The Manager-agent makes observations about the population of designs and the population of agents. This agent also communicates with the user to understand what objectives in the problem are most important (as discussed in Section 3.4). While the Maker- and Modification-agents can be viewed as adding a directed and deterministic nature to A-Design, the Manager-agent brings a stochastic and learning aspect to the process. Initially, the Manager-agent randomly invokes the Maker- and Modification-agents but as the process progresses and data is gathered about each agent's success rate more directed agent calls are made. The interaction of the various agent types is displayed in Figure 5.1. Recall that some of the design tasks happen independent of agent interaction. These include the evaluation of designs, the sorting of designs, the updating

of incomplete design states and a handful of bookkeeping functions. These tasks do not fit into our definition of an agent, detailed below, because they do not happen as a collaboration of uniquely defined directed strategies and do not directly deal with the uncertain design issues.

5.2.1 Definition of Agent

There is some ambiguity in computational research over the proper use of the word “agent”. Our use of agent is consistent with the definition of Russell and Norvig (1995) where agents are viewed as perceiving their environment through *sensors* and acting upon their environment through *effectors*, as is illustrated in Figure 5.2. Within the electromechanical A-Design system, agents are implemented as independent functions that perceive an environment of design states, and through some decision-making effect

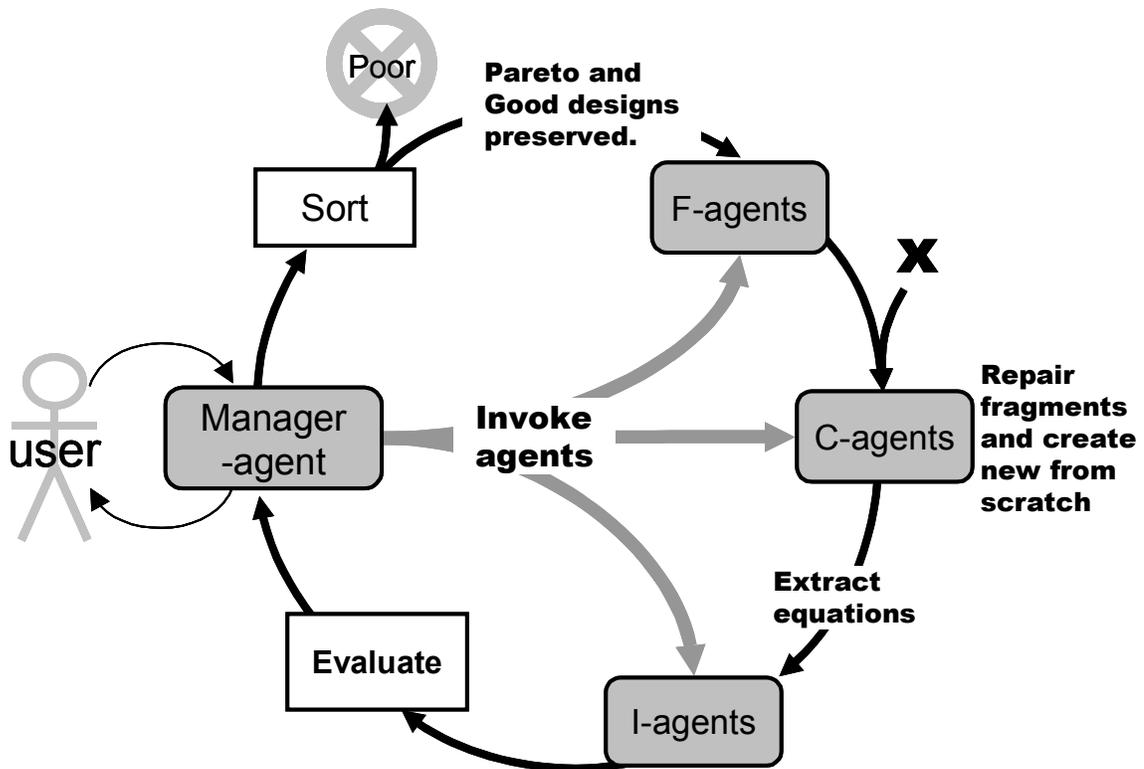


Figure 5.1: The interaction of the agents occurs with the designs and the user.

this environment by modifying a partial or completed design state.

While some might argue that without displaying specific behaviors of autonomy, mobility, or sociability (Sycara, 1998; Franklin and Graesser, 1996), the agents in A-Design do not fully conform to the definition of “agency”. However, within the Artificial Life community agents often are defined as simple strategies that, when combined, lead to more complex emergent behaviors (Langton, 1988). Similarly, our agents are defined as *knowledge-driven strategies for solving open-ended problems that, when collaboratively combined with other similar strategies, leads to a more complex and often emergent behavior for achieving a design goal*. For example, in designing travel paths between two set destinations, one can imagine different agents to accomplish the goal. One agent might try to determine the best path for driving while another might look at possible flight paths for connecting the start and end points. The various agents would accomplish the same goal but in different manners. Because no single strategy is best for solving this design task, we refer to such strategies as fulfilling a “disputed”

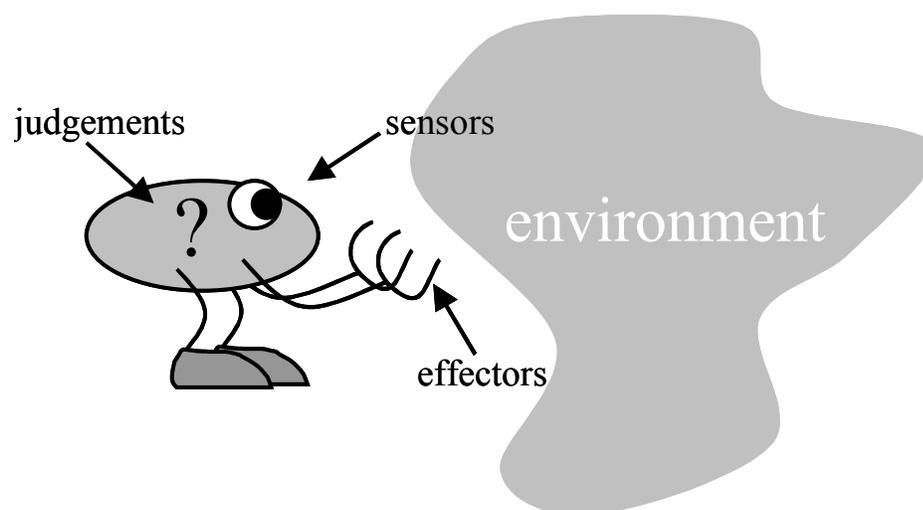


Figure 5.2: An agent is defined by its sensors, effectors and judgments.

functionality. If more than one approach exists to solve a problem then the various “disputed” strategies can be enumerated as computational design agents as is shown in the agent strategies in this chapter.

Because, in design, there is no single answer, different agents working on the same design problem can generate completely different solutions. By having agents with different abilities and preferences contributing to designs, the process gains robustness and variety in solving various conceptual design problems. This manner of working together on a problem is known as collaborative agent interaction. There are other multi-agent strategies, especially in the artificial life area where agents interact either *cooperatively* (Hoffman and Waring, 1996; Reynolds, 1987) or *competitively* (Steels, 1996; Hillis, 1991). In cooperative and competitive multi-agent systems, agents act on an environment consisting mainly of other agents. Depending on the goals of the agents, the interactions are either beneficial or detrimental to the other agents. Beneficial agent interactions cooperate to achieve a common goal, whereas detrimental agent interactions are usually the result of a competition for survival or resources. In this work, the agents’ environment consists mainly of designs and not other agents. The actions of any one agent are not detrimental to other agents and thus the system is not competitive, and since the agents are not directly aiding one another, the interaction is not cooperative. The agents interact indirectly through the common goal of producing successful designs, and as a result in this work and in related projects such as A-Teams research (notably Talukdar, 1999; Talukdar, 1998) the interaction is labeled *collaborative*. This collaborative interaction in design is analogous to a collaborating team of engineering designers.

5.2.2 Utility agent model

The underlying premise of the utility agent strategies is the concept that agents are trying to maximize their utility in their interactions (i.e. creation or reconstruction) with their environment (i.e. design state). What occurs inside each agent is, in a sense, what occurs in the system's model of the user. The utility agent model claims that the effectors of an agent lead to new states in the environment and such states can be predicted and mapped onto real numbers representing an agent's utility. Figure 5.3 shows the strategy used by utility agents in the context of design. Each agent has a unique evaluation function that is a combination of terms from individual agent preferences and outside learning. In the general utility agent model, a local search is performed over all possible actions that the agent can carry out. Each action is evaluated by the agent and a choice is

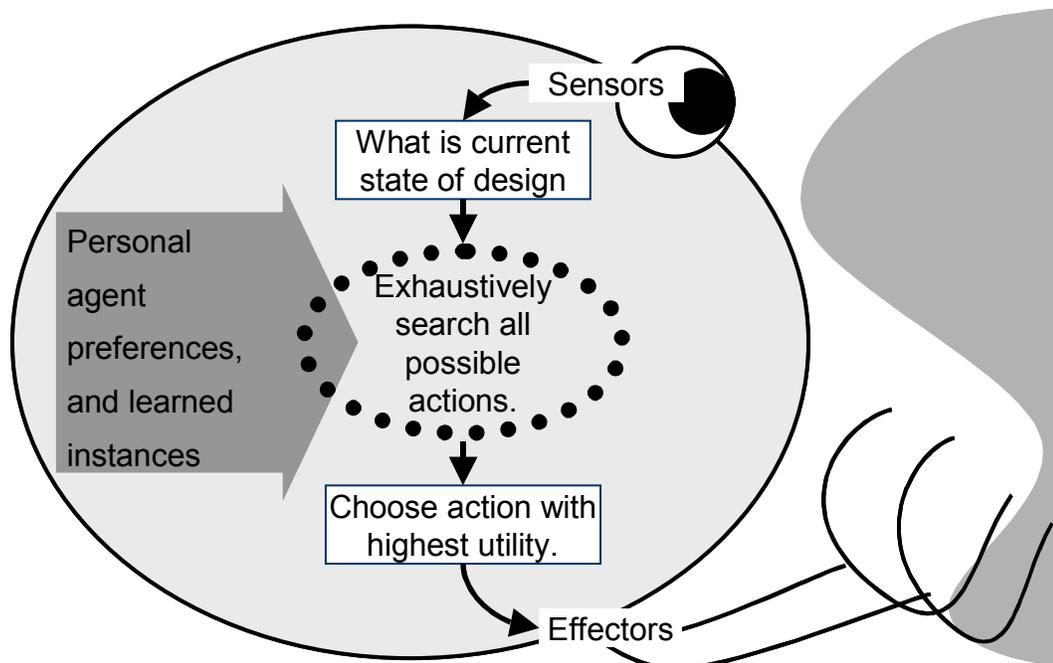


Figure 5.3: Utility agent tests all possible actions before performing the action with the highest utility.

made based on the action that maximizes the agent's evaluation function. In the following sections, an example of how the utility agent strategy is realized in A-Design is presented.

5.3 MAKER-AGENTS

5.3.1 Configuration-Agents (C-Agents)

C-agents are, by far, the most involved and interesting of the agent types. They have the ability to take the user-defined inputs and outputs and build designs to achieve the functionality specified in the design problem. In doing so, C-agents determine what portion of a configuration to address, and what Embodiments to connect to the configuration. For this reason, their operation is closely linked to the functional representation described in Chapter 4. As a utility agent, the C-agents perceive the current state of the environment and search for actions that lead to the most desired outcome. In the case of electromechanical design, the agent searches among possible EBs in the catalog to find one that leads to a new design state that maximizes the agent's evaluation function. In any given design configuration, each Embodiment represents the unique contribution of a specific C-agent. As a result of the different C-agent behaviors and combination of these behaviors an infinite variety of configurations can be created.

Initially, C-agents examine a partial design state to determine where to attach a new Embodiment. In an incomplete design state like that found in Figure 5.4, a new Embodiment can be connected to any of the Functional Parameters present in the system. The number of connection possibilities is not combinatorially large. Each C-agent exhaustively tests all 32 EBs (shown in Table 4.2) on all possible connection points in the

system and commits to the EB and target FP connection that maximizes the utility of the agent.

C-agents are created to encompass various preferences that can be envisioned in adding new Embodiments to a design. Table 5.1 shows the various preferences that characterize agents within each agent type. In the C-agent type, agents exist for each combination of characteristics in the first column. For example, some agents prefer electrical Embodiments while others prefer rotational, translational, or hydraulic Embodiments. As a result of this preference, an agent preferring electrical components will score connections made to “electrical” FPs higher than other connection types. This same preference might also elicit the introduction of a motor’s rotational connection into a system that has no electrical connections present, in order to introduce electrical FPs into the design state.

In addition to the domain preference, the decision-making for other preferences in Table 5.1 operates similarly. Agents can prefer connecting to FPs that are supplying energy (FP:direction = source) or receiving energy (FP:direction = sink). This is followed

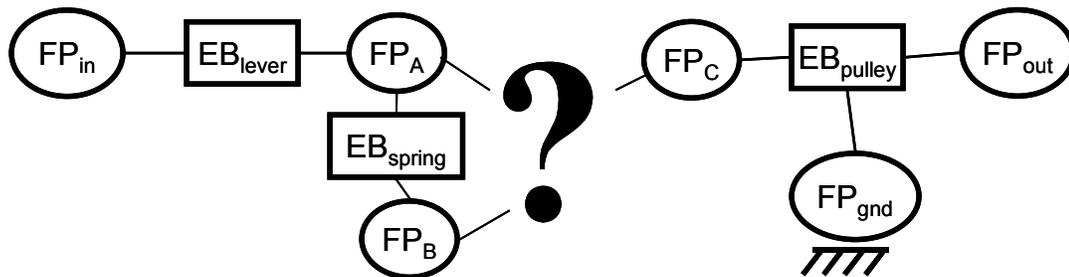


Figure 5.4: The C-agent's perception of an incomplete design state.

Table 5.1: Different preferences that exist in the current set of C-, I- and F-agents.

C-Agents (48 agents)	I-Agents (18 agents)	F-Agents (72 agents)
Domain Preference electrical EBs translational EBs rotational EBs hydraulic EBs	Objective Preference prefer inexpensive components prefer lightweight components prefer efficient components	Design Preference modify expensive designs modify inexpensive designs modify heavy designs modify light designs
Parallel vs. Series connect EBs in parallel connect in series	Variable Preference select component based on variables present in behavioral equations select component based on variables present in FP interfaces	Objective Preference remove expensive components remove heavy components remove inefficient components
Source vs. Sink connect to source FPs connect to sink FPs	Value for Instantiated Variable High Medium Low	Degree of Fragmentation remove component from design remove EB from design
Other EB connections link new EB to existing FPs link new EB to new FP link new EB to ground FP		EB Fragmentation remove repeated EBs remove dangling EBs remove ground connections

by another preference that chooses to connect Embodiments in parallel or series with existing EBs. Finally after an Embodiment is connected, agents have a preference for how to connect the remaining ports of the new Embodiment. These remaining ports can be linked to Functional Parameters currently in the configuration, can be used to establish new FPs, or can be connected to ground FPs. Each C-agent has all four preference types, and so as a result of all possible combinations of these preferences, 48 unique C-agent strategies can be constructed. As an example, one of these agent types is known as “C-agent-electrical-parallel-source-existing”, meaning the C-agent has a preference for

electrical Embodiments, connected in parallel, from source FPs, to existing Functional Parameters in a design.

In addition to these unique preferences, every C-agent's evaluation function contains terms to account for EB connections that complete a design goal. As discussed in Chapter 4, the initial input and output FPs have goal flags that act as placeholders in distinguishing actual values from those that are desired by the user. In testing candidate Embodiments, outcomes fulfilling goal specifications are emphasized, so that agents are also striving to solve the design problem as well as meet their personal preference.

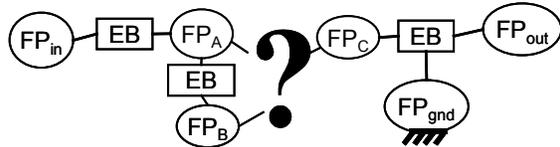
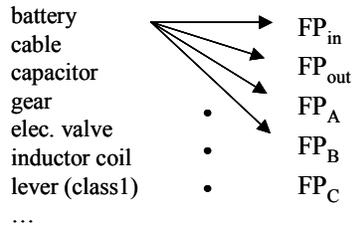
Finally, the agent model includes a method of learning. Chapter 6 examines how the Manager-agent finds both useful and detrimental commonalities in the designs of past iterations. The TODO list includes positive commonalities, while the TABOO list includes poor commonalities. So, in addition, to personal preferences, and strategies for fulfilling goal specifications, C-agents are also advised by the presence of good and bad past design exemplars. This method of guiding agent decisions is similar to the multi-agent reinforcement learning approach in Tan (1993). The evaluation function of the C-agent is rewarded when the addition of a new EB fulfills a design fragment that is on the TODO list, and conversely penalized for fulfilling subsystems of the TABOO list.

In Figure 5.5, pseudo-code is presented to clarify how the Configuration-agents operate. The utility functions, or evaluation functions, of the C-agents include five terms: the value of the Functional Parameter (V_{FP}), the value of the Embodiment (V_{EB}), the value of the other EB connections (V_C), and the two reinforcement learning terms (*TODO*, and *TABOO*). These values are incremented by one every time an agent's

Sensors = incomplete design state, TODO, TABOO

1. Check every EB with every FP in the system.

// For example, from Figure 5.4, check every EB from Table 4.2 with every FP in the configuration. //



a. Initialize values to zero ($V_{FP} = V_{EB} = V_C = TODO = TABOO = 0$).

b. What FP to connect to? What is the value of each FP? (V_{FP})

$V_{FP} := V_{FP} + 1$ for meeting series or parallel preference

$V_{FP} := V_{FP} + 1$ for meeting direction preference (source vs. sink)

$V_{FP} := V_{FP} + 1$ for meeting domain preference

$V_{FP} := V_{FP} + 1$ for containing a goal state (through or across goal)

c. What is the value of each EB? (V_{EB})

$V_{EB} := V_{EB} + 1$ for meeting domain preference

$V_{EB} := V_{EB} + 1$ for fulfilling goal state (through or across goal)

d. What to do with remaining EB ports? What is value of connections (V_C)?

$V_C := V_C + 1$ for proper connection type (existing vs. ground vs. new)

$V_C := V_C + 1$ for meeting series or parallel preference

$V_C := V_C + 1$ for fulfilling goal state (through or across goal)

e. Does this connection fulfill a TODO or TABOO fragment?

$TODO :=$ number of good designs fragment found in (if none then = 0).

$TABOO :=$ number of bad designs fragment found in (if none then = 0).

f. Calculate evaluation function

$$V = w_1 \times V_{FP} + w_2 \times V_{EB} + w_3 \times V_C + w_4 \times TODO - w_5 \times TABOO$$

2. Choose new EB connection producing the highest value of V

Effectors = design state (new-EB, connecting-FPs)

Figure 5.5: Pseudo-code for Configuration-agent.

preference is met for a particular action. In summing up the effect of these terms, each agent has a weighted utility function similar to the linear weighted approximation of the user's utility function. The weights (w_1, w_2, w_3, w_4, w_5) are predetermined in the current A-Design implementation for each agent. Various agents of each agent type are created with different weightings. For example, for the agent-type "C-agent-electrical-parallel-source-existing", various agents can be implemented each emphasizing different terms of the evaluation function. One agent might prefer the Embodiment (high value for w_2) more than the FPs it connects to in a design (lower value for w_1). Also, some agents can strongly consider learning influences (higher w_4, w_5), while others ignore learning ($w_4 = 0, w_5 = 0$). With this approach, a variety of agents can be constructed to produce more variety in the possible configurations. In the current A-Design system, four agents, each with different weights, are formed for each agent type. This results in a total of 192 C-agents in the electromechanical A-Design process (48 types multiplied by four of each type). Future work with this utility agent model will investigate alternative ways of choosing weights either automatically or through experimentation.

The specifics explained here are for the current set of C-agents. They are by no means the only way to implement such Configuration-agents and are shown here to portray the amount of programmed intelligence that can successfully lead to interesting design states. It is important that agents are directed enough to prevent infeasible design alternatives, but it is also equally important that they are not so developed as to unduly limit the exploration of the design space. For example agents that determine the complete configurations on their own will greatly limit the possible design configurations that are

visited. The iterative nature of A-Design relieves C-agents from having to produce viable alternatives at the start of the process. The space of possible solutions is better searched by the set of C-agents producing possibly extravagant alternatives at first and then progressively becoming focused on improving the better solutions over time.

5.3.2 Instantiation-Agents (I-Agents)

The Instantiation-agents have a simpler job than the Configuration-agents. They choose real components from the catalog for each Embodiment in a configuration. I-agents choose the EB to instantiate as well as the component for the instantiation. Component selection is performed by referencing a catalog of real components and the equations extracted from the configuration to determine which actual components best meet the design specifications.

Similar to C-agent preferences, I-agents can also exhibit different preferences in instantiating Embodiments with actual components. The I-agent preferences are shown in Table 5.1 and often include specific design objectives that the user wishes to optimize. For example, if cost and mass of a design artifact are to be minimized then specific I-agents are developed to have preferences for choosing inexpensive or lightweight components. Also, I-agents can have a preference for which variables of an Embodiment are the deciding factor in selecting a component, for example, choice of a gear component can be based on diameter or gear pitch. Finally, I-agents have a preference for instantiating such variables with high, medium, or low values. Similar to the C-agent utility model, the I-agents search all possible components and return the instantiation that maximizes the agent's utility. The I-agents are invoked until no Embodiments in the system are without a corresponding component.

Again, the strategies shown in Table 5.1 are not the only possible implementations; they merely reflect a combination of goal-directed selection of components and the stochastic interaction of agents.

5.4 MODIFICATION-AGENTS

In the final implementation of A-Design discussed in this dissertation, only one type of Modification-agent was utilized. It is possible to imagine other types of Modification-agents such as agents that exchange components, agents that merge designs, or agents that attempt to create function sharing in designs. The idea behind these agents is to establish strategies to find design states related to the ones that have been previously created. In other words, these agents explore the design space in the neighborhood of past successful design solutions. The Modification-agents are constructed to bridge the gap between the iterative exploration of the search space and the complexities of the design representation.

5.4.1 Fragmentation-agents (F-Agents)

The Fragmentation-agents are crucial in driving the system towards optimal design states. F-agents first identify a design to modify, and then further search over the EBs or components within the chosen design to find traits that are believed to be reducing the design's worth. The agents then remove parts of the design in an indirect effort to improve the current design solutions in the following iterations.

Various fragmenting preferences differentiate F-agent types from one another as is seen in Table 5.1. Similar to I-agents, these agents can be defined by the choice of objectives to address, and the choice of designs to modify. In addition, the degree of fragmentation can be a removal of instantiated components (to alter a design's

instantiation) or a removal of complete Embodiments (to alter a design's configuration). Finally, the strategy behind removing Embodiments from a system, can include the deletion of repeated Embodiments, of dangling EBs, or even of EBs that appear to be unjustifiably connected to ground. The result of these preferences is a set of 72 Fragmentation-agents. These agents fragment all Good and Pareto designs at the end of every iteration. While the Fragmentation-agents are not capable of completing design states, the fragmented solutions are returned to the C-agents and I-agents in the next iteration for reconstruction.

5.5 COLLABORATION AMONG AGENTS

In this chapter, three different agent types are described in reference to electromechanical design problems. In each of the design tasks in a given iteration of A-Design, agents are called sequentially until all design activities are accomplished. These agents are nearly deterministic in nature. That is, they always choose based on the action that maximizes their evaluation function. In cases of a tie, they pick randomly from the best evaluated actions. While the behavior of these agents collaborate in a well-defined manner, the sequence in which agents are invoked is done randomly through the process. As discussed in the next chapter, this random invoking of agents is guided by stochastic decisions made by a Manager-agent.

In order to create the most variety, agent strategies are enumerated to form as many different approaches to solve a design problem as possible. Each "disputed" strategy presents one possible approach used in the design process. Similar to design solutions, there is no clear answer as to which strategy is best for creating designs. The population

of agents represents the spectrum of different approaches interacting under a common goal to create a variety of successful design states.

Chapter 6

The Manager-Agent: Guidance for Iterative Search

Main Entry: **guide**

Function: *verb*

Inflected Form(s): **guid·ed; guid·ing**

Date: 14th century

transitive senses

1 : to act as a guide to : direct in a way or course

2 a : to direct, supervise, or influence usually to a particular end

b : to superintend the training or instruction of

intransitive senses : to act or work as a guide

- **guid·er** *noun*

This chapter returns to the *iteratively-guided* search subsystem described in Chapter 2. The iterative operation of A-Design is a simplification of the iteration occurring in human design. Human problem solving is more than just “trial and error”; people absorb an abundance of information from studying failed attempts and from observing successful or competing products. The learning that results from design activity can lead to an efficient search downstream in the process by building intuition about successful and unsuccessful regions of the search space.

Unlike Maker- and Modification-agents shown in the previous chapter, the Manager-agent acts as a moderator in the process between the user and the A-Design process. Through the advice supplied by the user, the Manager-agent filters the immense amount of data in the process and assesses how to best *guide* the search process. In this sense, the Manager-agent *collaborates* with the user and the other agents in the A-Design system.

In the implementation of electromechanical A-Design, only one Manager-agent was developed. Like the agents shown in the previous chapter, this Manager-agent also addresses ambiguous conceptual design issues and therefore embodies a “disputed” design strategy. Future models of A-Design may include a number of interacting Manager-agents similar to the interacting Maker- and Modification-agents.

The philosophy behind the Manager-agent is to glean information from the complex thought process of the user and from the extensive details of past iterations. The learning attained by the Manager-agent is then used to *guide* the process to better designs. This *guidance* can both narrow search to particular areas of the space of design solutions and broaden search to find new fruitful areas of the search space.

6.1 RELATED WORK

The Manager-agent model is built mainly upon machine learning research (see overview in Mitchell, 1996). Learning algorithms have many uses in computation because they can automatically determine strategies for complex environments and can adapt to new stimuli that might be encountered in such environments. Similar to stochastic optimization, A-Design uses statistics based on previous design states as the stimuli for making better design decisions in future iterations. Many learning approaches

require a “teacher” to articulate the proper responses to various stimuli. Unfortunately, proper response in design is not a simple matter of right and wrong. While there are degrees of success in design, there are no right answers or strategies that can be referenced in creating better designs in the future.

Constructing data to be used for learning in A-Design starts with the user/Manager-agent dialog. In Chapter 3, this dialog established a scale to sort designs. From these rankings, an evaluation function can be constructed. The method of extracting an evaluation function from the user’s rankings is based on Christensen and Korf (1986). This paper laid the groundwork for determining coefficients of a linear weighted function. Later, this was explored in depth by Abramson (1990) in reference to game-playing.

A second area of learning in A-Design is finding trends in past designs. The TODO/TABOO learning that is explained below in Section 6.3, is based on several different machine learning techniques. First the detection of good and bad design trends is similar to learning by analyzing differences, explored by Winston (1982, 1992). This work determines classifications of instances by observing differences and commonalities in test data. The algorithm performs best when the test data include “near-misses” – alternatives that, although similar, belong to different classes. The technique developed here to discover characteristics of good and bad designs has also been influenced by the SOAR system (Laird et al.1986) and the EURISKO system (Lenat, 1983). Although the domains in these projects are quite different than the configurations built in A-Design, the concept of dissecting alternatives to find commonalities is similar. The design fragments

that belong to the TODO and TABOO sets are similar to the “chunking” of ideas discussed in Laird et al.

The Manager-agent uses the information gained from past designs to *guide* the design activities of the other agents. The TODO and TABOO sets passed to the Maker- and Modification-agents influence the actions of these agents by a reinforcement learning technique (see overview in Kaelbling, 1996). The manner in which this new design data affects the decision-making process of the agents is similar to other approaches that have combined reinforcement learning with multi-agent systems (see Tan, 1993; Sandholm and Crites, 1995).

Finally, the Manager-agent also can adjust the probabilities of the randomly invoked Maker- and Modification-agents. This adjustment is based on the past contributions of these agents and is similar to techniques such as proportional selection in genetic algorithms (see Bäck and Hoffmeister, 1991) and move set probabilities in simulated annealing (Hustin, 1989). These approaches can allow stochastic optimization to be more directed and efficient in the search for successful design alternatives.

6.2 BASIC OPERATION

This section describes the complete operations of the Manager-agent. Figure 6.1 shows pseudo-code representing the Manager-agent’s procedure at every iteration in the process. The Manager-agent receives the design alternatives separated into Pareto and non-Pareto solutions. Because this separation can be determined mathematically, it is not part of a unique agent strategy. From these designs, the Manager-agent prepares data to present the current state of the process to the user (as shown in the example of Figure

3.5). From the interactions with the user, the Manager determines an evaluation function that approximates the user's utility function. Based on this evaluation function, the Manager-agent is able to further divide the remaining non-Pareto designs into Good solutions and Poor solutions.

Next, the Manager-agent initiates the TODO/TABOO learning. This process, described in depth in the next section, finds common trends among design states. These trends include both combinations of agents and design fragments from both good and bad designs. The TODO list is constructed by examining the best designs from an iteration to

<p>Sensors = design states (already divided into Pareto and non-Pareto)</p> <ol style="list-style-type: none"> 1. Find best design for current user preference. 2. Report to user details of process (see Figure 3.5). <ol style="list-style-type: none"> a. Inform on the amount of change to the Pareto set. b. Present best-design and inform when this changed last. 3. Query user ("Talk to M-agent (hit return) ? "). <p>if "yes"</p> <ol style="list-style-type: none"> a. Present random Pareto designs b. Retrieve rating from user c. Approximate preference as linear weights (see Figure 3.6) 4. Find Good as a fraction of remaining non-Pareto solutions that best meet this user preference. 5. Create TODO and TABOO lists <ol style="list-style-type: none"> a. find trends in agents, configurations, and components 6. rank occurrence of such trends 7. Update agent statistics <p>Effectors = Pareto designs, Good designs, Poor designs</p> <p>Throughout process:</p> <ul style="list-style-type: none"> • recommends Configuration-agents • recommends Fragmentation-agents • recommends Instantiation-agents
--

Figure 6.1: Pseudo-code for Manager-agent.

find trends of positive design activity. Conversely, the worst designs in an iteration are compared to find elements of the TABOO list representing poor design trends.

In the next iteration, the Manager-agent advises the Maker- and Modification-agents with the TODO and TABOO lists. The trends on the TODO list are strived for, while TABOO trends are avoided. Maker- and Modification-agents balance their preferences with these trends to make informed decisions. This method of reinforcement learning is discussed in Chapter 5 where the electromechanical A-Design agents are described in detail. Also, the effects of TODO and TABOO learning are tested in the experiments of Chapter 9, which show that keeping track of design trends allows the process to converge more rapidly and to achieve better quality solutions.

Finally, statistics on the agents are recorded based on previous design activity. The Manager-agent keeps track of the designs each agent produces. For example, a particular C-agent might contribute to two Pareto designs, one Good design and four Poor designs. This data is then used to adjust the probabilities of invoking an agent in future iterations, and is discussed in more detail in Section 6.4.

At this point in the process, the Manager-agent returns control back to the iterative process with the division of designs into Pareto, Good, and Poor populations, and the recommendations for invoking agents in the next iteration.

6.3 FINDING TRENDS

The TODO/TABOO learning determines good and bad trends in past designs to provide a reference to improve future design activity. These trends are found by intersecting System Configuration structures (shown in Figure 4.8). These intersections

can be either graphs (connections of EBs and FPs) or various lists such as agents and components. Three types of trends are currently found in the implementation: 1) design fragments consisting of EBs and FPs connected together to form small functional blocks such as a “rack-and-pinion”, 2) agents that act as design teams to create either good or bad design instances, and 3) groups of components that instantiate configurations.

From the approximated utility function determined by the dialog with the user, the designs are sorted from best to worst by collapsing the various objectives to a single metric as seen for two objectives in the example of Figure 6.2a. From this sorting, a

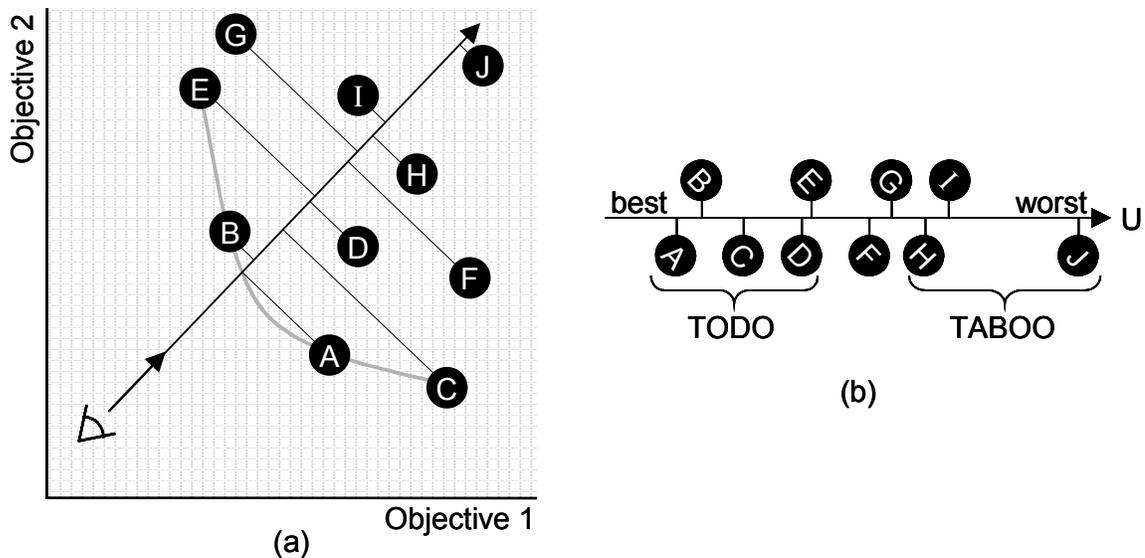


Figure 6.2: An example of detecting TODO and TABOO trends from a set of 10 designs (A through J). (a) The approximation of the user’s utility allows designs to be collapsed a single metric. (b) The sorted list is then divided into elements for TODO and TABOO comparison. (c) The top five TODO designs are intersected exhaustively for all possible combinations.

number of the best designs are selected for constructing the TODO list and a number of the worst designs are selected to determine the TABOO list (see Figure 6.2b).

Next the designs are exhaustively compared to find commonalities in the selected designs. In Figure 6.2, the top five designs separated for TODO comparison are examined to find common intersections in the designs. First, the process looks for an intersection in all five designs. Sometimes the intersecting all of the designs does not yield any commonalities. Therefore, the process also checks all possible combinations within four out of the five designs to find trends. This intersection continues for all possible combinations of designs down to comparing only two members at a time. The level corresponds to the number of designs that a particular trend is common to, and this number is assigned to that trend to be used in the reinforcement learning of the agents.

Finding the intersection of sets (such as common design teams of agents) is easier than finding the intersection of graphs (such as design fragments). While set intersection routines are rather quick and fundamental to computation, the intersection of graphs can be an intricate procedure. To find a design fragment, the set of Embodiments are first intersected to find a common set. Then, the set of common EBs are checked to see if they are connected in the same manner between designs. Repeated Embodiments and highly connected FP's make finding common design fragments difficult.

In the A-Design implementation, the intersection procedure is not particularly time-consuming, but because of the number of intersections (as in Figure 6.2c), the size of the TODO/TABOO lists can be constrained by resource constraints. Increasing the membership of the TODO and TABOO sets drastically increases the number of required

intersections. Also, if the designs that are created contain more components, then the individual intersection times would greatly effect the time required to find the TODO and TABOO trends. For example, airplanes contain orders-of-magnitude more components than weighing machines. To find commonalties in larger design domains will require considerable computational resources or the development of efficient heuristics to find commonalties. Currently, detecting these trends has produced promising results as seen in Section 9.3. Future research may find quicker ways to detect trends or specific ways to identify useful trends without extensive comparisons.

6.4 STOCHASTIC EXECUTION OF AGENTS

Throughout the modification and creation of designs, the basic operation of the iterative process is to randomly invoke C-, I- and F-agents until all design tasks for a given iteration have been completed. The Manager-agent stores statistics on agent contributions from previous iterations and uses these statistics to direct the choice of agents. These Manager-agent decisions give the process a stochastic guidance while still allowing for a certain degree of randomness in the search for new design states.

Before every agent call, the Manager-agent is queried by the process to generate a list of how to proportion the probabilities for selecting an agent. The Manager-agent rates each possible agent based on the agent's prior statistics and prior collaborations with other agents. These ratings for each agent are the result of the Manager-agent's own evaluation function, which combines TODO/TABOO agent teams and agent statistics.

Figure 6.3 illustrates an example of how this stochastically-guided proportional agent selection is accomplished. After a particular agent call, a partial design is constructed

from calling agents D, A, C, A (as seen in Figure 6.3, element 1). The Manager-agent then sets up probabilities for the next agent to call based on information from the agent statistics (Figure 6.3-2), and the TODO and TABOO lists of design teams (Figure 6.3-3). The agent statistics and the TODO/TABOO list information are filtered through the evaluation function. This function is a weighted sum of five terms that determines the probabilities of the next agent call:

$$U = 2 \times \text{num_of_pareto} + 1 \times \text{num_of_good} - 1 \times \text{num_of_poor} + 2 \times \text{TODO_team} - 2 \times \text{TABOO_team} \quad (4)$$

The number of past Pareto designs the agent contributed to (num_of_pareto), the number of past Good designs (num_of_good), and the number of past poor designs (num_of_poor) weigh in to the Manager-agent's evaluation. Also, if a particular agent completes an agent team from the TODO TABOO list of trends, then the degree of that

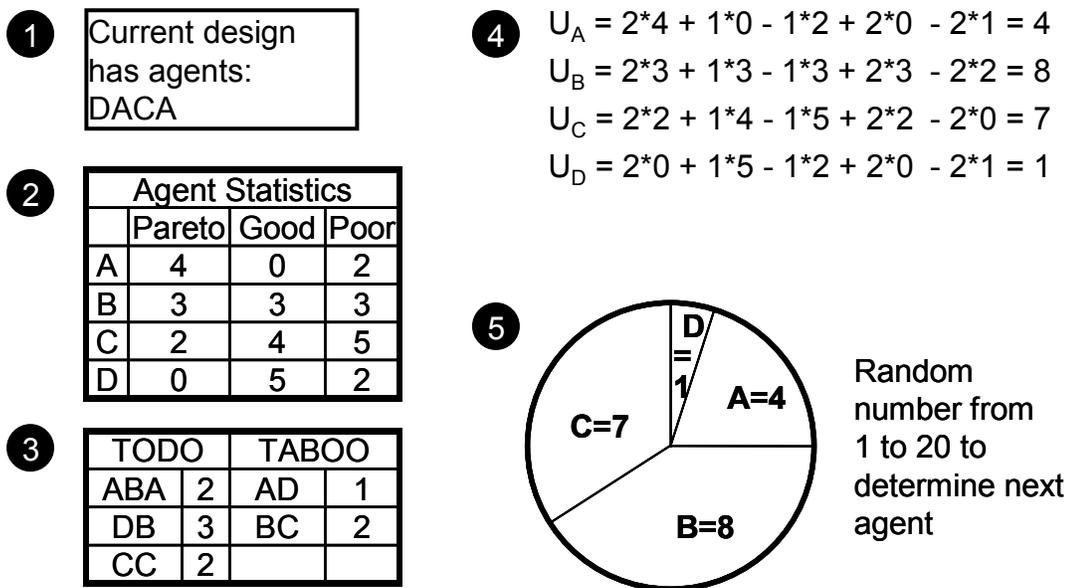


Figure 6.3: Manager-agent proportions probabilities based on evaluation function.

trend also weighs into the Manager-agent's evaluation. The agents within an agent team are not invoked in any particular order. They represent the intersection of several designs, where the number indicates how many designs the agent team was found in. As an example, agent B contributes to a TODO team ($BD = 3$) and a TABOO team ($BC = 2$) as seen in Figure 6.3-3. Since the addition of B fulfills both of these teams based on the previous agents contributing to a design, then the evaluation function for B includes $TODO_team = 3$, and $TABOO_team = 2$.

The Manager-agent considers all possible agents and gives each a score from the evaluation function (Figure 6.3-4). As a result of these values, the Manager-agent divides the probabilities as shown in Figure 6.3-5. The process then picks a random number to determine which agent to invoke based on the Manager-agent's division. The coefficients in Equation 4 have been implemented to produce a balanced yet effective division of agent probabilities. It is possible that these coefficients could be adjusted for different applications or agents. The means of optimally or automatically weighing the factors of this stochastic guidance is subject to future development and experimentation.

This manner of stochastically guiding the process is similar to the dynamic selection mechanism of proportional selection in genetic algorithms. However, the A-Design stochastic process is augmented by the fact that the Manager-agent updates the selection probabilities for each agent call. The case-by-case updating of probabilities allows designs to be tailored to their strengths and weaknesses, and encourages certain agents to collaborate on specific or diverse designs. The context of continually recommending agents based on the state of a design is a unique guidance mechanism to A-Design. The

range of possible M-agent behaviors is large and intricate and lends itself to future research, however current strategies of providing feedback have improved efficiency and performance as is shown in Chapter 9.

Chapter 7

Preliminary Test

Problems

This chapter introduces the second half of the dissertation, which sets out to test the various facets of the A-Design methodology presented in the first six chapters. The two design problems in this chapter test individual subsystems of the theory and illustrate the generality of the methodology. While these examples do exhibit some design-like qualities, they do not have an open-ended formulation like the electromechanical representation described in Chapter 4. The first example tests the multi-objective design selection and agent interaction subsystems, while the second numerical optimization problem explores the iterative process' ability to efficiently search a complex space for optimal solutions.

7.1 MANHATTAN TRANSFER

In this example, solutions are relatively easy to find, but contrasting objectives make meeting the user's preference difficult. The object of the Manhattan Transfer problem is to get from one location to another in a grid-based city in the minimum amount of time,

cost and effort. A user specifies the start and end location of a trip as the initial specification to the algorithm. The algorithm then finds solutions that connect the start and end locations via various transportation devices. A simulated two-dimensional grid of squares represents city blocks, and the transportation devices consist of bike, walk, run, bus, taxi and subway. Each of these devices has a unique cost, time and effort. For example, a taxi might cost an average of \$0.20 a block but requires little effort and is time efficient, while walking costs nothing but requires more time and effort. The problem specification requires A-Design to create alternatives using combinations of the 6 transportation devices in order to best satisfy the user's weighted criteria of minimizing cost, time and effort.

As in Figure 2.1, the process begins with the Maker-agents contributing partial travel paths along the way to the creation of complete trips. Rather than each agent solving the complete problem from start to end, the Maker-agents add individual travel segments that combine to make a complete solution. Designs result from the combined efforts of several Maker-agents. These Maker-agents differ in how they handle constraints in the system such as bus and subway stops, and maximum distances one can walk or run. After Maker-agents complete travel paths, the alternatives are evaluated on their cost, time and effort, and solutions are sorted into Pareto, Good and Poor populations. Next, Modification-agents remove undesirable segments of travel paths and return the fragmented designs back to the Maker-agents for reconstruction.

In the Manhattan Transfer experiments, problems were initiated with beginning (0, 0) and end (20, 20) locations as well as the relative importance of each objective. At the end of the process A-Design returns several solutions that best meet the user preference. Originally, the user preference placed more emphasis on minimizing cost than minimizing time or effort (cost was five times more important than time and two and a half times more important than effort). The solution shown in Figure 7.1 was created by A-Design in 62 iterations with a maximum design population of 160.

In addition, the user is free to change preference throughout the process allowing the system to adapt appropriately. By examining the results produced, the user can adjust the preference weighting to achieve a desired travel path. In this example, the adaptability of the system was tested by changing the user preference to prefer minimizing time twice as much as effort and 5 times as much as cost. This produced the result of “take taxi from

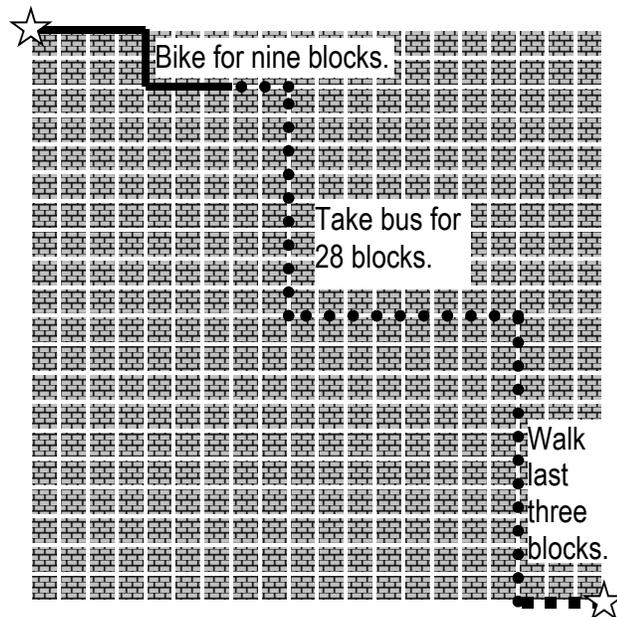


Figure 7.1: A solution to the Manhattan Transfer problem generated by the A-Design process.

start to finish” (TAXI (0 0) (20 20)). By starting from the population of designs created under the first preference, the process easily adapted to the new preference. As a result, the algorithm took only 11 iterations to converge after the preference change, thus illustrating the power of the recessive traits stored in the Pareto set. It was determined through a separate experiment that the space of possible designs numbers approximately 1.2 million and of these, only 99 are Pareto-optimal. After our experiment, A-Design had found 40 Pareto points of the 99 in only searching approximately 7,000 design states.

Although the Manhattan Transfer example deals with a highly abstracted situation, it illustrates that agents with different characteristics are able to work together to adapt to change and to find optimal design configurations. Alone, agents consider minimizing only single objectives. For example, low-cost designs are usually created by agents that prefer low-cost devices. Through the collaboration of different agents, solutions are constructed to exemplify a balance of attributes as suggested by the user’s preference. The combination of agents that prefer low-cost devices with agents that prefer low-effort devices will ideally lead to designs that are both low in cost and low in effort. In the example, the initial user preference shifted from low-cost designs to quick designs. A-Design was able accommodate this alteration by switching focus to agents with a preference for quick transportation devices.

7.2 NUMERICAL OPTIMIZATION

In addition to adapting to changes in user preference, the iterative search should also be able to improve or “optimize” for a given user preference. Therefore, this example tests the optimizing power of A-Design apart from its use as a conceptual design generation tool.

This multi-objective example contains two highly multi-modal numerical objective functions. The two functions shown in Figure 7.2, are expressed by

$$f_1(x, y) = 300(1-x)^2 e^{-x^2-(y+1)^2} - 1000 \left(\frac{x}{5} - x^3 - y^5 \right) e^{-x^2-y^2} - \frac{100}{3} e^{-(x+1)^2-y^2} - |x| - |y| \quad (5) \text{ (Figure 7.2a)}$$

where the minimum is -59.2 and is found at $(x = 0.023, y = -2.105)$, and

$$f_2 = x^2 + y^2 - 100 \sin(x) - 100 \sin(y) \quad (6) \text{ (Figure 7.2b)}$$

where the minimum is -234.6 and is found at $(x = -1.57, y = -1.57)$.

A-Design was compared with a robust SQP algorithm (Lawrence, et al., 1993) in its ability to find the optimum for a weighted sum of the two objectives. First, both A-Design and SQP were tested by weighting f_1 ten to one over f_2 . Then using the results of the first run tested again with a weighting of f_2 ten to one over f_1 . The 10:1 and 1:10 preferences are used to establish different optimal points for the lumped objective functions. Agents within this problem are simple functions that increase or decrease parameters within the equations to reduce objective values or avoid local minima.

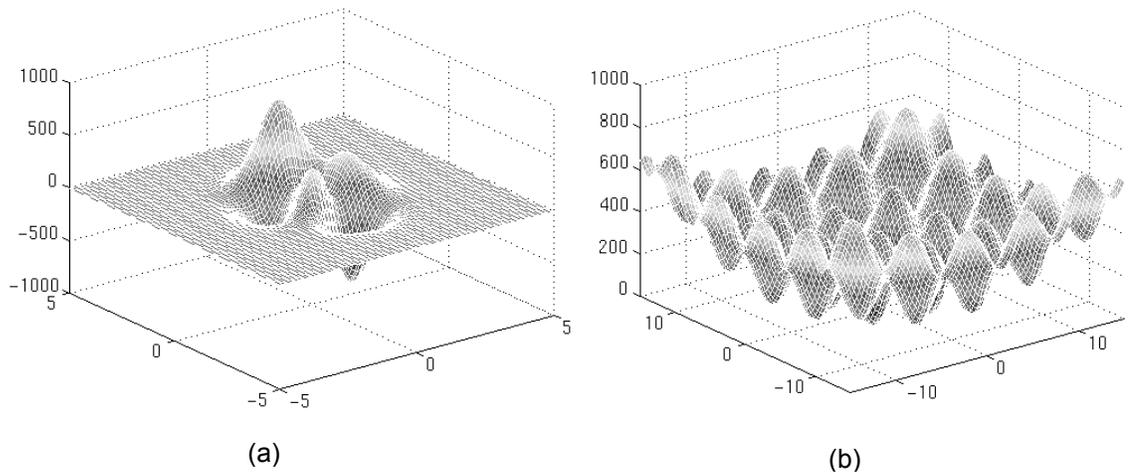


Figure 7.2: The two mathematical functions (a) f_1 and (b) f_2 used to test the optimizing power of A-Design.

A single run of A-Design proved to be a time-consuming process partly due to the number of design state evaluations and partly due to agent and design management. Although SQP found the optimal solution with fewer evaluations for the 10:1 weighting, it found the global minimum only one in seven runs of the process. The A-Design algorithm, due to its agent-based search and storing of design alternatives, found the solution in every run with a population size of 100 and an average of 88 iterations. When the weightings changed, A-Design only needed to perform a single iteration to arrive at the new optimum, while SQP had to be rerun 13 times before finding the new optimum. By retaining the Pareto-optimal set from the first run, the A-Design algorithm quickly adapts to changes in weighting of the two objectives. Certainly, SQP is a more efficient optimization strategy if it starts in the neighborhood of the optimum (requiring only on average 10 evaluations per run as opposed to the hundreds of evaluations required by A-Design). However, in general, A-Design is more robust in its ability to find the optimum and the time required in arriving at Pareto-optimal solutions for the first user preference results in a large savings when the process is reinitiated with different weights.

The two examples discussed above demonstrate the versatility of the A-Design methodology. Both examples illustrate the methodology's potential outside the intended use as a conceptual design tool. These examples show that A-Design, through its unique design selection scheme of preserving the Pareto front and iterative-based agent operations, can successfully and adaptively produce results for a wide variety of problems.

Chapter 8

Electromechanical

Test Cases

The two test cases presented in this chapter make full use of the potential of the A-Design conceptual design process. The implemented electromechanical A-Design system includes the iterative search process, the adaptive design selection, the functional representation, the C-, I-, F-agents, the Manager-agent, the equation extractor, the design evaluation mechanism, and a general framework for transferring designs to the various sub-processes. The system is written in LISP and is shown in abbreviated form in the Appendix.

8.1 WEIGHING MACHINE

In order to pose the weighing machine design problem, the user supplies the desired inputs and outputs, the objectives and the location of the files that make up the catalog of components. In this test case, four objectives were chosen to guide the system to successful design states: minimize cost, minimize mass, minimize dial error and minimize input displacement. The first two objectives (minimize cost and minimize

mass) are calculated by summing component values found in the catalog. The latter two (minimize dial error and minimize input displacement) are results of the behavior resulting from the extracted equations. The problem is described to the process by both the functional description and the objectives (see Figure 8.1). The catalog of components for this test example consists of the Embodiments shown in Table 4.2. For each of the 32 Embodiments shown, there exist actual components drawn from Allied Electronics, Nordex Inc. and Mc-Master Carr Supply catalogs. These total just over 300 components available for constructing designs. The catalog is expandable to allow the user to introduce new Embodiments and components.

Since the first several chapters have already described the problem, the results will now be presented. Figure 8.2 shows three weighing machines created by the process over

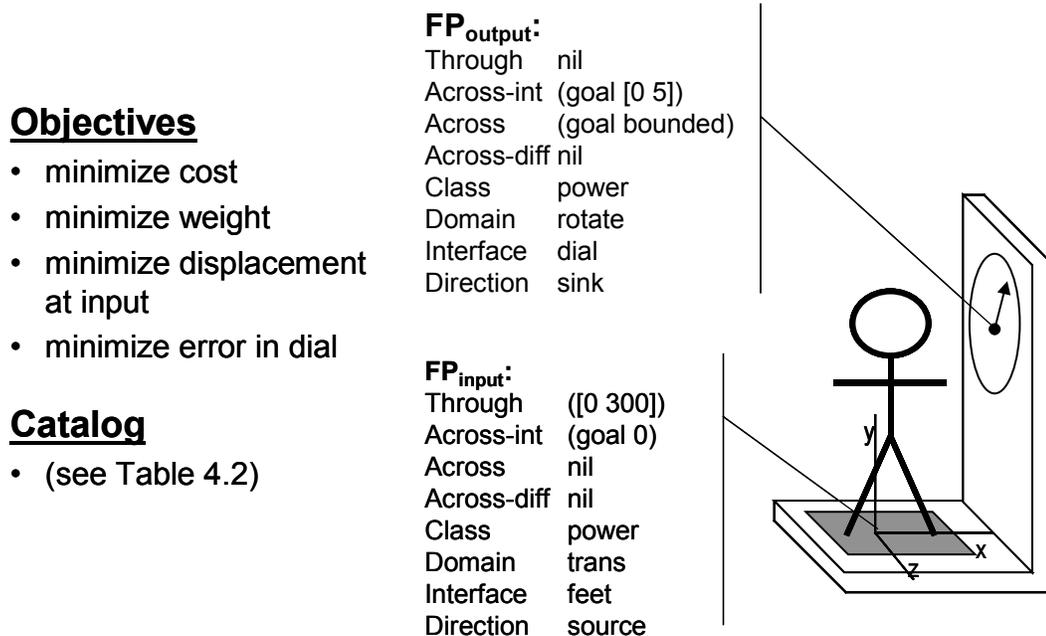
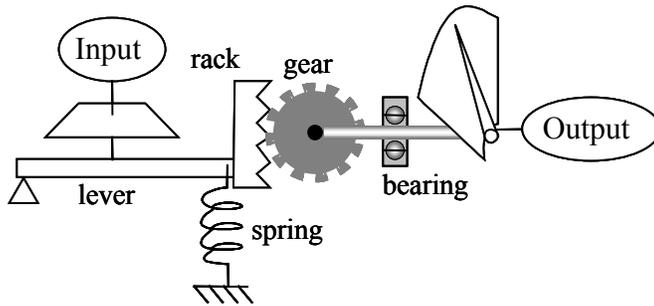


Figure 8.1: Description of weighing machine design problem as posed to A-Design.



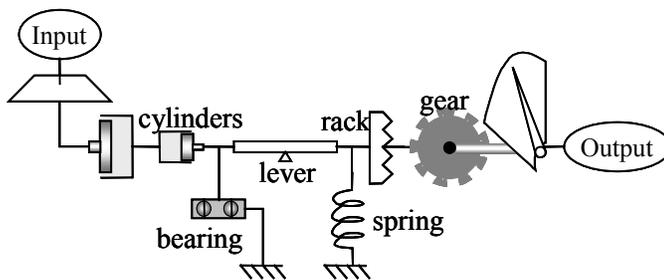
Components:

lever: 5 cm bar stock $w=1.0"$, $t=0.25"$
spring: ERS-A1-36 \$0.93, $K=16.0\text{lb/in}$
rack: KHS-F2-142 \$26.75, $\text{pitch}=64$
gear: LAS-F7-28 \$5.75, 28 teeth
shaft: AAS-A8-20
bearing: ABS-A2-19

Design objectives:

cost = \$46.82, mass = 0.2kg,
input dx = 4.1cm, accuracy = 0.4rad.

(a)



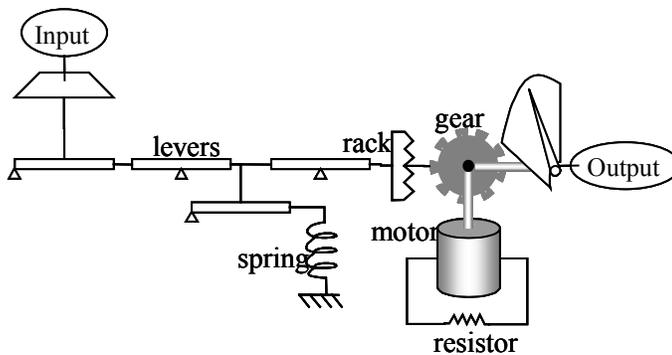
Components:

cylinder-1: 62205K77 \$368.49, $\text{Dia}=3.25"$
cylinder-2: 62205K71 \$198.63, $\text{Dia}=1.5"$
linear-bearing: ABS-L1-4 \$10.47, $\text{Dia}=0.25"$
lever: 10 cm bar stock $w=1.0"$, $t=0.25"$
spring: ERS-A1-2 \$0.89, $K=2.0\text{lb/in}$
rack: KHS-F2-142 \$26.75, $\text{pitch}=64$
gear: LAS-F7-28 \$5.75, 28 teeth

Design objectives:

cost = \$616.18, mass = 1.3kg,
input dx = 5 mm, accuracy = 0.4 rad.

(b)



Components:

lever-1: 4 cm bar stock $w=1.0"$, $t=0.25"$
lever-2: 4 cm bar stock $w=1.0"$, $t=0.25"$
lever-3: 13 cm bar stock $w=1.0"$, $t=0.25"$
lever-4: 7 cm bar stock $w=1.0"$, $t=0.25"$
spring: ERS-A1-7 \$0.78, $K=14.6\text{lb/in}$
rack: KHS-F2-142 \$26.75, $\text{pitch}=64$
gear: LAS-F7-128 \$12.03, 128 teeth
motor: 542-0130 \$34.19, 300rpm
resistor: 297-7751 \$0.01, 180K Ω

Design objectives:

cost = \$90.20, mass = 0.5kg,
input dx = 0.7mm, accuracy = 0.2 rad.

(c)

Figure 8.2: Three different alternatives created by the A-Design process. Design (a) is found by an equal preference for the four design objectives, whereas designs (b) and (c) are found by placing more importance on minimizing input displacement.

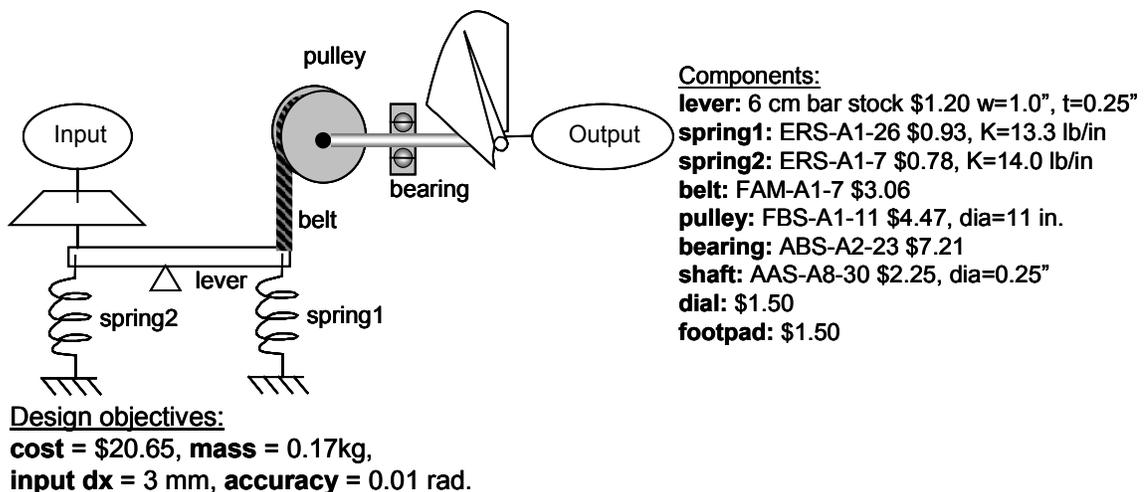
30 iterations with a maximum population of 100 designs⁵. The first solution (Figure 8.2a) was created when the user preference was for low-cost low-weight designs, and as a result the solution is relatively inexpensive and lightweight compared to the other two solutions. However, it is deficient on the two performance objectives lacking dial accuracy and stability at the input. As a result of this trade-off, the system was then adjusted to place more emphasis on minimizing movement at the input. As a result of this preference shift, designs were created that although more costly and more massive, addressed the issue of minimizing input displacement. The second and third designs (Figure 8.2b, c) show two different ways that A-Design was able to solve this problem. Figure 8.2b uses two hydraulic cylinders with different diameters while Figure 8.2c utilizes a series of levers to minimize input. These examples demonstrate the richness of alternatives created by the A-Design algorithm and the collaborative reasoning of the agents. The ingenuity of these solutions is especially evident in resolving the “minimize input displacement” objective. The search process overcame the challenges of the design problem by improving alternatives to better designs through the iterations.

8.1.1 Results Under the Influence of TODO and TABOO Learning

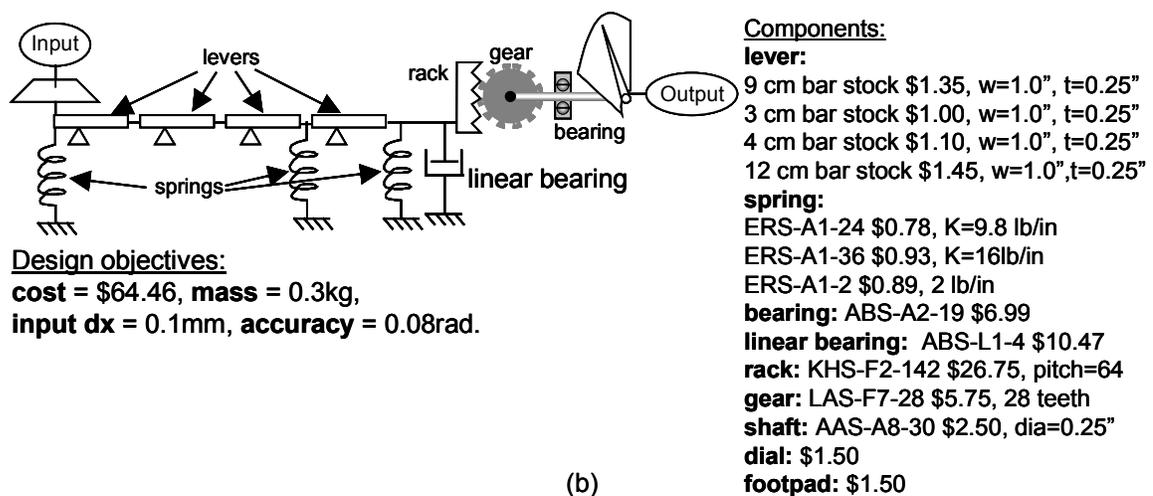
In addition to the results shown above, several experiments were run to test the validity of TODO and TABOO learning (a detailed description of this experiment can be found in Chapter 9). Two interesting artifacts from these experiments are shown in Figure 8.3. These two designs are able to solve the design problem better than when no learning

⁵ Figure 8.2, and 8.3 are the authors’ renditions of the appearance of the Embodiments and instantiations generated by the system. The system outputs System Configuration text structures, which include the Embodiments, their connecting FPs, and their instantiations (see Appendix).

is present. The design shown in Figure 8.3a makes use of a belt and pulley as opposed to the rack and pinion pairing found in previous designs. Here also, the challenge in reducing the displacement in the input leads to some interesting solutions such as that shown in Figure 8.3b. The learning seems to favor more components to solve the problem as is noted by multiple springs in these configurations.



(a)



(b)

Figure 8.3: Two additional weighing machines found under the influence of TODO and TABOO learning.

8.1.2 Discussion of Weighing Machine Results

All the designs in the previous two figures make interesting use of series and parallel connections. Also, these designs all contain several connections to ground - an essential factor in solving the design problem. The ground provides a reference and the sense of bounded state variables that is prescribed by the input/output FPs. The general update mechanism for adding new EBs to a configuration and the diverse agents leads to the wide variety of achievable designs for this problem. Figure 8.4 shows a performance plot for the five designs from Figure 8.2 and Figure 8.3. In this figure, the four objectives

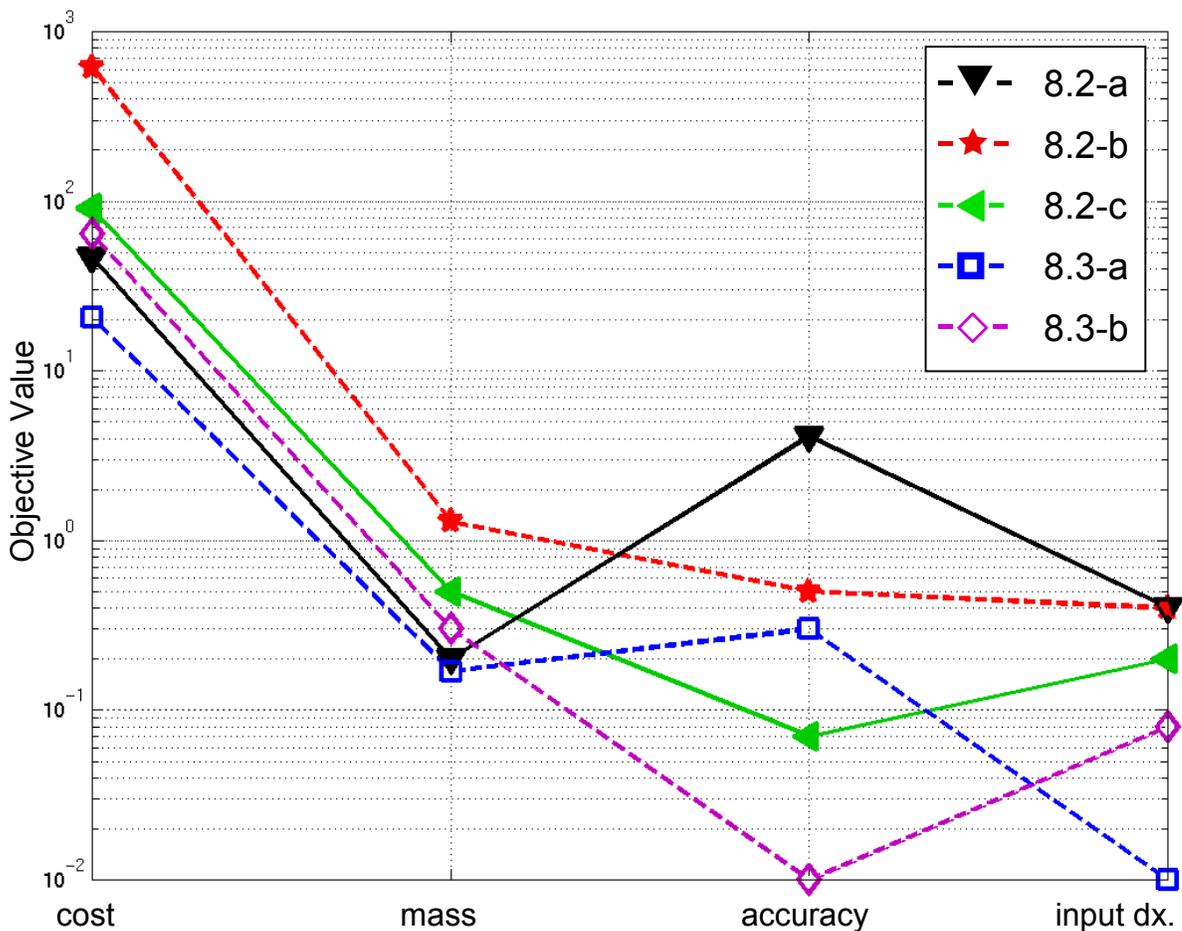


Figure 8.4: Performance plot of five designs from Figures 8.2 and 8.3.

along the bottom are all “minimizations”, therefore the lower the data point, the better. The two designs created under the influence of learning dominate those created in the absence of learning. Even the first design (Figure 8.2-a) created under the low-cost and low-weight preference was dominated on both cost and mass by the design from Figure 8.3a despite that fact that the design in Figure 8.3a was created under the preference for minimizing input displacement. The two designs created under the influence of learning are Pareto-optimal for this set of solutions.

One of the most interesting design structures in this set of solutions is the motor-resistor pairing in Figure 8.2c. This is one way to meet the goal in the output FP of a bound across variable (FP:Across-none = (goal bound) in Figure 8.1). This goal statement states that designs must have a “damped” motion at the output. All other designs solve this by including bearings (with friction damping) in the solution. However, the motor-resistor is similar to a damper due to the analogy of a resistor as an “energy dissipater” and a motor as a “transformer” from rotational to electrical energy.

This example also brings up an interesting point about the representation. While it is true that bearings lead to damped motion due to the inescapable friction that exists between a bearing’s moving parts, the true purpose of bearings is to contain rotation to a specific location in space. Currently this degree of functionality is not incorporated within the current functional syntax. However, it is possible that extensions could be made to incorporate this detail.

Furthermore, the designs shown are not complete in position information. The coordinate slot of the Functional Parameters has been implemented, however the

transformation of these coordinates within the Embodiment is not fully realized. This is due to the fact that such transformations include some variability and thus require further constructive power to be included within the Configuration-agents. For example, the gear EB in Figure 8.5 shows the ports of a gear as (A) the connection with the rotational shaft, and (B) the translational connection with the gear teeth. The position of the B Port can be located at any angle about the shaft. The choice for the value ϕ needs to be made to relate port A of the EB to port B in space. The decisions for position variables like ϕ can lead to a variety of different geometric layouts with a set configuration. This variability might mandate the construction of new objectives to describe how well the positions of the input and output FPs meet with the user specifications. Including this geometric information would establish a more authentic portrayal of the challenges in designing realistic weighing machines.

Also, the interface between components is a simplification of real component interfaces. All of the current 300 components in the catalog are constrained to the same interface dimensions. For example, only one gear pitch, shaft diameter and bolt size are used throughout the catalog. This is because dimensioning the interface would necessitate the inclusion of individual component's operating ranges or failure modes. A shaft with a one-inch diameter has very different characteristics than a shaft with a one-sixteenth inch diameter. Constraints on these shafts could specify the maximum values of the through and across variables, or the operating ranges of components (e.g. maximum shaft torque, or maximum current through a resistor). While it is obvious that the shaft with a one inch diameter can handle much higher torque loads than the one-sixteenth inch shaft, these

two shafts are not distinguishable on any characteristics other than cost in the current formulation. Without the operating range or failure mode data, the process would most likely create weighing machines using the smallest, most inexpensive components. Therefore, in order to include components with a variety of interface sizes, operating ranges need to be incorporated within component descriptions. The agents then need to be aware of how to design around these new constraints, or new objectives need to be created to measure the amount of operating range violations in a given design.

Despite these limitations to the current functional representation subsystem, it appears that diverse and interesting designs can be created. If a more detailed functional representation is developed, the A-Design electromechanical design process will be capable of more realistic designs. The adaptive user preference along with the various agent strategies created some exciting “proof of concept” results portraying a variety of

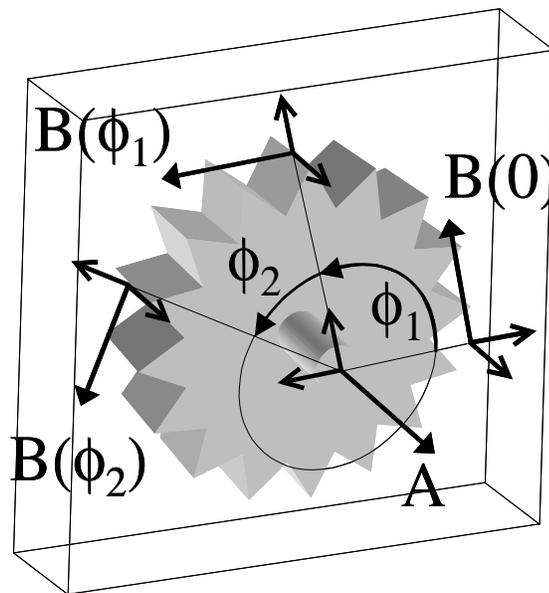


Figure 8.5: There is some variability for the position of ports on an EB. For the gear, this variability is the variable, ϕ , which relates the port A of the shaft hole to port B, the gear teeth interface point. This point can be anywhere on the radius of the gear.

configurations. Also, the learning in the iteratively-guided subsystem leads to better designs (Figure 8.3) than when the process is simply iterative (Figure 8.2).

8.2 MEMS ACCELEROMETER

The second electromechanical design problem for A-Design is the design of an accelerometer within the domain of Micro-ElectroMechanical Systems (MEMS). MEMS is a fairly recent technological endeavor with the basic principle of fabricating electromechanical systems using the manufacturing innovations pioneered by integrated circuit (IC) fabrication techniques. The technology behind constructing the micron-sized transistors of IC computer chips is enhanced in MEMS to construct micron-sized mechanical systems. Such techniques have been evolving for several decades and are capable of creating microscopic sensor and actuator devices with resolutions on the order of a few microns. The fabrication of Micro-ElectroMechanical Systems is accomplished through depositing and etching layers of silicon or other metals on a flat silicon substrate. By using various computer-aided design tools, one can construct a MEMS “layout” which is transferable to manufacturing houses for fabrication. The entire design of a MEMS device can quite easily be performed by a single engineering designer with access to the proper CAD tools.

There are many possible ways that computational design can improve the MEMS design process. MEMS design has an inherent formalism as a result of the extensive use of CAD tools and the constraints imposed by the manufacturing processes. Since most devices are constructed through the deposition and etching of layers, the space of possible solutions is confined to a series of two-dimensional planes. Although these constraints do not necessarily reduce the difficulties of finding successful designs, they do help in

establishing a formal representation of the design space. With its focus on the conceptual design process and electromechanical configuration design, A-Design is ideally suited to the application of synthesizing MEMS devices.

8.2.1 Description of design problem

Perhaps the most commercially successful MEMS device today is the ADXL accelerometer developed by Analog Devices, Inc. (1998). These devices, which come in a variety of acceleration ranges, are packaged on small computer chips along with corresponding digital circuitry that sense accelerations in the range of 0.1 mG to 100 G and can withstand shock up to 1000 G. Inside the chip is a miniature mass-spring system like any conventional accelerometer occupying a space less than half of a square millimeter. This mass-spring system is etched from silicon and is suspended between layers of the chip to allow it move in response to external accelerations. Figure 8.6 shows a layout of an ADXL accelerometer. At the center of the layout is a large plate of silicon that acts as a proof mass for sensing the accelerations. Connected on the top and bottom of the mass are electrostatic comb fingers used to sense the displacements of the mass as well as to position the mass for best sensitivity. On the sides of the mass, beams are configured to provide the compliance for the structure. The series of beams shown in the figures are often referred to as U-Springs, and are anchored at the ends to the substrate.

The complete design problem of making functional accelerometers requires many goals and constraints, including the details of the corresponding circuitry and the manufacturability constraints. However, the basic difference in the variety of accelerometers made by Analog Devices is in the dimensioning of the accelerometer configuration shown in Figure 8.6. By changing spring stiffnesses, mass size and comb

finger lengths, different acceleration ranges can be achieved. The previous work of Mukherjee et al. (1999) and Zhou (1998) made strides in automating the dimensioning of components of an ADXL-style configuration so that custom accelerometers can be automatically designed to specific demands.

The design problem that is posed to A-Design is to create novel configurations as opposed to the set topology used in the ADXL devices. It is believed that the evolution of this topology was performed under a number of design objectives, namely: minimize area, maximize sensitivity, maximize the maximum acceleration and minimize movement of the proof mass in the orthogonal direction by maximizing orthogonal stiffness. These four objectives will thus be used to guide the A-Design search process towards novel

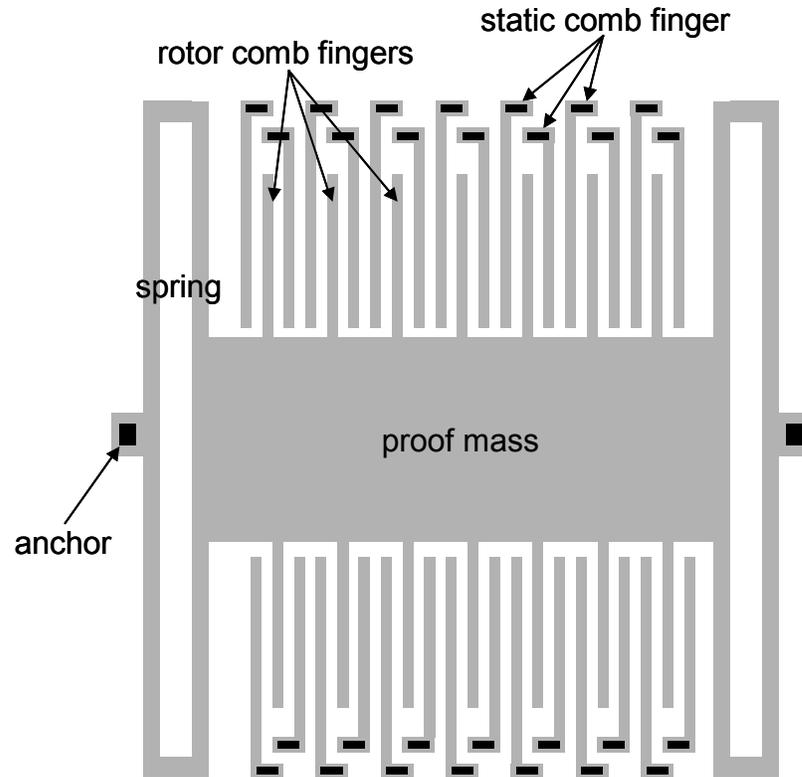


Figure 8.6: Configuration view of ADXL accelerometer.

accelerometer configurations. Because A-Design is formulated to minimize objectives, the objectives stated as “maximizations” will be inverted so as to minimize the reciprocal (e.g. maximize sensitivity, S_x , will become minimize the reciprocal of sensitivity, $1/S_x$). In addition to formulating objectives, A-Design requires that the functionality of the accelerometer be specified by input and output FPs (see Figure 8.7). The input FP has a value of 0 to 10 in the across differential slot to designate an acceleration of 0 to 10 G. The output FP is in the electrical domain, and has an across variable goal of 0 to 5 volts.

With the input and output description, and objectives in place, the catalog of components is all that remains in preparing the accelerometer problem for A-Design. In the MEMS domain, one can make a similar division of distinct components as that of the macroscopic electromechanical domain. Hence, we construct MEMS components using the FP-EB representation as a foundation. The basic Embodiments used in this accelerometer design and, in fact, the design of most MEMS devices are simply plate mass, beam and electrostatic gap. Our catalog contains only these Embodiments in horizontal and vertical forms as seen in Figure 8.7.

8.2.2 Evaluation

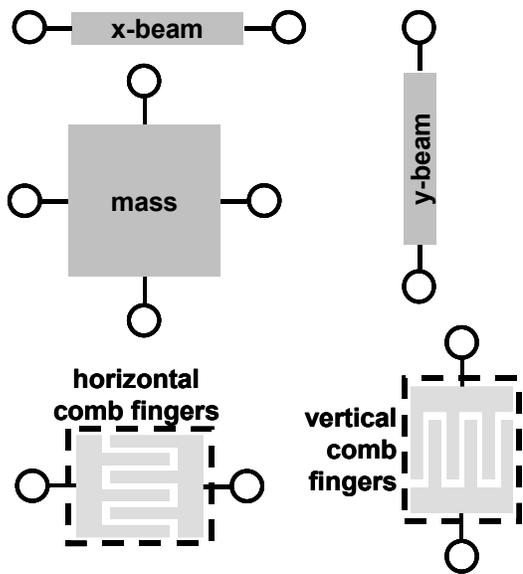
The four objectives of this problem are not closed-form equations. The values for area (A), sensitivity (S_x), maximum acceleration (a_{max}) and y-direction stiffness (K_{yy}) require additional analysis. Thus, some automated simulation of design states needs to occur as part of the larger automated design process. This is accomplished for the MEMS problem by invoking SABER (Analogy, Inc., 1995), a dynamic simulator that can handle mixed domain systems like MEMS. However, in order to use this application, an intermediate

Objectives

Minimize:

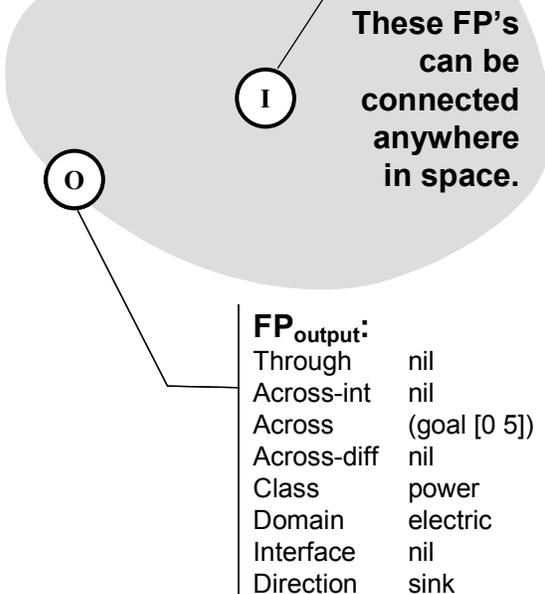
- Area (A)
- Inverse of sensitivity ($1/S_x$)
- Inverse of maximum acceleration ($1/a_{max}$)
- Inverse of orthogonal stiffness ($1/K_{yy}$)

Catalog



FP_{input}:

Through	nil
Across-int	nil
Across	nil
Across-diff	([0 10])
Class	power
Domain	trans-x
Interface	nil
Direction	source



FP_{output}:

Through	nil
Across-int	nil
Across	(goal [0 5])
Across-diff	nil
Class	power
Domain	electric
Interface	nil
Direction	sink

Figure 8.7: Description of accelerometer design problem as posed to A-Design

program was developed to automate the pre- and post-processing necessary to perform the simulation. This was done in a separate research project (Prakash and Cagan, 1999) and is part of the “evaluation” block of the A-Design flowchart as illustrated in Figure 8.8.

This automated analysis posed two new research challenges: the intermediate process must be robust enough to automate the analysis of the wide variety of configurations constructed by A-Design, and the evaluation of a single design state must be kept to a minimum amount of time as a result of the numerous evaluations required by A-Design. With these challenges, the research of Prakash and Cagan set out to develop a

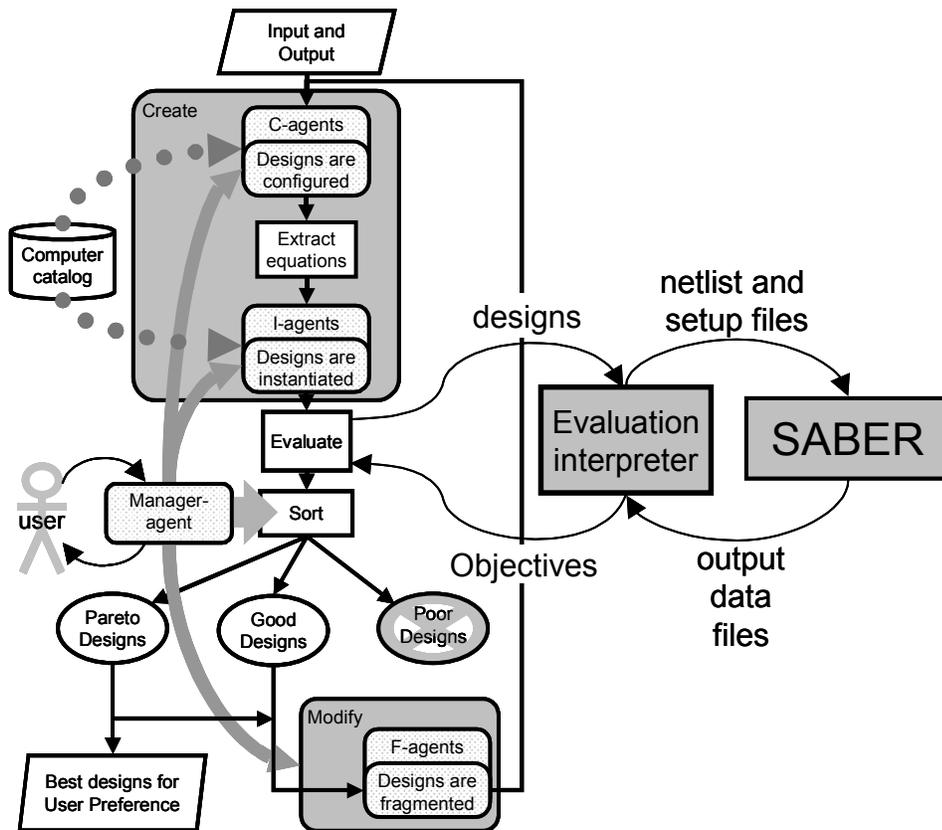


Figure 8.8: The additional computational demands of the evaluation process requires that external code be developed which is capable of further executing the SABER dynamic simulation.

hierarchical method for approximating the analysis of MEMS accelerometers for the A-Design system. This hierarchical method was structured to make large approximations at the beginning of the A-Design search process in order to reduce the computational time, and to increase accuracy of the evaluation as the search process begins to converge on successful solutions.

The basic function of the automated analysis is to structure problems for SABER to solve by setting up the proper input files and parsing the output files. In setting up the input files the boundary conditions and design schematic (or “netlist”) are created for each of A-Design’s configurations. Then SABER is invoked to determine the dynamic properties of the designs. To extract the results of the SABER simulation, output files are parsed by searching for values at key nodes in the results. The values of the nodes are then used to construct objective function values. Unfortunately, the implementation of this hierarchical evaluation method did not permit any analysis beyond the most simplified analysis. The variability in the configurations created by A-Design caused problems for automating analysis to the level of detail performed by experienced MEMS designers. This simplification in analyzing designs lead to fairly erroneous objective values.

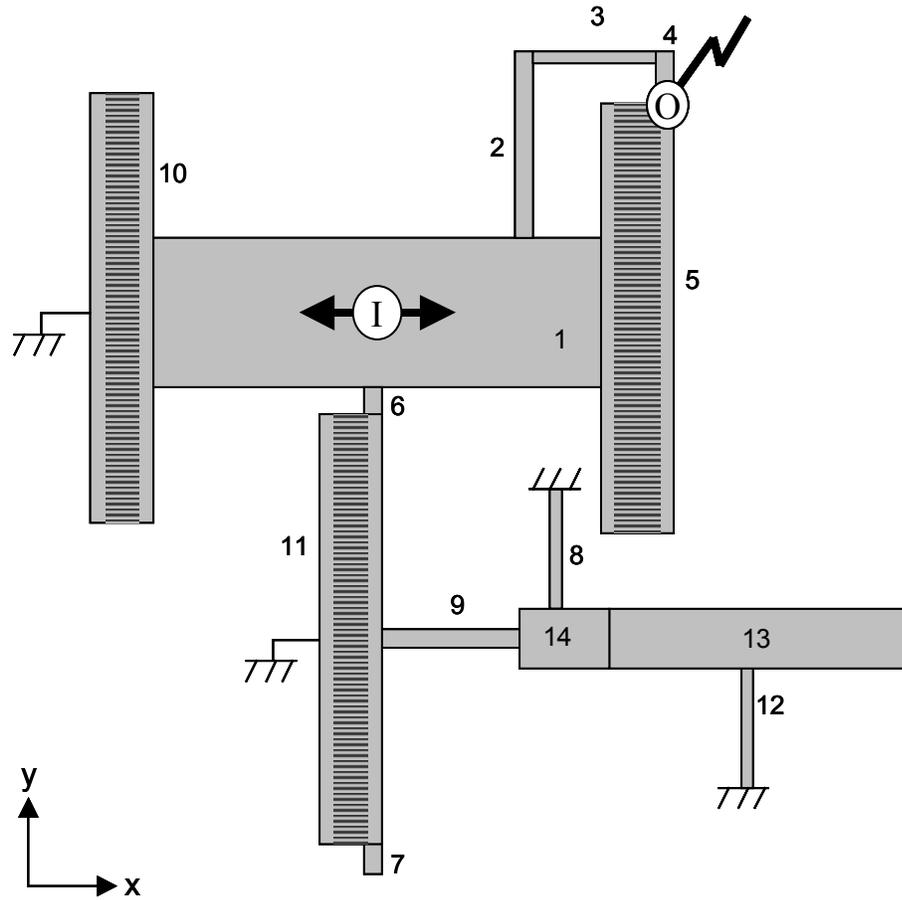
8.2.3 Results

Figure 8.9, 8.10, and 8.11 show MEMS accelerometers created by A-Design⁶. The search process follows the same model as the weighing machine problem with the exception that external analysis is used to solve the objectives. The three configurations

⁶ Like the weighing machine results, the figures are graphical renditions of the System Configurations produced by A-Design (examples shown in Appendix). In these figures, the input acceleration (FP_{input}) is applied at the I-label, while the output voltage (FP_{output}) is measured at the O-label. The instantiated components are listed below the figures with dimensions in meters (1.0e-6 meters = 1 micron).

in the figures show how this truly open-ended design problem can lead to some diverse solutions. Accelerometer A (Figure 8.9) is created with no TODO or TABOO learning. The time saved in omitting the trend-finding routines of the Manager-agent allows the process to run for more iterations (300 iterations with a population of 150 design states). Design A makes use of several electrostatic comb drives to sense the movement of the central mass but appears to lack an effective spring structure to provide the proper compliance in the x-direction.

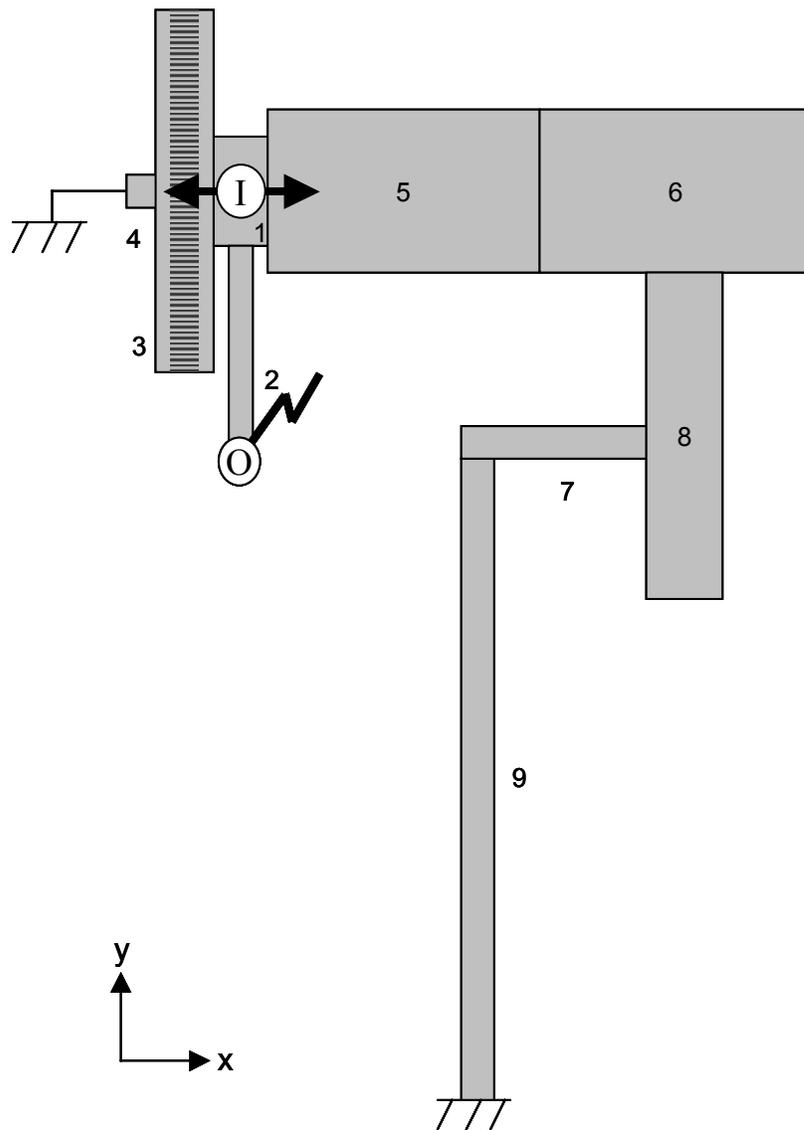
The MEMS accelerometer shown in Figure 8.10 (Design B) was conceived by the process in the presence of TODO and TABOO learning (population of 150 for 100 iterations). This design appears to utilize one long vertical beam as a useful strategy for making compliant structures in the x-direction while being relatively stiff in the y-direction. The final accelerometer, Design C in Figure 8.11, is designed by the system under the influence of a user-defined TODO element (also population of 150 for 100 iterations). At the beginning of the process, a U-spring design fragment is defined such that the search process has some domain knowledge of what features have previously been useful in accelerometer design. It appears that as a result of this clue about MEMS design, a more functional spring structure is created.



Components

1. **MASS-25-75** h=2.5e-5 w=7.5e-5
2. **V-BEAM-3-71** l=7.1e-5 w=3.0e-6
3. **V-BEAM-3-15** l=1.5e-5 w=3.0e-6
4. **H-BEAM-2-17** l=1.7e-5 w=2.0e-6
5. **H-ES-GAP-8** w=2.0e-6 l=10.0e-6
overlap=6.0e-6 #teeth=24
6. **V-BEAM-2-5** l=5.0e-6 w=2.0e-6
7. **V-BEAM-2-5** l=5.0e-6 w=2.0e-6
8. **V-BEAM-3-71** l=7.1e-5 w=3.0e-6
9. **H-BEAM-3-23** l=2.3e-5 w=3.0e-6
10. **H-ES-GAP-3** w=2.0e-6 l=5.0e-6
overlap=3.0e-6 #teeth=24
11. **H-ES-GAP-3** w=2.0e-6 l=5.0e-6
overlap=3.0e-6 #teeth=24
12. **V-BEAM-2-23** l=2.3e-5 w=2.0e-6
13. **MASS-10-50** h=1.0e-5 w=5.0e-5
14. **MASS-10-15** h=1.0e-5 w=1.5e-5

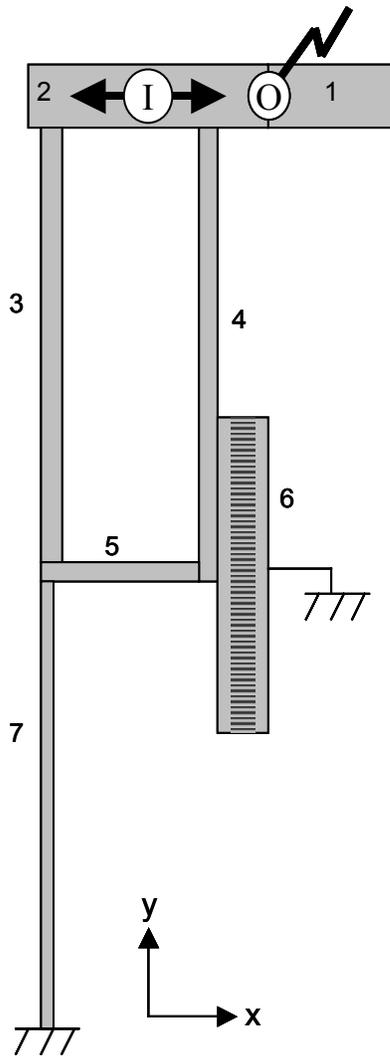
Figure 8.9: Accelerometer A is created by the A-Design process with no learning.



Components

1. **MASS-10-5** $h=1.0e-5$ $w=5.0e-6$
2. **V-BEAM-2-17** $l=1.7e-5$ $w=2.0e-6$
3. **H-ES-GAP-5** $w=2.0e-6$ $l=10.0e-6$
overlap= $4.0e-6$ #teeth=12
4. **H-BEAM-3-3** $l=3.0e-6$ $w=3.0e-6$
5. **MASS-15-25** $h=1.5e-5$ $w=2.5e-5$
6. **MASS-15-25** $h=1.5e-5$ $w=2.5e-5$
7. **H-BEAM-3-17** $l=1.7e-5$ $w=3.0e-6$
8. **MASS-30-7** $h=3.0e-5$ $w=7.0e-6$
9. **V-BEAM-2-3** $l=3.0e-6$ $w=2.0e-6$

Figure 8.10: Accelerometer B is created by the A-Design process under the presence of TODO and TABOO learning.



Components

1. **MASS-10-20** h=1.0e-5 w=2.0e-5
2. **MASS-10-20** h=1.0e-5 w=2.0e-5
3. **V-BEAM-3-71** l=7.1e-5 w=3.0e-6
4. **V-BEAM-3-47** l=4.7e-5 w=3.0e-6
5. **H-BEAM-3-30** l=3.0e-5 w=3.0e-6
6. **H-ES-GAP-3** w=2.0e-6 l=5.0e-6
overlap=3.0e-6 #teeth=24
7. **V-BEAM-2-71** l=7.1e-5 w=2.0e-6

Figure 8.11: Accelerometer C is created by initiating the TODO list with a U-Spring subsystem.

8.2.4 Discussion of Accelerometer Results

Inspecting these design states reveals the wide range of devices possible in this representation. The elements in this representation are more fundamental than the elements for the weighing machines problem. Even though there are fewer components to choose from, there is more variety in the possible configuration of these components.

In order to test the validity of these designs, an ADXL-style accelerometer was created for comparison through similar objectives. Using the optimization procedure outlined in Mukherjee et al., dimensions were chosen for the ADXL set topology⁷ resulting in an accelerometer with a sensitivity (S_x) of 10mV/G and a maximum acceleration (a_{\max}) of 10 G (shown in Figure 8.12). This design and the three A-Design accelerometers were compared manually using a thorough dynamic simulator (SPECTRE, Cadence Design Systems, Inc., 2000) as well as the automated approximation method used during the search process (Prakash and Cagan with SABER).

Figure 8.13 shows performance plots comparing all four designs under both evaluation methods. In this figure, each objective is multiplied by a normalization factor that scales the average for that objective to one. This is done to better visualize attribute tradeoffs because different attributes have very different ranges. For example, the areas of the four designs shown in this figure are: $A=5.8 \times 10^{-9} \text{m}^2$, $B=1.47 \times 10^{-8} \text{m}^2$, $C=8.32 \times 10^{-9} \text{m}^2$, $\text{ADXL-style}=2.38.8 \times 10^{-7} \text{m}^2$. The average of these values is $6.66 \times 10^{-8} \text{m}^2$. Therefore, each

⁷ This sizing can be run online at <http://www.ece.cmu.edu/~mems/memsyn/accsyn/index.html>.

point is divided by this average value so that the plotted values are $A=0.09$, $B=0.22$, $C=0.12$, and $ADXL=3.56$.

In Figure 8.13a, the in-depth SPECTRE analysis shows the three A-Design accelerometers with similar values for the objectives, while the ADXL-style accelerometer values are quite different. In three of the four objectives, one of the A-Design solutions performs best, however the ADXL performs extremely well on the sensitivity objective. In Figure 8.13b, the approximate method developed for A-Design shows very different values for the objectives. In fact, for each objective except area, the ordering of designs from best to worst differs between the two evaluation methods.

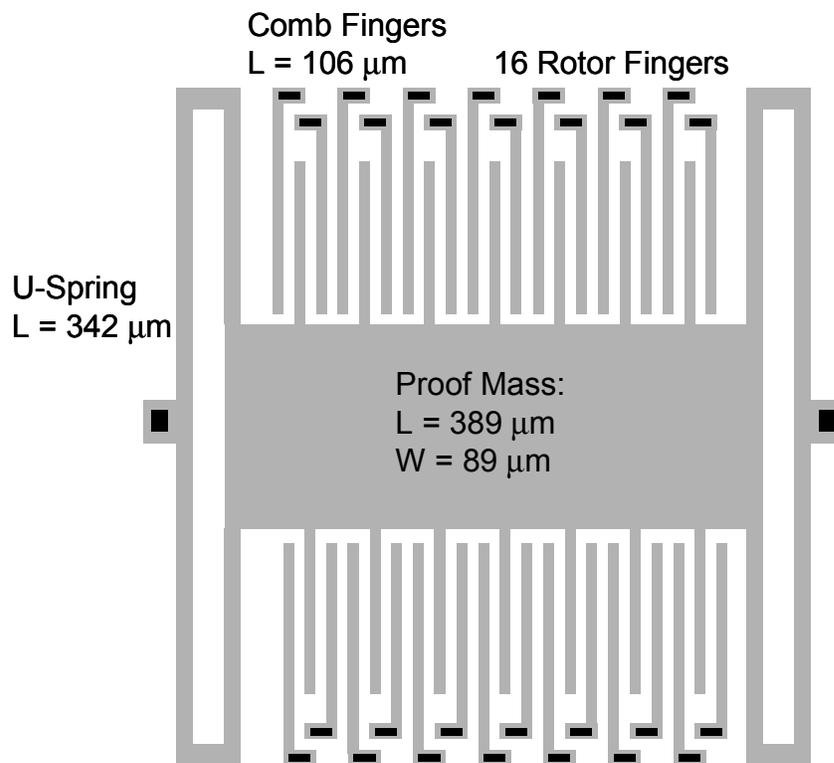


Figure 8.12: ADXL-style accelerometer optimized using approach from Mukherjee et al. (1999).

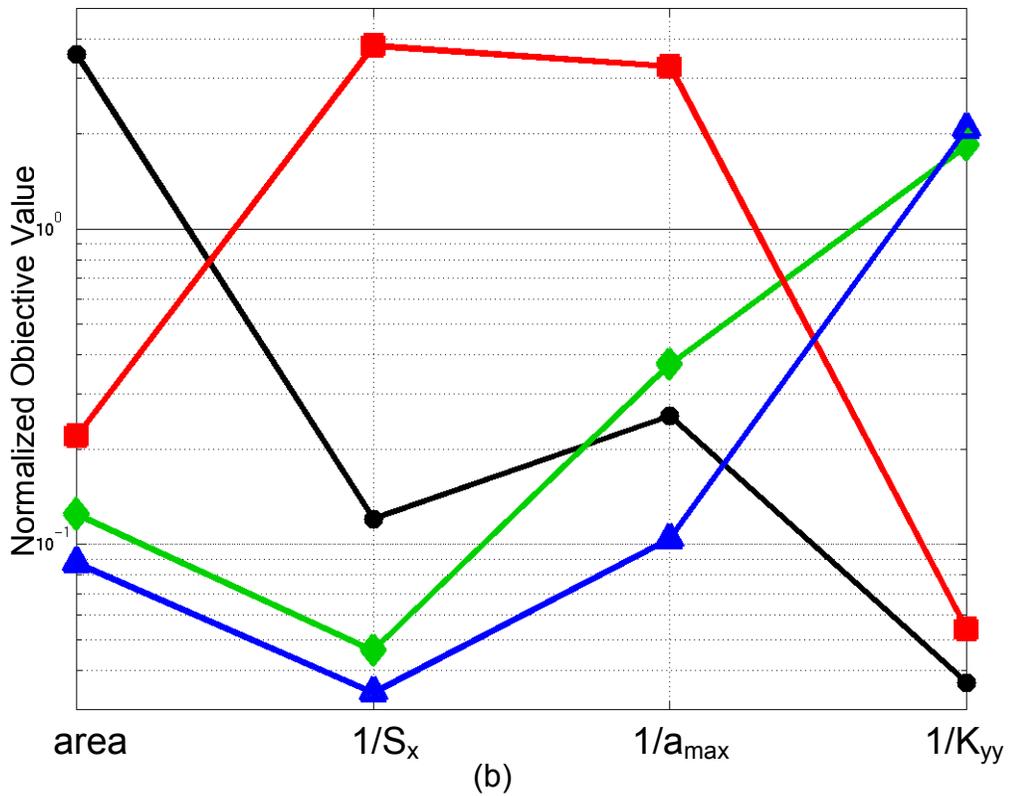
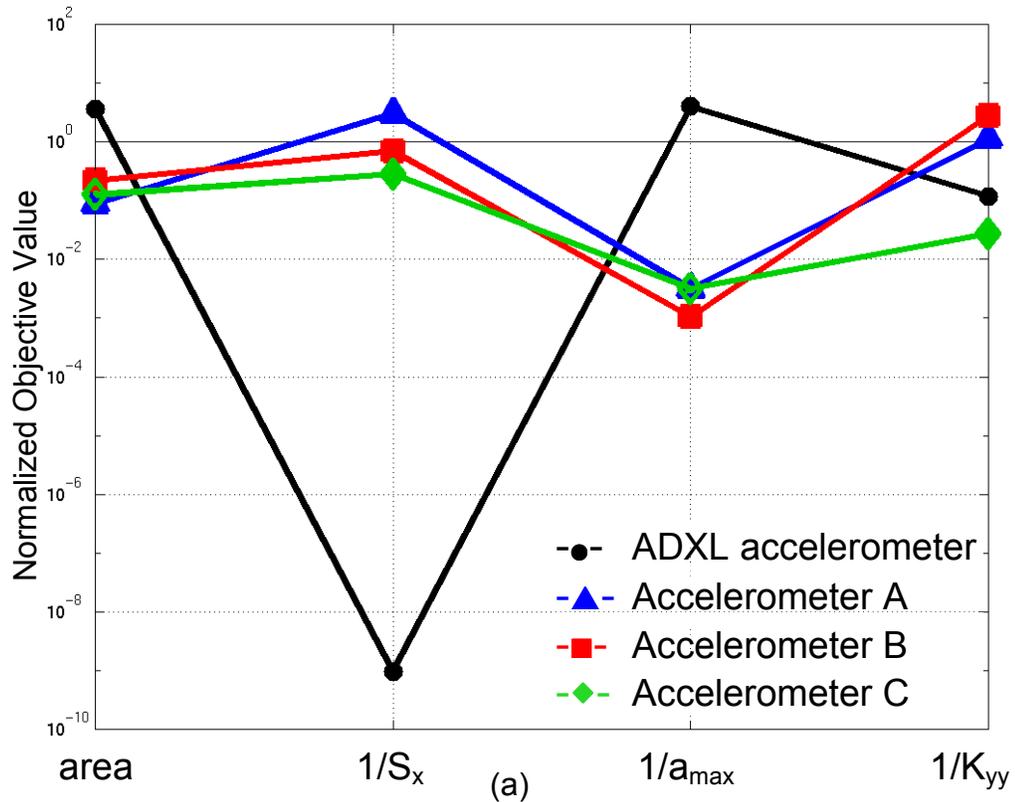


Figure 8.13: Performance plot of the three A-Design accelerometers and the ADXL-style accelerometer according to (a) SPECTRE analysis done by hand, and (b) heuristic based automated analysis.

According to the evaluation used by A-Design, the performance of ADXL configuration is not unique or exceptional. The strength of the ADXL design is not evident from the approximate analysis in Figure 8.13b. However, the more thorough analysis in Figure 8.13a shows the ADXL accelerometer producing much more useful sensitivity than the remaining designs.

In this example, the sensitivity objective turned out to produce highly erroneous results. While simpler objectives such as area are not complicated by varying configurations, the sensitivity objective was susceptible to such diversity. The evaluation interpreter did not include sophisticated enough preprocessing to handle the variety of configurations that A-Design passed to it. As a result many configurations left the evaluation process with default values because they could not be properly analyzed. In addition to analysis accuracy and robustness, there is the need to keep analysis times to a minimum. These challenges with evaluation demonstrate the difficulty in combining automated design with automated analysis. As the Introduction mentioned, computational analysis is at a more developed stage than computational design. However, as computational design advances and more ambitious design problems are addressed, a better understanding of how to assimilate the computational analysis with computational design becomes crucial.

The designs made automatically by A-Design are quite different from the ADXL-style accelerometers. Most notably the A-Design accelerometers have fewer components and lack symmetry. Part of this could be due to the lack of design insight provided in the

objectives. Experienced MEMS designers can immediately notice deficiencies in the three A-Design devices. These deficiencies relate to design constraints not stated in the design problem. Issues such as frequency modes of operation, and unexpected rotations may lead to erroneous or possible disastrous behaviors. Increasing the exactness of the functional representation or the evaluation might overcome these “second-order” effects not discernible from current analysis.

The results in this section demonstrate A-Design’s ability to invent a variety of design solutions for the complex domain of MEMS devices. With better analysis, the inventive power of the process would ideally lead to more useful designs. While the inexactness of evaluation scheme leads to simplistic results, the present form of A-Design could serve as a design-aid to the MEMS designer by producing novel configurations from scratch or from previously learned examples.

8.3 COMPARISON OF TWO DESIGN PROBLEMS

In comparing the two design problems described in this chapter, we gain insight into several aspects of the A-Design process. First, learning in the system can do more than make the process more efficient. In other stochastic search processes, statistics from previous iterations are used to make the process less time-consuming. In A-Design, the Manager-agent together with the iterative search leads to a similar stochastic process to guide design construction. However, previous search steps are also used to store useful design features. In both the weighing machine problem and in the MEMS accelerometer problem learning leads to better design states.

Also, in one version of the MEMS accelerometer problem, a priori design knowledge (TODO = U-Spring) was introduced to guide the process from the start, thereby allowing the computational process to start where previous design innovations left off. When an engineering designer starts a new problem, he/she uses previous learned experiences as the foundation for building new designs. Learned instances are a large part of our design process, and this method of adding knowledge similarly provides a computational system with the head start that human design experiences. Unlike many expert system approaches to design automation, this design experience does not restrict solutions to conform to previous instances. Rather, the design knowledge can guide innovation to build on past experiences to create entirely new concepts, as can be seen in the spring structure in Figure 8.11 built from the U-Spring design fragment.

The second insight gained is that different design domains challenge different areas of the A-Design methodology. In the weighing machine example, limitations of the current functional representation prevented the construction of even more realistic designs. However, in the MEMS test problem, the automated analysis proved to be the bottleneck to creating realistic designs. The reason for this could be the fact that MEMS offers simple component descriptions but complex behaviors for the sum of the components, while weighing machine components are complex in their immediate interactions but less complex in overall behavior.

It appears that the challenges of these design problems are the representation and evaluation. Both representation and evaluation, which are the crux of engineering analysis, are necessarily simplifications of what happens in the real world. Modeling for

analysis purposes always involves some simplification. So, it is not surprising that through similar simplification, the results of the A-Design process appear somewhat naïve.

Despite the limitations, the results of this chapter show that the implemented A-Design system is able to invent solutions for two open-ended engineering design problems. The input and output specifications, the objectives, and the catalog of components are all that is required by A-Design to invent novel design configurations. The collaborative unstructured design activity in A-Design is founded on human properties of design, and similar to human design, leads to the development of creative and diverse designs.

Chapter 9

Experimental Results

The results shown in the previous chapter exemplify A-Design as an invention machine for solving *open-ended* design problems through the *collaborative* interaction of agents. This chapter sets out to explore the *iteratively-guided* and *adaptive* claims of the A-Design theory. A series of experiments on the weighing machine design problem determine how these constituent parts of the theory add to the overall inventive power of A-Design.

9.1 SIGNIFICANCE OF RANDOM STARTING POINTS

The first test addresses whether the randomness of A-Design's stochastic nature is significant enough to prevent comparison of different runs. The randomness of initial starting points, or designs, might have a significant effect on the final results of the search process. Ideally, the starting points should have little effect on the end results, because the search process should find successful designs from any starting point in the process. To test this, a comparison was made of 20 runs all with the same initial population to 20 runs with random initial populations. Each run consisted of 60 iterations of the A-Design process using a population of 100 designs. As shown in Figure 9.1a, these two sets of

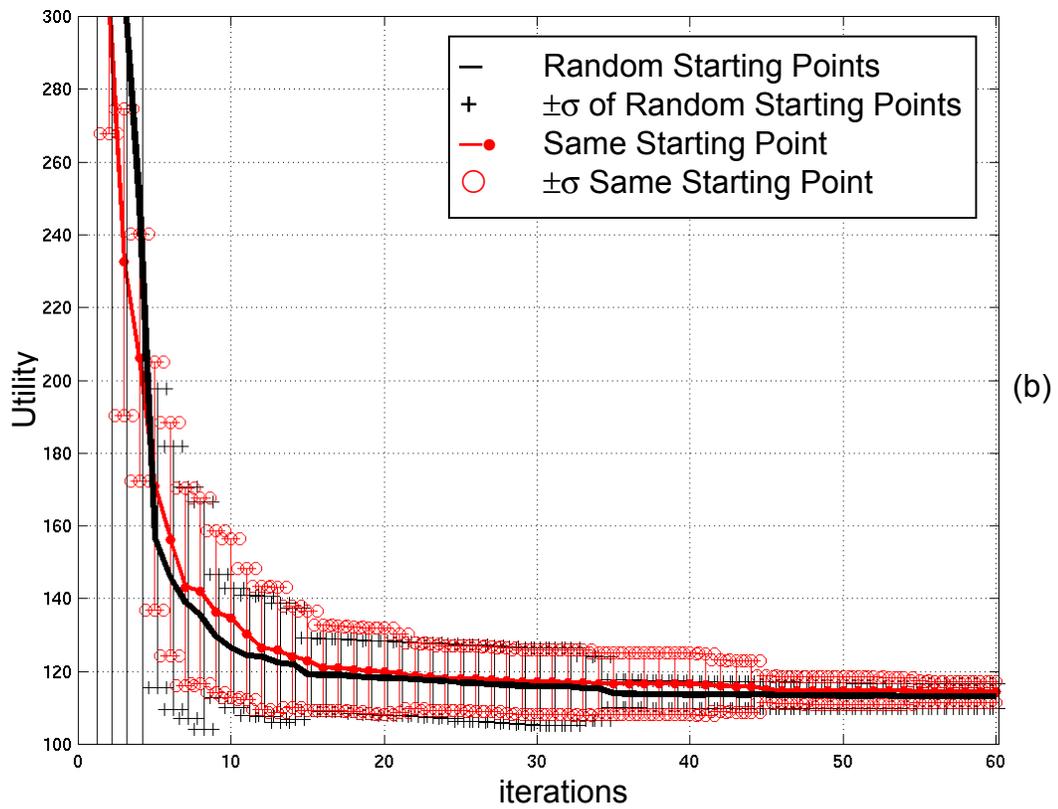
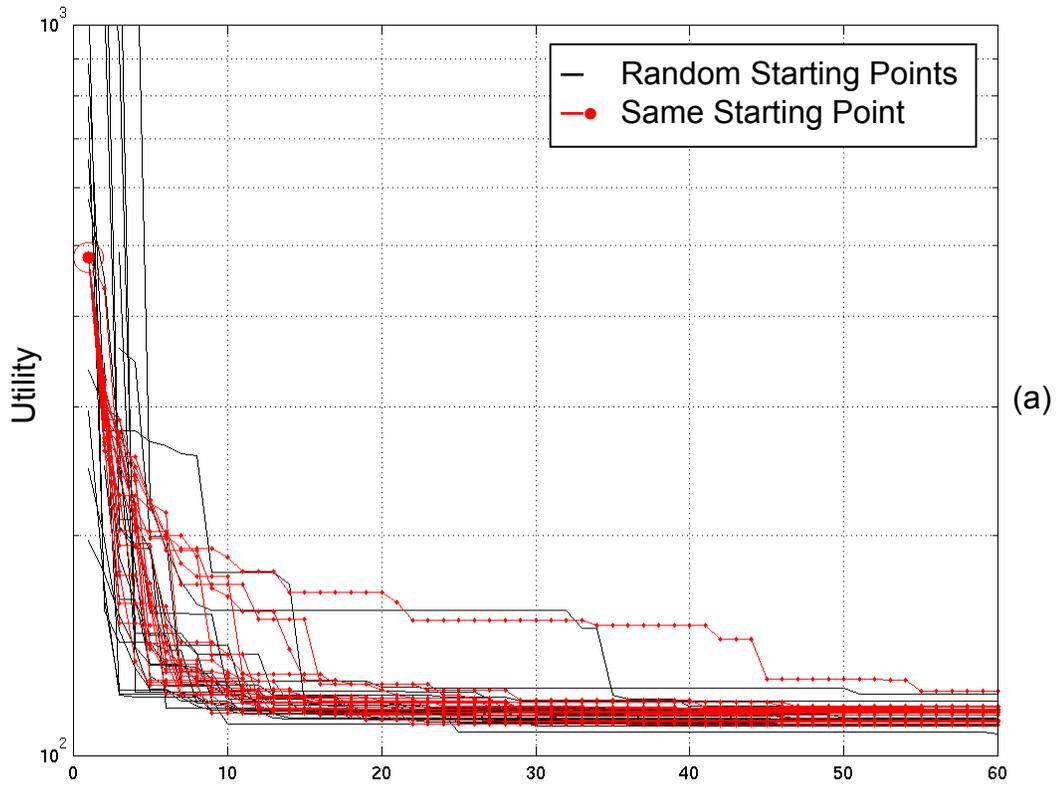


Figure 9.1: Comparison of random vs. set initial design populations a) best design at each iteration for 20 runs, b) average of best design from 20 runs.

runs are compared by plotting the best design at each iteration for a set user preference. The gray dotted lines show runs containing the same initial population and therefore all have the same points at the first iteration, while the dark solid lines have very different designs in the first iterations. Although there is much fluctuation in the beginning of the processes, all runs converge to a fairly uniform value. Figure 9.1b clarifies the results shown in Figure 9.1a by averaging the runs to single line and plotting the standard deviation of the 20 runs by an error bar (+/- one standard deviation, σ). Here the average values and the standard deviation values are nearly identical at the end of the process thereby demonstrating that there is little difference in the data produced through random starting points as opposed to set starting points. The pruning of designs and the execution of guided agents quickly eliminate the effect of starting at different points in the design space. Therefore in the following tests it is assumed that the use of random starting points has little effect on the statistical results.

9.2 ADAPTABILITY EXPERIMENTS

Next, a series of tests were performed to analyze the design selection subsystem of A-Design (see Chapter 3). The claim is made that A-Design is an effective mechanism for adapting to changing market demands or user preferences throughout the search for successful design states. In order to test this claim, a series of experiments were performed to test the importance of the Pareto population, the importance of the Good population and the adaptability of the process.

9.2.1 Good Set and Pareto Set Isolation

The A-Design theory is based strongly on the design selection methodology of dividing the population into Pareto, Good and Poor sets. In order to observe the effects of this division, we isolated the Good and Pareto populations in separate runs and compared their ability to optimize the user's objectives. Figure 9.2 shows the best designs at each iteration averaged over 20 runs as was done in Figure 9.1b. The three runs compared in this figure separate designs into two or more populations. The "Both Pareto and Good" run divides designs into three populations as described in Chapter 3, the "Pareto only" run divides designs into only Pareto and Poor solutions, and the "Good only" run divides designs into only Good and Poor solutions. Since the actual Pareto designs in this "Good

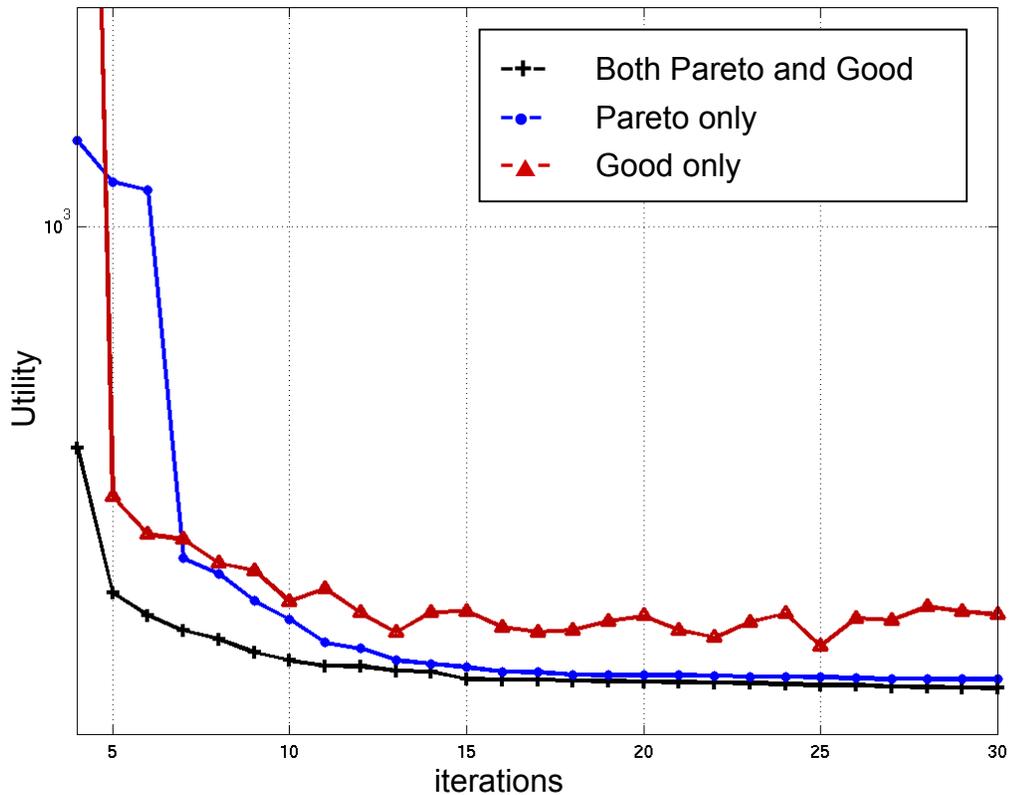


Figure 9.2: Comparison of best designs from Pareto only, Good only and both.

only run” are not separated out, those Pareto designs that are preferred by the user preference are also included in this run’s Good population.

In this figure, the “Good Population only” has a higher, and therefore inferior, lumped evaluation value than the runs containing a Pareto population. Comparing the final design from the “Pareto only” run with the “both Good and Pareto” run finds only small differences; both processes find a similar value for their best design.

From this experiment we can draw the conclusion that the Pareto population contributes significantly to the process’ ability to find successful solutions. This was a surprisingly useful conclusion to draw since the Pareto population was originally preserved to allow an adaptability to changing user preferences. However, this conclusion shows the importance of saving diverse solutions even in the exploration of a static design problem. It is believed that preserving diverse design states allows the system better coverage of the search space of possible designs. In a sense, the unpreferred Pareto designs are equivalent to biological recessive characteristics that are transferred but not expressed.

The results in Figure 9.2 do not prove the significance of the Good population in finding improved design states. Its inclusion in the “both Pareto and Good” run leads to slightly better solutions over the Pareto only run, however, not a substantial enough amount to validate the theory. There is, however, a substantially quicker convergence of the best design in the “both Pareto and Good” compared to the “Pareto population only” run seen in the earlier part of the process. Since the algorithm is run for an arbitrary number of iterations as decided by the user, quicker convergence may produce better

design states if the user stops the algorithm prematurely. Additionally, the results of the next experiment show how the Good population plays an important role in adapting to changes in user preference.

9.2.2 Changing User Preference

This experiment tests the true adaptive capability of A-Design by observing the effect of switching the user preference at the middle of a run of the search process. In the experiment, both Pareto and Good populations are in operation when the user preference changes at the thirtieth iteration (of 60 iterations). With this change, a new location for the Good population is defined with respect to the Pareto surface as is seen in Figure 9.3a. The results shown at the bottom of Figure 9.3 are created through averaging over 20 separate runs of the A-Design process and plotting the best design for a given user preference.

In this experiment, three separate runs are compared. Figure 9.3a and Figure 9.3b graphically depict the various runs plotted in Figure 9.3c and Figure 9.3d. These plots show how the process reacts to switching from a preference favoring low-cost and low-weight designs (W1, the same preference used to produce the weighing machine in Figure 8.2a) to one favoring minimal input displacement designs (W2, the preference used in Figure 8.2b and Figure 8.2c). Figure 9.3b shows a flowchart of how this experiment compares five sub-procedures; sub-procedure **E** is the focus of this experiment while **A**, **B**, **C** and **D** are the control for the experiment. Sub-procedures **A** and **B** are created under a constant user preference, W1, while sub-procedures **C** and **D** together represent the system under constant user preference of W2. After 30 iterations

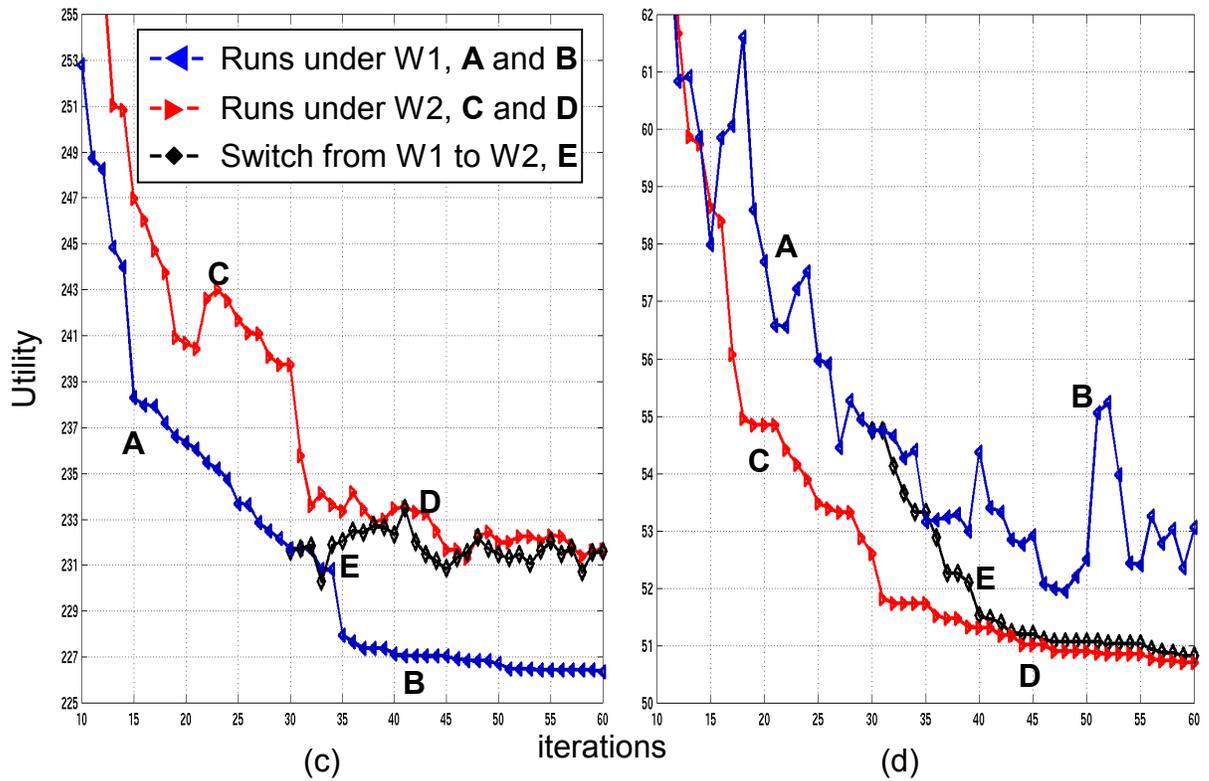
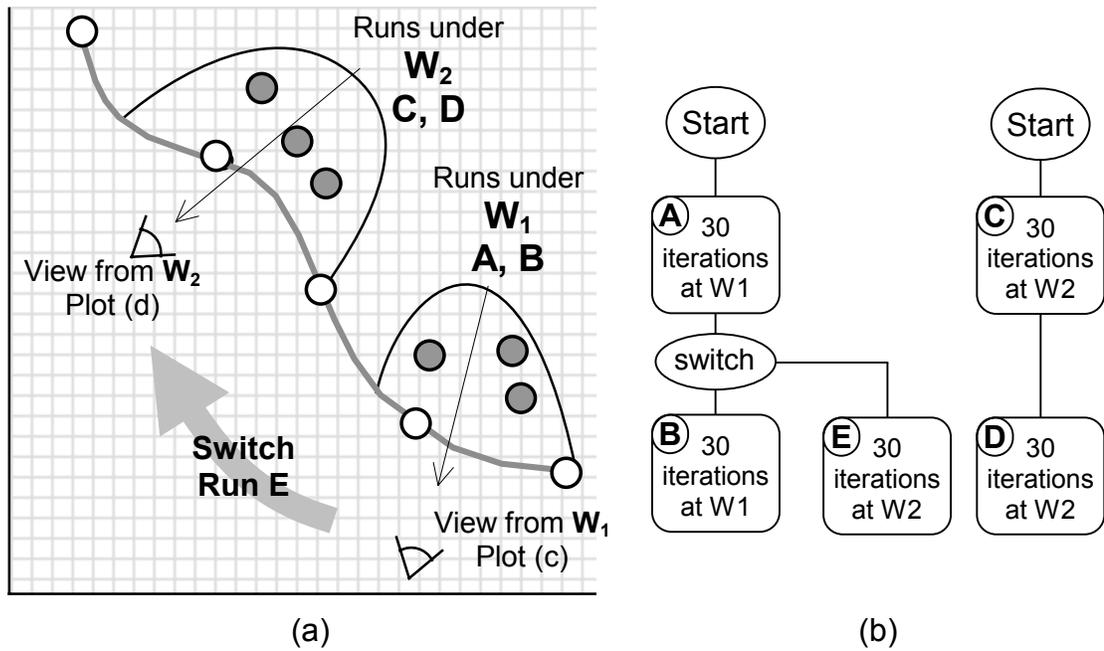


Figure 9.3: Adaptability of A-Design: a) the change in preferences is viewed on two different plots to see the effect, b) flowchart of adaptability experiment, c) best designs viewed by W_1 , d) best designs viewed by W_2 .

under preference W1 (**A**) the process is stopped and sub-procedures **B** and **E** are initiated with the agent and design data last achieved in sub-procedure **A**. To view the results, all five sub-procedures are plotted as viewed under W1 preference (Figure 9.3c) and under W2 preference (Figure 9.3d). Therefore, by comparing the best lumped evaluation under these preferences we can determine how the process responds to optimizing the objectives under these preferences. It is important to note that while all five sub-procedures are plotted in both Figure 9.3c and Figure 9.3d, each procedure concentrates on only one user preference. Thus, the plots are showing some procedures in a different preference than the preference governing the creation of the designs for that procedure. For example, **C** is created under user preference W2 despite the fact that in Figure 9.3c, **C** is plotted in reference to W1. It is for this reason that **A** and **B** have better values in Figure 9.3c while **C** and **D** have better values in Figure 9.3d.

The switch in preference (**A** to **E**) occurs at the thirtieth iteration at which point the process stops optimizing designs for W1 and the Good population shifts focus to user preference W2. Although viewing sub-procedure **E** in Figure 9.3c appears to offer no improvement after the switch, there is a significant gain shown when viewed from the perspective of W2 (Figure 9.3d) as is expected since the process is now optimizing under W2. The switched run, **E**, begins at much lower value compared to run **C**. The “recessive” Pareto solutions developed in **A** allow the search in **E** to begin at an advanced stage as opposed to high values at the beginning of run **C**. In addition, run **E** almost completely recovers from the time spent developing designs for W1 as seen when comparing **E** with **C** and **D**.

Due to the Pareto-based conservation of diverse designs, the process was able to accommodate the switch to the W2 preference after spending half of the iterations concentrating on a completely different user preference. While it is believed that the recessive characteristics in the Pareto set allow the system to start at an advanced stage in the search process instead of starting from undeveloped design states, this experiment has shown that the Good population allows the process to concentrate on the current user preference. In this experiment, the use of the Good population in conjunction with the Pareto population has shown to be a successful way of handling changing user preference. The Good population provides focus for the process in searching under the current user preference, while the Pareto population allows flexibility if the preferences should change.

9.2.3 Adding and Removing Objectives

In addition to changing the relative importance of objectives, it can also be the case that the number of objectives in a conceptual design problem change. Outside influences can invoke the demand for new criteria, or perhaps previous design objectives become no longer important. This experiment seeks to understand how A-Design adjusts to such changes.

Figure 9.4 shows the average of 20 control runs (40 iterations with a population of 100 designs) of the A-Design process with both four objectives and three objectives (shown as black dots). Note that the value for four objectives is higher due to the fact that the linear weighted sum of four objectives has an extra term. After twenty iterations in the process, separate runs are initiated based on these control runs. A fourth objective is added to the control with three objectives, and the fourth one is omitted from the control

with four objectives. Again, this experiment involves the design of weighing machines. The fourth objective that is being added or subtracted in this experiment is the “minimize input displacement” objective.

The first thing to note in this experiment is the large changes between iteration 20 and 21. This is due to the fact that the addition or subtraction of the fourth objective causes the current population of designs in the process to be evaluated quite differently. The control runs in this figure are at an advantage in that they have 20 iterations of search over these newly adjusted runs. However, in the remaining 20 iterations, the runs with an added or removed objective are able to quickly adapt to the new utility function and find solutions with nearly the same quality as when the process starts from inception with a

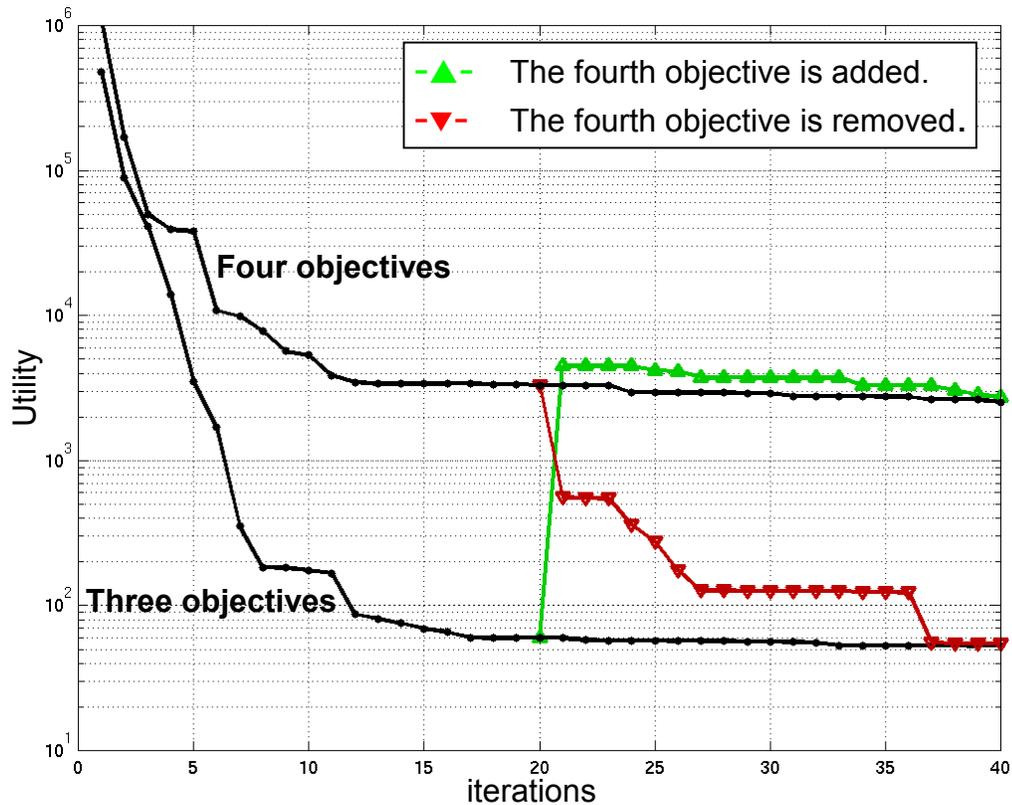


Figure 9.4: The addition and removal of an objective to further test the adaptability of A-Design.

static objective function.

Also, one might be surprised to see a significant difference in objective values for the case when the fourth objective is removed in Figure 9.4. It is expected that the inclusion of a new design goal or objective would cause some setback in the design process, but it is not clear that the removal of design parameters would have a drastic effect. The reason for this effect is due to the competing objectives in this formulation. We mention in the Introduction how design problems often contain contrasting goals that lead to the compromise of design objectives. In the weighing machine problem, the minimization of input displacement presents a real challenge to the search process. When this objective is removed (as is the case between iteration 20 and 21 in Figure 9.4), observe how the remaining objectives have been compromised in solving the design problem. The agents quickly “realize” that this objective is no longer a concern for the design. The three remaining objectives can be improved more now that the constraint implied by this fourth objective no longer remains.

In conclusion, the A-Design process is able to adapt to changes in the user formulation of the design problem. This is a crucial development in the advancement of automated conceptual design. No longer is the user's interaction with a design automation technique a static one. The process is able to adapt to changes that the user might introduce as a result of changes in personal user preferences or in larger changes such as market demands.

9.3 LEARNING EXPERIMENTS

As shown in the previous chapter, the existence of learning in the system can provide the automated process with a better understanding of the design problem's functionality and search space. Closely monitoring past design activity can lead to better future designs. The notion of gathering data on past design activity and learning from such data is the basis for the iteratively-guided subsystem of A-Design. This section seeks to understand how another learning mechanism, the TODO and TABOO lists, affect the process.

Figure 9.5 shows three curves whose points represent an averaging of 20 separate runs of the A-Design system. Each run contains 40 iterations with a population of 100 designs. The three curves show A-Design with no learning, with TODO learning, and with TABOO learning. The TODO and TABOO trends are stored on a queue of fixed size (12 elements in this experiment) that contains agent teams, and design fragments. When new trends are found, the oldest trends are removed from the set.

Early on, there is not much difference in the three runs, but as the runs converge, the effect of TABOO learning leads to a much improved design state over the other two runs. Interestingly enough, the TODO learning, although the quickest process in the first 15 iterations, begins to level off and experiences the same rate of improvement as when no learning is present; however, the end effect of the TODO learning still offers a significant improvement over the run with no learning.

Figure 9.6 compares some hybrid learning techniques that involve combinations of both TODO and TABOO learning. The runs in Figure 9.5 are shown lighter in this figure to accentuate the results of the hybrid approaches. This figure shows three TODO/TABOO combinations: BOTH, KICK and UPDOWN. BOTH contains the exact TODO and TABOO list size as in the independent cases, and this list size is maintained for all iterations. The KICK and UPDOWN tests contain dynamic TODO and TABOO list

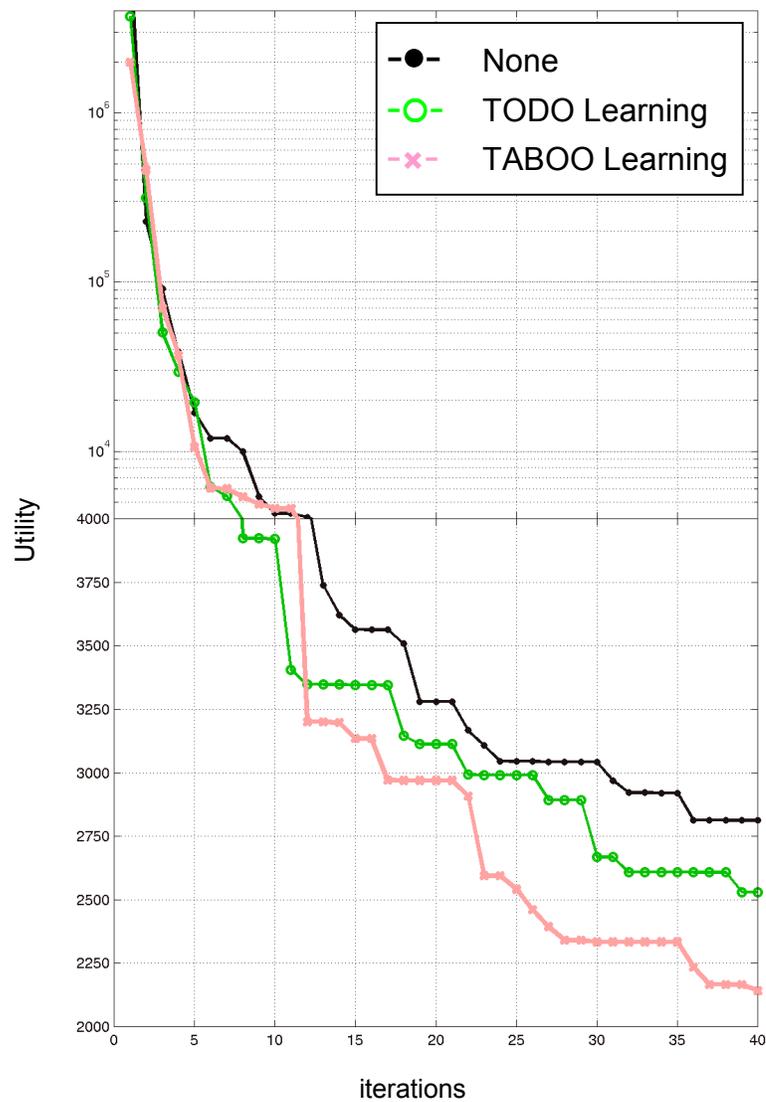


Figure 9.5: A comparison of TODO, TABOO and no learning.

sizes. In KICK, a constant TODO list size is maintained, and the TABOO list is zero at all iterations except at every tenth iteration where it contains up to 15 members. The motivation for this strategy is to “kick” the process intermittently so as to avoid getting trapped in local optima. The UPDOWN procedure attempts to take advantage of the quick learning shown in the TODO and TABOO comparison of Figure 9.5. In the first 15

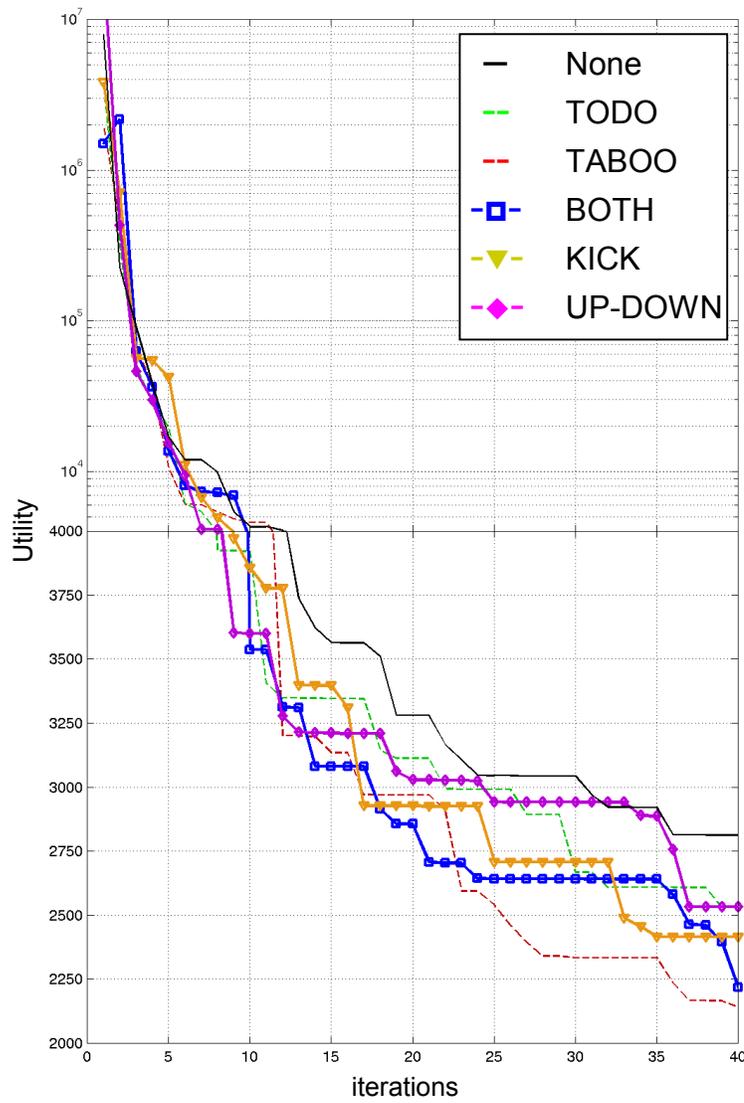


Figure 9.6: A comparison of three learning techniques that involve a combination of TODO and TABOO.

iterations, TODO has a population size of 15 members then diminishes to zero, while TABOO learning is increased from 0 to 15 members at the fifteenth iteration.

Upon examining the results in Figure 9.6, the system with simply TABOO learning performs better than any of the hybrid approaches. Interestingly enough, the hybrid approaches all fall somewhere between TODO and TABOO. The unadulterated TABOO learning appears to be the most successful feedback for these problems. This type of avoidance feedback is also the basis for Tabu search (Glover, 1989), which operates under a similar principle. The advantage in remembering bad design trends is twofold. First, it provides a means of tracking common mistakes that occur during design construction so that these can be avoided in future moves to make the process more efficient. Second, the avoidance of past moves can produce atypical designs or innovative search that allows the process to escape local optima.

In addition to the comparison shown in Figure 9.6, a performance plot shown in Figure 9.7 compares the best final design found from the 20 runs in each learning technique. In this plot, the best design without learning appears to be deficient in nearly all comparisons. The three best designs from BOTH, KICK and TABOO are nearly equivalent, while the best solutions for the TODO and UPDOWN algorithms actually appear to find better values for the “minimize input displacement” objective. The reason for this is not well understood. It is possible that the early TODO learning in these two approaches provides insight into the most challenging design issues. TODO learning exploits the benefits of successful past solutions. This might not lead to design diversity but it does allow A-Design to concentrate on improving known successes. Conversely,

TABOO learning promotes the exploration of new avenues of invention by avoiding past trends.

Future experiments with TODO and TABOO learning within various design domains might clarify the advantages of each learning technique. However, it is safe to say that the learning mechanism incorporated in the A-Design process better guides the process to successful designs, and makes strides towards incorporating the kind of human learning that is used in conceptual design.

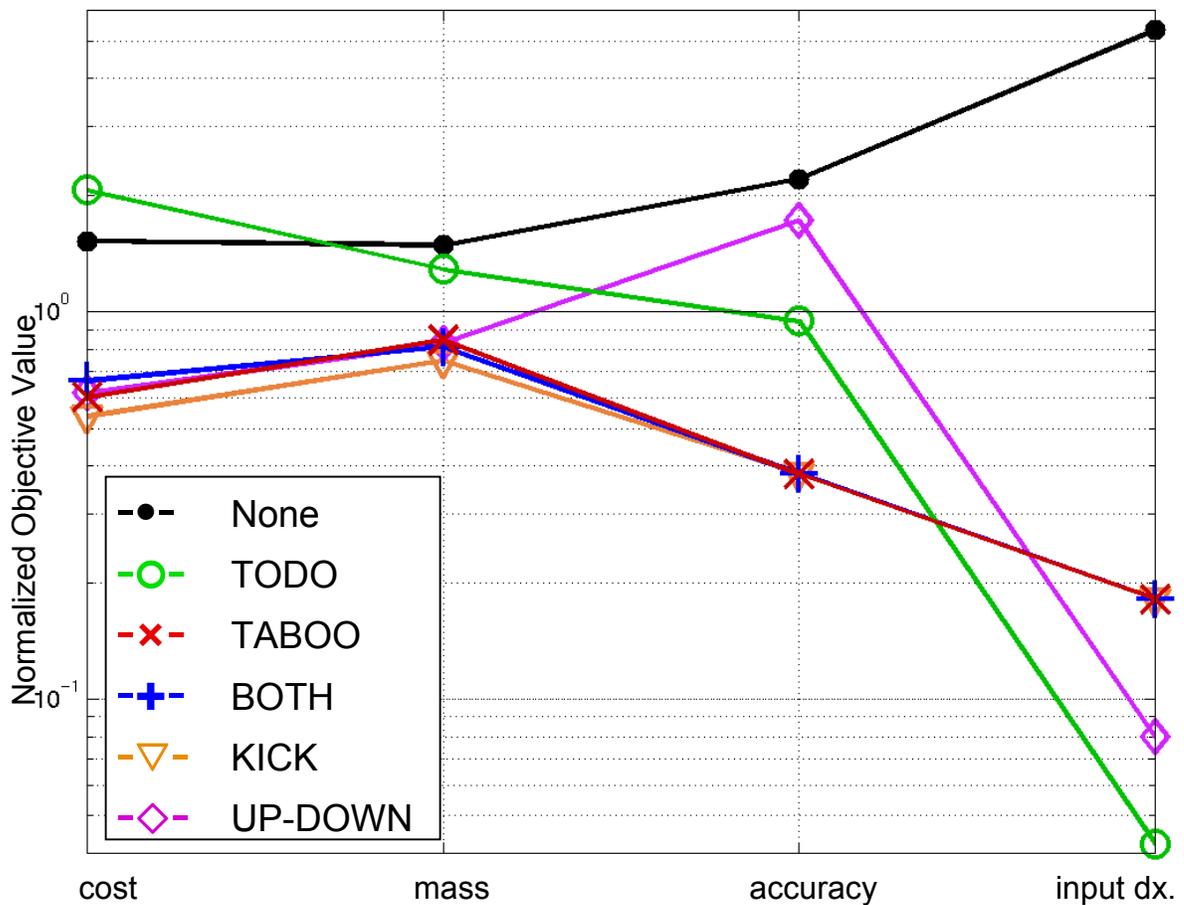


Figure 9.7: Performance plots for learning techniques shown in Figure 8.5 and Figure 8.6.

Chapter 10

Conclusions

10.1 SUMMARY

This dissertation has introduced the A-Design automated approach to conceptual design. Through its implementation, A-Design invents solutions to open-ended design problems. The creation of designs occurs as a result of the interactions of a multitude of agents folded into a stochastic iterative process capable of adapting to changes in user preference. The innovations of A-Design are based on a number of related research topics including Stochastic Optimization, Genetic Algorithms, Artificial Life, Multi-Agent Systems, Qualitative Physics, Bond Graphs and Utility Theory. Table 10.1 provides a summary of these related research topics and the A-Design subsystems they influence. The table also shows how A-Design expands upon or diverges from these topics to create the unique constituents of the theory by including characteristics of human conceptual design. The goal of A-Design is to investigate and integrate qualities of the human design process into a computational algorithm. This has been accomplished by creating four subsystems that each embody different characteristics found in human conceptual design.

Table 10.1: Derivation and innovations of the A-Design theory and the electromechanical implementation.

A-Design Subsystems	Agent architecture	Search Process	Design selection	Functional Representation
Human Design Characteristics	Collaboration	Iteratively-guided and Learning	Adaptive	Open-ended
Related Work	Artificial Life, A-Teams, other Multi-agent Systems	Stochastic Optimization, Genetic Algorithms, Tabu Search, Reinforcement Learning	Pareto-optimality, Satisficing, Multi-attribute Utility Theory	Qualitative Physics, Function Grammars
A-Design Innovations	Agents are goal-directed and work within a framework where feedback and collaboration yield diverse design solutions.	A-Design iteratively improves upon existing design alternatives and creates new ideas through management of Maker- and Modification-agents.	Designs are divided into 3 unique populations that are used to add adaptability and diversity to design states.	Two-tier representation describes both abstract and real components and allows for more general connectivity.

One of the key aspects in A-Design is the combination of optimization algorithms and knowledge-driven techniques. Knowledge-driven strategies are contained within software agents that interact iteratively to search the design space in a stochastically guided manner. This combination takes advantage of searching over numerous iterations while making sense of ill-defined problem spaces.

Initial test examples help to demonstrate the effectiveness of the algorithm's adaptability and search success. In the Manhattan Transfer test example, A-Design found solutions for a specific user preference and quickly adjusted when a change in preference occurred. The numerical optimization test example pits the A-Design theory against a traditional SQP algorithm. Although the SQP method runs faster than A-Design, it often

gets stuck in local optima while A-Design more consistently finds the global optimum. Agent interaction and Pareto design selection in A-Design allow solutions to be searched in the non-monotonic and multi-modal spaces of the test examples.

To address electromechanical problems, A-Design is supplied with a description of the design problem in the form of inputs and outputs, a set of objectives to be optimized and a library of electromechanical components. The results from the weighing machine test problem display a diverse set of possible design alternatives that can be created. These design configurations depict a successful combination of the agent architecture and functional representation subsystem. The application of A-Design to the design of a MEMS accelerometer shows promise for A-Design in solving a very different design problem. Learning from past iterations allows for a better understanding and coverage of the design space, and hence better final design solutions.

Experiments were performed to test the adaptability and learning in the system. These experiments validate the theory's separation of designs into Pareto, Good and Poor subsets as an effective way to both optimize objectives and retain flexibility. Also, the TODO and TABOO learning guide the agents through both positive and negative exemplars. This learning leads to the creation of improved design solutions that are not easily attained in the absence of learning.

10.2 DISCUSSION

In the Introduction, the development of "computer as analyzer" compared to the development of "computer as designer" illustrated several of the challenges to the latter that have inhibited progress in this area. Understanding and overcoming these challenges

has been the motivation for researching and developing the A-Design system. The first challenge, incorporating traits of human design, has been the basis for the A-Design methodology, while the second challenge (investigating the foundation of design) and third challenge (integrating design and analysis) have been referenced intermittently throughout this dissertation. Each of these three challenges is discussed in the following sections.

10.2.1 Automating Conceptual Design: Theoretical Claims of A-Design

The results of the previous chapters show a computational system capable of conceptualizing innovative and interesting designs given merely a functional description and the metrics for measuring good designs. These results of the A-Design implementation provide a “proof of concept” for the underlying theory of A-Design that is implicitly stated throughout this dissertation. Explicitly, the underlying theoretical claim of A-Design is that by understanding the broad characteristics of the human design process, a computational process can be created to model these behaviors in order to invent, create, or solve conceptual design problems. The successful manner in which these human characteristics are incorporated within A-Design offers proof of the validity of A-Design’s theoretical claim. These proofs are tied to the development of the A-Design subsystems, which are each discussed a final time below.

10.2.1.1 Iteratively-Guided Search Process

Quite possibly the most natural computational property, the concept of iteration affords us several advantageous qualities for conceptual design. In any human design problem, creation occurs as the result of several steps. These steps can be a development from the abstract to the concrete, as in the progression from identifying a need to

developing a prototype, or the critical steps that improve existing designs after the original conceptualization. Designs that have evolved over a series of improvements from previous instances are often the most robust. As stated in Jones (1980) in the analysis of traditional design methods, “This slow and costly sequential searching for the ‘invisible lines’ of a good design can, in the end, produce an astonishingly well-balanced result and a close fit to the needs of the user.” While product evolution can be measured in centuries for some human design problems, we can bank on the intrinsic speed of computational systems to iterate and evolve design concepts in a reasonable time.

Given that objectives are definable for a design problem, iteration allows initially weak alternatives to progress to successful solutions. As is evident in the A-Design system, agents initially create unsuccessful solutions but as the design selection scheme isolates better alternatives and feedback is provided to the agents, the process improves designs to better meet the design specification. The iteration also allows parts of a design problem description to be expressed as objectives. For example in the weighing machine problem, accuracy of dial is an objective that partially defines the functionality of a device. Initially, agents may create solutions that do not even cause the output dial to rotate. As the process unfolds, solutions appear that better meet the accuracy objective, and feasible weighing machines are produced.

While the iterative process alone can lead to a natural evolution of better designs, human design also learns from the successes or deficiencies of past designs to guide the modifications of future artifacts. One way that this guided iteration is implemented in A-Design is with TODO and TABOO learning. It has been shown in the experiments of

Chapter 9 that this learning does not constrict search in future iterations but instead allows for more efficient search and more design innovation.

10.2.1.2 Adaptive Design Selection

The incorporation of adaptability into A-Design embodies two traits of human design. First, design takes place within a dynamic environment. While it is true that designs evolve to create improved design solutions, designs can also be modified to meet different product demands. Variations in consumer markets or the introduction of new technologies can change how a design is viewed (the utility of a design). Human design is capable of adapting, however current computational design strategies require reinitiating the process to solve a newly formulated design problem. Within A-Design, this reinitiation is avoided by a unique pruning of designs at each iteration of the process. This pruning stores key designs so that the process is capable of quickly adapting to changes in the relative importance of objectives as well as the addition or deletion of objectives.

Second, this adaptability within A-Design allows a user of the process to understand the trade-offs among the objectives. By tracking the automated design procedure, the user can steer the process to meet the designer's needs. In addition, the dialog of the Manager-agent provides an interaction between the user and the search process so that the resulting design process is part human and part computer.

As a by-product of storing the Pareto-set for adaptability, it also has been shown that the preserving of diverse design solutions leads to improved design activity for even static user preferences (shown by testing the effects of the Pareto set in Section 9.2).

Perhaps, human design also benefits from this storing of diverse design ideas. In her book on creativity, Margaret Boden (1990) cites the following Poincaré adage:

Among chosen combinations the most fertile will often be those formed of elements drawn from domains which are far apart...Most combinations so formed would be entirely sterile; but certain among them, very rare, are the most fruitful of all.

10.2.1.3 Open-Ended Representation

Representing information in the human design process or, in fact, any human thought process is not a simple matter. The storing and manipulating of information to solve problems is the basis for the cognitive science study of procedural knowledge (see overview Holland et al., 1986). It is generally accepted that design is an open-ended problem-solving phenomenon that lacks a single correct answer. The analogy of conceptual design as search (see Section 1.2) provides a starting point for understanding how design can occur within a computational system. Other than this analogy, there is no general scheme for formalizing design problems. Therefore, the power of a conceptual design engine will depend upon the generality of the representation. With this claim, we are challenged to develop a representation to handle a wide variety of possible design solutions. The language that was developed in this dissertation for electromechanical devices provides the grammar for design construction without imposing large restrictions on the search space. This is demonstrated by observing the wide variety of designs in Chapter 8. While this representation is not capable of representing all possible design solutions for a given problem, it does provide a starting point for generating a significant degree of unstructured invention.

10.2.1.4 Collaborative Agents

A theoretical claim of the agent subsystem is that many approaches are better than one. In human design teams, the interaction of various designers produces a more than additive effect. The total combined expertise and preferences of the designers are further compounded by the expertise and preferences resulting from interactions of the designers. This is confirmed in studies of Osborn (1963) where he states, “Most of us can work better creatively when teamed up with the right partner because collaboration tends to induce effort, and also to spur our automatic power of association.” The artifacts that result from the cooperation of numerous engineering designers are often more robust and effective than designs conceived by a single designer.

In computational systems, it has been claimed on a number of occasions that collaborative or cooperative agent strategies lead to scale-effective computation (Talukdar, 1996; Lander, 1997). This means that the addition of new strategies only seeks to increase the capabilities of the system. With this in mind, agents are constructed in A-Design with various unique strategies and preferences. While these strategies are somewhat constrained and deterministic, their stochastic interaction can produce a wide range of results. Again, this is shown by the resulting artifacts of Chapter 8.

10.2.2 Investigating Conceptual Design

The second challenge of computational design involves discovering the foundations of the design process. While it could be said that engineering is part design and part analysis, most of the past engineering research has focused on analysis. This is due to the nature of engineering analysis, which is founded on mathematical formalisms. In the last fifty years or so, cognitive psychology has provided a great deal of insight into

how the human mind operates. This research has blossomed as a result of computation, which has allowed models of cognition to be formalized and studied by artificial models of intelligence. The combination of cognitive psychology and engineering design has yet to be fully realized. While this project has assimilated four human design traits in a computational system, there is no restriction on integrating and investigating other human design traits.

In addition to the computational research in cognitive psychology, there are also countless experiments seeking to understand the behaviors involved in solving design-like problems. Through these psychological experiments, we gain insight into human qualities such as functional fixedness (Gick and Holyoak, 1980), representation of new knowledge (Clark and Karmiloff-Smith, 1993), and isomorphic representations (Kotovsky and Simon, 1990). There are also numerous historical accounts of design (see Weber, 1992; and Petroski, 1990) that touch on the cognitive details behind engineering invention. On the basis of the work described here, it is believed that the rigorous psychological studies and the historical texts could be combined to yield new insight into how design is accomplished both naturally and artificially.

10.2.3 Uniting Automated Design and Automated Analysis

One of the key issues of design, either human or automated, is the need for evaluating design alternatives. In our construct of design as search, we have found that evaluation is necessary in arriving at the objective functions used to compare alternatives.

Unfortunately, conceptual design evaluation is not straightforward and can include qualitative effects and complex performance objectives. Simon (1969) confirms this challenge by laying out two issues in the evaluation of alternatives:

1. *Utility theory and statistical decision theory as a logical framework for rational choice among given alternatives.*

2. *The body of techniques for actually deducing which of the available alternatives is the optimum.*

He goes on to say, “Only in trivial cases is the computation of the optimum alternatives an easy matter. If utility theory is to have application to real-life problems, it must be accompanied by tools for actually making the computations.”

The challenge of incorporating automated analysis with automated design is a very real theoretical challenge, and this challenge will only become more apparent as more complex engineering design problems are addressed by design automation. As is shown in the MEMS accelerometer example, the external computation involving the invocation of SABER presents challenges both in automatically solving a wide variety of designs and in balancing the trade-off between accuracy and time. Coincidentally, this challenge was also addressed in this researcher’s previous research with VLSI layout under heat transfer constraints (see Campbell et al. 1997). From these experiences, there seems to be three possible solutions in combining automated analysis as a subset of automated design.

First, develop an automated analysis that increases in accuracy throughout the search process of design. Early in the search, states are coarsely evaluated to allow the coverage of a wide variety of design solutions similar to brainstorming in human design (Osborn, 1963). Then, as the process continues and search becomes more focused, comparison of solutions requires a finer and finer evaluation and thus analyses are increased in accuracy to better distinguish design worth. This has been the approach used by Prakash and Cagan (1999) and in Campbell et al. (1997). However, if the coarseness is not well controlled early in the process, search may converge on erroneous areas of the space in

later iterations. Also, changes in the accuracy of the evaluation may lead to unexpected shifts in search. These shifts may result from different amounts of statistical error present at different levels of analysis. For example, if one level of analysis consistently underestimates attribute values by 20%, and the process shifts to a more accurate analysis with only a 5% error, then the coarsely evaluated solution will be erroneously favored. Therefore, it is not judicious to make comparisons of alternatives evaluated at different levels of analysis.

Second, develop a means of evaluating similar to how human engineers evaluate. That is, develop computational analysis tools capable of the “intuition” that occurs when human designers rule out or approve design decisions. This development of human heuristics for evaluation can be a shortcut to more intricate computational analysis. Perhaps machine learning techniques can be developed to identify trends in designs and correlate them with infrequent computational analysis. This machine “intuition” could greatly benefit large conceptual design search processes requiring numerous or complex evaluations.

Third, create a search process that is efficient in the choice of alternatives. Many stochastic approaches require numerous design states to be visited. However, if a less random search process is developed that follows well-formulated heuristics as is done here with the agents and similarly in Yin and Cagan (2000) and Storn (1995), then computational analysis does not need to sacrifice accuracy for time since fewer design states require analysis.

This discussion has presented the challenges of computational design. The innovations of A-Design have opened the door to understanding the limitations and realizing the possibilities of automating the conceptual design process.

Chapter 11

Coda

11.1 CONTRIBUTIONS

The main contribution from the theory and implementation of A-Design is the production of an artificial process capable of invention. Several specific contributions can be identified in making a significant step towards automating conceptual design:

- The construction of an adaptive process that can change to meet user needs or market demands during the search process thereby capturing the interactive nature of conceptual design,
- The creation of a search process that can accommodate the most appropriate representations for a given problem via agent interfacing,
- The combination of stochastic and knowledge-driven methods to create a unique stochastic search process capable of learning.
- The development of a functional representation for electromechanical systems that includes a general connectivity of components, the ability to model incomplete design states and the inclusion of real components,

- The first automated topology generation of MEMS devices given only the functional specifications,
- The classification of human design characteristics to provide a basis for computational design and to establish a computational testbed for investigating the behaviors of both the human and computational design,
- The establishment of a design tool capable of searching infinite design spaces for component configurations to meet a user's specifications.

11.2 FUTURE WORK

The development of the A-Design system allows for future work to proceed in a variety of different directions. The four subsystems each can be further investigated both in the context of A-Design and as separate research endeavors.

The functional representation developed in Chapter 4 could be further developed and implemented to solve more complex designs than those developed here. These would include geometry transformations, operating range issues, and better dynamic modeling mentioned in the discussion of Section 8.1.2. The contents of the Functional Parameter and Embodiment structures have been created to represent electromechanical components with transformations of *power*, *signal*, and *material* (the three fundamental engineering classes constructed by Pahl and Beitz). In the current implementation, only power transformations are explored. More interesting design problems could be addressed if the representation is extended to include signal and material. For example, the input and outputs of a coffeemaker could be posed as produce coffee (*material*), given electric power (*power*), water (*material*), and coffee beans (*material*). These representation issues

could be investigated alone as a basis for, or in combination with, other engineering representations such as boundary or feature-based representations.

The development of the adaptive subsystem demonstrated how the interaction between user and Manager-agent could lead to new ways of modeling user utility functions. Here, the dialog and approximation of utility is confined to cardinal rankings and simple matrix manipulation. More detailed dialog and models for utility have been developed for multi-attribute problems (see example in Thurston, 1991); future research could combine these approaches with search techniques to yield a more interactive multi-objective design process.

The Manager-agent strategy explained in Chapter 6 presents one possible way that guided search can lead to improved design activity. Instead of a single Manager-agent strategy, a multitude of interacting Manager-agents could be explored to represent different strategies for conducting design. The different learning approaches tested in Chapter 9 could provide a basis for the interacting Manager-agent strategies.

Furthermore, the TODO and TABOO learning is a fertile area of future research. While agent teams and design fragments trends are currently determined from past experiences, other design characteristics such as phenomenological concepts (as discussed at the end of Chapter 4), or geometric similarities could greatly increase the learning and inventive power of A-Design.

The future work possibilities in developing new evaluation methods are briefly discussed in Section 10.2.3. The need for automated analysis can pose a significant bottleneck for conceptual design. Future work in interfacing automated design with

automated analysis might require the development of sophisticated interpreters to handle the pre- and post-processing or heuristic approaches to make simplifications or judgments based on individual design state properties.

Finally, the A-Design methodology is shown to have a general applicability to a wide variety of conceptual design domains. The implementation is developed to a significant level in this dissertation to show a proof of concept for the automated design theory. It would be exciting to further develop A-Design to address new design problems that may yield novel, or possibly even patentable, inventions.

11.3 CONCLUDING REMARKS

The A-Design theory presented here makes strides towards understanding conceptual design, automating conceptual design, and assisting a user with conceptual design. This dissertation provides a theoretical foundation for future design automation tools that could be developed to aid industries in innovating new concepts by bringing together human design and computational processes.

References

- Abramson, B., 1990, "Expected-Outcome: A General Model of Static Evaluation", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 12, No. 2, pp. 182-193.
- Analog Devices, Inc., 1998, "ADXL150/ADXL250: ± 5 g to ± 50 g, Low Noise, Low Power, Single/Dual Axis iMEMS[®] Accelerometers", *Datasheet*, http://www.analog.com/pdf/ADXL150_250_0.pdf, Norwood, MA.
- Analogy, Inc., 1995, "SaberDesigner[®]: Simple Solutions. Powerful Results", *Datasheet*, <http://www.analogy.com/Products/DataSheets/prodbro.pdf>, Beaverton, OR.
- Bäck, T, and Hoffmeister, F., 1991, "Extended Selection Mechanisms in Genetic Algorithms", *Proceedings of the Fourth International Conference on Genetic Algorithms*, eds. Belew, R. and Booker, L., Morgan Kaufman Publishers, San Mateo, CA, pp. 92-99.
- Balachandran, M., J. S. Gero, 1984, "A Comparison of Three Methods for Generating the Pareto Optimal Set", *Engineering Optimization*, Vol. 7, pp. 319-336.
- Boden, M.A., 1990, *The creative mind: Myths and mechanisms*, Basic Books, New York, NY.
- Bracewell, R. H., and Sharpe, J. E. E., 1996, "Functional Description Used in Computer Support for Qualitative Scheme Generation- 'Schemebuilder'", *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, Vol. 10, pp. 333-345.
- Brown, D.C., and Chandrasekaran, B., 1986, "Knowledge and Control for a Mechanical Design Expert System", *IEEE Computer*, Vol.19, No.7, pp. 92-100.
- Brown, D. R., and Hwang, K. Y., 1993, "Solving Fixed Configuration Problems with Genetic Search", *Research in Engineering Design*, Vol. 5, pp. 80-87.

- Cadence Design Systems, Inc., 2000, "Spectre® Circuit Simulator Advanced Simulator for Complex Circuits", *Datasheet*, http://www.cadence.com/datasheets/spectre_cir_sim.html, San Jose, CA.
- Campbell, M. I., C. H. Amon, J. Cagan, 1997, "Optimal Three-Dimensional Placement of Heat Generating Electronic Component", *Journal of Electronic Packaging*. Vol. 119, No. 2, pp. 106-13.
- Campbell, M., "Knowledge discovery in Deep Blue", *Communications of the ACM*, vol.42, no.11, p. 65-7.
- Chakrabarti, A., Bligh, T. P., 1996, "An Approach to Functional Synthesis of Mechanical Design concepts: Theory, Applications, and Merging Research Issues," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, Vol. 10, pp. 313-331.
- Christensen, J.; Korf, R.E., 1986, "A Unified Theory of Heuristic Evaluation Functions and Its Application to Learning," *Proceedings AAAI-86: Fifth National Conference on Artificial Intelligence*, pp. 148-152.
- Clark, A., Karmiloff-Smith, A., 1993, "The Cognizer's Innards: A Psychological and Philosophical Perspective on the Development of Thought", *Mind and Language*, Vol. 8, No. 4, pp. 487-519.
- D'Ambrosio, J. G., and W. P. Birmingham, 1995, "Preference-directed design," *Artificial Intelligence in Engineering Design, Analysis and Manufacturing*, Vol. 9, pp. 219-230.
- Dong, A., A. M. Agogino, 1995, "A Spectral Optimization Algorithm for Multi-Objective Prototype Selection", *Proceedings of the ASME Design Engineering Technical Conferences*, DE-Vol. 83, Vol. 2, pp. 447-453.
- Eschenauer, H., J. Koski, A. Osyczka (eds.), 1990, *Multicriteria Design Optimization*, Springer-Verlag, Berlin, Germany.
- Finger, S. and Rinderle, J. R., 1989, "A Transformational Approach to Mechanical Design Using a Bond Graph Grammar," *Proceedings of the 1st ASME Design Theory and Methodology Conference*, Montreal, September, 1989.
- Fonseca, C. M., P. J. Fleming, 1995, "An Overview of Evolutionary Algorithms in Multiobjective Optimization", *Evolutionary Computation*, Vol. 3, pp. 1-16.
- Forbus, K. D., 1988, "Qualitative Physics: Past, Present, and Future", *Exploring Artificial Intelligence*, ed. Howard Shrobe, Morgan Kaufmann Publishers, pp. 239-296.
- Franklin, S. and A. Graesser, 1997, "Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents", *Proceedings of the Third International Workshop on Agent*

Theories, Architectures, and Languages, published as Intelligent Agents III, Springer-Verlag, pp. 21-35.

Fu, Z., De Pennington, A., and Saia, A., 1993, "A Graph Grammar Approach to Feature Representation and Transformation", *International Journal of Computer Integrated Manufacturing*, Vol. 6, No. 102, pp. 137-151.

Gage, P. J., and I. M. Kroo, 1995, "Representation Issues for Design Topological Optimization by Genetic Methods", *Industrial and Engineering Applications of Artificial Intelligence and Expert Systems: Proceedings of the Eighth International Conference*, eds. G. F. Forsyth, and M. Ali, Melbourne, Australia; June 6-8, pp. 383-388.

Gick, M. L. and Holyoak, K. J., 1980, "Analogical Problem Solving", *Cognitive Psychology*, Vol. 12, pp. 306-355.

Glover, F., 1989, "Tabu Search-Part 1," *ORSA Journal on Computing*, Vol. 1, No. 3, pp. 190-206.

Goldberg, D. E., 1989, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison Wesley, Reading, MA.

Goldberg, D.E., Deb, K., Kargupta, H., and Harik, G., 1993, "Rapid, Accurate Optimization of Difficult Problems Using Fast Messy Genetic Algorithms", *Proceedings of the International Conference on Genetic Algorithms*, Vol. 5, pp. 56-64.

Goldstein, D., 1994, "An Agent-based Architecture for Concurrent Engineering", *Concurrent Engineering: Research and Applications*, Vol.2, pp. 117-123.

Greco, D. L., and D. C. Brown, 1996, "Design Agents that Learn", *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, Vol.10, pp. 149-150.

Hillis, W. D., 1991, "Co-Evolving Parasites Improve Simulated Evolution as an Optimization Procedure", *Artificial Life II: SFI Studies in the Sciences of Complexity*. Vol. 10, eds. C. G. Langton, C. Taylor, J. D. Farmer, & S. Rasmussen, Addison-Wesley.

Hoffman, R., and Waring, N., 1996, "The Localization of Interaction and Learning in the Repeated Prisoner's Dilemma", *Tech. Report SFI-96-08-064*, Santa Fe Institute, Santa Fe, NM.

Holland, J. H., 1992, *Adaptation in Natural and Artificial Systems*, The MIT Press, Cambridge, MA, 2nd edition.

- Holland, J. H., Holyoak, K. J., Nisbett, R. E., and Thagard, P. R., 1986, *Induction: Processes of Inferences, Learning, and Discovery*, The MIT Press, Cambridge, MA.
- Jones, J. C., 1980, *Design Methods: Seeds of Human Futures*, John Wiley & Sons, New York, 1980 edition.
- Kaelbling, L. P., Littman, M. L., and Moore, A.W., 1996, "Reinforcement Learning: A Survey", *Journal of Artificial Intelligence Research*, Vol. 4, pp. 237-285.
- Keeney, R. L., and Raiffa, H., 1976. *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*, John Wiley & Sons, New York.
- Kirkpatrick, S., C. D. Gelatt Jr., and M. P. Vecchi, 1983, "Optimization by Simulated Annealing," *Science*, Vol. 220, pp. 671-679.
- Kotovsky, K., and Simon, H. A., 1990, "What Makes Some Problems Really Hard: Explorations in the Problem Space of Difficulty", *Cognitive Psychology*, Vol. 22, pp. 143-183.
- Koza, J. R., F. H. Bennett III, D. Andre, 1996, "Automated Design of Both the Topology and Sizing of Analog Electrical Circuits using Genetic Programming," *Artificial Intelligence in Design*, eds. J. S. Gero and F. Sudweeks, pp. 151-170.
- Laird, J. E., Newell, A., and Rosenbloom, P. S., 1986, "Soar: An Architecture for General Intelligence," *Technical Report CMU-CS-86-171*, Carnegie Mellon Univ., Pgh., PA.
- Lander, S. E., 1997, "Issues in Multiagent Design Systems", *IEEE Expert*, Vol.12, pp. 18-26.
- Langton, C. G. (ed.), 1988, *Artificial Life: SFI Studies in the Sciences of Complexity*, Addison Wesley, Reading, MA..
- Lawrence, C. T., J. L. Zhou, and A. L. Tits, 1993, CFSQP: C-implemented Functional Sequential Quadratic Programming, *Computer Program*, University of Maryland, Institute for Systems Research, College Park, MD.
- Lenat, D. B., 1983, "EURISKO: A Program That Learns New Heuristics and Domain Concepts", *Artificial Intelligence*, Vol. 21, pp. 61-98.
- McDermott, J., 1982, "R1: A Rule-Based Configurer of Computer Systems", *Artificial Intelligence*, Vol.19, No.1, pp. 39-88
- McDermott, J., 1993: "R1 ('XCON') at Age 12: Lessons from an Elementary School Achiever", *Artificial Intelligence*, Vol.59, No.1-2, pp. 241-247.

- Mitchell, T. M., 1997, *Machine Learning*, McGraw-Hill, New York.
- Mitchell, M., 1996, *An Introduction to Genetic Algorithms*, MIT Press, Cambridge, MA.
- Mittal, S., Dym, C., and Morjara, M., 1985, "PRIDE: An Expert system for the Design of Paper Handling Systems", *IEEE Computer*, Vol.19, No.7, pp. 102-114
- Mukherjee T., Zhou, Y., Fedder, G. K., 1999, Automated Optimal Synthesis of Microaccelerometers", *Proceeding of the 12th Annual IEEE International Micro Electro Mechanical System Conference*, Orlando, Florida, January 17-21, pp. 326-331
- Murthy, S. S., 1992, *Synergy in Cooperating Agents: Design Manipulators from Task Specifications*, Ph.D. Dissertation, Carnegie Mellon University.
- Navinchandra, D., K. P. Sycara, and S. Narasimhan, 1991, "A Transformational Approach to Case-Based Synthesis", *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, Vol. 5, pp. 31-45.
- Nayak P., 1992, *Automated Modeling of Physical System*, Tech. Report STAN-CS-92-1443, PhD. Thesis, Stanford University.
- Newell, A., Simon, H. A., 1972, *Human Problem Solving*, Prentice-Hall, Englewood Cliffs, NJ.
- Osborn, A. F., 1963, *Applied Imagination*, Charles Scribner's Sons, New York.
- Pahl, G. and W. Beitz, 1988, *Engineering Design - A Systematic Approach*, Springer-Verlag, New York.
- Palmer, R. S., and Shapiro, V., 1993, "Chain Models of Physical Behavior for Engineering Analysis and Design," *Research in Engineering Design*, Vol. 5, pp. 161-184.
- Paynter, H. M., 1961, *Analysis and Design of Engineering Systems*, MIT Press, Cambridge, MA.
- Petrie, C. P., T. A. Webster, and M. P. Cutkosky, 1995, "Using Pareto Optimality to Coordinate Distributed Agents", *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, Vol. 9, pp. 269-281.
- Petroski, H., 1990, *The Pencil*, Alfred A. Knopf Publishing, New York.
- Pinilla, J. M., Finger, S., and Prinz, F. B., 1989, "Shape Feature Description Using an Augmented Topology Graph Grammar", *Proceedings of the NSF Engineering Design Research Conference*, Amherst, MA, June 11-14, pp. 285-300.

- Prakash, S. and Cagan, J., 1999, *Hierarchical Method for Approximating MEMS Analysis*, Master's Thesis, Carnegie Mellon University.
- Pylyshyn, Z. W., 1992, "Computers and the Symbolization of Knowledge", *Minds, Brains, and Computers*, eds. Morelli, R.; Brown, W. M.; Anselmi, D.; Haberlandt, K.; Lloyd, D., Ablex Publishing Corporation, Norwood, NJ, pp. 82-94
- Quadrel, R., R. Woodbury, S. Fenves, and S. Talukdar, 1993, "Controlling Asynchronous Team Design Environments with Simulated Annealing," *Research in Engineering Design*. Vol. 5, No. 2, pp. 88-104.
- Queipo, N., R. Devarakonda, and J. A. C. Humphrey, 1994, "Genetic Algorithms for Thermosciences Research: Application to the Optimized Cooling of Electronic Components," *International Journal Heat and Mass Transfer*, Vol. 37, pp. 893-908.
- Reynolds, C. W., 1987, "Flocks, Herds, and Schools: A Distributed Behavioral Model", *Computer Graphics*, Vol. 21, No. 4, pp. 25-34.
- Russell, S. and P. Norvig, 1995, *Artificial Intelligence: A Modern Approach*, Prentice-Hall, Inc., Englewood Cliffs, NJ.
- Sandholm, T.W.; Crites, R.H., 1995, "On Multiagent Q-learning in a Semi-Competitive Domain", *Adaptation and Learning in Multi-Agent Systems. IJCAI 95 Workshop. Proceedings*, eds. G. Weiss, S. Sen., Vol. 6, 191-205
- Schmidt, L.C., 1995, *An Implementation Using Grammars of an Abstraction-based Model of Mechanical Design for Design Optimization and Design Space Characterization*, PhD. Thesis, Carnegie Mellon University.
- Schmidt, L.C., and J. Cagan, 1995, "Recursive Annealing: A Computational Model for Machine Design", *Research in Engineering Design*, Vol. 7, pp. 102-125.
- Schmidt, L.C., and Cagan, J., 1998, "Optimal Configuration Design: An Integrated Approach Using Grammars", *Journal of Mechanical Design*, Vol. 120, pp. 2-9.
- Scott, M. J. and Antonsson, E. K., 1996, "Formalisms for Negotiation in Engineering Design," *Proceedings of the 1996 ASME Design Engineering Technical Conferences and Computers in Engineering Conference*, DETC96/DTM-1525, Irvine, CA, August 18-22.
- Shea, K., J. Cagan, and S.J. Fenves, "A Shape Annealing Approach to Optimal Truss Design with Dynamic Grouping of Members", *Journal of Mechanical Design*, Vol. 119, No. 3, pp. 388-394, 1997.
- Simon, H. A., 1969, *The Sciences of the Artificial*, The MIT Press, Cambridge, MA.

- Simon, H.A., 1986, "Rationality in Psychology and Economics", *The Journal of Business*, Vol. 59, No. 4, pp. 209-224.
- Stahovich, T. F., R. Davis, and H. Shrobe, 1998, "Generating Multiple New Designs from a Sketch", *Artificial Intelligence*, Vol. 104, pp. 211-264.
- Steels, L., 1996, "Discovering the Competitors", *Journal of Adaptive Behavior*, Vol. 4, No. 2, pp. 173-199.
- Stevens, S. S., 1975, *Psychophysics: Introduction to its Perceptual, Neural, and Social Prospects*, John Wiley & Sons, New York.
- Stone, R. and Wood, K., 1999, "Development of a Function Basis for Design", *Proceedings ASME Design Engineering Technical Conferences*, DETC99/DTM-8765, September 12-15, Las Vegas, NV.
- Storn, R., 1995, "Differential Evolution Design of an IIR-Filter with Requirements for Magnitude and Group Delay" *IEEE International Conference on Evolutionary Computation ICEC 96*, pp. 268-273, Technical Report TR-95-026, ICSI University of California, Berkeley, CA.
- Subramanian, D.; Cheuk-San Wang, 1995, "Kinematic Synthesis with Configuration Spaces," *Research in Engineering Design*, Vol.7, no.3, p. 193-213.
- Sycara, K., 1998, "Multiagent Systems", *AI Magazine*, Vol. 19, No. 2, pp. 79-92.
- Talukdar, S., 1993, "Asynchronous Teams", *Fourth International Symposium on Expert Systems Applications to Power Systems*, La Trobe University, Australia, Jan 4-8.
- Talukdar, S., L. Baerentzen, A. Gove, and P. de Souza, 1996, "Asynchronous Teams: Organizations for Algorithmic Computation", *EDRC Tech-Report 18-56-96*, EDRC Carnegie Mellon University, Pittsburgh, PA.
- Talukdar, S., 1998, "Autonomous Cyber Agents: Rules for Collaboration and Concurrency", *Proceedings of the Thirty-First Hawaii International Conference on System Sciences*, Vol. 7 pp. 57-61.
- Talukdar, S., 1999, "Collaboration rules for autonomous software agents", *Decision Support Systems*, Vol.24, No.3-4, pp. 269-278
- Tan, M., 1993, "Multi-agent Reinforcement Learning: Independent vs. Cooperative Agents", *Tenth International Conference on Machine Learning*, Amherst, MA, pp. 330-337.
- Thurston, D. L., 1991, "A Formal Method for Subjective Design Evaluation with Multiple Attributes" *Research in Engineering Design*, Vol. 3, pp. 105-122.

- Thurston, D. L., 1999, "Real and Perceived Limitations to Decision Based Design", *Proceedings ASME Design Engineering Technical Conferences*, DETC99/DTM-8750, September 12-15, Las Vegas, NV.
- Ulrich, K., T., 1989, *Computation and Pre-Parametric Design*, PhD. Thesis Massachusetts Institute of Technology, Tech. Report AI-TR-1043.
- Ulrich, K., and Seering, W., 1989, "Synthesis of Schematic Descriptions in Mechanical Design," *Research in Engineering Design*, Vol. 1, pp. 3-18.
- Weber, R.J., 1992, *Forks, Phonographs, and Hot Air Balloons: A field guide to inventive Thinking*, Oxford University Press, New York.
- Welch, R. V., 1992, *Conceptual Design of Mechanical Systems: A Representation and Computational Model*, PhD. Thesis, University of Massachusetts.
- Welch, R. V., and Dixon, J., 1994, "Guiding Conceptual Design Through Behavioral Reasoning," *Research in Engineering Design*, Vol. 6 pp. 169-188.
- Williams, B.C., 1990, "Interaction-based invention: designing novel devices from first principles", *AAAI-90 Proceedings. Eighth National Conference on Artificial Intelligence*, Vol.1, Boston, MA, pp. 349-356.
- Winston, P. H., 1982, "Learning New Principles from Precedents and Exercises," *Artificial Intelligence*, vol.19, no.3, p. 321-350.
- Winston, P. H., 1992, *Artificial Intelligence*, Addison-Wesley, 3rd edition, Reading, MA.
- Yin, S., and Cagan, J., 2000, "An Extended Pattern Search Algorithm for Three-dimensional Component Layout", *Journal of Mechanical Design*, Vol. 122, No. 1, pp. 102-108.
- Zhou, Y., 1998, *Layout Synthesis of Accelerometers*, M.S. Thesis, Carnegie Mellon University.

Appendix:

Implementation

A. GENERAL A-DESIGN PROCEDURE

In the flowchart of Figure 2.2, the four subsystems of A-Design are combined in a process of interacting functions and agents. This section presents an overview to the general algorithmic procedure of A-Design through detailed pseudo-code. As can be seen by the top of Figure A.1, the main function, *A-Design*, is invoked with no arguments. The initiation of variables and the design problem description is implemented in the **init.lisp** file shown in Section C of this Appendix. Information on each step of the iterative process is outputted through the *Write-design-data* function which details the agent populations, design populations, and the best and average objective values found at each iteration. Data on the best final designs is written with the *Write-Results* function at the end of the process.

In the following pages of pseudo-code, the nested functions show the subroutines within the main functions. The italicized phrases correspond to function names, and the phrases in parentheses denote the arguments to a particular function. Comments are indicated by double slashes.

A-Design ()

- Initiate variables and structures by loading **init.lisp** (see Appendix Section C)
- Read in program files, agents, and catalog of components.
- Begin iterative process
 - iteration := iteration + 1
 - configs := *Create-and-Repair-Configurations* (fragments, agent-stats, TODO, TABOO)
 - designs := *Instantiate-Configurations* (configs, agent-stats, TODO, TABOO)
 - *Evaluate-designs* (designs)
// The routines to evaluate designs are unique to each design problem. For the accelerometer problem and the weighing machine problem, details can be found in Section C in the input file, **evaluate.lisp** //.
 - [Pareto-designs, Non-Pareto-designs] = *Find-Pareto-designs* (designs)
 - [Good-designs, Poor-designs, TODO, TABOO, agent-stats]
:= *Get-Manager-agent-response* (Pareto-designs, Non-Pareto-designs)
// See pseudo-code for Manager-agent in Figure 6.1. //
 - *Write-design-data* (Pareto-designs, Good-designs, Poor-designs, iteration, agent-stats)
 - *if* (iteration = *tot-iter*)
 then break
 else
 - fragments := *Modify-Designs* ((Pareto-designs, Good-designs), agent-stats)
 - *repeat*
- *Write-Results* (Pareto-designs, Good-designs)

// The pseudo-code for the major functions of this main procedure are shown below. //

Create-and-Repair-Configurations (fragments, agent-stats, TODO, TABOO)

- *for* (each fragment)
 - *Until design-is-complete* (fragment)
 - Fragment := *Create-from* (fragment, agent-stats, TODO, TABOO)
- *for* (number-of-new-configs = *design-pop* - *number-of* (fragments))
 - *Until design-is-complete* (config)
 - config := *Create-from* (config, agent-stats, TODO, TABOO)

design-is-complete (config)

- *AND* (*design-goals-are-met* (config))
// checks to see that no more goal flags in input and output FPs. //
- (*design-is connected* (config))
// checks to see that a connection of EBs exist from input to output FPs. This is accomplished by a depth-first recursive search of the graph. //

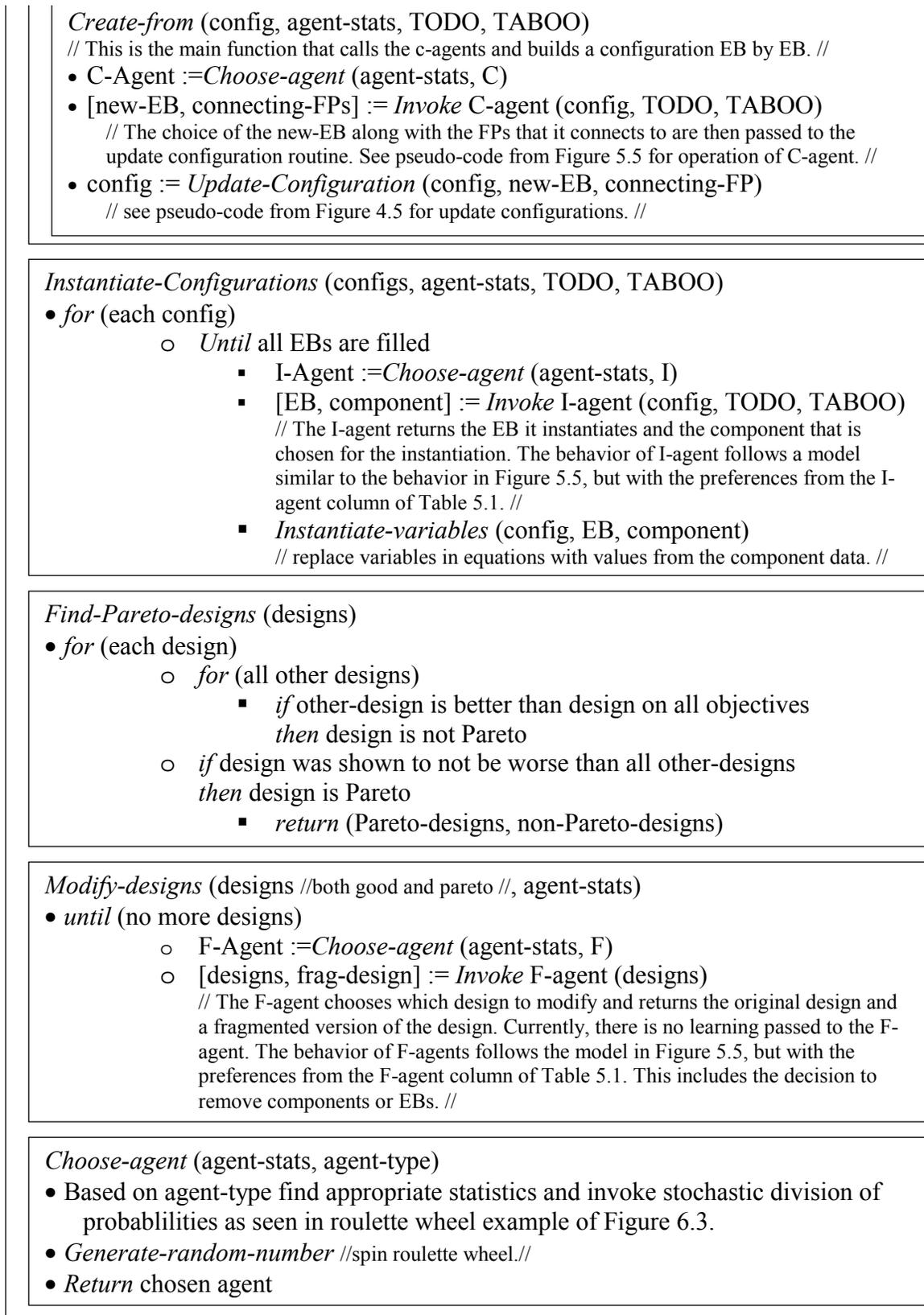


Figure A.1: Pseudo-code for general A-Design procedure. Subroutines shown in nested frames.

B. DIRECTORY STRUCTURE

The code for A-Design is implemented in LISP and C and is divided into several directories as seen in Figure A.2. The **codeGeneral** directory includes files that maintain the basic workings of A-Design. The C-, I- and F-agents that are created for specific applications are stored in separated directories: **codeEM** contains agents for the weighing machine problem, and **codeMEMS** contains agents for the MEMS accelerometer problem.

The catalog of components for A-Design are placed in a separate directory. For the weighing machine problem, the directory, **libraryEM**, contains electro-mechanical Embodiments and components used for building weighing machines. For the MEMS problem components are stored in **libraryMEMS**.

The files **init.lisp** and **evaluate.lisp** are the input files for A-Design, and are described in the next section. Output files are stored in **outputEM** and **outputMEMS** for the weighing machine and accelerometer problems respectively. The details of these output files are shown in Section E of this appendix.

A-Design

init.lisp

evaluate.lisp

codeGeneral

design.lisp io.lisp

create.lisp magents.lisp

functions.lisp trend.lisp

update.lisp

codeEM

cagents.lisp

iagents.lisp

fagents.lisp

codeMEMS

cagents.lisp

iagents.lisp

fagents.lisp

libraryEM

Embodiments elec_valve.comps

battery.comps Inductor.comps

cable.comps lever1class.comps

capacitor.comps lever2class.comps

gear.comps (etc., see Table 4.2)

libraryMEMS

Embodiments mass.comps

horiz_beam.comps horiz_comb.comps

vert_beam.comps vert_comb.comps

outputEM

iter.out topdesigns.out

pareto.out cagents.out

good.out fagents.out

poor.out iagent.out

outputMEMS

iter.out top_data.out

(etc., same files as **outputEM**)

Figure A.2: The directory structure of files used in the Weighing Machine and MEMS accelerometer problems.

C. INPUT FILES

For the results in Chapter 8, the design problem is presented to A-Design in terms of the inputs and outputs of the design, the objectives of the design problem, and the catalog of components. In the LISP implementation of A-Design, each of these is stored in a separate file or files. The input and output descriptions are included in **init.lisp**. This file also includes all the data structures used in A-Design, and constants that can be adjusted by the user. The objectives of the design problem are created in a separate file, **evaluate.lisp**, which lists all the code for analyzing designs. These files are shown below.

The catalog of components for A-Design are placed in a separate directory. For the weighing machine problem, the directory is called **libraryEM** because it contains electro-mechanical Embodiments and components. Within this directory, one file, **Embodiments** contains all the EBs for the catalog. Separate files with for each EB contain the various components used to instantiate the EB. These are stored in filenames with a **.comps** extension to the EB they instantiate.

init.lisp

```
;;; INIT.LISP - Contains data structures and constants used throughout the
;;; process. This can be called with the (a-design) executable or loaded
;;; beforehand to run a multitude of tests.

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; The following are constants used by the process.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(sys:resize-areas :new 20000000 :old 40000000)
;; this sets up the RAM needed to run this space-extensive program

(setf *random-seed* (make-random-state t))
(setf *random-state* *random-seed*)
;; seed the random generator so that no two runs are the same
```

```

;; save the initial seed in a separate variable such that it can be stored
;; for a particular run.

(setf *num-of-objectives* 4)
;; the number of objectives in the design problem.

  (setf *obj-constraints* '(1000.0 1000.0 100000.0 100000.0))
;; the ceiling of the design spaces, any designs above this are
;; automatically eliminated from the process

(setf *attempts-to-reconstruct* 15)
;; after designs are fragmented sometimes they are impossible to repair. This
;; constant sets how many attempts at reconstructing a design are performed
;; before giving up on it.

(setf *design-pop* 100)
;; the maximum number of designs at any given time.

(setf *designs-per-config* 4)
;; the number of designs to instantiate per config in beginning.

(setf *pareto-cap* (/ *design-pop* 3))
;; the maximum number of designs in the pareto before pruning.

(setf *good-cap* (/ *design-pop* 4))
;; the maximum number of designs on good.

(setf *tot-iter* 40)
;; total number of iterations for the process.

(setf *topdesigns-num* 25)
;; the number of top designs reported for a completed run.

(setf *iter_dump_designs* 50)
;; the number of iterations at which to dump the design population

(setf *percent-kept* 0.75)
;; the approximate percentage of designs kept from one iteration
;; to the next. Otherwise C-agents generate some from scratch.

(setf *remove-similar-designs* t)
;; boolean that determines whether or not to prune designs when
;; the population caps are reached.

(setf *min-agent-U* 0.1)
;; the minimum value an agent population can have.
;; if <= to min-agent-pop then = to min-agent-pop.

(setf *num-of-discretize-points* 20)
;; the number of points in the range that define the objective calc-range

(setf *max-num-ebs* 15)
;; the maximum number of ebs that can be put into one design

(setf *gravity* 9.81)
(setf *pi* 3.1416)
(setf *pi/4* 0.7854)

```

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; The following are structures used by the process.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; The FUNCTIONAL PARAMETER, FP describes the energy state used in all
;;; connections and ports in all the systems. Borrowed from the Welch&Dixon
;;; representation but expanded with the interface character and direction
;;; character. By the way, the 9 variables that make up the FP are known
;;; as characters or characteristics.
;;; Create-fp is the constructor for making an FP.
(defstruct (fp (:constructor create-fp
                (&optional through across class domain coord
                 inter direct index)))
  (THROUGH nil)           ;once effort, now through
                        ;it's a list of lists of lists
                        ;list of the through variable for each
                        ;connection in index repeated for each
                        ;domain in domain
  (ACROSS nil)           ;once flow, now across
                        ;it's a list of lists of lists
                        ;list of the time differentiation of
                        ;the across variable for each
                        ;domain in domain
  (CLASS nil)           ;class = {signal power material}
  (DOMAIN nil)          ;energy domain
                        ;can be a list if more than one domain
                        ;= {trans, rotat, elect, hydra, therm}
                        ;= {trans-x, trans-y, rot-z, etc.}
  (COORD nil)           ;4 x 4 coord frame position
  (INTER nil)           ;interface = {any accepted interface
                        ;symbol eg. 9/16-in-bolt}
  (DIRECT nil)          ;direction of energy flow
                        ;= {sink, source}
  (INDEX nil)           ;index of comps connecting to this fp,
  )

;;; The CONSTRAINT PARAMETER, CP is a functional parameter for constraints
;;; on component connections. It identifies the bounds that a particular
;;; FP must have to match a connection. Any character of an FP can be
;;; constrained. In the through/across pair of FP is identified in TY, and
;;; its maximum magnitude is identified by MG. If TY = across, then MG is
;;; a triple noting the maximum magnitudes of the (integral none derivative)
;;; of the across variable.
(defstruct (cp (:constructor create-cp (&optional throughrange acrossranges
                                       oper class domain coord
                                       inter direct)))
  (THROUGH RANGE nil)   ;range of through var.
  (ACROSS RANGES nil)  ;ranges of across vars. - triple list
  (OPER nil)           ;time operator
                        ;= {deriv none integ}
  (CLASS nil)         ;class = {signal power material}
  (DOMAIN nil)        ;energy domain
                        ;= {trans, rotat, elect, hydra, therm}
  (COORD nil)         ;4 x 4 coord frame position
  (INTER nil)         ;interface = {any accepted interface
                        ;symbol eg. 9/16-in-bolt}
  (DIRECT nil)        ;direction of flow = {source, sink}
                        ;not what is supplied by that
                        ;component but what is required
  )

```

```

;;; The EMBODIMENT, EB structure is used to describe all components read in
;;; by the catalog. Borrowed from the W&D representation with a behavior
;;; change, constraints and evaluations. For simplifying the structure,
;;; behavior is split into three things on this level: MG-change, PO-change,
;;; and BG.

```

```

(defstruct eb
  data ;characteristic data for the
      ;following device
  MG-change ;matrix for overall change in
           ;magnitude of component
  PO-change ;matrix for position change of
           ;component
  const-param ;list = (DO MG-limit OT PO IT)
             ;if not constrained
             ;by one of these then nil
)

```

```

;;; The COMPONENT, COMP structure is used to describe all components read in
;;; by the catalog. Borrowed from the W&D representation with a behavior
;;; change, constraints and evaluations. For simplifying the structure,
;;; behavior is split into three things on this level: MG-change, PO-change,
;;; and BG.

```

```

(defstruct comp
  data ;list of values of data in the EB's
  evals ;list = (cost weight efficiency etc.)
)

```

```

;;; The SYSTEM CONFIGURATION, SC structure holds a complete or possibly
;;; incomplete design state. The graph contains the information about the
;;; systems components and connectivity including components and FP's. The
;;; c-agents holds the responsible maker-agents for the device. The c-agents
;;; holds the responsible fragment-agents for the device. And evaluations
;;; contains the final evaluations of the device as determined in the
;;; evaluate stage of the process.

```

```

(defstruct sc
  graph ;list of fps' involved in design
  behavior-eq ;list of functionality of inter-
             ;acting component characters
             ;in FP-ID found in graph
  embodiments ;list of emobidments in the design
  c-agents ;list of conceptual agents
           ;responsible for design
  components ;list of components in the design
  i-agents ;list of instantiation agents
           ;responsible for design
  f-agents ;list of fragment agents
           ;responsible for design
  evaluations ;list of evaluatable criteria
)

```

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; The interface-list contains the possible matches of interface types.   ;;;
;;; If an interface doesn't match with any on the list than it is assumed to ;;;
;;; only match with itself.                                               ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(setf *interface-list* '((belt pulley)
                        ((gear-teeth gear-teeth) . gear-teeth)
                        ((shaft shaft-hole) . shaft)
                        ((male-pipe female-pipe))
                        ((chain sprocket-teeth))
                        ((dial dial))
                        ((feet flat-user-interface))
                        ((hand flat-user-interface))
                        ((hand handle-user-interface))
                        ((hand button-user-interface))
                        ((bolt hole) . bolt)
                        ((bolt belt) . bolt)
                        ((bolt bolt) . bolt)
                        ((wire wire) . wire)
                        ))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; The interface-list for MEMS is quite simple. Ends of components are   ;;;
;;; described by their open faces.                                         ;;;
;;; If an interface doesn't match with any on the list than it is assumed ;;;
;;; to only match with itself.                                             ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(setf *interface-list-mems* '((east west)
                              (north south)
                              (up down)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; The following are the files and design desicription used by the process. ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(setf *library-dir* '(:relative "libraryEM"))
;; directory name of where the library of components/embodiments is stored
(setf *code-dir* '(:relative "codeEM"))
;; directory name of domain specific code mostly agent code

(setf *gen-code-dir* '(:relative "codeGeneral/"))
;; directory name of general code mostly agent code

(setf *library-file*
      (make-pathname :directory *library-dir* :name "Embodiments"))

(setf *input-agents-file* (make-pathname :directory '(:relative "output")
                                         :name "agents"
                                         :type "out"))
(setf *input-designs-file* (make-pathname :directory '(:relative "output")
                                         :name "alldesigns"
                                         :type "out"))
(setf *input-optimal-designs-file* (make-pathname
                                     :directory '(:relative "output")
                                     :name "topdesigns"
                                     :type "out"))

(setf *input-agents-file* (make-pathname :directory *code-dir*

```

```

                                :name "initagents"
                                :type "lisp"))
(setf *input-designs-file* nil)

(setf *input-optimal-designs-file* nil)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; Load in the other lisp files involved in the process,
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(tenuring (load (make-pathname :directory *gen-code-dir* :name "design")))
(tenuring (load (make-pathname :directory *gen-code-dir* :name "create")))
(tenuring (load (make-pathname :directory *gen-code-dir* :name "io")))
(tenuring (load (make-pathname :directory *gen-code-dir* :name "functions")))
(tenuring (load (make-pathname :directory *gen-code-dir* :name "trend")))
(tenuring (load (make-pathname :directory *gen-code-dir* :name "update")))

(tenuring (load (make-pathname :directory *code-dir* :name "cagents")))
(tenuring (load (make-pathname :directory *code-dir* :name "fagents")))
(tenuring (load (make-pathname :directory *code-dir* :name "iagents")))
(tenuring (load (make-pathname :directory *code-dir* :name "equer")))
(tenuring (load (make-pathname :directory *code-dir* :name "evaluate")))

(tenuring (setf *eb-library* (read-library)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; Set the all-important grounds for the system
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(setf *new-connects*
      (build-fps
        '((nil (((0 120) (0 120) (0 120))) power (elect) nil
              three-prong-outlet source nil)
          (nil ((0 0 0)) power (trans) nil bolt sink (ground))
          (nil ((0 0 0)) power (rotat) nil bolt sink (ground))
          (nil ((0 0 0)) power (rotat) nil shaft-hole sink (ground))
          (nil ((0 0 0)) power (hydra) nil female-pipe sink (ground))
          (nil ((0 0 0)) power (elect) nil wire sink (ground)))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; Finally, describe the design problem at hand in terms of inputs and
;;; outputs of the system. nil can be placed anywhere to denote
;;; no particular specification.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(setf *io-fps*
      (build-fps
        '(((0 297)) ((nil nil (goal (0 0)))) power (trans)
            ((0 1 0 0) (-1 0 0 0) (0 0 1 0) (0 0 0 1)) feet source
            (goal))
          (nil ((nil (goal bound) (goal (0 5)))) power (rotat)
            ((-1 0 0 1) (0 1 0 5) (0 0 -1 0) (0 0 0 1))
            dial sink (goal)))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; MEMS design problem in terms of inputs and outputs of the system.
;;; nil can be placed anywhere to denote no particular specification.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```

```
(setf *io-fps*
      (build-fps
        '((nil nil nil nil)
          ((nil nil (goal (0 0))) (nil nil (goal (0 1)))
           (nil nil (goal (0 0))) nil)
          power (trans-x trans-y rot-z elect) (0 0 0 1) (down) sink (goal))
          ((nil nil nil nil) (nil nil nil ((goal (0 25)) nil nil))
           power (trans-x trans-y rot-z elect) nil
           (west north east south up down) source (goal))))))
```

evaluate.lisp (for Weighing Machine Problem)

```
;;; Evaluations.lisp for Weighing Machine Problem
;;; Contains functions called in evaluate in design.lisp
;;; These functions are specified in parameters
```

```
(setf *evaluators* '(cost mass dial-accuracy input-dx))
```

```
;;; The following functions perform evaluate
(defun evaluate-designs (designs eval-level)
  (cond ((endp designs) nil)
        (t (setf (sc-evaluations (car designs)) (evaluate-each-design
                                         (eval (car designs))
                                         *evaluators*))
              (evaluate (cdr designs) eval-level))))
```

```
(defun evaluate-each-design (design evaluators)
  (cond ((endp evaluators) nil)
        (t (cons (funcall (car evaluators) design)
                  (evaluate-each-design design (cdr evaluators))))))
```

```
(defun cost (design)
  (apply '+ (mapcar #'(lambda (x) (first (third x))) (sc-components design))))
```

```
(defun mass (design)
  (apply '+ (mapcar #'(lambda (x) (second (third x))) (sc-components design))))
```

```
(defun calc-ineff (design)
  (- 1 (apply '* (mapcar #'(lambda (x) (third (third x)))
                          (sc-components design)))))
```

```
(defun dial-accuracy (design)
  (let* ((equation (find-if #'(lambda (x) (equal (car x) '(0 5)))
                           (sc-behavior-eq design)))
         (answer (cond ((car equation) (discretize-interval (car equation))))
         (result (cond ((cadr equation) (solve-equation (cadr equation))))))
    (cond ((and result answer (and-list (mapcar 'numberp result))
              (and-list (mapcar 'numberp answer)))
          (/ (apply '+ (mapcar
                       #'(lambda (x y)
                           (* (- x y (- (car answer) (car result)))
                              (- x y (- (car answer) (car result))))
                       (cdr answer) (cdr result)))
             (1- (length answer))))
          (t (third *obj-constraints*))))))
```

```

(defun input-dx (design)
  (let* ((equation (find-if #'(lambda (x) (equal (car x) '(0 0)))
    (sc-behavior-eq design)))
    (answer (cond ((car equation) (discretize-interval (car equation))))
    (result (cond ((cadr equation) (solve-equation (cadr equation)))))
    (cond ((and result answer (and-list (mapcar 'numberp result)))
      (and-list (mapcar 'numberp answer)))
      (/ (apply '+ (mapcar
        #'(lambda (x y)
          (* (- x y (- (car answer) (car result)))
            (- x y (- (car answer) (car result)))))
        (cdr answer) (cdr result)))
      (1- (length answer))))
    (t (fourth *obj-constraints*))))))

;;; Discretize-Interval
;;; This function discretizes an interval by operator (third interval) which is
;;; a unary function. The number of discretized points comes from the
;;; constant that it set *num-of-discrete-points*. Basically, for interval
;;; (x0 xf), the results are return as
;;;          (xn-x0)
;;; yn = x0 + f(-----)*(xf - x0)
;;;          (xf-x0)
(defun discretize-interval (interval)
  (do ((x0 (first interval))
      (xf (second interval))
      (operator (third interval))
      (spacing (/ (- (second interval) (first interval))
        *num-of-discretize-points*))
      (xn (first interval) (+ xn spacing))
      (y nil (backcons
        (cond ((= xf x0) x0)
          (operator
            (+ x0 (* (- xf x0)
              (funcall operator (/ (- xn x0) (- xf x0)))))))
        (t xn))
      (y))
    (i 0 (1+ i)))
  (> i *num-of-discretize-points* y)))

(defun solve-equation (eq)
  (do* ((interval (find-if #'(lambda (x) (and (listp x) (numberp (car x))))
    (apply #'append
      (mapcar #'return-state-vars *io-fps*))))
    (equation nil (subst (car points) interval eq :test 'equal))
    (values nil (backcons (ignore-errors (eval equation)) values))
    (points (discretize-interval interval) (cdr points)))
  ((endp points) values)))

```

evaluate.lisp (for MEMS accelerometer Problem)

```
;;; Evaluations.lisp for MEMS
;;; Contains functions called in evaluate in design.lisp
;;; These functions are specified in parameters

(setf *evaluators* '(area Sy Kxx Amax))
(setf *global-ground* nil)
(setf *C_para* 120e-9)
(setf *V_m* 10)
(setf *Vn_circuit* 0.1)
(setf *Boltzman_K_b* 1.381e-23)
(setf *Temp* 298)
(setf *epsilon* 8.854e-12) ;8.854 pF/m
(setf *freq_range* 10000)
(require :foreign)
(load "./c_code/evaluate.so")
(ff:def-foreign-call return_objs ((level :int fixnum)
                                  (f :foreign-address array))
                                :returning :int)
(setf *objs-from-c* (make-array 4 :element-type `double-float))

;;; The following functions perform evaluate
(defun evaluate-designs (designs level)
  (cond ((endp designs) nil)
        (t (format nil "Creating netlist...~%"
                     (create-netlist (add-joints-to-design (car designs)))
                     (format nil "calling c_code...~%"
                               (cond ((zerop (return_objs level *objs-from-c*))
                                      (setf (sc-evaluations (car designs)) *obj-constraints*))
                                     (t (format nil "return from c_code...~%"
                                                  (let* ((Kxx (aref *objs-from-c* 0))
                                                         (Kyy (aref *objs-from-c* 1))
                                                         (By (aref *objs-from-c* 2))
                                                         (My (aref *objs-from-c* 3))
                                                         (electro-params (find-electro-params
                                                                    (sc-embodiments (car designs))
                                                                    (sc-components (car designs))))
                                                         (A_c (car electro-params))
                                                         (g0 (cadr electro-params))
                                                         (C0 (cond ((zerop g0) 0)
                                                                (t (/ (* A_c *epsilon*) g0))))
                                                         (Sy (calc-sensitivity-y My Kyy C0 g0)))
                                                         (setf (sc-evaluations (car designs))
                                                               (list (calc-b-b-area (car designs))
                                                                     (/ 1 Sy)
                                                                     ;(calc-a-min By My Sy)
                                                                     (/ 1 Kxx)
                                                                     (/ 1 (calc-a-max Kyy My C0 g0 Sy))
                                                                     ))))))
                     (evaluate (cdr designs) level))))))

(defun calc-b-b-area (design)
  (do* ((coords nil (mapcar #'eval (fp-coord (car fps))))
        (max-x 0 (cond ((and (numberp (car coords)) (> (car coords) max-x))
```

```

      (car coords)) (t max-x)))
(max-y 0 (cond ((and (numberp (cadr coords)) (> (cadr coords) max-y))
  (cadr coords)) (t max-y)))
(min-x 0 (cond ((and (numberp (car coords)) (< (car coords) min-x))
  (car coords)) (t min-x)))
(min-y 0 (cond ((and (numberp (cadr coords)) (< (cadr coords) min-y))
  (cadr coords)) (t min-y)))
(fps (sc-graph design) (cdr fps)))
((endp fps) (* (- max-x min-x) (- max-y min-y))))))

(defun find-electro-params (ebs comps &optional (A_c 0) (g0 0))
  (cond ((endp ebs) (list A_c g0))
        ((or (equal (car ebs) 'h-electrostatic-gap)
              (equal (car ebs) 'v-electrostatic-gap))
         (list (+ A_c (* 2 2.0e-6 (third (cadar comps)) (fifth (cadar comps))))
               (fourth (cadar comps))))
        (t (find-electro-params (cdr ebs) (cdr comps) A_c g0))))
(defun calc-sensitivity-y (My Kyy C0 g0)
  (cond ((zerop g0) (/ 1 (second *obj-constraints*)))
        (t
         (/ (* 2 C0 My *V_m*) (* (+ (* 2 C0) *C_para*) Kyy g0))))))
(defun calc-a-min (By My Sy)
  (cond ((zerop Sy) (third *obj-constraints*))
        (t
         (sqrt (+ (/ (* *Vn_circuit* *Vn_circuit*) (* Sy Sy))
                  (/ (* 4 *Boltzman_K_b* *Temp* By *freq_range*
                    (* My My)))))))
(defun calc-a-max (Kyy My C0 g0 Sy)
  (cond ((zerop g0) (/ 1 (fourth *obj-constraints*)))
        (t
         (let* ((E0 (/ (* C0 *V_m* *V_m*) 2))
                (D (expt (- (* E0 g0 g0 g0 (sqrt (* Kyy Kyy Kyy))
                    (sqrt (+ E0 (* g0 g0 Kyy))))
                          (* E0 g0 g0 g0 g0 Kyy Kyy)) (/ 1 3)))
                (R (sqrt (- (+ 1 (/ (* 2 D) (* Kyy g0 g0))) (/ (* 2 E0) D))))))
           (cond ((or (= 1 R) (typep R 'complex))
                  (/ 1 (fourth *obj-constraints*)))
                 (t
                  (abs (/ (* Kyy g0 R (- 1 (/ (* 4 E0) (* Kyy g0 g0 (- 1 (* R R))
                    (- 1 (* R R))))))
                        My))))))))))

```

```

;;; This function takes the linked list and actuator data and
;;; creates a netlist of the format recognized by Analogy SABER
;;; software.

```

```

(defun create-netlist (design)

```

This function and the remaining parts of this file, set up the A-Design configurations to be used with the external approximation method of Prakash and Cagan and SABER.

```

)

```

D. SYNOPSIS OF PROGRAM FILES

CodeGeneral

- **design.lisp**

This is the main file that contains the function, *A-Design ()*, that begins the process, and *Find-Pareto-designs* function that separates designs into Pareto and non-pareto populations.

- **create.lisp**

This file contains functions to control the designs as they are being constructed by the C-agents and the I-agents. The functions from Section A in this appendix *Create-and-Repair-Configurations*, *Create-from*, and *Modify-designs* are included in this file.

- **update.lisp**

The updating of configurations and their instantiations is handled by functions in this file. The *Update-Configuration* function with the pseudo-code presented in Figure 4.5 is performed by functions in this file.

- **io.lisp**

The input and output functions are all contained in this file. Reading in the catalog of components, and writing output files are functions found in this file. The functions *Write-design-data*, and *Write-results* are found in this file.

- **functions.lisp**

General functions, used by the process are stored here (e.g. sorting functions, random number generators, etc.). The *design-is-complete*, and *choose-agent* functions are found in this file.

- **magent.lisp**

Because the workings of the Manager-agent are independent of the functional representation, the Manager-agent functions are contained in this file in the **codeGeneral** directory. The pseudo-code shown in Figure 6.1 is implemented in this file.

- **trend.lisp**

Finding intersections in the detection of TODO and TABOO trends prove to be an intricate process. The functions in this file are invoked by the Manager-agent.

codeEM & codeMEMS

- **cagents.lisp**

The details of the Configuration-agents described in Section 5.3 are included here. The pseudo-code from Figure 5.5 is implemented in each C-agent in this file.

- **iagents.lisp**

Functions describing the Instantiation-agents and their workings are found in this file. The functions *Instantiate-Configuration* and *Instantiate-variables* are found here.

- **fagents.lisp**

Fragmentation-agents and their subroutines are found in this file which are invoked by the *Modify-designs* function.

E. OUTPUT FILES

Various output files have been created in testing the A-Design implementation. The graphs shown throughout the dissertation are the result of output files stored after executing the A-Design process. Data on the design process can be extracted from these files: **iter.out** (details the iteration data - how much process has improved, size of populations, etc.); **pareto.out**, **good.out**, and **poor.out** (list the members of these populations over the iterations); **cagents.out**, **fagents.out**, and **iagents.out** (list statistics on the agents as is similarly used by the Manager-agent); and **topdesigns.out** (the details of the top designs at the end of the search process).

The following are two designs from the **topdesigns.out** file that are used in this dissertation. The basic form of the design follows the SC structure shown in Figure 4.8.

topdesigns.out (SC of weighing machine shown in Figure 8.3a)

FPs: (Elements of these list corresponds to the slots in the FP structure. The first is through, the second is across, and so on. Note that the across variable slot has

three members. These correspond to across, across-differentiated, and across-integrated.)

```
((((BOUND) ((0 0 0)) POWER (TRANS)
  ((-1 0 0 1) (0 1 0 5) (0 0 -1 0) (0 0 0 1)) BOLT SINK
  ((4 1) GROUND))
 (BOUND) ((NIL (GOAL-MET BOUND) (GOAL-MET BOUND))) POWER (TRANS)
  ((-1 0 0 1) (0 1 0 5) (0 0 -1 0) (0 0 0 1)) BOLT SOURCE
  ((7 0) (6 1) (5 0)))
 (BOUND) ((NIL (GOAL-MET BOUND) (GOAL-MET BOUND))) POWER (TRANS)
  ((-1 0 0 1) (0 1 0 5) (0 0 -1 0) (0 0 0 1)) BOLT SOURCE
  ((6 0) (5 1) (4 0) (3 0)))
 (BOUND) ((NIL (GOAL-MET BOUND) (GOAL-MET BOUND))) POWER (TRANS)
  ((-1 0 0 1) (0 1 0 5) (0 0 -1 0) (0 0 0 1)) BOLT SINK
  ((5 1) GROUND))
 (BOUND) ((NIL (GOAL-MET BOUND) (GOAL-MET BOUND))) POWER (TRANS)
  ((-1 0 0 1) (0 1 0 5) (0 0 -1 0) (0 0 0 1)) NIL SOURCE
  ((3 1) (1 0)))
 (BOUND) ((0 0 0)) POWER (ROTAT)
  ((-1 0 0 1) (0 1 0 5) (0 0 -1 0) (0 0 0 1)) BOLT SINK
  ((2 1) GROUND))
 (BOUND) ((NIL (GOAL-MET BOUND) (GOAL-MET BOUND))) POWER (ROTAT)
  ((-1 0 0 1) (0 1 0 5) (0 0 -1 0) (0 0 0 1)) NIL SOURCE
  ((2 0) (1 1) (0 1)))
 (BOUND) ((NIL (GOAL-MET BOUND) (GOAL-MET (0 5)))) POWER (ROTAT)
  ((-1 0 0 1) (0 1 0 5) (0 0 -1 0) (0 0 0 1)) NIL SINK ((0 0) GOAL))
 (((0 297)) ((NIL (GOAL-MET BOUND) (GOAL-MET (0 0)))) POWER (TRANS)
  ((0 1 0 0) (-1 0 0 0) (0 0 1 0) (0 0 0 1)) NIL SOURCE ((7 1) GOAL)))
```

EBs: *(The order in this list is important to the connection of the configuration.)*
(DIAL PULLEY BEARING-ROTAT BELT SPRING LEVER1STCLASS SPRING FOOTPAD)

C-agents: *(Each C-agent in this list has a direct correspondence with the EBs list above. The list of numbers represents the agent's preference for the terms of the evaluation function.)*

```
((AGENT-TRANS-SINK-PARALLEL-GROUND (0.6 0.1 0.1 0.1 -0.1))
 (AGENT-ROTAT-SOURCE-PARALLEL-DANGLE (0.1 0.1 0.6 0.1 -0.1))
 (AGENT-ELECT-SOURCE-PARALLEL-GROUND (0.1 0.1 0.6 0.1 -0.1))
 (AGENT-TRANS-SINK-SERIES-CONNECT (0.1 0.6 0.1 0.1 -0.1))
 (AGENT-ROTAT-SOURCE-PARALLEL-CONNECT (0.1 0.6 0.1 0.1 -0.1))
 (AGENT-TRANS-SOURCE-SERIES-DANGLE (0.1 0.1 0.6 0.1 -0.1))
 (AGENT-TRANS-SINK-PARALLEL-CONNECT (0.2 0.2 0.2 0.2 -0.2))
 (AGENT-ROTAT-SOURCE-PARALLEL-DANGLE (0.1 0.1 0.6 0.1 -0.1)))
```

Components: *(The components that correspond with each of the EBs above.)*

```
((IMAG-DIAL-1 (0) (1.5 0.01 0.99))
 (PULLEY-FBS-A1-11 (0.0127 0.00635 0.00318)
  (4.47 0.004 0.99 5.08e-5 1.27e-5))
 (BEARING-ABS-A2-23 (1.9999999e-4 0.00635 0.015875) (7.21 0.03 0.99))
 (BELT-FAM-A1-7 (0.638 0.00318) (3.06 0.009 0.97))
 (SPRING-ERS-A1-26 (1874 0.00476 0.0381 7.3700005e-4)
  (0.93 0.012 0.98))
 (IMAG-LEVER1ST-5 (0.001 0.005) (1.2 0.001 0.95))
 (SPRING-ERS-A1-7 (2557 0.00317 0.00635 4.0600002e-4)
  (0.78 0.003 0.98))
 (IMAG-FOOTPAD-1 (0) (1.5 0.1 0.99)))
```

I-Agents: *(Each I-agent in this list has a direct correspondence with the Components list above.)*

(AGENT-CHEAP-UPPER-MG-LEAST-USED-DATUM
AGENT-LIGHT-MIDDLE-MG-LEAST-USED-DATUM
AGENT-LIGHT-UPPER-MG-MOST-USED-DATUM
AGENT-LIGHT-MIDDLE-MG-LEAST-USED-DATUM
AGENT-CHEAP-UPPER-MG-LEAST-USED-DATUM
AGENT-LIGHT-UPPER-MG-LEAST-USED-DATUM
AGENT-LIGHT-MIDDLE-MG-LEAST-USED-DATUM
AGENT-LIGHT-MIDDLE-MG-LEAST-USED-DATUM)

F-agents: *(The list of F-agents that have modified design in past iterations.)*

(AGENT-CHEAP-CHEAPER-IN-COMPS)

Evaluations: *(The list of attributes determined for this design.)*

(20.650002 0.169 1.441736e-2 2.78708e-3))

topdesigns.out (SC of MEMS accelerometer C shown in Figure 8.11)

FPs:

((((NIL NIL NIL NIL)
((NIL NIL (GOAL-MET BOUND)) (0 0 0) (NIL NIL (GOAL-MET BOUND))
(0 0 0))
POWER (TRANS-X TRANS-Y ROT-Z ELECT) (4.4999996e-5 1.2799999e-4 0 1)
(UP) SOURCE ((2 0)))
(NIL NIL NIL NIL)
((NIL NIL (GOAL-MET BOUND)) (0 0 0) (NIL NIL (GOAL-MET BOUND))
(0 0 0))
POWER (TRANS-X TRANS-Y ROT-Z ELECT) (4.4999996e-5 1.23e-4 0 1) NIL
SOURCE ((3 0) (2 1)))
(NIL NIL NIL NIL) ((0 0 0) (0 0 0) (0 0 0) (0 0 0)) POWER
(TRANS-X TRANS-Y ROT-Z ELECT) (4.8e-5 1.23e-4 0 1) (SOUTH EAST WEST)
SINK ((8 0) GROUND))
(NIL NIL NIL NIL)
((BOUND NIL (GOAL-MET BOUND)) (0 0 0) (NIL NIL (GOAL-MET BOUND))
(0 0 0))
POWER (TRANS-X TRANS-Y ROT-Z ELECT) (4.8e-5 5.2e-5 0 1) (NORTH WEST)
SOURCE ((8 1) (6 0)))
(NIL NIL NIL NIL) ((0 0 0) (0 0 0) (0 0 0) (0 0 0)) POWER
(TRANS-X TRANS-Y ROT-Z ELECT) (3.8e-5 5.2e-5 0 1) (EAST) SINK
((7 1) GROUND))
(NIL NIL NIL NIL)
((BOUND NIL (GOAL-MET BOUND)) (0 0 0) (BOUND NIL (GOAL-MET BOUND))
(0 0 0))
POWER (TRANS-X TRANS-Y ROT-Z ELECT) (4.4999996e-5 5.2e-5 0 1) NIL
SOURCE ((7 0) (6 1) (4 0) (3 1)))
(NIL NIL NIL NIL)
((NIL NIL (GOAL-MET BOUND)) (0 0 0) (NIL NIL (GOAL-MET BOUND))
(0 0 0))
POWER (TRANS-X TRANS-Y ROT-Z ELECT) (4.4999996e-5 5.0e-6 0 1) NIL
SOURCE ((4 1) (2 3)))
(NIL NIL NIL NIL)
((NIL NIL (GOAL-MET BOUND)) (0 0 0) (NIL NIL (GOAL-MET BOUND))
(0 0 0))
POWER (TRANS-X TRANS-Y ROT-Z ELECT) (6.5e-5 7.9999995e-6 0 1) NIL
SOURCE ((1 0) (0 1)))

```

((NIL NIL NIL NIL)
 ((NIL NIL (GOAL-MET BOUND)) (0 0 0) (NIL NIL (GOAL-MET BOUND))
  (0 0 0))
 POWER (TRANS-X TRANS-Y ROT-Z ELECT) (6.5e-5 5.0e-6 0 1) NIL SOURCE
 ((1 1) (0 3)))
((NIL NIL NIL NIL)
 ((NIL NIL (GOAL-MET BOUND)) (0 0 0) (NIL NIL (GOAL-MET BOUND))
  (0 0 0))
 POWER (TRANS-X TRANS-Y ROT-Z ELECT) (5.5e-5 0 0 1) NIL SOURCE
 ((2 4) (0 2) GOAL))
((NIL NIL NIL NIL)
 ((NIL NIL (GOAL-MET BOUND)) (0 0 0) (NIL NIL (GOAL-MET BOUND))
  (0 0 0))
 POWER (TRANS-X TRANS-Y ROT-Z ELECT) (3.5e-5 0 0 1) NIL SOURCE
 ((5 0) (2 2)))
((NIL NIL NIL NIL)
 ((NIL NIL (GOAL-MET BOUND)) (0 0 0) (NIL NIL (GOAL-MET BOUND))
  (0 0 0))
 POWER (TRANS-X TRANS-Y ROT-Z ELECT) (1.0e-5 0 0 1) NIL SOURCE
 ((5 1) (0 4)))
((NIL NIL NIL NIL)
 ((NIL NIL (GOAL-MET (0 0))) (0 0 0) (NIL NIL (GOAL-MET (0 0)))
  (0 0 0))
 POWER (TRANS-X TRANS-Y ROT-Z ELECT) (0 0 0 1) NIL SINK
 ((0 0) GOAL)))

```

EBs:

```
(MASS V-BEAM H-ELECTROSTATIC-GAP H-BEAM MASS MASS H-BEAM MASS V-BEAM)
```

C-agents:

```

((AGENT-SINK-SERIES-CONNECT-1 (0.2 0.2 0.2 0.2 0.2))
 (AGENT-SINK-SERIES-CONNECT-4 (0.1 0.1 0.6 0.1 0.1))
 (AGENT-SINK-PARALLEL-DANGLE-1 (0.2 0.2 0.2 0.2 0.2))
 (AGENT-SINK-SERIES-DANGLE-3 (0.1 0.6 0.1 0.1 0.1))
 (AGENT-SINK-SERIES-CONNECT-4 (0.1 0.1 0.6 0.1 0.1))
 (AGENT-SINK-SERIES-CONNECT-4 (0.1 0.1 0.6 0.1 0.1))
 (AGENT-SINK-SERIES-CONNECT-4 (0.1 0.1 0.6 0.1 0.1))
 (AGENT-SOURCE-PARALLEL-GROUND-2 (0.6 0.1 0.1 0.1 0.1))
 (AGENT-SINK-SERIES-CONNECT-1 (0.2 0.2 0.2 0.2 0.2)))

```

Components:

```

((MASS-10-5 (1.0e-5 5.0e-6))
 (V-BEAM-2-17 (1.7e-5 2.0e-6))
 (H-ES-GAP-5 (2.0e-6 10.0e-6 4.0e-6 12))
 (H-BEAM-3-3 (3.0e-6 3.0e-6))
 (MASS-15-25 (1.5e-5 2.5e-5))
 (MASS-15-25 (1.5e-5 2.5e-5))
 (H-BEAM-3-17 (1.7e-5 3.0e-6))
 (MASS-30-7 (3.0e-5 7.0e-6))
 (V-BEAM-2-3 (3.0e-6 2.0e-6)))

```

I-Agents:

```

(AGENT-2-MIDDLE-MG-MOST-USED-DATUM
 AGENT-3-MIDDLE-MG-MOST-USED-DATUM
 AGENT-2-MIDDLE-MG-MOST-USED-DATUM
 AGENT-3-UPPER-MG-MOST-USED-DATUM
 AGENT-2-MIDDLE-MG-LEAST-USED-DATUM
 AGENT-3-MIDDLE-MG-LEAST-USED-DATUM
 AGENT-3-MIDDLE-MG-MOST-USED-DATUM)

```

AGENT-1-MIDDLE-MG-LEAST-USED-DATUM
AGENT-3-UPPER-MG-MOST-USED-DATUM)

F-agents:
NIL

Evaluations:
(8.32e-9 4679.943142077555d0 1.811013219321773d-5 636.022287028054d0)

