EXPLORING INNOVATIVE DESIGNS BY RELAXING CRITERIA AND REASONING FROM PRECEDENT KNOWLEDGE

by

Dundee J. Navinchandra

S.M. M.I.T., 1985 & B.Tech. I.I.T., New Delhi, India, 1982

SUBMITTED TO THE DEPARTMENT OF CIVIL ENGINEERING IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 1987

Signature of Author _____
Department of Civil Engineering
September 25, 1987

Certified by _____
Professor R. Patil and Professor D. Sriram
Thesis Supervisors

Certified by _____
Professor David H. Marks
Thesis Committee Chairman

Accepted by _____
Professor Ole S. Madsen
Chairman, Departmental Committee on Graduate Students

Exploring Innovative Designs by Relaxing Criteria and Reasoning from Precedent Knowledge

by
Dundee J. Navinchandra

Submitted to the Department of Civil Engineering on September 25, 1987
in partial fulfillment of the requirements for the degree of Doctor of Science.

Abstract

This dissertation investigates computational models for innovative design. We propose that innovative design involves **exploring** a wide variety of design alternatives, identifying those which are **interesting** and **adapting** them if they have problems. These ideas have been implemented in a program called CYCLOPS.

Exploration plays an important role in innovative design. The process generates diverse design alternatives which can sometimes serendipitously lead to interesting solutions. The more diverse these alternatives, the better the chances of finding an innovative design. One way of generating diverse design alternatives is to gradually break away from the norms and pre-specified constraints and to look at alternatives that would otherwise not be considered. We present an algorithm for exploration which is based on gradually relaxing the constraints and objectives imposed on the design. The algorithm is built into CYCLOPS' multi-objective search mechanism which generates **pareto optimal** designs.

During exploration, CYCLOPS also looks for **interesting** aspects of design alternatives by comparing them to a database of past experiences called **precedents**. When CYCLOPS finds a design to be similar to some favorable past experience, it introduces a new criterion to reflect its discovery. The emergence of **new criteria** can significantly influence the design process and the resulting solutions.

Further, during the process of exploration, CYCLOPS might identify situations which provide certain opportunities, yet are flawed in some respects. CYCLOPS attempts to exploit these situations by eliminating the flaws using knowledge drawn from similar situations in its database. This process is called design **adaptation**. It has been observed that the adapted designs appear innovative when they incorporate knowledge drawn from diverse sources which, at first glance, seem unrelated to the problem at hand. CYCLOPS is able to make such innovative adaptations by drawing analogies from a diverse set of precedents. An algorithm, **demand posting**, is proposed as a means for precedent-based analogical problem solving.

CYCLOPS is implemented in the domain of Landscape Architecture. It solves urban layout design problems. At present, the program has a modest precedent database and has been used to solve small, but representative landscape design problems. Our results are preliminary, however, CYCLOPS represents another step towards a new generation of innovation-enhancing design tools.

Thesis Committee:

Dr. R. Logcher, Professor of Civil Engineering
Dr. D. Marks, Chairman, Department of Civil Engineering
Dr. R. Patil, Assistant Professor of Computer Science
Dr. D. Sriram, Assistant Professor of Civil Engineering
Dr. D. Tomlin, Associate Professor of Architecture, Harvard University

Dedication

To my parents, Mrs. and Mr. Jithachandra for making me study and for emphasizing the importance of education.

To Lord Sri. Venkateshwara.

I would like to thank ....

my advisor, Professor David H. Marks for inculcating in me an aptitude for research. For guiding me, and supporting me through some very rough times. For suggesting and encouraging my investigations into new research areas. His experience and foresight have played a major role in molding my professional life. Above all, his personal warmth and caring have made my stay at MIT very enjoyable. It is to him that I am most deeply grateful.

my thesis committee, Professors Robert Logcher, David Marks, Ramesh Patil (MIT-CS), D. Sriram and Dana Tomlin (Harvard-GSD) for their comments and guidance. There were times when I had too many ideas - all in different directions, things seemed chaotic for a while. It was my committee's patience and understanding that helped my develop a coherent thesis.

Professor Robert Logcher for inculcating in me a love for computers and computer aided engineering - something which will stay with me for the rest of my life. He has always been very helpful to me and has been open to new ideas.

Professor Ramesh Patil for helping me organize my thoughts and for providing valuable criticism. I appreciate the patience and care he put into reading and improving this dissertation. He was always available for stimulating discussions about various research issues.

Professor Sriram for getting me excited about computer aided design, the very topic of this dissertation. We had many long discussions which helped me identify and pursue interesting research directions. I appreciate his open-minded approach to "blue-sky" ideas and his dedication to original research.

Professor Tomlin for helping me understand design and designing. It was he who told me about the role of criteria emergence in design. This idea is central to the thesis presented here.

members of the Design Theory and Methods Group (MIT Architecture): James Anderson, Stephen Ervin, Professor Aaron Fleisher and Dr. Mark Gross for the weekly meetings where we discussed half-baked ideas and argued different points of view.

Dr. Ravinder Jain (CERL) for his foresight in recognizing the importance of AI in engineering - very early in the game. For encouraging me and for giving me the opportunity to research new ideas at CERL and MIT.

Professor M. Kim (Univ. of Illinois Urbana-Champaign) for helping me realize that design isn't just constraint satisfaction. We had several long discussions about AI, design and innovation. It was through my interaction with him, that the idea of design exploration was developed. This idea is an important part of my thesis.

my dorm-mates, for interesting gossip.

Mark Gross, for helping me develop an interest in constraint based design. For making many valuable contentful, editorial and stylistic suggestions to this dissertation. For being very supportive during times when things were slow and I was very low emotionally.

Mukesh Ahuja, for his optimism. Something I needed from time to time.

Ann Brach, for philosophical discussions on AI and everything else under the sun.

Frank Chen, for being so practical.

Fadi Chehayeb, for his interesting ideas and for stimulating discussions.

Jonathan Cherneff, for his humor.

Robert Clyatt, for making me develop the IMST system.

Mark Coe, for his enthusiasm and interest in my work. For philosophical discussions.

Mark Fox, for helping me develop an interest in constraint directed systems.

Jaideep Ganguly, for interesting discussions about AI.

Nick Groleau, for his interesting views.

Ayman Hindi, for his company.

Nitin Joglekar, for helping me get my priorities straight.

Kewan Khawaja, for allowing me time for my research by taking up the GIS project.

Anil Khullar, for his untiring interest in new ideas.

Robert Petrossian, for good discussions during the early stages.

The Smurfs for entertaining me.

Fouad Tamer, for his company.

Mingtch Wang, for trying to convince me to get married - but in vain.

Jim Westervelt and Michael Shapiro, for their friendship and hacking tips.

Jacqueline Winton for being so nice to me and for all those muffins.

Hera, for always being up. For staying by my side through all those grueling nights.

the sponsors, U.S. Army Corps of Engineers, Construction Engineering Research Laboratory, Champaign, Ill. for supporting the research.

Table of Contents

List of Figures

Chapter 1

Introduction and Overview

Designing is among the more complex tasks that humans perform. It is the process of producing artifacts that have desired properties. Designing is a central skill in many human tasks: Architects deal with shape and form to create new buildings, Financial planners manipulate money to design profitable portfolios and Mechanical engineers design functional machines with parts such as gears and cams.

Designing is an important part of engineering. Designers are constantly producing newer and better artifacts. In the modern competitive world, they are under constant pressure to turn out new and innovative products quickly. In order to improve the productivity of designers, computer aided design tools have been developed. In the short term, these efforts have focused primarily on automating the more routine and tedious tasks involved in design. Applications have been limited to computer-aided drafting, solid modeling, simulation and analysis. Some of the more sophisticated design tools have been built to assist primarily with the analysis and detailed-design phases of the overall design process. Examples of such systems are: STRUDL (Structural Design Language [Roos 66]), ADINA (Finite Element Analysis [ADINA Engg. Inc. 75]) and GDS (a drafting tool [McDonnel Douglas 84]). Having made substantial contributions towards the automation of design analysis, researchers are now focusing attention on aids/tools for conceptual design.

Conceptual design is that part of the design process in which: problems are identified, functions and specifications are laid out and appropriate solutions are generated through the combination of some basic building blocks. Conceptual design precedes analysis and detailed design. Conceptual design, unlike analysis, is non-algorithmic, is not purely numerical and encompasses several "soft" issues such as tradeoffs, judgment and creativity. Research into understanding and automating conceptual design has raised a set of issues very different from those

governing the automation of design analysis and detailing. Conceptual design is a mixed symbolic-numeric process whose automation draws upon ideas from Operations Research (OR) and Artificial Intelligence (AI).

Over the past five years researchers in AI have been developing knowledge-based systems for conceptual design. The majority of these systems view design as a ***search*** process. For example, a system that designs buildings might search for solutions by trying various combinations of structural elements such as beams, columns and slabs. One of the problems with using a search technique is that if it is not properly controlled, it could lead to a combinatorial explosion. Researchers have addressed this problem by using constraints and heuristics to guide the search [Mackworth 77, Stefik 80, Fox 83]. Based on these developments, research efforts in design automation were started. A majority of these efforts have concentrated on routine design problems. An advantage of taking this approach is that, as routine design is a fairly well understood task, there is a rich set of design heuristics which can be used to effectively control the search process. Routine design systems can be organized in several ways. For example, (1) some routine design tasks can be broken up into discrete steps, where each step performs some specific design sub-task, (2) in some cases the problem can be approached hierarchically where, each level in the hierarchy is pre-determined, and (3) in other cases, specific heuristics are available for the different parts of the design. These heuristics can be packed into modules, where each module is responsible for one part of the overall design. Using strategies such as these, several systems have been built for various routine design applications. Examples of such applications are: building design [Sriram 86], integrated circuit design [Tong 86a] and the design of mechanical devices [Mittal 85, Steinberg et al. 86].

Based on the success of the routine design systems, interest is now being generated in building systems that can handle complex problems that are not as well understood. Researchers are now working on systems that take a non-routine, innovative approach to design. For example, the EDISON project at UCLA is aimed at building an invention system for mechanical devices [Dyer et

al. 86]; another example is the PROMPT system that develops new structural elements [Murthy & Addanki 87].

This is a thesis about innovative conceptual design. Unlike a routine design situation, an innovative design problem provides us with far fewer guiding heuristics to work with. Several assumptions that could be made in the context of routine design cannot be used anymore. Here are some prevailing issues: (1) As there is no prior knowledge about the shape or structure of the solution to a problem, it is not possible to pre-determine a set of steps that can be followed to produce a desired design. (2) The design criteria are liable to change during the design process. Criteria can either be relaxed or intensified, they can even be deleted or new criteria can be brought into consideration. (3) There may be multiple design objectives that have to be all maximized simultaneously. In non-routine design situations, there is no prior knowledge about the tradeoffs involved. (Little work has been done on multi-objective optimization of conceptual designs [Mackenzie & Gero 87]). (4) In certain cases a design might have problems that cannot be solved by search alone, requiring that knowledge from previous design experiences be used either directly or analogically.

This dissertation presents a computational model for conceptual design that addresses the issues listed above. We will show how some aspects of human innovative-design behavior can be cast in computational terms. In particular, we will focus on two aspects of conceptual design. The first is on how designers ***explore*** many design alternatives before choosing a final one. We will examine why ***exploration*** plays an important role in innovative design. The second focus is on how designers apply past experiences (precedents) to solve design problems. It is by reasoning from precedents that designers are able to solve problems in interesting ways. This aspect is termed design ***adaptation***.

In the rest of this chapter we will examine a model of design and its relation to our central thesis. Following this, a description of our computational model of design is presented. Finally, a summary of the dissertation is provided.

1.1 Main Points

There is no formula for design. The process of designing draws upon a wide range of knowledge and problem-solving techniques. No single algorithm will ever replace designers. However, it is possible to develop tools which can help designers design. It is our ***aim*** to develop an architecture for a CAD tool which will help designers explore alternatives and adapt designs which have problems. We would like our system to automatically explore a large variety of design alternatives and report only the interesting ones to the user. In addition, we would like the system to help the designer adapt designs which have problems. Normally, a designer might adapt designs by reasoning from his experiences about previous design cases. We would like the system to aid the designer by suggesting adaptations drawn from a large database of experiences stored in the computer. We hypothesize that a tool which can explore alternatives and adapt designs can help designers be more innovative. This dissertation argues this point and presents a program that demonstrates the idea.

1.1.1 A Model of Conceptual Design

In this subsection a model of design is presented to serve as a context for the work presented in this dissertation. Conceptual design can be viewed as a five stage process, involving: (1) **Problem Identification** - perceiving of a need, (2) **Formulation** - choosing an approach to the problem, (3) **Specification** - laying out the design requirements and performance specifications, (4) **Synthesis** - finding possible solutions, and finally, (5) **Evaluation** - checking designs for compliance with the specifications. The process of designing an artifact, however, need not strictly follow the stages outlined above. A designer might loop between stages, refining his product in each iteration (Figure 1-1).

The feedback loop plays an important role in conceptual design. Every time a we loop through the process, we develop a better understanding of the problem. We might discover new problems or opportunities relevant to the solution. Further, we might decide to extend or modify the representation being used. There are four types of feedback that can occur:

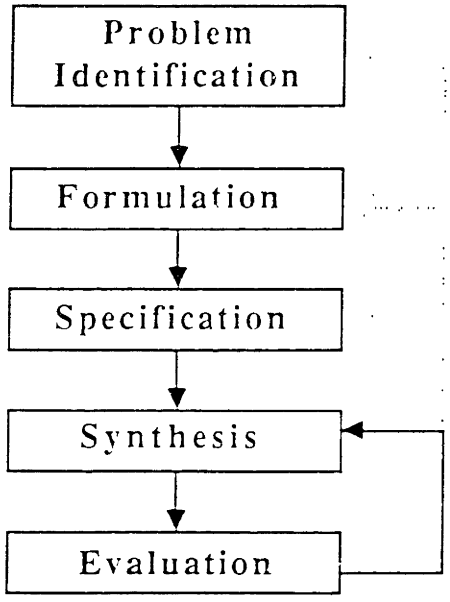

Figure 1-1: A model of conceptual design

- **Feedback to Synthesis from Evaluation** occurs when the design generated by synthesis is unsatisfactory. This happens when some constraints are violated or some goals are not realized. A majority of the design automation (DA) systems of today operate in the Synthesis-Evaluation Loop.

- **Feedback to Specification** occurs when the designer feels that the specifications need changing. Only few DA systems allow feedback to this level. Specifications can change in several ways. This thesis looks at three ways in which governing criteria can change: the addition of new criteria, the relaxation (weakening) of existing constraints and the making of tradeoffs among objectives.

- **Feedback to Formulation** requires the ability to reason about the representation itself. In order to change the representation meaningfully, the system would need knowledge about how and why representations are chosen for given problems. In this thesis only a narrow aspect of this behavior is examined. The thesis shows how new objects can be added to the representation, which in effect, extends the search space. The program has no knowledge about representation formalisms and all changes it makes are purely syntactic.

- **Feedback to Problem Recognition** requires the ability to ask questions such as "Are we looking at the right problem?". This kind of reasoning is presently beyond the scope of computer-aided design.

1.1.2 The Thesis

Our thesis has three major parts. The first thesis is that many useful design tasks can be modeled as configuration problems and that, search techniques can be used to find solutions to these problems. A configuration problem is defined as follows: "Given a pre-defined set of generic objects, find a way of arranging some or all of the objects such that the final arrangement (configuration) satisfies a given set of requirements (criteria)." For example, a building is composed of generic objects such as beams, columns, slabs etc. and satisfies criteria about stability, economics and access.

The second thesis is that innovation designs can be obtained by exploring a wide variety of alternatives. Innovative designs are not obtained through some deliberate attempt at producing them, but by generating lots of design alternatives and throwing away the bad ones. Consequently, the ability of a system to innovate depends on how well it can generate diverse alternatives that break away from the norms and the governing constraints. This idea is based on the observation that new alternatives sometimes serendipitously lead to interesting solutions.

The third thesis is that innovation, in a given design domain, comes from the ability to use knowledge drawn from sources inside or outside the current domain. In particular, a design will appear novel if it incorporates knowledge acquired from past experiences which come from sources that seem unrelated to the current design problem. For example, consider a person who is trying to design a new type of tent that deploys itself by snapping into place. As there are no existing tents that have this ability, the designer may not be able to solve the problem by using standard tent parts. She/He[1] will have to draw upon experiences from outside the domain of tent design. An obvious precedent he could use is one about automatic umbrellas. The central point is that, some design problems cannot be solved if one is restricted to using only the standard components and methods. The process of trying to use non-standard precedents is key to innovative design. The perception of

[1]The designer is addressed as a "he" throughout the rest of this document. This is, however, no indication of the designer's gender.

an artifact as being innovative lies, not in some inherent property of the artifact but in the eyes of the beholder. For a given observer or a given group of observers there is a set of design styles they are accustomed to seeing in the artifacts designed by their peers. For example, a construction equipment designer will view an excavation robot as being an innovation. On the other hand, a robot designer will view the same robot as being just another application of familiar techniques. The robot's sophisticated vision, tactile and position sensing systems are not new to the robot designer and are part of the ***design culture*** he works within. For the construction engineer, on the other hand, an excavation robot is novel because robotics is outside his design culture. *A design culture is defined by the common practices, design styles and technologies used by people who operate within the culture. It defines the context they operate within, it's their weltanschauungen.* Consequently, it is our thesis that innovation is based on the ability to reason from precedents taken from across design cultures.

1.2 The Implementation

The thesis presents a program, CYCLOPS[2] that implements the ideas listed above. CYCLOPS operates in three primary modes: the ***normal search*** mode, the ***exploration*** mode and the ***adaptation*** mode.

1.2.1 Modes of Operation

In the ***normal search*** mode, the program performs a tree like search of design alternatives. It uses the given design criteria to guide the search by pruning off sub-optimal solutions. Left to itself, the normal search mode will produce designs that satisfy all the given criteria, provided such solutions exist, or fail to produce any solution.

In the ***exploration*** mode, CYCLOPS relaxes the governing criteria and searches alternatives

[2]Criteria Yielding, Consistent Labeling, Optimization and Precedents-based System

outside the original space of solutions. There are two reasons why we would like our system to relax criteria: Firstly, criteria relaxation is required in situations where the normal search mode is unable to find solutions. This happens when the problem is over-constrained. The second reason is based on our finding that criteria relaxation is useful, not only in over-constrained problems, but also in situations where solutions to the design problem are readily available. This is because criteria relaxation could find a design alternative which, by accident, turns out to be better than the ones available. There are two reasons why happens: First, the design space tends to be non-continuous, non-convex and non-monotonic. Due to the fragmented nature of the space it is possible that a relaxation might expose unseen peaks in the design space. A second way a new alternative can suddenly become important is when it is recognized as being interesting in some unexpected way. The program can recognize such opportunities by using a database of precedents. These precedents are used to recognize interesting designs. A design is said to be interesting if its characteristics cause the program to be reminded (retrieve) a precedent that was previously labeled as interesting. In addition to recognizing interesting designs, precedents can also be used to recognize potential problems.

In the ***adaptation*** mode, CYCLOPS uses precedent knowledge to solve design problems. For example, if the program finds an interesting solution by relaxing criteria in the exploration mode, it will have to find some way of compensating for the relaxation. For instance, consider a landscape designer who is trying to find a good location for a new hotel. Let's assume he finds a hillside location which provides an excellent view. This location, however, has a problem. As the hillside is steep, the designer has to violate the constraint on maximum allowable slope. If he is sure he wants to locate the hotel on the hillside, he will have to find some way of compensating for the violation. CYCLOPS tries to fix the problem by reasoning either directly or analogically from some precedent case. For example, CYCLOPS might be reminded of how terracing is used for hillside farming and consequently, might attempt to terrace the site for the hotel. This is a form of analogical reasoning, as the purpose of terracing in the precedent is different from the purpose it serves in the hotel location problem.

1.2.2 Architecture of CYCLOPS

The program CYCLOPS has several modules and its architecture can be viewed in layers. At the heart of the system is the basic search process which consists of a synthesis module (*synthesizer*) and a selection module (*selector*). The synthesizer takes a set of partial solutions as input and adds detail to them. For example, a synthesizer might take a partial job-shop schedule and add a few more jobs to it. The designs produced by the synthesizer are tested by a selector (Figure 1-2, Part I). The selector uses the given design criteria to select partial designs for further detailing. If, at any stage, a complete design passes selection, it is output as a solution. This basic cycle is the normal-search process that CYCLOPS performs. If CYCLOPS is unable to find a solution using this process or, if the user decides to examine alternatives beyond the normal solution space, the program enters the exploration mode.

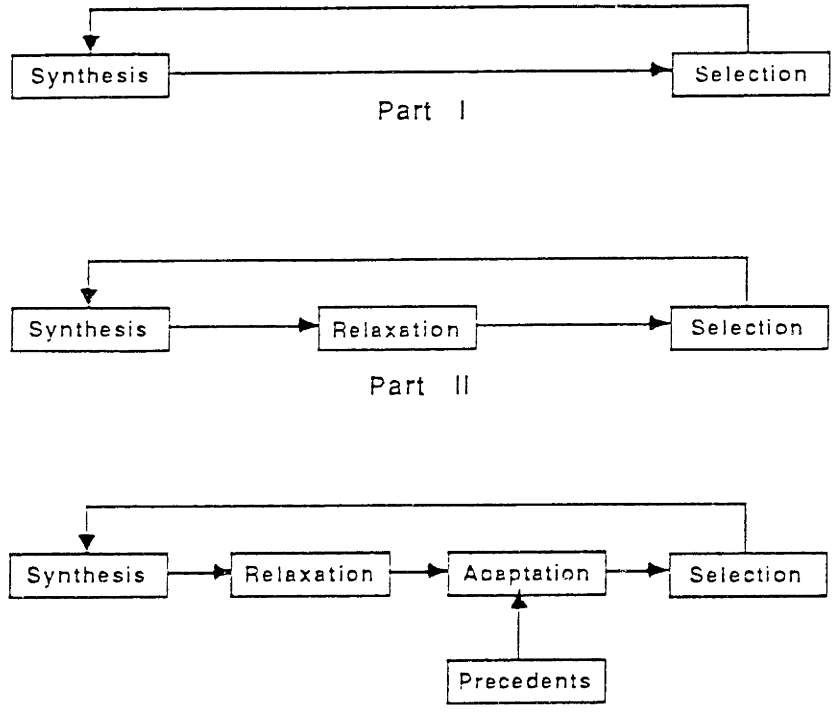

Figure 1-2: Architecture of CYCLOPS

CYCLOPS is able to explore alternatives by using a ***relaxation*** module (Figure 1-2, Part II). In this step the governing criteria are relaxed to allow more partial solutions to pass selection than would have normally. It is by using the relaxation module, that CYCLOPS can explore alternatives.

Finally, CYCLOPS has a precedent database it can draw upon (Figure 1-2, Part III). These precedents are used in two ways. Firstly, the precedents can be used as a source of new criteria. When the synthesis and relaxation modules produce alternatives, they are checked by the precedents. If a precedent finds a problem or an opportunity, a corresponding criterion is added to the selector. Secondly, precedents are used is for patching problems in design alternatives. This step is called ***adaptation*** and is done by reasoning either directly or analogically from precedent cases.

1.3 Limitations of the Research

The research and the implementation presented in the dissertation have several limitations:-

1. It is assumed that the design can be represented as a labeling problem. That is, the designed artifact consists of variables that have values assigned to them. Even though the number of variables can change during the design process, the assumption limits us to configuration-type designs.

2. The program does not actively verify analogies, it does not perform a quantitative or qualitative simulation/analysis of actions it takes. We felt this aspect was not central to our investigation. The program verifies analogies in a passive fashion. When an analogy is drawn, the modified design is posted to the precedents data-base. If none of the precedents find a problem, the analogy is deemed correct. We make the assumption that any clause that cannot be inferred from the knowledge possessed is untrue. This is a ***closed-world assumption*** on the knowledge base. In other words: If you don't have the knowledge to recognize a problem you will make the mistake of not recognizing the problem, and no amount of sophistication in the reasoning mechanism can help. In the future, some standard simulation or analysis technique could be used for verification.

3. Currently, all criteria are treated equally. This problem is alleviated, to some extent, by using a dimensionless measure of utility. Further, the effect of choosing wrong weights is further reduced (not eliminated) by the program's ability to relax criteria. In effect, if a particular criterion has a slightly lower weight than it ought to, then relaxations on other criteria will give the criterion due consideration.

4. The precedent's explanations are hand-coded. The explanation assigned to a precedent pre-determines how the precedent may be used. Further, the program presently does not have the ability to generalize and learn new strategies from successful designs.

5. CYCLOPS has only a limited knowledge base. Innovations the program makes is with respect to the knowledge the program has and not with respect to current state-of-the-art in the domain of inquiry.

1.4 Organization of the Thesis

The dissertation is centered around an example. The main body of the thesis starts with a scenario chapter. This chapter presents all the main points of the thesis through an extended example. Reading the scenario chapter should give the reader a complete, intuitive picture of the thesis and its contributions. The body of the dissertation addresses our three major theses. Chapters 3 and 4 are about design representation and search. Chapter 5 discusses design exploration and the emergence of new criteria. Chapter 6 is about precedent based adaptation. Chapter 7 presents the overall architecture and shows how all the pieces fit together into one coherent system. The dissertation ends with two concluding chapters. The first of which (Chapter 8) presents relationships to other research efforts. The last chapter (Chapter 9) talks about applications, future directions and contributions of the thesis.

In the following paragraphs the major topics of the thesis are presented in a summary form. The body of the thesis discusses these topics in detail and presents numerous examples covering both theoretical and implementation details.

Chapter 2.

Scenario. A landscape design scenario is presented in detail.

Chapter 3.

Representing the design. In this thesis we are concerned with configuration design. A design is said to be configured if it is a combination of generic objects. A configuration design is represented by a configuration language which is simply a collection of statements about objects and their inter-relationships. The design representation used here is called ***local labeling***. The labeling problem is represented as a set of variables. Each variable has associated with it, a set of values

from which one value may be chosen and assigned to the variable. A set of assignments that covers all the variables constitutes a complete configuration.

Chapter 4.

Design Criteria. Acceptable designs are characterized by criteria. Criteria can come in two forms: constraints and objectives. It is required that the final design simultaneously satisfy the given set of constraints. Further, the final design should be optimal with respect to the given objectives.

Selection by Optimality. Design problems often involve many complex and often conflicting objectives. It is important to be able to handle such complexity in the solution process. The solution technique uses traditional A^* search. However, the A^* algorithm has been modified to handle multiple objectives. This is done by using pareto optimality as the method for evaluating partial designs in the search tree. The technique is called the ***Pareto Optimality*** based A^*.

Criteria Relaxation. As design is a criteria satisfaction process, it is the aim of the designer to attain the "best" design possible. However, it is often not possible to find a design that simultaneously satisfies all the constraints and is simultaneously optimal with respect to all the objectives. Designers often find themselves dealing with over-constrained, sub-optimal situations. Under these conditions the designer has to relax constraints and tradeoff among the objectives in order to find a solution. In the thesis, a unified technique for handling both mechanisms is developed. The technique is called criteria relaxation, where, a criterion is any proposition about design, be it a constraint or an objective. Making a tradeoff in any multi-criteria situation is a difficult decision for the designer. There is a vast literature concentrating in this area. In this thesis, it was decided not to address utilities, values and tradeoff mechanisms.

Chapter 5.

Design Exploration. Designing goes beyond the mere satisfaction of criteria. Designers like to explore alternatives and like to be adventurous while solving hard design problems. If a designer is presented with a design that satisfies all the criteria, it is possible that, he may not be fully satisfied with it. He may still want to explore new designs in an attempt to find something interesting. In the

model presented, exploration is done by relaxing the expected utility on each criteria. When we relax criteria, we are prepared to accept designs that would normally have been pruned off the search tree. These alternate designs, through a synthesis process, go on to produce new designs.

Criteria Emergence. The criteria imposed on a design problem are always subject to change. While exploring alternatives, a designer may come across a situation that may influence him to change the original set of criteria: add new criteria or relax old ones. The designer is continually recognizing problems and opportunities in his design. He does this by drawing upon a knowledge base of precedents. All criteria, prior or emergent, come from this knowledge base.

A technique for representing precedents is proposed. A precedent represents some past experience. It is a frame-based [Minsky 75] data structure with slots for:

- the **conditions** of the experience,
- the **effects**, emotional or physical, and
- an **explanation** of how the effects followed from the conditions.

Whenever exploration unearths new design alternatives, it is possible that new criteria might emerge. This process can often change the course of the design process.

Chapter 6.

Design Adaptation. When a design alternative runs into problems it is possible that the problem can be solved by applying some previously acquired precedent. A technique, called **demand posting** is presented. The process of resolving design problems starts with posting a demand on the database of precedents. If a precedent is found, it is applied. If not, the problem's causal explanation is retrieved and posted. This process continues recursively till an appropriate precedent is found. If all fails, the demand is posted to the user.

Chapters 7, 8, 9.

These are concluding chapters.

Appendix A.

Formal Treatment of CLOP. The first appendix presents a formal treatment of the Consistent

Labeling Optimization Problem. The appendix presents the Pareto-Optimality based A^* algorithm with a proof of its optimality.

Appendix B.

A Trace of CYCLOPS in action. A trace of one of the implementations of CYCLOPS is provided. The different modules in CYCLOPS are introduced and described.

1.5 Summary

Thesis	Techniques	Chapters
Design as a multi-criteria configuration problem.	Modified A^* algorithm for multiple criteria.	Chapters 3 and 4
Design innovation through exploration.	Exploration through criteria relaxation and the emergence of new criteria.	Chapter 5
Design innovation through precedent-based reasoning.	Analogical Reasoning used for design adaptation.	Chapter 6

Chapter 2

Scenario

In this chapter a hypothetical scenario of a designing process is presented. The purpose of this scenario is to provide an intuitive grasp of the concepts presented in this dissertation. The scenario follows the steps that might be taken by a designer as he goes about solving a design problem. Interspersed with this is an account of how similar steps can be taken using the computational framework developed as part of this research. The purpose of the example is to focus on the **process** of design and not on its end product.

This chapter is based on one central example. The different sections in this chapter gradually develop the example and introduce various important issues. Section 2.1 describes a landscape design problem. Section 2.2 shows how search techniques may be used to find a solution. Section 2.3 introduces the notion of optimal design. Section 2.4 explains how search can be used to optimize solutions. Section 2.5 shows how multiple objectives interact and make design decisions difficult. Section 2.6 provides an example of how criteria relaxation can be used to explore new design alternatives. Section 2.7 is about the emergence of new criteria during the design process. Section 2.8 shows how precedents can be used to solve design problems. Sections 2.9 and 2.10 introduce the actual techniques used for reasoning from precedents both directly and analogically. Finally a summary and conclusions is provided in section 2.11.

2.1 The Problem

Consider the following design problem: A region in the south-west section of town is to be developed. Eight lots of land are available for purchase and subsequent development. Five landuses, namely, a *recreational-area*, an *apartment-complex*, a cluster of 50 *single family houses*, a large *cemetery* and a *dumpsite* are to be sited in this region.

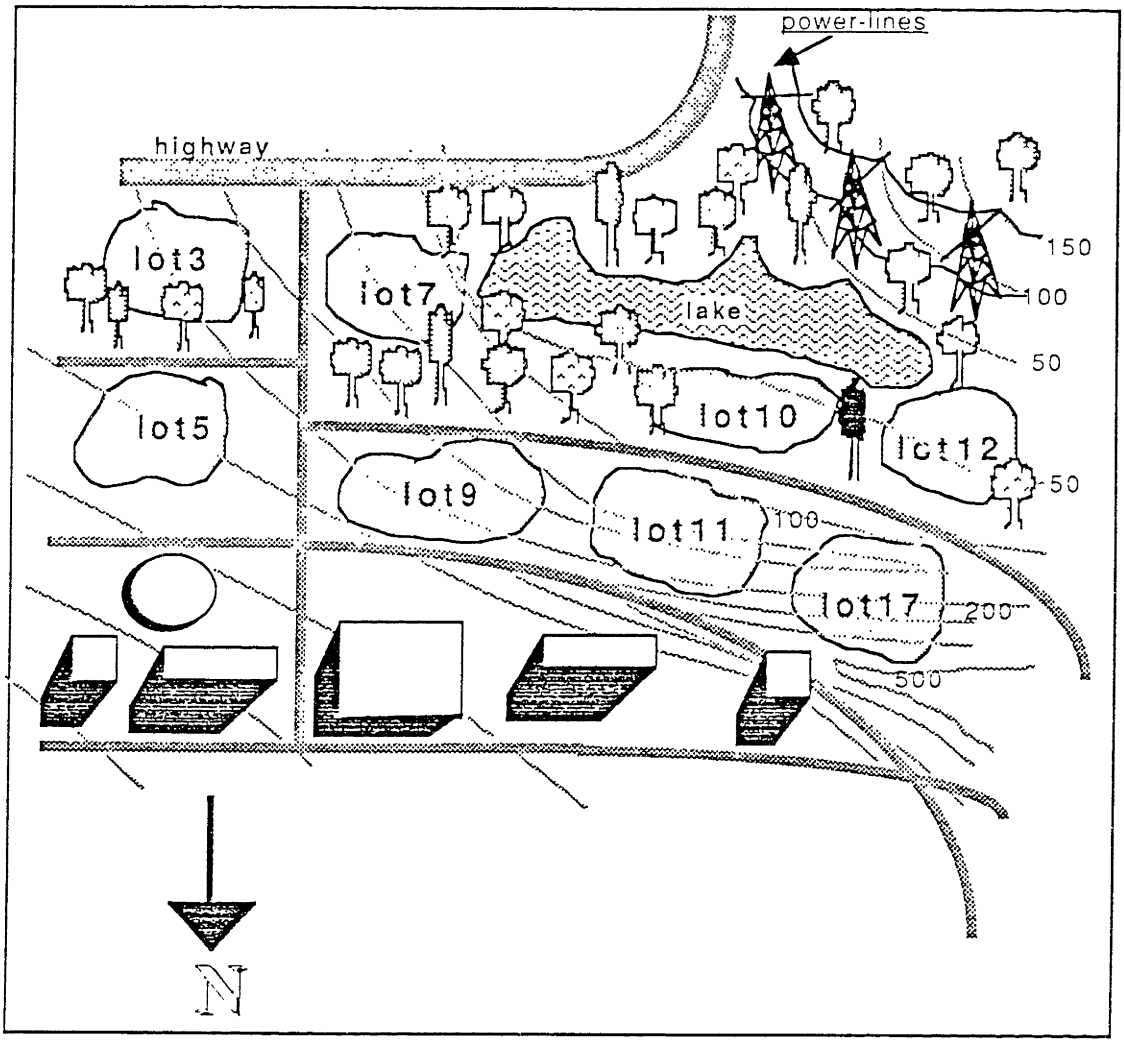

Figure 2-1: The site

Assume that the characteristics of the site have been determined using a standard geographic mapping system. A map of the site is shown in figure 2-1. The map shows eight lots of land which are available for development.

Lot3, lot5, lot7 and lot9 are all relatively flat sites with fairly good soil conditions. Lot10 and lot12 are moderately sloped sites, which have a nice wooded location but also have very poor soil conditions. Lot11 is a steep site with good soil conditions. Lot17 is a very steep site. Lot11 and lot17 are elevated sites which face south west and look down into the valley which has a lake and some wooded area.

The designer's task is to come up with assignments of landuses to sites. A complete design is one in which each of the landuses have been assigned to a lot of land. The final design should be one which complies with a given set of criteria.

Starting with a preliminary analysis, the designer first lays down some basic criteria:

1. Dumpsite and cemetery should not be visible from either of the dwellings (i.e. houses or apartments).

2. The recreational area should be near some water body.

3. Steep slopes are to be avoided for building.

4. Poor soil should be avoided for those landuses that involve construction.

5. The highway is noisy and ugly and should be avoided when locating the apartments, housing-complex and recreational areas.

Based on these criteria, the designer then prepares a table (Figure 2-2). The table shows the compatibility of each of the proposed landuses with respect to the different available lots of land. In the table, criteria 2,3,4 and 5 have been depicted. It is not possible to represent criterion 1 in this way. This is because it is a relationship between two landuses which have not been sited yet. Such criteria are said to be binary (i.e. they relate two sitable objects). A binary criterion can also be represented as a table, Figure 2-3 shows a criterion between the dumpsite and the housing complex. The other four criteria are called unary.

Landuse	lot3	lot5	lot7	lot9	lot10	lot11	lot12	lot17
Recreat-ional	no lake	no lake	OK	no lake	OK	no lake	OK	no lake
Apartment-complex	highway too close	OK	highway too close	OK	poor soil	steep	poor soil	very steep
Housing-sector	highway too close	OK	highway too close	OK	poor soil	steep	poor soil	very steep
Dumpsite	OK	OK	OK	OK	OK	steep	OK	very steep
Cemetery	OK	OK	OK	OK	OK	steep	OK	very steep

Figure 2-2: Tabular representation of the unary constraints

		Dumpsite							
		3	5	7	9	10	11	12	17
H	3	X	OK	X	OK	OK	OK	OK	OK
o	5		X	X	X	OK	OK	OK	OK
u	7			X	OK	OK	OK	OK	OK
s	9				X	OK	X	OK	OK
i	10					X	X	X	X
n	11						X	X	X
g	12							X	X
	17								X

Figure 2-3: Binary Criterion: Dumpsite should not be visible from the housing complex.

2.2 A Solution is found

Let's assume that the designer comes up with the following assignments: (cemetery lot3), (housing-sector lot5), (dumpsite lot7), (apartment-complex lot9) and (recreation-area lot10). This design has several interesting ideas. The dumpsite and the cemetery which are relatively noise insensitive, are placed next to the noisy highway. The housing is placed on lot5 and the apartment-complex is placed on lot9, by so doing, the designer has made sure that neither the dumpsite nor the cemetery can be seen from the dwellings. This is because the apartment-complex, being a high-rise building, has a far reaching view requiring that the apartments to be placed far from the dump and the cemetery. The single family housing complex, being a group of low rise buildings, will not be able to view the cemetery, even though it's close by (i.e. because of the trees in between). The recreational area is near a lake and is sufficiently far from the dumpsite. This is a good design as it satisfies the given criteria.

It is important to try and understand the type of reasoning that underlies the generation of the design presented above. Here are two ways of looking at the problem:

- The first approach involves thinking about the problem deductively. A deductive process might proceed something like this: *"The apartment-complex is a high rise structure and hence has a far reaching view, consequently it should be far from the dumpsite and the cemetery. The housing is a low rise building and could be placed next to the cemetery but not the dumpsite. That's because the cemetery has trees around it and the dumpsite is noxious. The lots near the highway are noisy and could accommodate noise insensitive landuses....."*. This is a technique which people might use to solve the problem. If the problem had 50 landuses, over a 100 sites and 30 criteria, the problem could drive anybody bananas!
- The second approach follows a different philosophy. Instead of thinking through deductively, a search technique is used. The idea is to generate partial designs and to prune away only when problems are recognized. In the context of a partial design, one does not spend time thinking about all the criteria and strategies for the placement of landuses, one just checks the design for problems. If problems are found, then the partial design falls from favor. An example of the search process is shown in Figure

2-4. The figure shows three landuses (variables) being synthesized. The synthesis is done in stages, where detail is added in each stage. The synthesis is started by first listing down the possible values the different variables can take. In the figure, the three variables: recreational-area, apartment-complex and housing-sector are listed with their corresponding set of possible assignments. For example, the apartment-complex can be located at either of lot5 or lot9 or lot12 (the other choices have been dropped in order to keep the figure simple). In the first stage of synthesis the values of the variables are combined into pairs. For example, the leftmost solution at the end of the first stage of the synthesis has the recreational-area on lot7 and the apartment-complex on lot5, and as this is only the first stage of synthesis, the solution has no information of where the housing-sector will be sited. The solution is denoted as: (7 5 ?), where the "?" stands for the location of the third variable: housing-sector. At the next stage of the synthesis, the solutions are paired into new ones which have all three variables instantiated. For example, the leftmost solution in the second stage: (7 5 9) is formed by joining (7 5 ?) and (7 ? 9). The technique is called ***latticed synthesis***. This systematic exploration of alternatives is difficult for humans but is amenable to computerization.

The latticed synthesis technique can be used to generate alternatives that satisfy the given constraints.

2.3 Adding some objectives

Let us now return to the landscape design scenario. The designer, meanwhile, has unearthed two more criteria he had noted earlier, but had forgotten to mention.

6. The cost of acquired land should be minimized

7. The total of excess noise for all the landuses should be minimized.

Solving the design problem now involves satisfying criteria 1 to 5 and optimizing criteria 6 and 7 simultaneously. This requires the calculation of total cost and total noise levels. The cost (in millions) and noise levels (in Noise Factors) for the sites are (lot3 = \$1.2m, 45NF), (lot5 = \$1.3m, 30NF), (lot7 = \$0.9m, 37NF), (lot9 = \$1.6m, 20NF), (lot10 = \$1.7m, 22NF), (lot11 = \$1.0m, 20NF), (lot17 = \$0.8m, 20NF) and (lot12 = \$1.4m, 22NF), respectively. The total cost is simply the sum of the costs of the plots of land. The total noise is the sum of the excess noise at each site. The excess

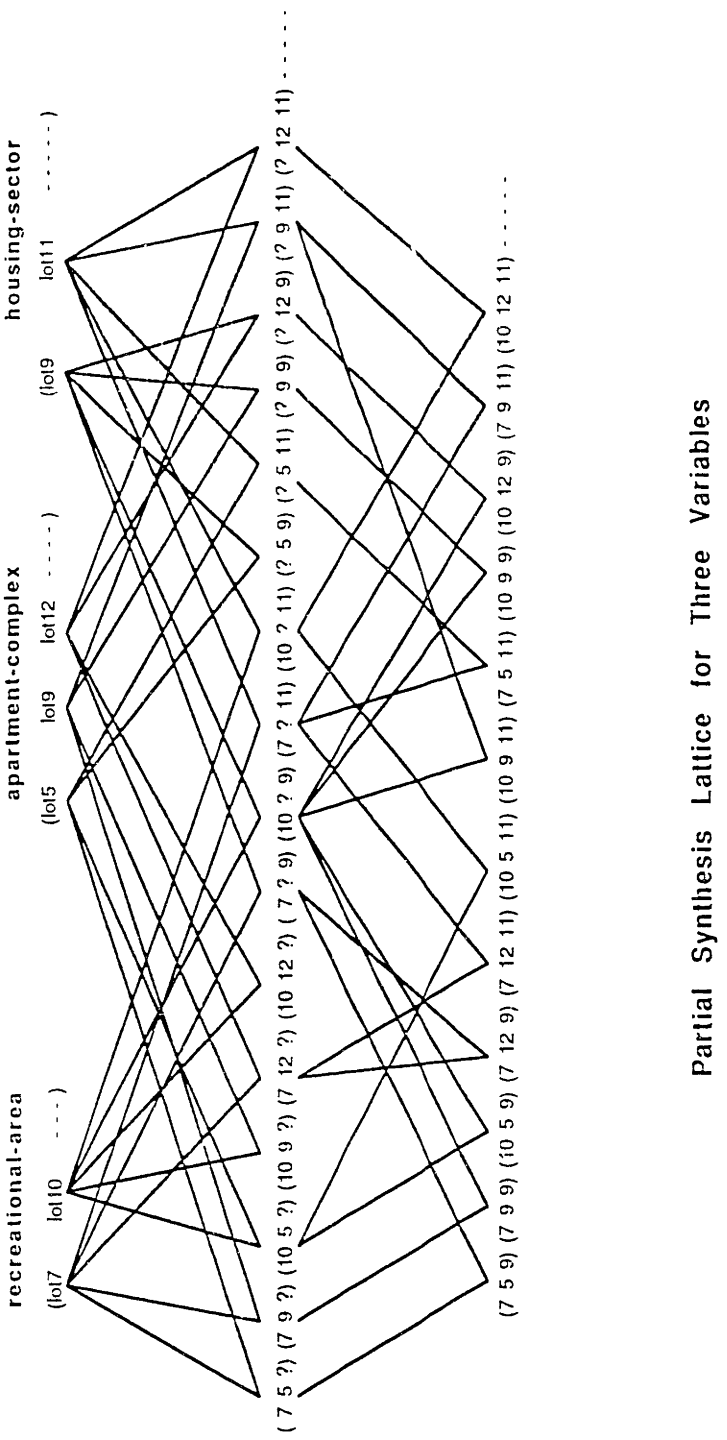

Figure 2-4: Partial Synthesis lattice of the example design problem.

noise at a site is found by comparing the site's noise with the acceptable NF level of the landuse sited there. The total excess noise is called the Effective Noise Factor[3] (ENF). The acceptable NF for the different landuses are: (recreational-area 20NF), (apartment-complex 30NF), (housing-complex 25NF), (dumpsite 50NF) and (cemetery 35NF) respectively.

We now have a multi-objective optimization problem at hand. The difficulty faced in handling multi-objective problems is that the objectives are often at odds. In our example, the cost minimization objective would prefer the cheaper lots: lot11, lot17 and lot7. On the other hand, the noise minimization objective will prefer that lot7 be avoided. Further, the cost minimization objective prefers lot11 and lot7 which happen to violate the constraint on minimum slope. In real problems we may have dozens of such objectives, all interacting with each other.

	Recr-ation	Apart-ments	Hous-ing	Dump site	Ceme-tery	Cost	ENF
Noise-Levels	20	30	25	50	35		
Alter-natives:							
A	lot10	lot5	lot9	lot7	lot3	6.7	14
B	lot7	lot9	lot5	lot12	lot3	6.4	32
C	lot12	lot5	lot9	lot7	lot10	7.1	4
D	lot10	lot12	lot7	lot5	lot3	6.5	24
E	lot10	lot7	lot12	lot3	lot5	6.5	9

Figure 2-5: A set of design configurations, their costs and ENF

Under these conditions, the solution to the design problem involves tradeoffs among the objectives. A simple and intuitive way of handling tradeoffs is to plot the different partial designs,

[3]The author would like to remind the reader that this is only a scenario. The actual techniques used for calculating cost and noise are not important. Do not stop to verify calculations, the purpose of this chapter is to provide the reader an intuitive grasp of the process not the product.

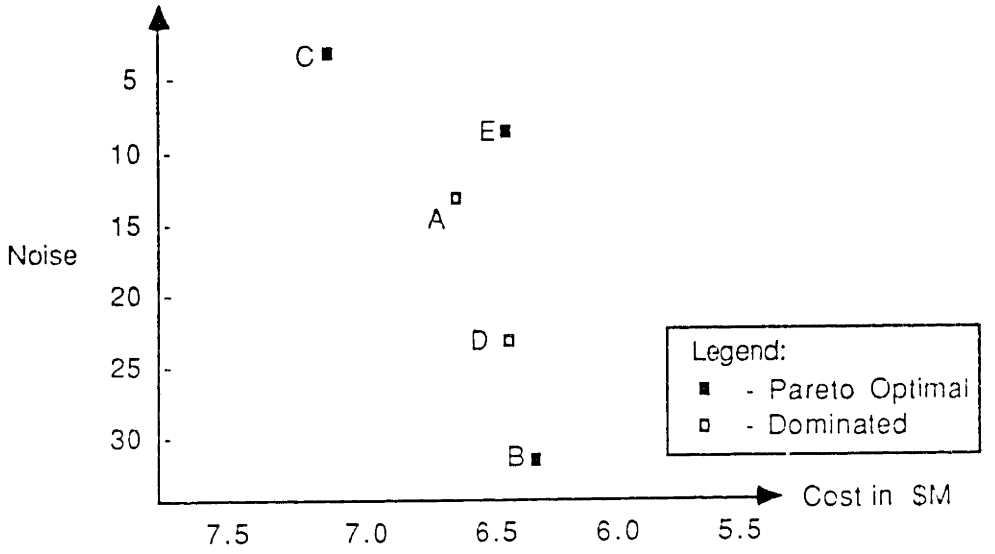

Figure 2-6: Pareto curve for cost vs noise.

onto a graph. For example, consider a set of alternate design configurations (A through E) shown in Figure 2-5. The cost and noise factors for the alternatives shown in Figure 2-5 can be plotted on a graph of noise vs. cost (Figure 2-6). In the graph, solutions A and D are inferior to E. The solutions B, C and E are said to be ***pareto optimal.*** A particular solution is said to be pareto optimal if it is not dominated by any other solution. A solution **X** is said to be dominated if there is any other solution **Y** on the graph, such that, **Y** has better values than **X** with respect to all the objectives (dimensions of the graph). This definition of pareto optimality gives us a method of dividing a set of solutions into two groups: the dominated set and the non-dominated set.

2.4 Evaluating Partial Designs

In the example above, we saw how a given set of design alternatives may be evaluated using pareto-optimality conditions. This approach, however, is not appropriate for large problems. In order to apply the technique, completed designs must be first generated and then tested for optimality. Substantial savings in computation may be achieved by moving the evaluation up into the generation phase allowing us to discard bad designs long before they are completed. This

approach may be viewed as a continuous synthesis process where, designs are evaluated at the end of each synthesis step with only the optimal partial design being allowed to pass on to the next synthesis step.

The synthesis-pruning process presented above, however, is not guaranteed to produce optimal solutions. By pruning off a partial design early in the synthesis process, it is possible to inadvertently cut off a branch leading to an optimal solution. This problem can be solved by using a technique which orders the search such that, the more attractive alternatives are searched first with, however, the option of returning to previously discarded solutions when none of the chosen paths lead to the optimal. This technique is called Best-First Search [Nilsson 80].

To implement Best-First search, we need some method for evaluating partial designs. In our example, optimality based on calculating the total cost and total noise factor (NF) of designs. The question is, "How can we calculate the total cost of a design that is only partially complete?" Although we can calculate the cost of whatever has been designed so far, we cannot find the total cost until the design is complete. It is possible, however, to *estimate* the cost of completing the partial design. For example, consider a partial design in which we have selected lot3 for the dump, lot5 for the cemetery and lot9 for the housing and we have yet to place the apartment-complex and the recreational-area. In this case, the best way to estimate the cost of the last two landuses is to make an optimistic estimate of the final cost. From Figure 2-2 (page 28) we can see that the recreational area can be sited on either lot7 ($0.9m) or lot10 ($1.7m) or lot12 ($1.4m)and the apartment complex can be sited on lot5 ($1.3m) or lot9 ($1.6m). In order to estimate the cost of the yet-to-be-sited landuses one may optimistically assume that the landuses will be assigned to the cheapest of the possible sites. In the example, the recreational area gets assigned to lot7 ($0.9m) and the apartment complex gets assigned to lot5 ($1.3m). The most optimistic estimate of the cost for the completion of the design is hence $2.2m (0.9 + 1.3).

The estimate is optimistic because it disregards other criteria that might inhibit the design from taking the best possible configuration. By choosing the best possible locations for the

remaining landuses, the estimate, though inaccurate, is optimistic. Interestingly, the use of optimistic estimates for searching the design space is guaranteed to yield optimal solutions. This notion, in the artificial intelligence literature, is called A^* search [Nilsson 80].

2.5 Tradeoffs: Making a choice

The use of search techniques, however sophisticated, cannot help us actually choose among pareto optimal designs. For example, in figure 2-6 solutions B, E and C are all optimal along some dimension. Choosing one of these designs involves making a tradeoff which prefers one dimension over the other. By choosing a design or a set of designs, the designer expresses his preference. This thesis does not examine the utility and preference structures that underlie such decisions. It is left to the user to choose among equally good (pareto-optimal) design alternatives. If the user cannot choose, he can just indicate that he is indifferent among the different pareto optimal solutions. The user always has the last word, this is because, there always will be many subjective criteria that cannot be expressed quantitatively or symbolically, even in the context of the narrow domain chosen.

Many techniques have been developed for handling tradeoffs among multiple objectives [Goicoechea 82]. A problem with these techniques is that they can only handle tradeoffs among objectives. The question is, if we can relax the objectives to find answers, why don't we relax the constraints too? One of the central hypothesis of this thesis is that constraint relaxation coupled with objective relaxation plays an important role in design. Let's examine why.

In Figure 2-6 designs B, C and E are non-dominated. The designer can choose the final design from among these alternatives. If, however, the designer wishes to find better solutions he could do so by exploring new alternatives. A way of finding new alternatives is by relaxing the governing criteria. In our scenario, let us assume the designer decides to relax the slope constraint. In Figure 2-2 (Page 28), lot11 and lot17 were discarded because of steep slopes. As lot17 is steeper than lot11, a marginal relaxation of the slope constraint will bring only lot11 into consideration.

Two new designs F and G are now considered. These new designs are added to the graph shown in Figure 2-6, the new graph is shown in Figure 2-7. The configurations of F and G are shown below.

	Recr-ation	Apart-ments	Hous-ing	Dump site	Ceme-tery	Cost	ENF
F	lot12	lot9	lot11	lot3	lot5	6.5	2
G	lot12	lot7	lot11	lot3	lot5	5.8	9

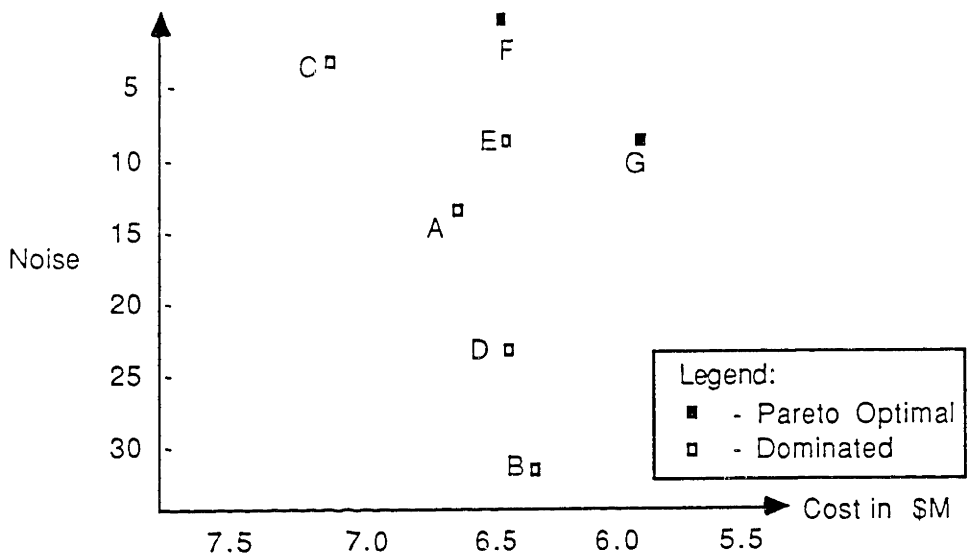

Figure 2-7: Pareto graph after the marginal relaxation of the slope criterion

The pareto set has just changed, F and G together are now pareto-superior to all prior solutions. Relaxing the slope constraint has paid off! We have a potential solution that could be superior to all others.

The relaxation of criteria can help us examine design alternatives that lie outside the original solution space. This exploratory process can uncover designs that turn out to be better than the original designs. The following sections expand on this idea.

2.6 Design Exploration

When a designer, through a design process, finds a suitable solution, he might decide to stop or continue to explore. He might decide to explore in order to convince himself that the solutions he has are really the best, he does not want to miss the opportunity of finding something more interesting. A way of performing such an exploration is by relaxing the criteria on the problem. If criteria relaxation can be automated, then the computer can be used as an alternatives generator. The computer can generate alternatives that may not be obvious to the designer. All alternatives, as far as the computer is concerned, are equally obvious.

How does one make the computer relax constraints? Traditionally speaking, constraints are expressed as predicates that either evaluate to true or false. For example, the constraint: "*The homes should face south*" can either be satisfied or not. If we are interested in relaxations of the above constraint, we need some way of dealing with situations where the constraint is "almost" satisfied. It is not necessary that the homes face exactly south, slight deviations from south is acceptable. In order to allow such forms of relaxations, the constraints are converted into objectives. For example, the above constraint may be replaced with the following objective: "*Minimize the deviation of the homes from the Southern direction*". With this conversion, it now becomes possible to talk about the constraint being partially satisfied.

Earlier we saw how a design problem's specifications consists of constraints and objectives. We now suggest that all the constraints be converted into an objective form. This means that all the design specifications are now objectives, some of which were derived from constraints. To make the distinction clear, the set of all the original objectives and converted constraints are called ***criteria***. A criterion is expressed in the form of an objective and its origins might lie in some original objective or constraint.

2.7 Finding a New Problem

After having determined the cost of designs F and G to be $6.5M and $5.8M respectively, our designer has now uncovered a new problem: Designs F and G place the single family homes on lot11 after relaxing the slope constraint. This means that extra cost is going to be incurred in making homes on steeper slope. It is estimated that this action might cost some $0.9 million more. After adding $0.9M to F and G, a new graph is plotted (figure 2-8). The new pareto set consists of F, C, E and B, making E, B and C pareto optimal once again. By recognizing the existence of a new problem, the designer has to reconsider choices that were dominated earlier.

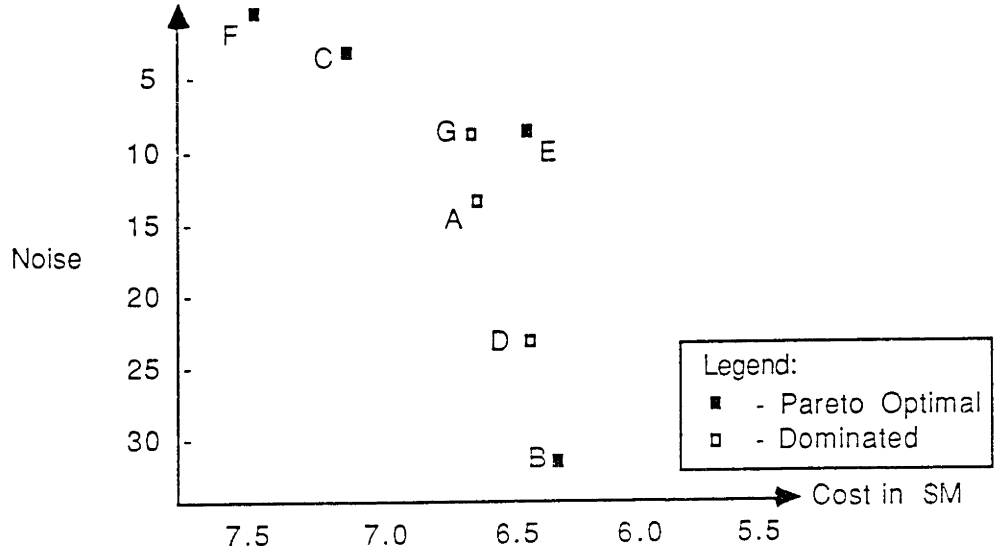

Figure 2-8: Building homes on steeper slopes is going to cost more

2.8 New Criteria

Wait! our designer has just changed his view about the now, pareto inferior, design G. In design G, the single family home complex is placed on a high slope that overlooks a wooded valley with a lake. These conditions have caught his fancy. He now introduces a new criterion:

8. The view from the homes should be nice and panoramic

All of a sudden, this new criterion is introduced as an additional dimension to the pareto space. Our designer is now excited about G, he likes the idea of having a good view.

What happened here? When our designer suddenly took a fancy to having the homes on a hillside, he was perhaps reminded of some favorable past experience. For example, he may have realized that good view can yield higher revenues form the sale of the homes. The introduction of a new criteria in this fashion is called ***criteria emergence***. Criteria can emerge *during* the process of design. This is an important characteristic of design.

2.9 Emerging new Criteria from Precedents

While working on a design, a designer might recognize some part of the design as either having a problem or providing an opportunity. Such a recognition may occur when a pattern in the design triggers some past experience about the corresponding problem or opportunity. When this happens, the designer might introduce a new criterion based on the experience he retrieves. This behavior is emulated in the computer by reasoning from a library of past experiences called ***precedents***.

Precedent#4232

location: hometown, 1962

conditions: (home on high slope) AND (home over-looking valley) AND
(valley is heavily wooded and has lakes) AND (home in Seattle)

effect: (very beautiful view)

explanation:

(very beautiful view) BECAUSE
((valley is beautiful) AND (home over-looking valley))

(valley is beautiful) BECAUSE
(valley is wooded and has lakes)

Consider a precedent (precedent#4232) about the designer's childhood home that had a nice view because it was high on a mountain slope overlooking a scenic valley. The **conditions** is a set of clauses that were true during the precedent experience. A clause is a statement such as: *(home on*

high slope). The **effect** is a set of clauses which denote the direct effects of the observed conditions. The **explanation** shows how the effect is associated to the initial conditions. It is expressed as a set of relations among clauses. In the example above, the first relation says that the view is beautiful because the valley is beautiful and because the home is overlooking the valley. The second one explains why the valley is considered beautiful. The purpose of the explanation will be addressed later in this chapter.

The above precedent can be used to identify a new criterion. For example, if the program encounters a situation that matches the conditions of the above precedent, the complete precedent is retrieved from memory and the corresponding effect: *(very beautiful view)* is inferred. The program can also emerge new criteria by reasoning analogically from precedents. The program then introduces quality of view as a new criterion to evaluate designs with. Precedents can also recognize problems. For example, if the system finds that some design alternative matches an unfavorable precedent, then a corresponding criterion is emerged.

In this way, triggered precedents can lead to new criteria that are either favorable or unfavorable to the design alternative in question. The initial set of criteria is also drawn from precedents. In fact, all criteria have their roots in precedents. As one goes about designing some artifact, one draws upon one's precedent knowledge during the process. After having completed a design task, one would have used possibly, thousands of such precedents.

In our example, we had two types of criteria: (1) criteria identified prior to designing (***prior-criteria***) and, (2) ***emergent-criteria***[4]. These two types of criteria are all drawn from precedents. A question that one may ask in this context is: *"If all criteria come from precedents, then why is it not possible to come up with all the criteria prior to designing? It's all in the designer's head anyway!"* The reason criteria emerge is because humans have so many precedents stored in memory that, often precedents do not trigger unless an appropriate pattern is encountered

[4]A third class is criteria identified in the post-design phase. These criteria are realized by observing the performance of the designed artifact in the real world.

and/or the person is given ample time to "sleep over" the problem. Prior-criteria and emergent-criteria all come from the same source: experiences of the designer as an individual and shared experiences from others in his design culture. As a person designs more and more artifacts of the same type he gains experience. Each time he approaches the problem, the criteria he lays down will be influenced by those emerged in previous experiences. In effect, prior-criteria in use today were probably emergent-criteria in some previously encountered design.

2.10 Design Adaptation and Innovation

Let's assume our hypothetical designer decides to work with design G, as it provides a panoramic view. The design has a problem, however. It involves placing the homes on a steep slope requiring some special action be taken. In this section we will see how our designer might solve the problem by reasoning from past experiences.

Assume that our designer first tries to solve the problem by terracing the hillside. He then realizes that terracing might lead to excessive erosion. Following, is a hypothetical "thinking aloud" protocol of how the designer may continue in his attempt to solve the problem:

"Let me analyze the problem....

The homes at site11 are unfavorable BECAUSE they are on ground which is steeply sloped.

One approach is to get rid of the steepness by terracing the slope. This however, gives raise to erosion problems. What can I do next?

Let's see, the unfavorability is dependent on two clauses: *the homes are on the ground* AND *the ground is steeply sloped.* I tried negating the second clause with terracing but it did not work. What about negating the first clause?

Can I think of some way of not putting the buildings on the ground at all?

.. a long pause ..

Yes! I remember seeing how people in Thailand keep their homes from getting wet during floods. They have devised a way of keeping their homes off the flood prone ground: they put their homes on stilts.

OK. I'll use stilts for my homes too. That seems like a good way of keeping a building off the ground.

Are there any new problems created? No. None, to the best of my knowledge."

There are two precedents used above. Firstly, the designer recognized that placing the house directly on the slope will cause the house to be unfavorably tilted. This could be recognized by a precedent which contains knowledge about how, putting an object on a sloped surface makes the object tilted[5].

[5]Simple precedents such as this one could have been acquired during one's childhood days when one played with blocks and balls

precedent#123

condition: (ground is sloped) AND (block on ground)

effect: (block is tilted)

explanation:

(block is tilted) BECAUSE
(ground is sloped) AND (block on ground)

The above precedent (#123) is used to recognize that the house will be tilted if it is directly placed on the steep slope. The problem faced by the designer can be expressed by the clause *(house is tilted)*. To solve the problem, he has to find some way of making the clause *(not (house is tilted))* true. Let's assume that no relevant precedent is found. An approach to solving the problem under these conditions is to find the causes of the original problem. The idea is that, *if you cannot address a problem directly, try addressing its causes*. In our example, the causes of *(house is tilted)* are *((house on ground)* and *(ground is steep))*. The problem may now be solved by finding precedents that can help achieve either *(not (house on ground))* or *(not (ground is steep))*. Let us assume that the only precedent for the *(not(ground is steep))* is terracing and that the terracing option has already been rejected. Alternatively, the requirement that *(not(house on ground))* triggers precedent #623 which is about how villagers in Thailand use stilts to protect their homes from flooding.

The precedent#623 is about the use of stilts for avoiding flooding. But flooding is not our problem here, our problem is: *(house on ground)*. However, if you examine the explanation of the precedent, you will find that one of the subgoals of precedents is to achieve *(not(house on ground))*. The associated action of the subgoal is *(action: house on stilts)*, which can be used to solve the steepness problem. Note that the purpose for using stilts in the precedent is very different from the purpose in the application. The program's ability to draw such analogies help it solve design problems by drawing knowledge from a diverse set of precedents.

precedent #623

location: Thailand countryside

condition1: (ground is flooded) and (house on ground)

effect1: (house is water-logged) and (house is unfavorable)

action: (action: house on stilts) and (action: stilts on ground)

effect2: (not(house is water-logged))

explanation:

(not (house is water-logged)) BECAUSE
((not (house on ground)) and (ground is flooded))

(not (house on ground)) BECAUSE
((action: house on stilts) and (house on ground))

The rest of the dissertation provides theoretical and implementation details about the various issues touched upon in this scenario.

2.11 Summary and Conclusions

2.11.1 Summary

This chapter presented some of the major ideas of the thesis with the aid of a hypothetical scenario. The scenario was about landscape design. It followed the steps taken by a hypothetical designer as he goes about solving a design problem. In the process of describing the designer's steps, the scenario introduced the actual techniques used in the implementation.

Here is a list of the key issues the scenario presented:

- Certain design problems, such as landscape planning, can be represented as **configuration problems**.
- **Search** can be used as a means of solving discrete design problems.
- Design problems which have **multiple criteria** often require that **tradeoffs** be made among criteria, where relevant tradeoffs are identified using a **pareto optimality** based evaluator.

- Optimal designs can be found by using **optimistic estimates** for the values of the objectives. This idea is a multi-objective variation of the standard A^* search algorithm.
- **Exploration** is an important part of design and that exploration in a space is the product of systematic **constraint relaxation.**
- **New criteria** can emerge during the design process. Such criteria can come from the past experiences of the designer.
- **Past experiences** can be represented as data structures called precedents. Typically, a precedent is represented as having: (1) a description of the conditions, (2) the effects of the conditions, and (3) an explanation of how the conditions cause the effects.
- Precedents can be used to solve design problems both directly and **analogically.** This is based on a technique which tries to address the causes of a problem if the problem cannot be solved directly.
- Finally, the idea that criteria-relaxation based exploration, coupled with **knowledge based adaptation** can lead to **innovative** designs.

2.11.2 Conclusions

In the scenario we sketched the three major points of this thesis through the use of an extended example. First, we saw how design can be represented as a configuration problem consisting of objects and criteria. Second, we showed how criteria relaxation can be used to generate alternatives that could, serendipitously, lead to better solutions. For example, the scenario showed how a designer relaxed a slope constraint to find a solution that provided a very good view. Third, we saw how precedents can be used to adapt designs. Finally, we argued that innovative design can be achieved by using a combination of the following two techniques: criteria relaxation and precedent-based adaptation.

In many respects, this chapter has covered all the major points of the thesis. The next four chapters provide details about the ideas presented in the scenario and show how they can be implemented in a computer program. Chapters three and four cover the representation of configuration problems. Chapter five covers ideas relating to design exploration through criteria relaxation. Finally, Chapter six discusses precedent-based reasoning.

Chapter 3

The Design Problem

Designing is an intellectual activity aimed at producing the description of an artifact that meets a given set of criteria which express desired properties and functional characteristics.

In this chapter we will examine how design can be formulated as a problem. The formulation involves two major steps: choosing a technique with which to represent the artifact, and choosing a technique for representing the criteria which will be used to evaluate generated solutions.

This chapter comprises of four sections. The first Section (3.1) presents the use of representation languages for design domains. Section 3.2 shows how a class of designs can be represented as a labeling problem. Section 3.3 emphasizes the use of constraints in design problems posed as labeling tasks. Section 3.4 describes the importance of optimization in design and shows how a design task can be represented as a consistent labeling optimization problem.

3.1 Design Representation

In order to design and to talk about design, it is important to be able to represent and communicate the design. The most popular method form of representation is a two dimensional sketch. An architect's sketch of a building, an engineer's drawing of a turret lathe and a musician's composition of a symphony are examples of two dimensional (2D) representations.

3.1.1 Layout Languages for Artifact Representation

A 2D representation, at the basic level, is composed of lines and points. At a slightly higher level, a 2D representation can be viewed as a collection of graphical primitives such as rectangles, circles, arcs, curves etc. At an even higher level, the 2D representation is a set of relationships among depicted objects such as tables, chairs, fasteners, walls, doors, columns etc. There is a

hierarchy of representations one can use to represent and communicate design. This hierarchy points towards a particular class of design problems known as ***configuration*** design. An artifact is said to be configured, if it is made by combining objects that are chosen from a given finite set of generic components. Where, the final configuration satisfies a given set of criteria expressed either as constraints and objectives. For example, a job-shop schedule can be viewed as a configuration of jobs which satisfies a given set of time and resource constraints. In another example, preparing a financial portfolio can also be viewed as a configuration task where, one assigns monies to different investments such that a given set of constraints and objectives are satisfied. Now for a counter-example: the design of a baroque Venetian vase is not a configuration design because, it is composed of a complex collection of convex and concave surfaces. These surfaces, though mathematically definable, are not drawn from some finite set of primitives. Such a design cannot be said to be configured, rather, it is said to be formed and, consequently, is known as ***formative*** design.

In this dissertation, we will concentrate on simple two-dimensional artifacts. 2D artifacts can be represented by a class of representational formalisms known as layout languages. A layout language is essentially a set of propositions about a design. The propositions represent relationships among objects that make up the artifact. There are several applications of layout languages: equipment layout in a computer facility [Pfefferkorn 75], building layout [Habraken 83, Gross 86], component layout on a VLSI chip [Soukup 81]. An example of a layout and its representation is provided in Figure 3-1.

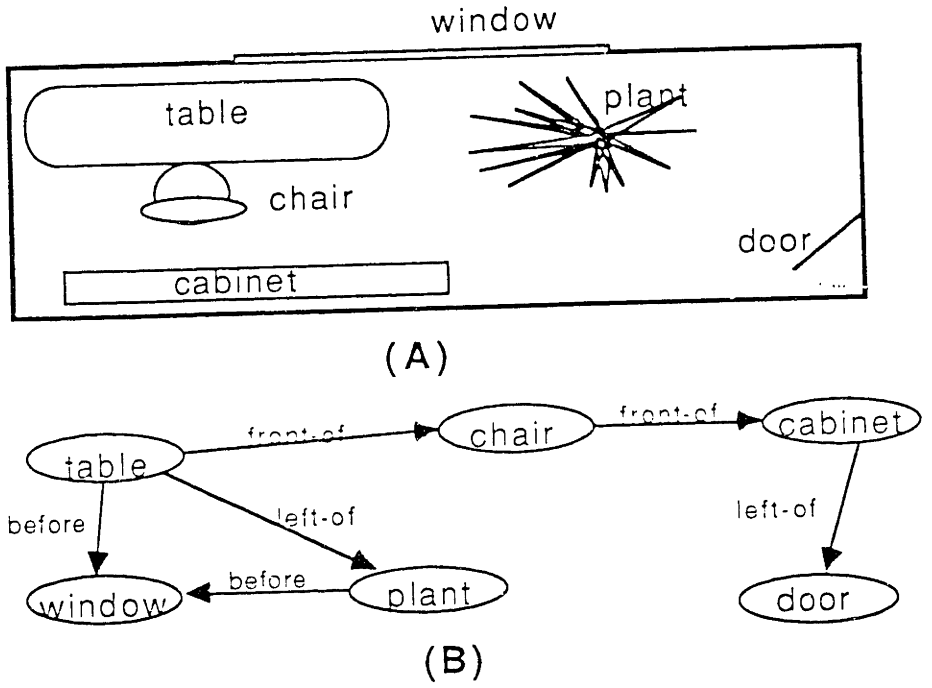

Figure 3-1: Layout language example

Part (a) of the Figure 3-1 is a plan view of a room. Part (b) of the Figure shows a set of relations. Each object (depicted as an oval) contains a description of its own dimensions, its orientation and its x-y coordinates. The relationships depict inter-object distance, relative orientation and direction. The layout language used in the example is ad-hoc - as there is no universal layout language. Researchers usually design layout languages to suit their own particular problems. The purpose for using a layout language is to provide a framework with which designs can be generated and tested for legality. In this dissertation, we will represent artifacts (landscape designs) as a set of assignments of landuses to plots of land. Implicit in this representation is the notion of a standard layout language, as we will be using predicates to test inter-object relations such as distance, visibility, bearing etc. The next section introduces our formalism.

3.2 Design as a Labeling Problem

In Chapter 2, we worked through a landscape design problem which was formulated as a configuration task. In the example, the designer had the task of assigning each of a given set of landuses to a a suitable site, where, the sites were selected from a given set.

This representation is uni-dimensional as, the design involves assigning lots to landuses. When we say: "the apartment-complex is located at lot9", there is no notion of where lot9 is, or what direction the apartment complex is facing. Inter-relationships among landuses are not a part of the representation. In fact, the full design can be expressed as a two column table; with the first column being the landuse's name and the second column being the name of the lot the corresponding landuse is assigned to.

This special type of configuration problem is called a ***labeling problem*** [Nadel 85a]. The landuses are like variables and the lots are like values that can be assigned to the variables. The act of assigning a value to a variable is known as "labeling" the variable.

Labeling problems are amenable to automation. If a computer is given a labeling problem, all it need do is to randomly pick labels (values) for all the variables in order to complete the labeling. Achieving a legal labeling, however, requires the computer to have some way of evaluating the alternatives it generates. One way of doing this is to provide the computer with a set of constraints. These constraints evaluate to either true (1) or false (0). In order to pass evaluation, a design should score 1 with respect to each of the given constraints. This kind of labeling problem is called a ***consistent labeling problem.***

3.3 Design as a Consistent Labeling Problem

In the Artificial Intelligence literature the Consistent Labeling Problem (CLP) is formulated as shown below[6]: (Adapted from [Nadel 85a])

1. There is a set of variables.
2. Each variable has an associated set of values. These sets are known as the **domains** (or **ranges**) of the variables.
3. There is a set of constraints on the values that various combinations of variables may compatibly take on, and
4. The goal is to find a few (or all) ways to assign to each variable, a value from its associated domain in such a way that all constraints are simultaneously satisfied.

A problem in the domain of landscape design can be represented as the following CLP:

1. There is a set of landuses that are to be sited.
2. Each landuse is to be assigned to a lot of land which is chosen from a set of given possible assignments.
3. There is a set of constraints on the lots assigned to the landuses.
4. The problem is to find few (or all) ways of assigning landuses to lots such that the constraints are all simultaneously satisfied.

3.3.1 Background

The Consistent Labeling Problem has received much attention in Artificial Intelligence and Operations Research. The CLP was first introduced by two seminal papers [Walker 60, Golomb and Baumert 65a]. Over the years, the CLP formalism has been successfully applied to a wide variety of problems.

One of the important applications of consistent labeling is in machine vision. A typical problem in machine vision is scene analysis. Given an image of a scene that is comprised of several regular shaped blocks, the computer is supposed to find out how many blocks are in the scene, even if the blocks partially obscure one another. Scene analysis involves three major steps: line/edge

[6] A formal, mathematical formulation of the CLP is provided in Appendix A

detection, line labeling and object interpretation. All three steps have been formulated as CLPs: [Waltz 75, Haralick 80, Barrow & Tenenbaum 76] respectively.

The CLP formalism is also well suited for dealing with graphs. Applications include: the map coloring problem [Nijenhius and Wilf 75], the bin packing problem [Deutsch 66] and crypt arithmetic [Burstall 69].

Another interesting application is relational database retrieval. Some relational calculus operators, such as **join**, can be formulated as a CLP. Consistency-maintenance, redundancy-checking and query-answering in databases have also been treated as CLPs [Grossman 76].

For a complete list of applications and for a formal treatment of CLP, refer to the work of Nadel [Nadel 85a, Nadel 85b, Nadel 85c, Nadel 85d].

3.3.2 Representing Constraints

On page 50, a formulation of the CLP was presented. In this subsection we will, with the aid of an example, examine how a CLP may be represented in a computer.

Consider the following example:

---

1. There are three variables: x_1, x_2 and x_3
2. Their corresponding domains are: $d_{x_1} = \{0, 1\}$, $d_{x_2} = \{1, 2, 3\}$ and $d_{x_3} = \{a, b\}$.
3. The constraints on a labeling are:
   a. $(2x_1)^2 + x_2^2 \quad >= \quad 5$
   b. $e^{-(x_1 + x_2)} \quad <= \quad 0.05$
   c. x_2 and x_3 should be chosen according to the following compatibility matrix: ("1" stands for compatible)

	x2 = 1	x2 = 2	x2 = 3
x3 = a	0	1	0
x3 = b	1	0	1

   d. $\cosh(x_2) > 2$

The constraints on a CLP need not be linear, polynomial or monotonic. Further, constraints

need not be numeric at all. In the CLP above, the third constraint (c), is symbolic and cannot be expressed as one equation. Characteristics such as these, make the CLP different from its close cousin, Integer Programming. Algorithms developed for Integer Programming problems assume linear monotonic constraints. For many problems, particularly design problems, this is a serious limitation. The CLP is able to handle much more complexity than the standard discrete value methods such as Integer Programming. The CLP, however, is not without problems. There is no elegant analytic technique for solving CLPs. Techniques in use today are based on search which is, by nature, a weak method. We are forced to use weak methods because the CLP makes very few assumptions about the constraints with which it works. This is in contrast to certain Integer Programming algorithms which assume linear constraints and are able to exploit the linearity in order to find solutions. Search techniques for solving CLPs are discussed in the next chapter. In this chapter we are concerned only with the formulation.

In the example above, we have a mixture of numeric and symbolic variables. Constraints among numeric variables are easily expressed as equations, however for symbolic variables a different form of predicate is required. We have found compatibility matrices to be a fairly general way of representing relationships among variables. Even some complex design criteria which are non-numeric or non-monotonic are easily expressed as matrices. For example, if one is scheduling manpower in a job-shop situation, one might want to make sure that the workers assigned to the same time slots are compatible and get along (socially) with one another. A matrix could be used to easily represent the constraint. Constraints such as these, on the other hand, could also be formulated as equations but not without unnecessary complexity and loss of elegance.

In this dissertation, all design criteria are expressed as compatibility matrices. This allows us to develop a general solution technique that can be used for problems regardless of the type of variables or criteria. In the example below, we will see how the constraints of the example introduced above can be converted into matrices:

(a) $(2x_1)^2 + x_2^2 >= 5$ Is converted into the following table which relates the compatibility between the domains of x_1 and x_2. This table is called a ***compatibility matrix.***

	x2 = 1	x2 = 2	x2 = 3
x1 = 0	0	0	1
x1 = 1	1	1	1

(b) $e^{-(2x_1 + x_2)} <= 0.05$

	x2 = 1	x2 = 2	x2 = 3
x1 = 0	0	0	1
x1 = 1	1	1	1

(c) This constraint is already in the form of a compatibility matrix:

	x2 = 1	x2 = 2	x2 = 3
x1 = 0	0	1	0
x1 = 1	1	0	1

(d) $\cosh(x_2) > 2.0$

x2 = 1	x2 = 2	x2 = 3
0	1	1

In the example above, there are only three consistent labelings for x_1, x_2 and x_3: (1 2 a), (0 3 b) and (1 3 b) respectively.

Much work has been done on understanding CLPs with constraints that relate only two variables [Mackworth 77]. Such constraints are known as binary constraints. Binary constraints are used because they are easy to represent and easy to use. CLP solution techniques developed for binary constraints are usually applicable to ***n***-ary constraints. An ***n***-ary constraint will be represented as a ***n***-dimensional compatibility matrix.

3.4 Design as a Consistent Labeling Optimization Problem

When we design an artifact, we would like the product to not only satisfy all the constraints, but also to be a good design. A particular design can be said to be better than other designs only if there is some method for comparing designs. The CLP formalism introduced in the previous section has no notion of "best" solution. It's goal is to find a labeling that satisfies all the constraints. Real world design problems, however, are not mere constraint satisfaction problems, they can also involve multiple objectives. Here are some examples of objectives in different design domains:

- **Financial Portfolio design**: minimize risk, maximize diversification
- **Landscape design**: minimize cost, maximize access, minimize noise
- **Machine design**: minimize cost, minimize weight, minimize vibration
- **Diet design**: minimize fat, maximize nutrition, maximize variety
- **Structural Member Design**: minimize weight, minimize cost

Many instances of the above design problems could be formulated as CLPs. With the addition of optimization component, the problem is called a Consistent Labeling Optimization Problem (CLOP).

3.4.1 Formulating a CLOP

In a CLOP:

1. there is a set of variables,
2. each variable has associated with it, a finite set of values known as its domain,
3. there is a set of constraints on the variables, that evaluate to either true or false,
4. there is a set of objectives that evaluate to a finite number, and
5. the goal is to find a few (or all) ways to assign to each variable, a value from its associated domain in such a way that all the constraints are simultaneously satisfied and all the objectives are simultaneously either maximized or minimized.

3.4.2 Representing objectives

In subsection 3.3.2, it was shown how constraints can be represented as compatibility matrices. Here we will see how objectives can be also be represented as compatibility matrices.

Please refer back to the CLP presented in section 3.3.2 (page 51). The problem had three variables and four constraints. The addition of the following two objectives converts the previous CLP into a CLOP:

- **Objective 1.** The first objective is symbolic and, let's assume, is based on reasons that cannot be expressed in words. Preferences, however, are expressible.

  This objective is about x_2 and x_3. Where, x_2 can take one of the values {1 2 3} and x_3 can take one of {a b}. A list of compatibilities among the variables is shown below. The measures of compatibility are: highly compatible, just compatible, moderately compatible and incompatible. The preferences on the values of x_2 and x_3 are:

  - a is highly compatible with values 1 and 2
  - a is just compatible with value 3
  - b is highly compatible with 3 but moderately compatible with 1
  - b is incompatible with 2

  The objective is to maximize the compatibility between the labelings chosen for x_2 and x_3.

- **Objective 2.** Maximize the function: $\log(x_1 + x_2)$

This CLOP example can be solved by first listing down all the solutions to the corresponding CLP and then selecting the optimal. The CLP has three consistent labelings for (x_1, x_2 and x_3) : (1 2 a), (0 3 b) and (1 3 b). Let's determine which one is optimal.

Just as we converted complex constraints into compatibility matrices, the objectives can also be be represented in this fashion. Let's start with the first objective.

Conversion of the first objective: There are four types of relationships among the values in x_2 and x_3's domains: highly-compatible, just-compatible, moderately-compatible and incompatible. These four relations are assigned ranks 1, 2, 3 and 4 respectively. The lower the rank the better. Here is the matrix that expresses the objective:

	x2 = 1	x2 = 2	x2 = 3
x3 = a	1	1	2
x3 = b	3	4	1

The combinations (of x_2 and x_3) which maximize the objective are (1 a), (2 a) and (3 b).

Conversion of the second objective: The second objective is an equation that can be evaluated and expressed as a table:

	x2 = 1	x2 = 2	x2 = 3
x1 = 0	0.0	0.3	0.47
x1 = 1	0.3	0.47	0.6

The value 0.6 is the maximum and should be assigned a rank of 1. All other boxes, being sub-optimal, are ranked with an infinitely large number. The best rank is achieved by the labelings $x_1 = 1$ and $x_2 = 3$. Consequently, the solution (1 3 b) emerges as the only consistent labeling that maximizes both the objectives (achieves a rank = 1) simultaneously, making it a solution to the given CLOP.

3.5 Summary and Conclusions

3.5.1 Summary

Designing is a problem solving process. Formulating a design problem involves two major steps: choosing a technique with which to represent the design, and choosing a technique for representing the criteria which will be used to evaluate generated solutions.

Representation. Certain design problems can be viewed as configuration tasks. An artifact is said to be configured if it is made by combining objects selected from a given finite set of generic components. The designer's task is to find a configuration that satisfies a given set of criteria. A landscape design problem is a configuration task which can be formulated as a labeling problem. A labeling task involves assigning values to a given set of variables which represent the design. The value assigned to a particular variable has to be selected from a pre-defined set associated with that variable.

Evaluation. The evaluation of landscape designs is based on a set of criteria comprising of constraints and objectives. A labeling problem with constraints is called a consistent labeling problem (CLP), and a CLP with objectives is called a consistent labeling optimization problem (CLOP).

As design problems can involve symbolic or numeric variables, and as the governing constraints and objectives can be non-linear, non-monotonic and even non-continuous, we need an elegant way of representing the constraints and objectives. We have found compatibility matrices to be a fairly general way of representing relationships among variables. Even some complex design criteria which are non-numeric or non-monotonic are easily expressed as matrices. For example, if one is scheduling manpower in a job-shop situation, one might want to make sure that the workers assigned to the same time slots are compatible and get along (socially) with one another. A matrix could be used to easily represent the constraint.

In this dissertation, all design criteria are expressed as compatibility matrices. This allows us to develop a general solution technique that can be used for problems regardless of the type of variables or criteria.

3.5.2 Conclusions

In the CLOP example described above, we found only one consistent labeling which maximized both the objectives simultaneously. The criteria used in the example were carefully selected to yield this final solution. This, however, is not possible in real world problems. In practice, there can be many objectives all preferring different solutions. Under these circumstances choosing any one solution requires that different objectives be traded off against one another. This raises several questions: How are tradeoffs identified? What techniques can we use to find optimal solutions when there are multiple criteria? Can such techniques yield global optimal solutions or should we settle for local optima? The following chapter addresses these questions, among others, and presents a search-based solution technique for consistent labeling optimization problems.

Chapter 4

Design Synthesis

In the previous chapter we discussed the formulation and representation of **labeling**, **consistent-labeling** (CLP) and **consistent-labeling-optimization** (CLOP) problems. In this chapter we will examine solution techniques for the these three types of problems.

Labeling problems are solved by heuristic search methods. Search is a slow and inefficient method of finding solutions. Much of the research in this field is directed towards finding clever methods of reducing the search space. This chapter presents one such method for solving CLOPs.

The chapter begins with an introduction to the notion of search and its relation to the state space (Section 4.1). The next section (4.2) introduces an example CLP and shows how a simple depth-first search can be used to find a solution. In the next section (4.3), the example is converted into a CLOP by introducing objectives into the formulation. An algorithm for solving CLOPs is proposed. This algorithm, called ***Pareto Optimal-A****, is a modified form of the standard A^* algorithm and can be used for finding global optimal solutions to problems with multiple-objectives.

4.1 Search as Enumeration

Search techniques can be used to enumerate points in the state space of all possible combinations. In our implementation, solutions are generated by performing a tree-like search. For example, if we had three variables x1, x2 and x3 with domains {1 2}, {a b c} and {x y} respectively, labelings in the state space can be generated as shown in Figure 4-1. The leaves of the tree represent labelings. There are twelve labelings in total. Had there been some constraints on the solutions, then the consistent labelings will be from among these twelve solutions.

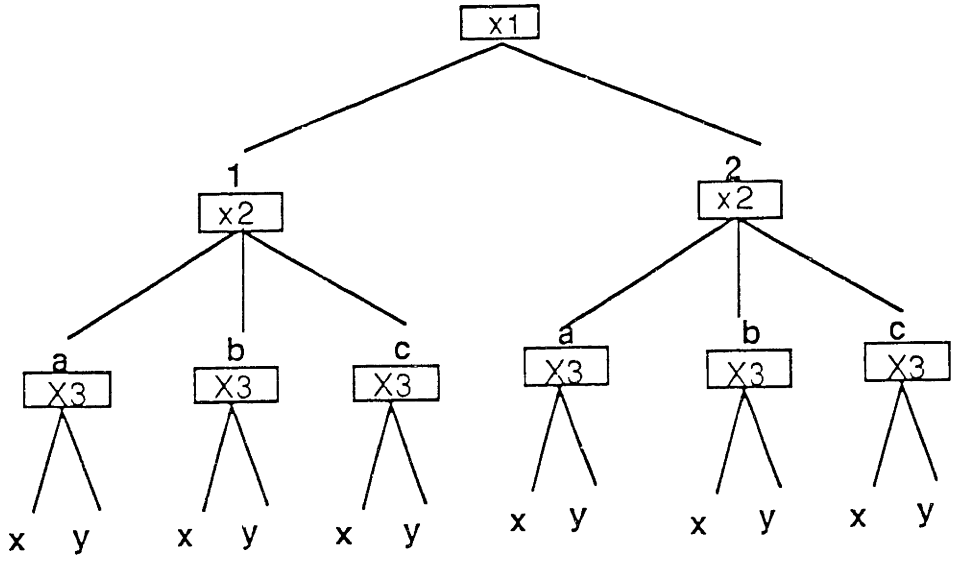

Figure 4-1: The state space

4.2 Consistent Labeling and Pruning

A way of generating consistent labelings is to use the *generate and test* paradigm [Newell 69, Lindsay 80]. This technique has two components: a generator and a tester. The generator continuously produces labelings and the tester checks them for consistency. When the first consistent labeling is found, the process stops.

Another approach is to search for solutions in a tree like fashion, checking for consistency during the synthesis process rather than at the end. We shall use this approach. To illustrate the use of search, let us now work through an example. The following subsection introduces an example CLP. The example is a simplified version of the problem presented in Chapter 2. Treat this new example independent of the one presented in that chapter. This new example will be used through the rest of the dissertation.

4.2.1 The Example Problem Statement

There are three facilities to be sited: a dumpsite, an apartment-complex and a single-family housing complex. The set of possible labelings for the three landuses are shown below:

variable	domain (list of lot numbers)
apts	(5 7 9 10 11 12 17)
housing	(5 9 10 11 12 17)
dump	(3 5 7 9 10 11 12 17)

The corresponding map is shown in Figure 4-2,

There are four constraints:

Constraint 1: The slope of the lot should be less than or equal to 8% for all three landuses. The actual % slopes are shown below:

lots:	3	5	7	9	10	11	12	17
slope:	2%	3%	2%	4%	6%	10%	7%	14%

Constraint 2: The soil conditions should be good. The soil conditions of the different lots is shown in the table below:

lots:	3	5	7	9	10	11	12	17
soil:	1	1	1	1	2	1	3	1

Legend: 1 = good, 2 = moderate, 3 = poor

Constraint 3: The total cost of all acquired land should be less than $35Million. The costs (in millions of dollars) of the lots are:

lots:	3	5	7	9	10	11	12	17
cost($M):	10	13	9	16	17	10	14	8

Constraint 4: The total Excess Noise Factor (ENF) should be less than 15. The ENF is the total of the excess noise at each of the facilities. The excess noise for a particular facility is the difference between the acceptable noise for the facility and the actual noise of the lot it is sited at. The acceptable Noise Factors (NF) for the apts, housing and dump are 20NF, 18NF and 50NF respectively. The noise levels at each of the sites are:

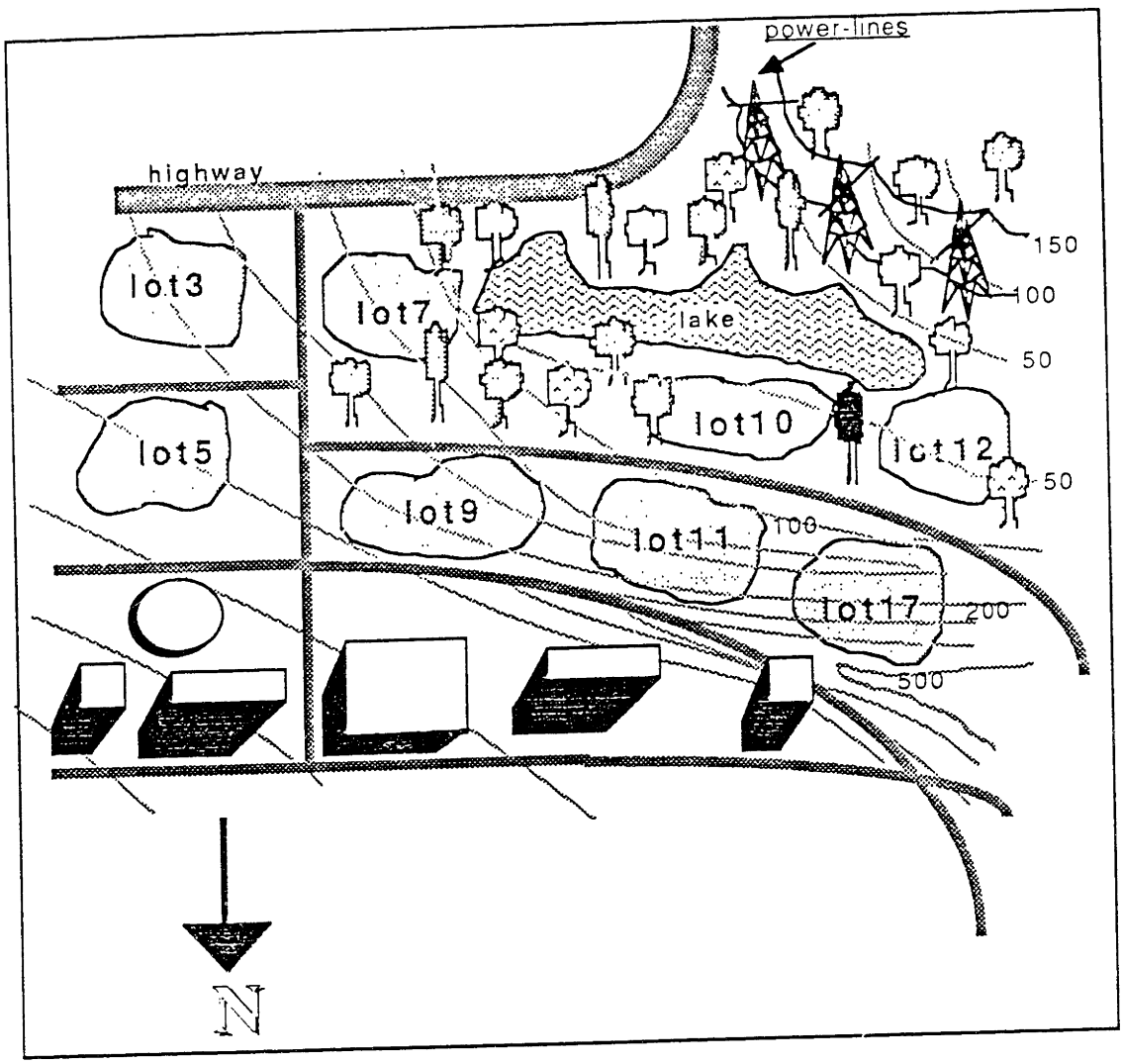

Figure 4-2: Example map

lots:	3	5	7	9	10	11	12	17
NF:	45	30	32	20	22	20	22	20

The goal is to find a consistent labeling over all four constraints.

4.2.2 Searching for a Solution

Let us start solving the above problem using a ***depth-first*** strategy [Golomb and Baumert 65b]. The search is carried out by successively choosing variables and expanding their domains into branches. The first variable is "apts" and the expanded tree is shown in Figure 4-3.

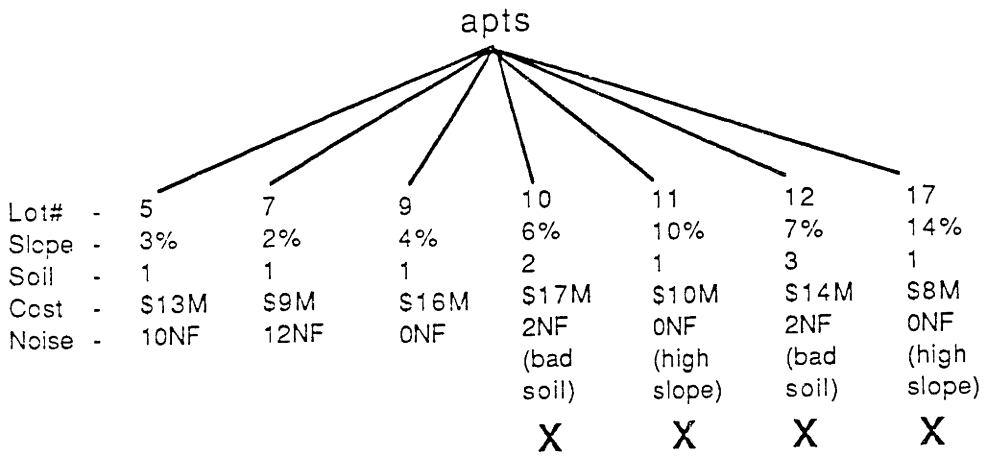

Figure 4-3: Variable "apts" expanded

At the end of each branch, there are five numbers. The first number is the lot number for placing the apartments (apts), the second is the % slope, the third is the rank of the soil type, the fourth is the total cost and the last is the ENF. After laying out these numbers, a constraints check is performed. In the figure, X's are marked under the branches that do not satisfy the four constraints. In other words, they represent inconsistent labelings and are *pruned* off the search tree.

This process of branching and pruning is continued till a complete solution is found. The final solution is shown in Figure 4-4. The consistent labeling found is: (apts = 7, housing = 9 and dump = 3).

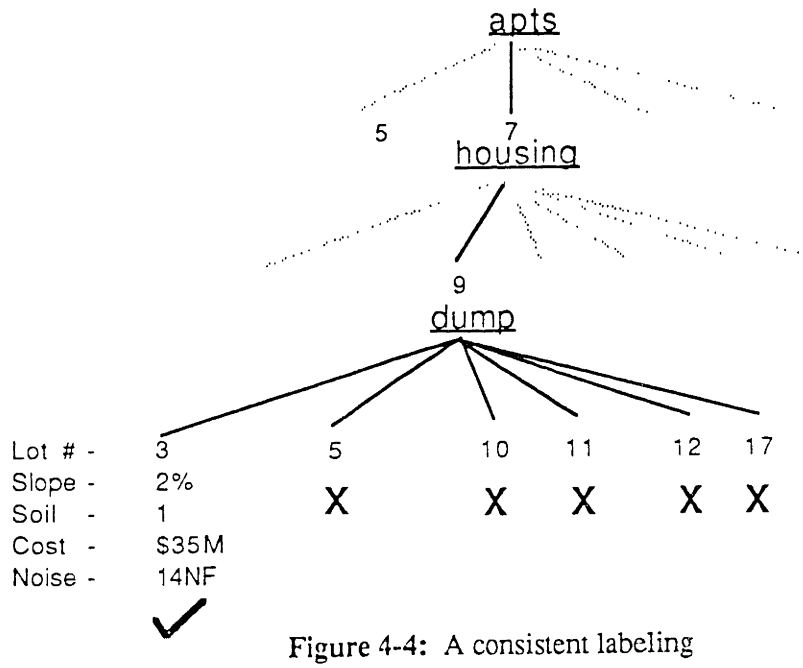

Figure 4-4: A consistent labeling

This example shows how designs can be synthesized using a simple search technique. There are other more sophisticated methods[7] for handling constraints and constraint satisfaction problems. In the research presented here, it has been our final aim to study design exploration and innovation and consequently we have used a simple search technique which adequately serves our purposes. In the context of our research goals it was felt that pursuing sophisticated, efficiency improving techniques was not going to qualitatively change our results.

4.3 Looking for Optimal Consistent Labelings: the CLOP

In this section we will see how optimal consistent labelings are found. Once again, we are going to use a search technique. This time, however, we are going to perform a best-first and not a depth-first search. The example CLP introduced in section 4.2.1 from page 60 will now be

[7]Such methods include: dependency-directed backtracking [Stallman & Sussman 77], constraint-posting [Stefik 80, Navinchandra 86], generalized backtracking [Golomb and Baumert 65b], forward-checking [Nadel 85d].

converted into a CLOP. The revised problem statement is shown below. The revision involves converting the cost and noise constraints into objectives. The other two constraints are left unchanged.

4.3.1 The Revised Problem Statement

As before, there are three facilities to be sited: a dumpsite, an apartment-complex and a single-family housing complex. The set of possible labelings for the three landuses are shown below:

variable	domain (list of lot numbers)
apts	(5 7 9 10 11 12 17)
housing	(5 9 10 11 12 17)
dump	(3 5 7 9 10 11 12 17)

The first two constraints are unchanged:

Constraint 1: The slope of the lot should be less than or equal to 8% for all three landuses. The actual % slopes of the are shown below:

lots:	3	5	7	9	10	11	12	17
slope:	2%	3%	2%	4%	6%	10%	7%	14%

Constraint 2: The soil condition should be good.

lots:	3	5	7	9	10	11	12	17
soil:	1	1	1	1	2	1	3	1

Legend: 1 = good, 2 = moderate, 3 = poor

The cost and noise constraints are now converted into objectives:

Objective 1: The total cost should be minimized. The costs (in millions of dollars) of the lots are:

lots:	3	5	7	9	10	11	12	17
cost($M):	10	13	9	16	17	10	14	8

Objective 2: The ENF should be minimized. The acceptable Noise Factors (NF) for the apts, housing and dump are 20NF, 18NF and 50NF respectively. The noise levels at each of the sites are:

lots:	3	5	7	9	10	11	12	17
NF:	45	30	32	20	22	20	22	20

The goal is to find a consistent labeling that simultaneously minimizes both objectives.

4.3.2 Searching for an optimal solution

The best search algorithm for finding optimal solutions is A^* [Hart, Nilsson & Raphael 68]. The A^* algorithm is defined for problems with a single objective function. The CLOP, however, is a multi-objective optimization problem which involves finding the non-dominated set of complete designs. In this section we will see how the A^* algorithm can be extended to find optimal solutions using pareto-optimality[8] as a measure of goodness. This extended algorithm is called ***Pareto Optimal-A**** (***PO-A****).

The standard A^* is a best-first algorithm where the "goodness" of a node, $f^*(n)$ is the sum of the actual cost of reaching that node $g(n)$ and the optimistic estimated cost of reaching the solution from that node, $h^*(n)$. The first complete solution A^* finds, is guaranteed to be optimal. The ***PO-A**** algorithm uses pareto optimality to determine the "goodness" of a node. All pareto optimal nodes are expanded at each stage of the search (unless the user chooses otherwise). The first complete solution found is pareto optimal over the state space of solutions. The algorithm can be continued till all pareto optimal solutions are generated. The admissibility of ***PO-A**** is based on using optimistic, heuristic estimates of the values for each of the criteria in the CLOP. This means that *a particular node will be **PO-A**** admissible over the entire search tree if the values of the criteria that govern it are individually A^* admissible.* This condition is necessary and sufficient for obtaining solutions which are pareto optimal over the entire state space. A proof of this result is provided in Appendix A.

We will now work through an example to illustrate the ***PO-A**** algorithm. The process starts

[8]We shall use the phrases pareto optimal and non-dominated interchangeably.

by expanding the domain of any one of the variables. Once again, let us start with the "apartments". The resulting tree is shown in Figure 4-5. The technique used for calculating the ENF and the cost (shown in the figure) is based on an admissible heuristic. Let's see how this is done.

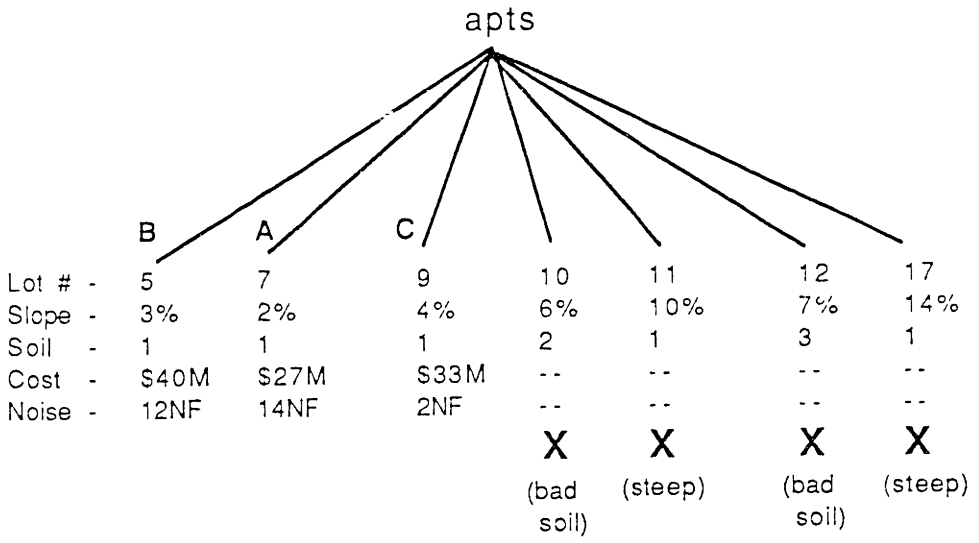

Figure 4-5: Expanding "apts"

In the previous section we calculated the cost of a node as the total cost incurred till that node. We used this cost to prune the tree and search forward. As this cost is only partial, it often happens that the search goes down paths that eventually lead to dead ends. This problem could be eliminated if one could accurately determine the actual cost of completing a given partial design. As this determination is not possible, we can reduce the problem by using an estimate of the cost to completion. The estimated total cost of a particular branch/node (n) is composed of two parts: $g(n)$ and $h^*(n)$. $g(n)$ is the total cost incurred till the node, and $h^*(n)$ is an optimistic estimate of how much more might be incurred in achieving the goal. The total cost $f^*(n)$ is given by:

$$f^*(n) = g(n) + h^*(n)$$

Let us now see how to calculate the $f^*(n)$ of the node "A" in Figure 4-5 (page 66). The cost of choosing lot 7 is $9M (a given). This is the $g(n)$ of the branch. At this stage there are two more landuses yet to be sited: housing and dumpsite. Their domains are {5 9 10 11 12 17} and {3 5 9 10 11 12 17} and their corresponding costs are {13M 16M 17M 10M 14M 8M} and {10M 13M 16M

17M 10M 14M 8M}, respectively. This means that the housing and the dumpsite have to be sited at a lot chosen from their domains and the corresponding cost of the lot will be incurred. We, however, do not know exactly which sites will be chosen for the two remaining landuses; consequently, the remaining cost, $h^*(n)$ must be estimated. In order to calculate $h^*(n)$ we make the assumption that the housing and the dumpsite will be located at the cheapest of the available sites[9], and that no additional costs will be incurred in doing so. In our example, $h^*(n)$ is calculated by assuming that the housing is placed on the cheapest site: lot 17 and the dump is placed on the second cheapest site: lot 3.

This makes $h^*(n) = (8 + 10) = \$18M$, and
$f^*(n) = g(n) + h^*(n) = 9 + 18 = \$27M$

The same is done for the calculation of total Excess Noise Factor (FNF). As CLOPs have finite domains for their variables, it is always possible to use the above heuristic to calculate $h^*(n)$.

Having examined how criteria values are estimated, we can now return to the continuing example. We last expanded variable, "apts" (Figure 4-5) we have to now choose the best branch and expand it further. Branch 9 (C) has the lowest NF but branch 7 (A) has the lowest cost. Which one do we choose? There is a graphical technique to aid in making such choices. This technique is discussed below.

The three consistent labelings are plotted on a graph as shown in Figure 4-6. Solutions A and C are *pareto optimal.* Solution B is dominated by C.

[9]One might ask: "Where did you get the heuristic from?" The answer is simple: Heuristics are heuristic in nature and often do not have sound, provable origins. There is no analytic way of telling whether a given heuristic is either good or bad. A heuristic is good as long as it is in use and no other heuristic refutes it by showing better search efficiencies.

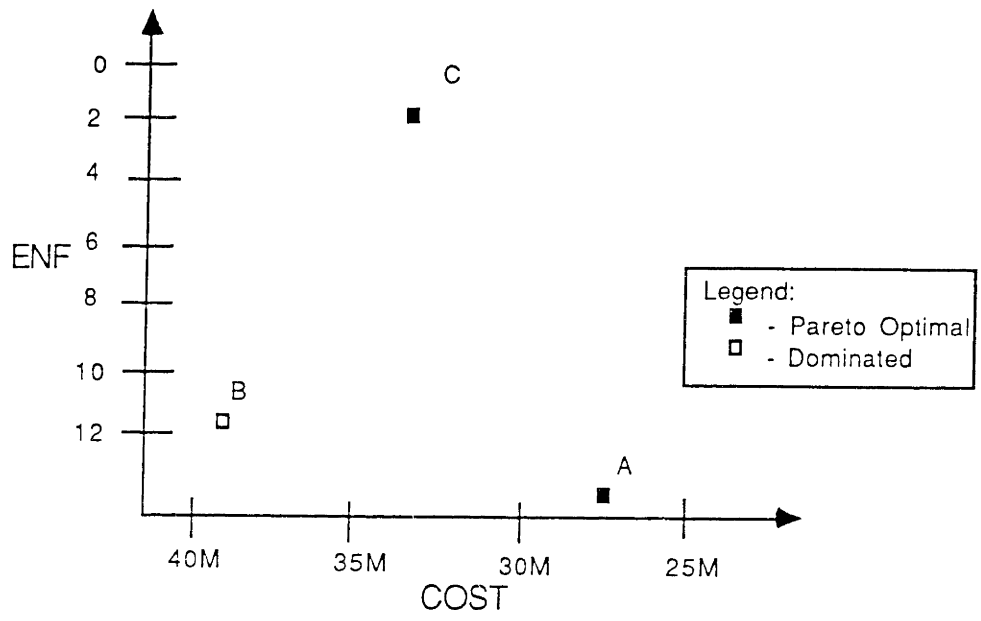

Figure 4-6: Plot of the first three consistent partial solutions.

Having eliminated branch B, we still do not know which of the branches A or C is the best. Which branch should one expand next? Unfortunately, there is no definite way of differentiating among non-dominated solutions, and that's because each non-dominated solution is optimal in its own right (i.e. with respect to at least one of the objectives). Choosing one solution over another requires a value judgment that trades off advantages in one dimension over another. In the current implementation, the computer relies on the human user to make such tradeoffs. If the user does make the tradeoff, then the chosen branch is expanded. If, on the other hand, the user is indifferent or otherwise chooses not to articulate his preferences, the program assumes that all the non-dominated solutions on the pareto surface are optimal and proceeds to expand them. The program has a mode of operation called "automatic" (no user assumed), which expands all pareto optimal points and leaves out all nodes that are completely dominated. Nodes that are dominated are never really pruned off the tree, this is because one might have to return to them if all else fails.

Expanding the pareto set of Figure 4-5, we get the tree shown in Figure 4-7. The pareto optimal solutions are now D, E and F. Let's assume the user chooses D and F for further expansion. This is shown in Figure 4-8.

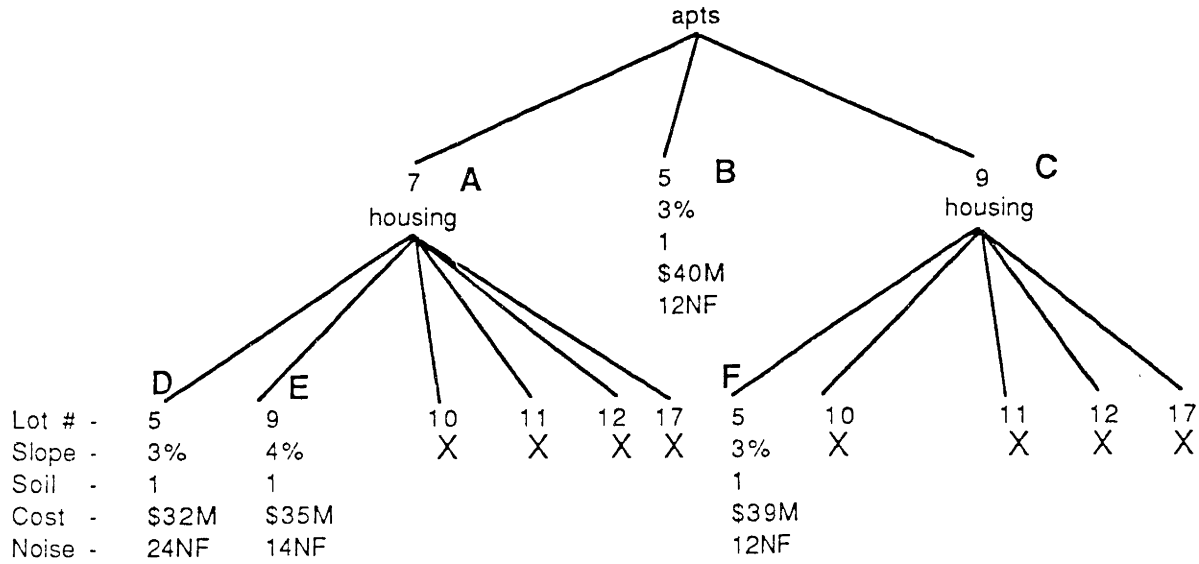

Figure 4-7: Expanding "housing"

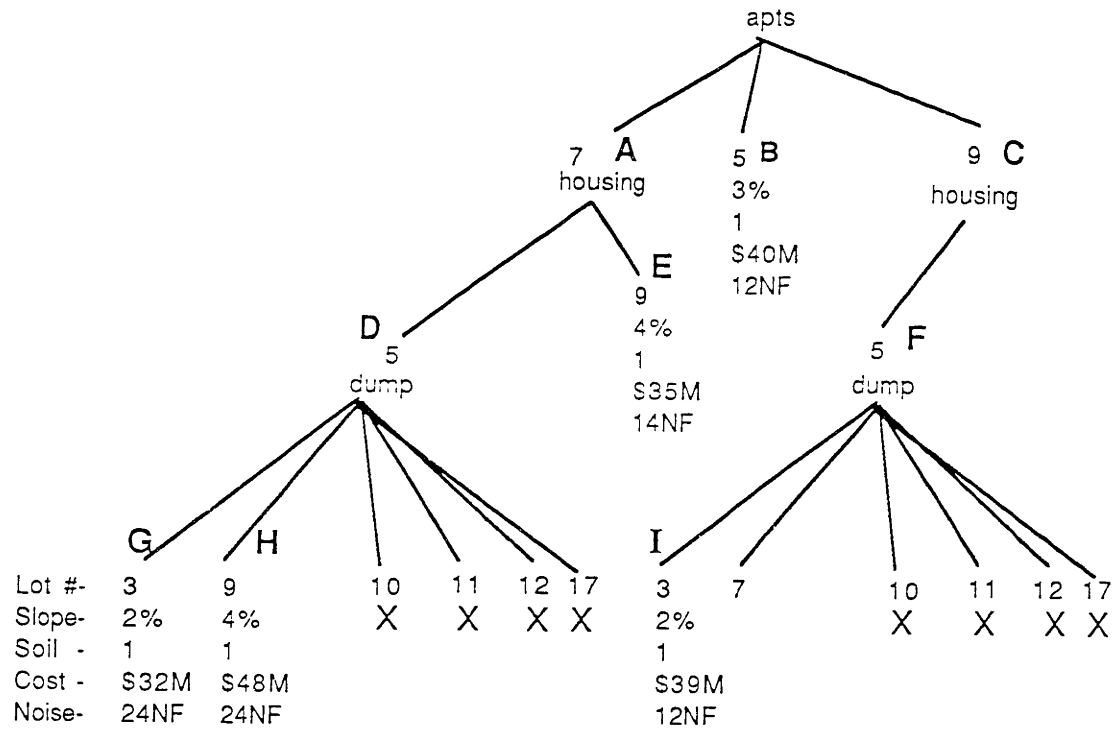

Figure 4-8: Expanding "dump"

Once again, we have several pareto optimal solutions. All the final nodes and unexpanded nodes are plotted on a pareto graph, see Figure 4-9.

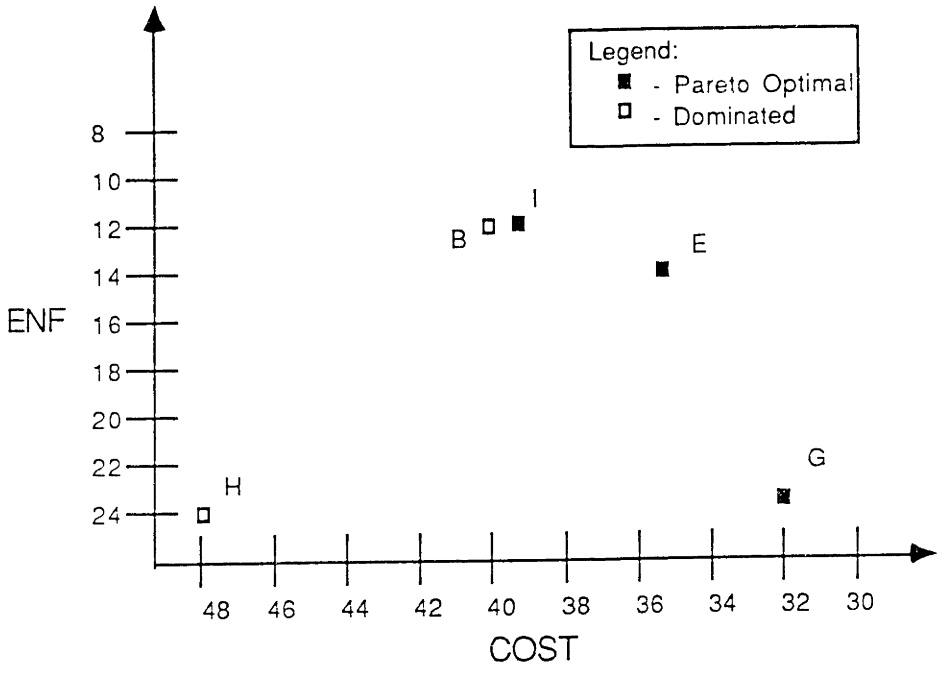

Figure 4-9: Pareto plot showing all partial designs.

Solutions I, E and G are non-dominated. I and G are complete solutions but E is yet to be expanded. Once again, the choice is up to the user. If the user feels that E is worth pursuing, the node is expanded as usual. For now, let's assume the user is interested in saving money and knowing that expanding E will not improve its cost, chooses G as the final design. The *PO-A** algorithm may be terminated at this point, or may be continued in order to find other non-dominated solutions.

4.4 Summary and Conclusions

4.4.1 Summary

Search proceeds in stages. Each variable expansion is a stage in the process of generating complete solutions. A design which has been cast as CLP or CLOP is said to be complete when all its variables have been instantiated. The path leading to each node in the tree (except for the bottom-most nodes) represents a partial design. Each stage of the search process adds more detail (new

instantiations) to the partial designs. In effect, each stage in the search adds detail to the partial designs. Figure 4-10 shows this process. Partial designs are fed into a synthesis module, where details are added. Detailed designs which are incomplete, re-enter the loop and go through the process till completion.

Figure 4-10: The Labeling Process

The process shown in Figure 4-10 is capable of generating all labelings. If a consistency check is added, it becomes a CLP solver (Figure 4-11).

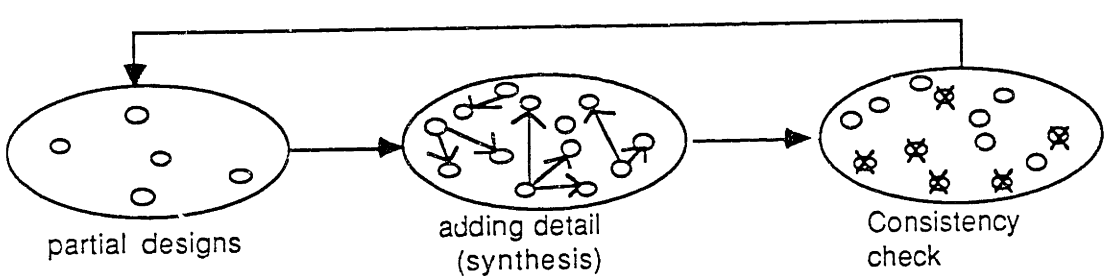

Figure 4-11: The Consistent Labeling Process

After a consistency check, if we use pareto optimality with A^* evaluation we get a CLOP solver. This is shown in Figure 4-12. Note that the consistency check prunes the bad partial designs (indicated by X's) but the pareto optimality check only suppresses pareto inferior solutions (indicated by little boxes). This is called the ***Pareto Optimal-A**** algorithm. The algorithm is very similar to the standard A^* algorithm [Nilsson 71]. A comparison of the two algorithms is shown in Figure 4-13.

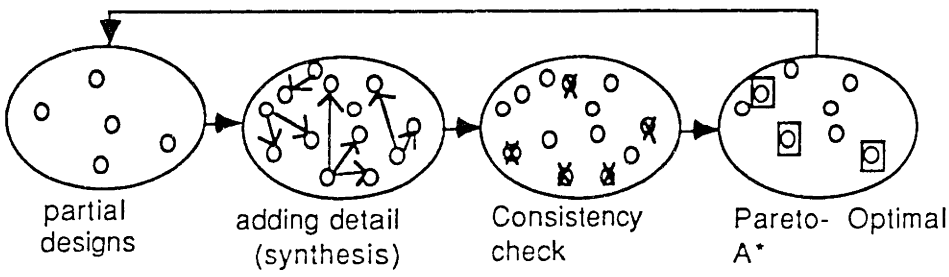

Figure 4-12: The Consistent Labeling Optimization Process

Pareto inferior solutions are not pruned off because they have a chance of becoming pareto optimal as the synthesis proceeds. This is because, as detail is added to designs, the pareto surface either stays unchanged or might slowly migrate towards the origin. The reason the Pareto Surface cannot move away from the origin is because it is based on optimistically estimated values. Adding detail to the partial designs is not going to improve their ratings over the optimistic estimates[10].

4.4.2 Conclusion

In this chapter we seen how A^* search and pareto optimality are used together to generate solutions to Consistent Labeling Optimization Problems. The technique is called *PO-A** (Pareto-Optimal A^*). A formal treatment of this algorithm is provided in Appendix A.

The *PO-A** algorithm is based on dividing the search space into two sets: the dominated and the non-dominated. The algorithm treats all pareto-optimal solutions alike, even if some of them are at the extremes. This behavior is desirable for design exploration. Extreme solutions can sometimes serendipitously lead to the emergence of a new criterion. We would like our program to automatically examine these extreme solutions for hidden opportunities. Furthermore, exploration can also be done by examining solutions that are dominated but happen to be very close to the pareto set. Such solutions can also lead to the emergence of new criteria and new opportunities

[10]A formal proof of this phenomenon is provided in Appendix A.

The A* algorithm:	**The PO-A* algorithm** (criteria emergence not shown):
1 Put the start node 's' on a list, called OPEN, of unexpanded nodes. Calculate f^* and associate its value with node s.	1 Put the start node 's' on a list, called OPEN, of unexpanded nodes. Calculate $f_1^*(s)$, $f_2^*(s)$ ..... $f_n^*(s)$ and associate its value with node s. (Where, $f_q^*(p)$ is an optimistic heuristic estimate of the final value that would be attained by the q^{th} criterion given the p^{th} node.)
2 If OPEN is empty, exit with failure; no solution exists.	2 If OPEN is empty, exit with failure; no solution exists.
3 Select from OPEN a node 'i' at which f^* is minimum. If several nodes qualify, choose a goal node if there is one, and otherwise choose among them arbitrarily.	3 Select from OPEN a node 'i' which is non-dominated. If several nodes qualify, choose a goal node if there is one, and otherwise choose among them arbitrarily.
4 Remove node 'i' from OPEN and place it on a list, called CLOSED, of expanded nodes.	4 Remove node 'i' from OPEN and place it on a list, called CLOSED, of expanded nodes.
5 IF 'i' is a complete solution, exit with success; a solution has been found.	5 IF 'i' is a complete solution, exit with success; a solution has been found.
6 Expand node 'i', creating nodes for all its successors. For each successor node 'j' of 'i'	6 Expand node 'i', creating nodes for all its successors. For each successor node 'j' of 'i'
a. Calculate $f^*(j)$	a. Calculate $f_1^*(j)$, $f_2^*(j)$ ..... $f_n^*(j)$
b. Add 'j' to OPEN and attach a pointer from 'j' back to its predecessor. (in order to trace back a solution path once a goal node is found).	b. Add 'j' to OPEN and attach a pointer from 'j' back to its predecessor. (in order to trace back a solution path once a goal node is found).
7 Go to (2)	7 Go to (2)

Figure 4-13: Comparison of A^* and $PO\text{-}A^*$

different from those available in the pareto set. We would like our program to also examine solutions just below the pareto surface.

The process of exploration, its role in design innovation, and implementation details is the topic of the next chapter.

Chapter 5

Design Exploration

Exploration is the basis of innovation. All designers explore alternatives and play "what-if" games in order to find innovative solutions to problems. This chapter introduces exploration and discusses its role in design.

Exploration is the process of generating and evaluating design alternatives that normally would not have been considered. Normal synthesis processes, such as search, are aimed at only considering those design alternatives that are within the solution space defined by the governing criteria. An exploratory process, on the other hand, tries to generate a wide variety of alternatives from outside the solution space, some of which might unearth new opportunities or solve design problems in unexpected ways. In this chapter we will propose a way of exploring design alternatives. The proposed technique explores outside the solution space by relaxing the criteria that bound the space. Alternatives that are generated by the above process are then examined for opportunities. This is done by reasoning from a database of past experiences about other design situations. If a design alternative has some interesting characteristics, a corresponding new criterion is added to the problem. The emergence of criteria during the design process can sometimes change the focus of problem solving and could lead to a solution completely different from what a standard synthesis technique (such as ***PO-A****) would produce.

Exploration is not guaranteed to always produce better designs. It is possible that the best design is within the normal solution space, furthermore, new favorable criteria could emerge while examining designs from within the solution space. This, however, does not happen too often. Exploration is important for innovative design as it is a way of breaking one's mindset about the types of solutions to a given problem.

This chapter is composed of three sections. The first (5.1) presents a scenario which shows

how exploration could lead to new possibilities and the emergence of new criteria. Section 5.2 presents an algorithm for exploration. The section ends with is discussed of how new criteria can emerge during the design process. Finally, the ideas presented in the chapter are summarized in perspective (Section 5.3).

5.1 A Scenario

Consider the following scenario[11]:

You have just been admitted to MIT as a graduate student. Your first assignment, and probably the toughest at MIT, is to find housing in the city of Cambridge.

You go to a realtor and place the following criteria in front of him:

- Constraint1: Cost of the studio should be less than or equal to $400.
- Constraint2: The place should be within a 15 minutes walk to school.
- Objective1: I want the largest studio.
- Objective2: I want the quietest studio you have.

The realtor has only four studios that satisfy both the constraints. After some discussion, you decide to go look at the studios. The first studio, on River Street, is large but close to a noisy highway. The second, on Harvard Street, is a small studio tucked away in a large apartment complex which is located in a relatively quiet neighborhood. The third, on Main Street, is noisy and small. Finally, the fourth, on Brattle St, is relatively quiet and is of medium size. You have a clear tradeoff among the apartments on River St. Brattle St. and Harvard St. - a tradeoff between noise and size. The Main St. apartment is inferior to all the other choices. The plot in Figure 5-1 illustrates this situation graphically. The figure shows a curve drawn through the non-dominated points. This curve is called the ***pareto curve***.

---

[11]The text of the scenario is bounded by two horizontal bars

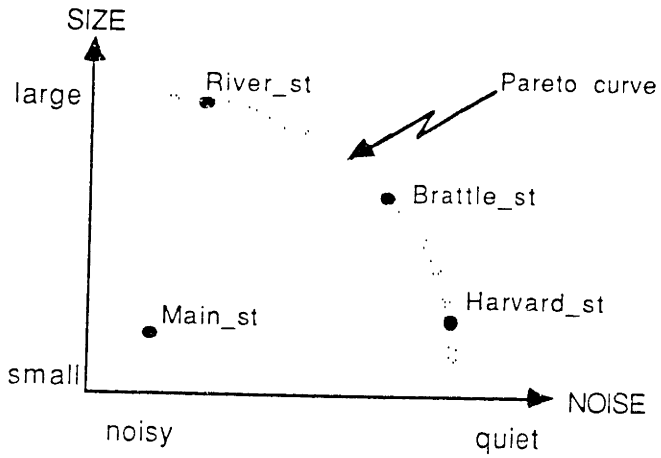

Figure 5-1: Trading-off noise and and size

Having seen the three houses, you are now confused and indecisive. The realtor then starts taking you to a fifth studio:

"Where are you taking me now?"

"I have a fifth place that I'd like to show you?"

"Is it less than $400 a month?"

"Come with me, and check this place out"

"How much does it cost? I need to know!"

"Why don't we look at the place first. If you don't like it you don't have to take it."

"Oh well, if you insist."

The fifth house, on Brookline street, is big and relatively quiet, however, it is dominated by the Harvard street apartment. This is shown in Figure 5-2.

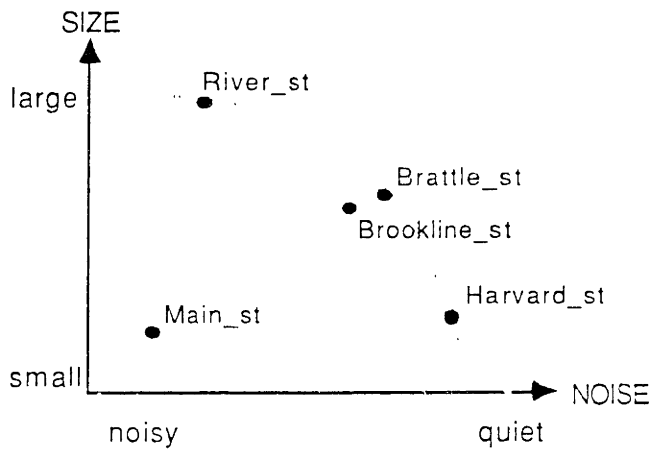

Figure 5-2: The four studios

The conversation continues:

"Do you like the place?"

"Yes! But how much?"

"Do you have a car?"

"No.."

"This is a good place sir, it's 10 minutes to school from here and you are only 3 minutes from the Subway station. The other four places are very far from public transportation."

"Hmm... I don't have a car, I really do need to be close to public transportation."

"The central square bus station is only three minutes away, a large Grocery Store is also within four to five minutes from here. You like this place sir?"

"Yes! This place is really convenient I had not thought of these other things in my original list of criteria. So, how much?"

"Four twenty five"

Things have changed, there is a new objective to consider. You now want a studio with minimum distance to public transportation. This new objective can be represented as a third dimension.

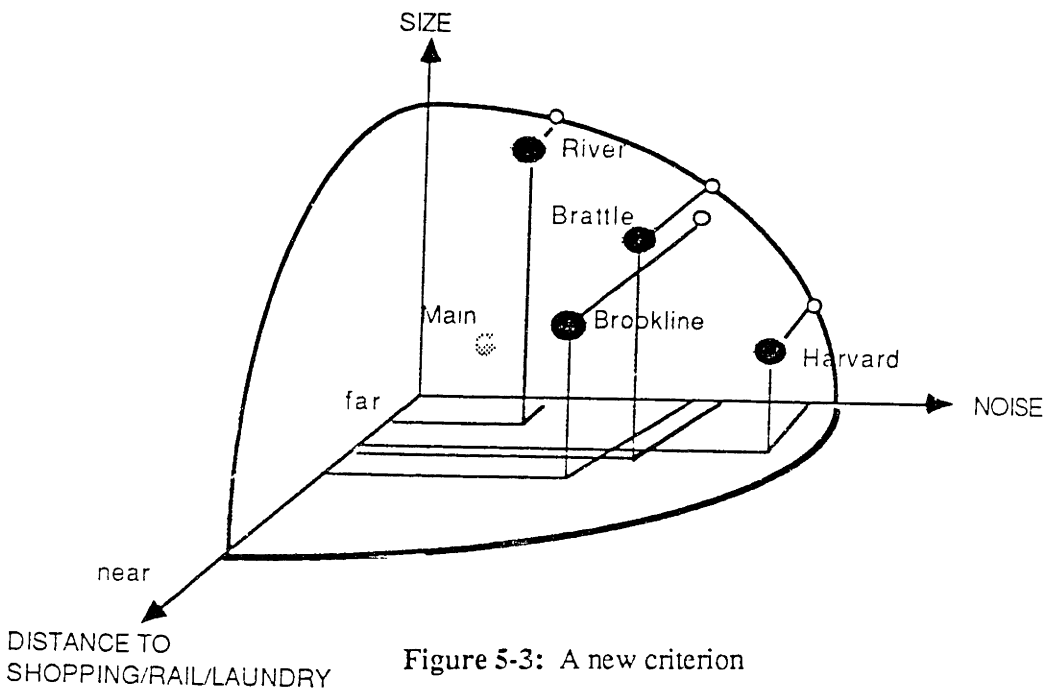

Figure 5-3: A new criterion

Let us assume that only Brookline_st fairs well on the third objective. It goes from being pareto inferior to pareto optimal. Which of the three studios would you now choose?

In the scenario, relaxing the constraint on rent lead to the consideration of the Brookline St. studio. This studio, turned out to be favorable because it had very easy access to public transportation. A new objective was introduced as a result of seeing the Brookline St. studio.

The same happens in design. When constraints are relaxed, new design alternatives come into consideration. It is possible that some of these new designs evoke, in the designers mind, some new criteria that change the terms of evaluation and choice. The alternatives generated by constraint relaxation serve as cues which might give the designer a new idea. This is the *Emergent Criteria* phenomenon [Tomlin 86].

In the conversation above, notice how the realtor insists that you see the fourth studio. He hopes that the studio might have certain characteristics that may catch your fancy. Note how he avoids questions about the actual price of the fourth studio. He is afraid that, if he tells you the price

up-front, you will discard the alternative immediately. He hopes that you might relax the cost criterion *after* looking at the studio. The program (CYCLOPS) adopts a similar approach.

5.1.1 Exploration in Design: Some Intuitions

In this dissertation, criteria relaxation is proposed as a means of exploring design alternatives. The proposed technique explores outside the normal solution space by relaxing the constraints and objectives that bound the space. In effect, relaxation increases the size of the solution space allowing us to examine designs in the state space which would normally have been pruned off.

There are two types of criteria: constraints and objectives. Both these criteria define the solution space as a subset of the larger state space of possible designs. We can explore the space by relaxing either of these criteria.

Constraint Relaxation. In the previous section the spirit of constraint relaxation was presented with the aid of a hypothetical scenario. The reason for relaxing constraints is that it gives rise to new alternatives, some of which may fortuitously lead to improved designs. For example, a landscape designer who relaxes the constraint that "all homes be on slopes less than 8%" to some higher value (say 10%), makes available to him, plots of land that are between 8% and 10% slope. It is possible, that some of these new alternatives may provide opportunities such as better soil conditions or better view. Relaxation overcomes the artificial precision built into a criterion. When we say that the slope should be less than 8%, it does not mean that lots with slightly higher slopes of say 9% or 10% be completely avoided.

Objective Relaxation. In addition to constraint relaxation, there is another form of relaxation that can produce design alternatives, and that is objective relaxation. Objectives can be relaxed by deliberately pushing the pareto-surface towards the origin. This is shown in Figure 5-4. In Part A it is shown that only a few of a large group of designs are non-dominated and lie on the pareto surface. The curve defines the tradeoff between objectives O1 and O2. The situation depicted in Part A of the figure can be relaxed by deliberately pushing the pareto surface towards the origin. In

effect, all solutions which lay just below the pareto surface are available for consideration. It is in this way that new alternatives can be brought into consideration through objective relaxation. An important property of this technique is that it prefers to surface designs that are very close to being non-dominated.

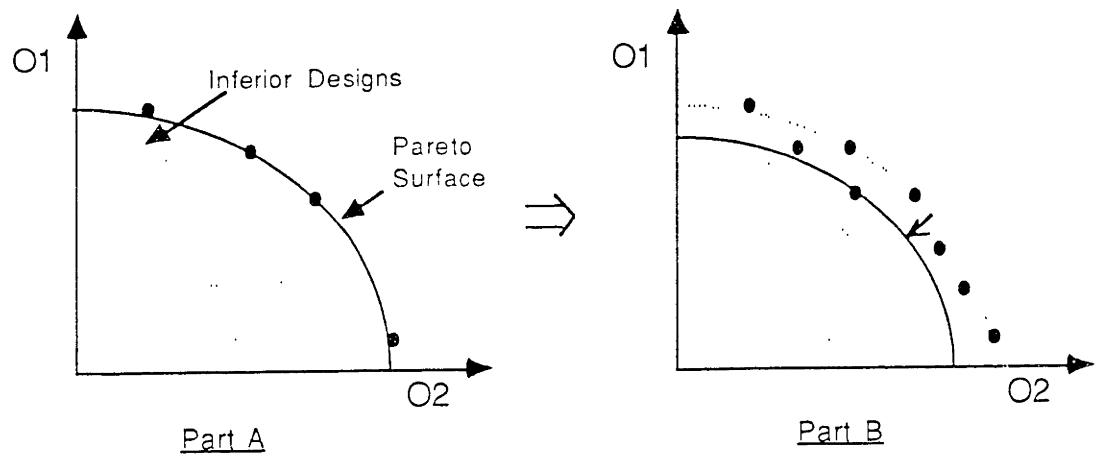

Figure 5-4: Pareto Slip

We now have a notion of how constraint relaxation and objective relaxation can be used to generate design alternatives. The question is, "Are these two relaxation types related? If so, how?" What is their relation to design? How does one decide when to relax constraints and when to relax objectives?

5.1.2 Criteria: Representation and Relaxation

The notion of relaxation is not new. Researchers in Operations Research have developed methods for relaxing objectives. The literature on multi-objective problem solving abounds with techniques for managing and presenting tradeoffs among objectives [Goicoechea 82]. A question this raises is: "If we are amenable to relaxing objectives by trading off among them, then, why are we so rigid about the constraints on the problem?" When tradeoffs are being made, it essentially means that we are in a state of bargaining among different viewpoints. It is not clear why constraints should be treated as sacrosanct and why they should not be included in the bargaining and trading-off process. To address this problem, we developed a simple, unified method for representing,

using, and relaxing constraints and objectives alike. This is done by converting constraints and objectives into a generalized form called the ***criterion***. A criterion is represented as a matrix of ranks. Here are some examples of how constraints and objectives can be converted into criteria.

Relaxed Representation. Consider the objective: "Minimize the cost of the design." This may be rewritten as: "Maximize the closeness of the actual cost to the theoretical minimum." Graphically speaking, the reformulation looks as follows:

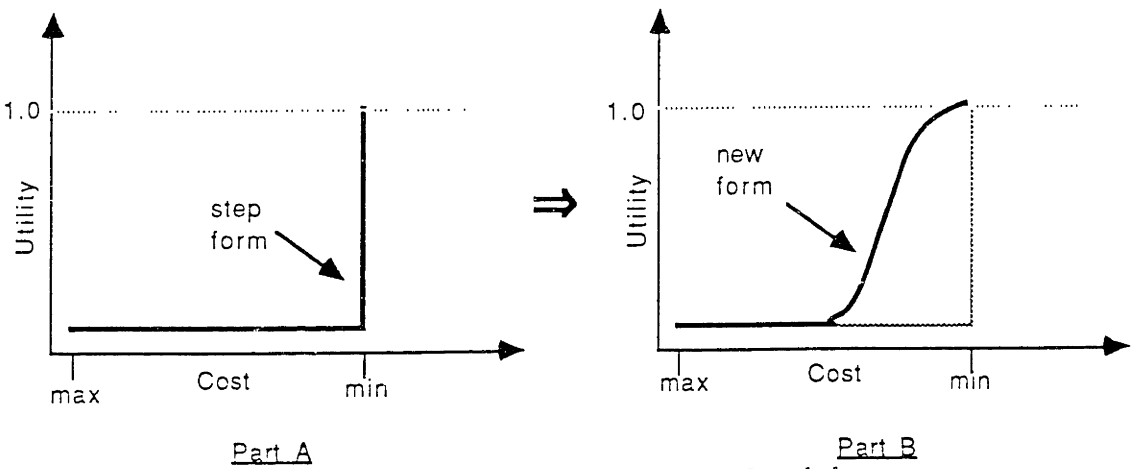

Figure 5-5: From a step change to a relaxed change

Part A shows how we normally specify objectives. An objective is said to be satisfied if and only if its value is at the extremum. In the figure, the utility of obtaining the minimum is unity and any other value, higher than the minimum, has zero utility. This form of representation in a multi-objective problem makes it difficult to find solutions that satisfy all the objectives simultaneously. However, it is possible to find solutions that are in the neighborhood of the theoretical optimal. Such solutions achieve values close to the extremum, with correspondingly lower utilities. This concept is illustrated in Part B of Figure 5-5. The figure shows how a hard step change is converted into a soft, relaxed one. In the current implementation the relaxed form of a criterion is represented as a matrix of ranks. This is done by assigning ranks to ranges of utility.

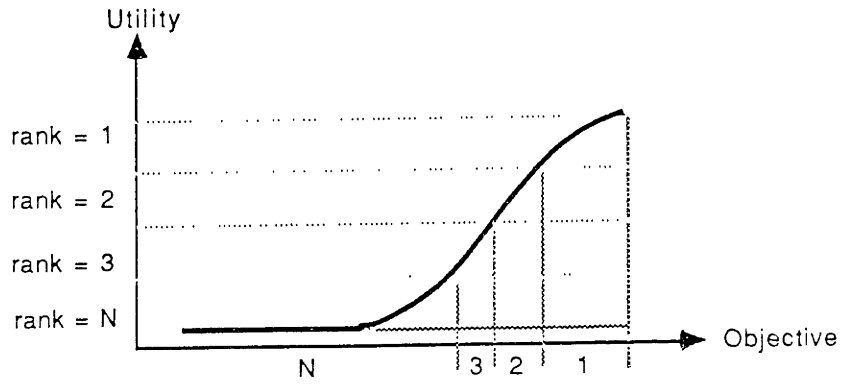

Figure 5-6: Ranks and ranges of utility

In the Figure above, it is shown how regions can be assigned ranks. All designs that fall within a region are given the corresponding rank. The best value obtainable is assigned a rank of 1. The region just next to the best is assigned a rank of 2, the next best region is assigned a rank of 3 and so on. This is carried out till some lower bound is reached. Values lower than the lower bound are assigned rank N. Where, N indicates "Not-to-be-considered". In the program, N is set to a very large number which, in effect, eliminates the region. A modified form of the Figure 5-6 is shown below. The modified form shows how the function takes a sudden step where the rank is equal to N.

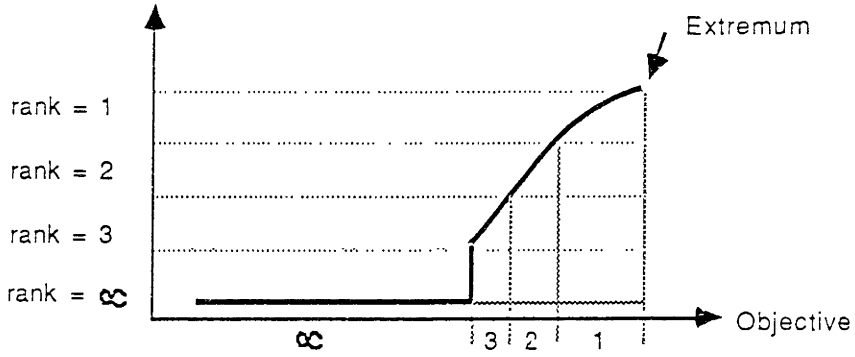

Figure 5-7: The function starts with a step and then smoothes out

The same idea is extendible to constraints. For example, consider the constraint: "Build

homes only on south facing slopes. Re-wording it as a criterion, we get: "Build homes facing a direction that has a bearing as close to the Southern direction as possible." The transformation is shown below:

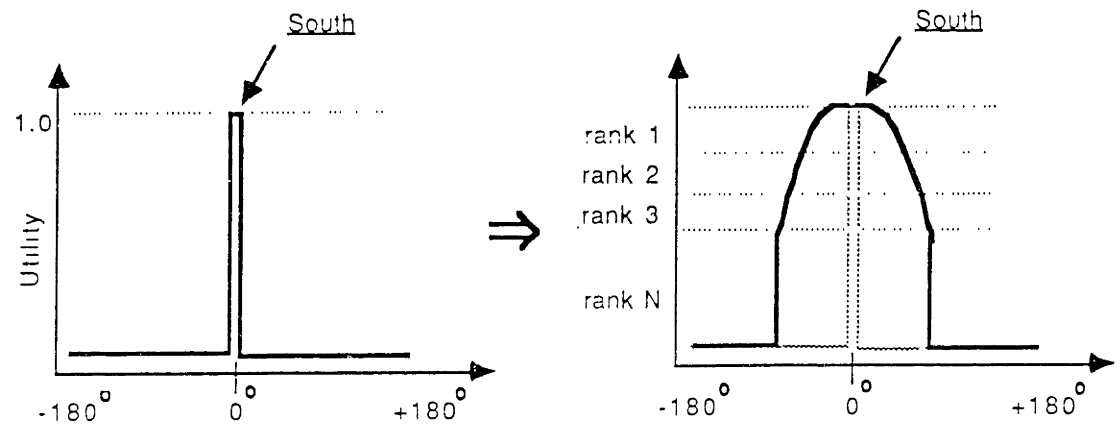

Figure 5-8: Homes facing south

In yet another example, the constraint: "Cost of housing <= $2M" can be changed into a ranked criterion as shown below:

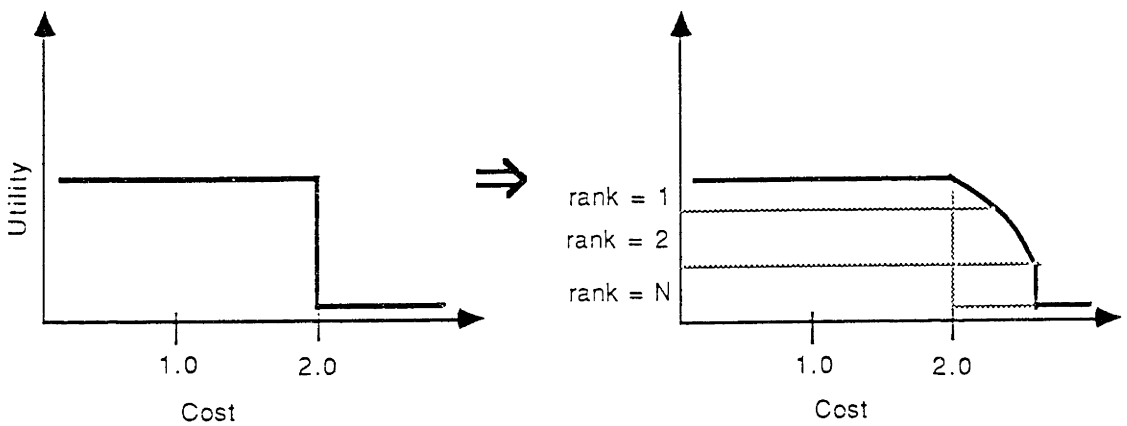

Figure 5-9: Cost of housing

Using this reformulation technique, we shall now return to the landscape example and re-solve the problem using exploratory techniques.

5.2 The Landscape Example, Revisited:

Let us redefine the design problem introduced in subsection 4.3.1 (page 64) in the light of what we have just discussed. In this section, we will convert the constraints and objectives into relaxed criteria.

5.2.1 A reformulation

As before, there are three facilities to be sited: a dumpsite, an apartment-complex and a single-family housing complex. The sets of possible labelings for the three landuses are shown below:

variable	domain (list of lot numbers)
apts	(5 7 9 10 11 12 17)
housing	(5 9 10 11 12 17)
dump	(3 5 7 9 10 11 12 17)

There are four criteria:-

Criterion 1: The slope of the lot should be less than or equal to 8% for all three landuses. The actual % slopes are shown below:

lots:	3	5	7	9	10	11	12	17
slope:	2%	3%	2%	4%	6%	10%	7%	14%

The above values are to be ranked. Any site with slope less than or equal to 8% is assigned a rank of 1. The sites 11 and 17 have to be assigned worse ranks. If we adopt a 5 rank scale, then the best value of 8% is assigned a rank of 1 and the rank 5 is assigned to the worst: 14%. By linear interpolation, the 10% slope is assigned a rank of 2. The ranked matrix is shown below:

lots:	3	5	7	9	10	11	12	17
slope:	1	1	1	1	1	2	1	5

Criterion 2: The soil ranks for each of the landuses should be equal to 1 .

lots:	3	5	7	9	10	11	12	17
soil:	1	1	1	1	2	1	3	1

Criterion 3: The total cost is to be minimized. The costs (in millions of dollars) of the lots are:

lots:	3	5	7	9	10	11	12	17
Cost($M):	10	13	9	16	17	10	14	8

These too can be ranked on a scale of 1 to 5:

lots:	3	5	7	9	10	11	12	17
cost:	2	3	1	4	5	2	4	1

Criterion 4: The ENF[12] should be minimized. The acceptable Noise Factors (NF) for the apts, housing and dump are 20NF, 18NF and 50NF respectively. The noise levels at each of the sites are:

lots:	3	5	7	9	10	11	12	17
NF:	45	30	32	20	22	20	22	20

The differences in the NF of the site and the landuses is tabulated below:

lots:	3	5	7	9	10	11	12	17
apts:	+25	+20	+12	0	+2	0	+2	0
housing:	+27	+12	+14	+2	+4	+2	+4	+2
dump:	-5	-15	-13	-25	-23	-25	-23	-25

The total ENF that is to be minimized is dependent upon positive deviation of the NF of a lot and the acceptable NF of the landuse sited there. The above table shows the differences in NF for all the possible sitings. The best rank of 1 is be assigned to a NF difference of zero or less. The NF differences +25 and above are assigned a rank of N. The worst NF difference of +14 is assigned a rank of 5 and all other ranks are obtained by linear interpolation. The resulting rank matrix is shown below:

[12]As defined before, the ENF is the total of the excess noise at each of the facilities. The excess noise for a particular facility is the difference between the acceptable noise for the facility and the actual noise of the lot it is sited at.

lots:	3	5	7	9	10	11	12	17
apts:	N	3	4	1	2	1	2	1
housing:	N	4	4	2	2	2	2	2
dump:	1	1	1	1	1	1	1	1

The aim is to find a labeling such that the rank attained by each of the given criteria is equal to unity.

5.2.2 Solving the Problem

We shall use *PO-A** to solve the above problem. The search is carried out by choosing variables and expanding their domains into branches. The branches are then evaluated and the pareto optimal set is extracted for the next stage of the search. In order to ensure optimality, the branches are evaluated using optimistic estimates of the costs to completion.

The first variable is "apts" and the expanded tree is shown in the figure below:

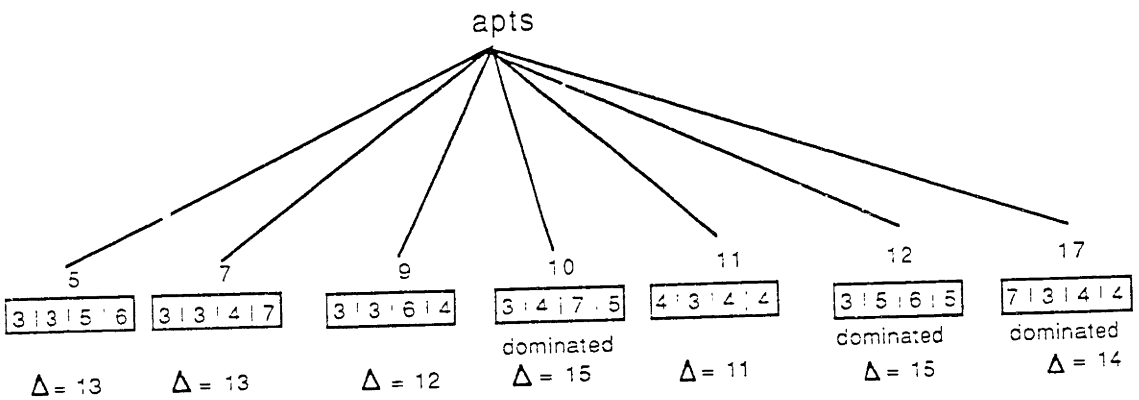

Figure 5-10: Expanding "apts"

At the end of each branch there is a box with four numbers.[13] The four numbers represent

[13]For now, ignore the numbers below the boxes.

the rankings attained by the four criteria: slope, soil, cost and noise. This set of values is called the ***spectrum*** of the partial design. Consider the third branch ("9") for a moment, let's examine how the values in the corresponding spectrum are calculated.

slope: The decision to place "apts" at lot "9" yields a slope rank = 1 (page 85). As we are using ***PO-A**** , we need to estimate (optimistically) the ranks for "housing" and "dump". We can make the assumption that the rest of the facilities will be sited at the best of the available lots: 3 and 5. The estimated slope rank is:

$f^*(n)_{slope} = g(n)_{slope} + h^*(n)_{slope}$

$f^*(9)_{slope} = g(9)_{slope} + h^*(9)_{slope}$

$f^*(9)_{slope} = 1 + (1 + 1) = 3$

The first box on branch "9" is a three.

soil: The soil rank is calculated in the same fashion

cost: The decision to place "apts" at lot "9" yields a cost rank = 4. The best lots available for the dump and housing are 17 and 7.

$f^*(9)_{cost} = g(9)_{cost} + h^*(9)_{cost}$

$f^*(9)_{cost} = 4 + (1 + 1) = 6$

noise: The noise rank is calculated in same way

Having expanded all the branches, the next step is to select the best branch to expanded further. This is done by finding the pareto optimal set. Pareto optimality is now based on four criteria: slope, soil, cost and noise. Comparing the spectra of the branches in Figure 5-10, branches 5, 7, 9 and 11 emerge as the non-dominated set. The next step, according to the ***PO-A**** algorithm, is to expand the non-dominated set of branches and to continue the process till complete solutions are found.

The search process, as described above, has a problem. As the ***PO-A**** algorithm expands the full non-dominated set at each stage of the search, it produces too many alternatives at each stage making it unsuitable for interactive situations. As we wish to have the designer examine alternatives and look for opportunities in them, it is important not to overwhelm him. A way of alleviating this problem is to order the non-dominated set using some measure of "goodness" and to then expand the best, first. The essential idea is to avoid examining the complete non-dominated set all at once, but to examine the set in some order. The question is, what order? "Are some pareto optimal solutions more optimal than others? What other measure can we use?"

5.2.3 Ordering the Non-dominated Set

Ordering of the non-dominated set is based on deviations from the ideal solution. This ordering technique is best explained with the aid of an example. Consider three partial solutions A, B and C which have the following spectra:

```
A ( 2  5  3  4  6  3  2)
B ( 1  4  2  1  5  1  2)
C (78  2 35 45 81 72 11)
```

In the set above, B dominates A. The solution C, which has very poor ranks in its spectrum has one criterion with such a good rank that it cannot be dominated. As a result, B & C are pareto optimal. Clearly, C is an extreme solution, as it has much worse ranks than B on all but one of the criteria. For this reason, it is desirable to expand B before C. It is important to note, however, that we do not want to eliminate C, we only want to look at it a little later as it has only one good rank in its spectrum. The reason for not eliminating C is based on the possibility that it could lead to an interesting solution. We do not want the algorithm to eliminate any such opportunity, how-so-ever remote. The decision to eventually examine C or not, should lie with the user.

The ordering technique we have used is based on a simple measure of deviation of a solution from the best possible. The best spectrum for a partial design is one with all ranks equal to unity. Let's call this the zeroth order optimal design. The second best spectrum is one which has one of it's criteria attaining a rank of two. Let's call this a first order optimality. We can use the order of a design to select designs from sets of pareto-optimal designs. Let's see how this is done:

The spectrum of a design i is indicated by S_i:

$S_i = \{ s_{i1}, s_{i2}, ... s_{in} \}$

Where s_{ij} is the rank attained by the j^{th} criterion in the spectrum S_i . If the total number of criteria is n, then the order of a spectrum (Δ) is given by:

$$\Delta = \sum_{j=1}^{n} w_j (s_{ij} - 1) \qquad (5.1)$$

Where w_j is the weight attached to the j^{th} criterion. In the current implementation all weights are set to unity.

Using Δ for search. We shall now modify the ***PO-A**** algorithm using the measure Δ, for ordering the non-dominated set. The modified algorithm is called the ***Ordered Pareto Optimal-A**** (***OPO-A****). The ***OPO-A**** algorithm uses a threshold value of Δ (denoted by Δ_g) as a means of limiting the search. The search process is similar to ***PO-A****, the difference is that instead of expanding the complete non-dominated set, we now expand only those branches in the non-dominated set which have a Δ less than or equal to the threshold (Δ_g). Before the search starts, Δ_g is set to 0. The search starts by expanding the root node which has a Δ of 0 (most optimistic). The search process continues till the first complete solution is found or till there are no branches with Δ less than or equal to Δ_g. If the process stops with no solutions, Δ_g is incremented and the search restarts from where it last stopped. The process continues till a complete solution is found. The first complete solution found is guaranteed to be pareto optimal (non-dominated) over the entire state space[14].

Using Δ for Criteria Relaxation. Interestingly, the process of increasing the threshold, as described above, serves as a means of performing systematic criteria relaxation! As we accept designs with higher Δ we are, in effect, moving away from the theoretical best and gradually lowering our expectation on the values attainable in the designs' spectra. Considering designs with higher values in their spectra means that the governing criteria are being relaxed. Hence, the process of gradually increasing Δ_g, to bring higher order solutions into consideration, is equivalent to criteria relaxation. This process of increasing the threshold is how CYCLOPS explores new alternatives. In the rest of this section we will work through an example to illustrate the ***OPO-A**** algorithm.

The Example. Having described the ***OPO-A**** algorithm, let us return to our continuing example. Our last step was the branching shown in Figure 5-10 (page 87) where the best Δ was 11. This means that Δ_g will have to be increased to 11 before any new branching can begin. Setting Δ_g to 11 we get:

---

[14]A proof of this result is provided in Appendix A

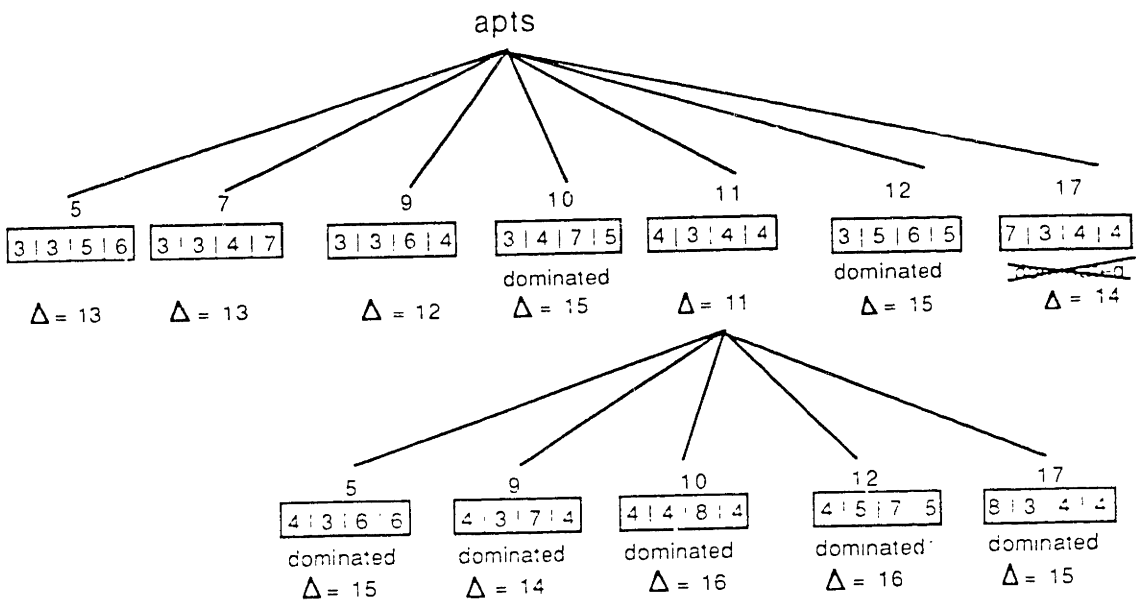

Figure 5-11: Expanding "housing"

The next expansion is to start at a branch with Δ less than or equal to Δ_g. However, there is no such branch. Δ_g has to be increased to 12. We next expand the non-dominated designs which have a Δ less than or equal to 12. The branch "9" in Figure 5-10 (page 87) is now expanded. On expanding this branch the best Δ exceeds Δ_g. Once again, Δ_g is relaxed to 13. Consequently, branches "5", "7" and "12" of Figure 5-10 are expanded. This process is continued for a few steps. Finally, all the expansions (at Δ_g = 14) are shown in Figure 5-12. All partial designs with Δ higher than 14 have been marked with an "X" to indicate that they are not to be considered. Further, all solutions which are non-dominated are marked with a "O".

The figure has two complete optimal designs (i.e. all the variables are instantiated): *A* (apts=9, housing=11, dump=7) and ***B*** (apts=11, housing=9, dump=7). Both the designs are very similar. Their costs are $35M each, the noise is 4NF in both designs and the slope at lot "11" is over the stipulated 8%. Let's see how these designs compare to designs found earlier. In the previous chapter the final design was (apts=7, housing=5, dump=3) (page 69) with a cost of $32M and a total ENF of 24NF. In that chapter, all the designs which used the relatively steeper lot: "11" were eliminated. In the current example, by relaxing the slope constraint, we have a design which is 8.5%

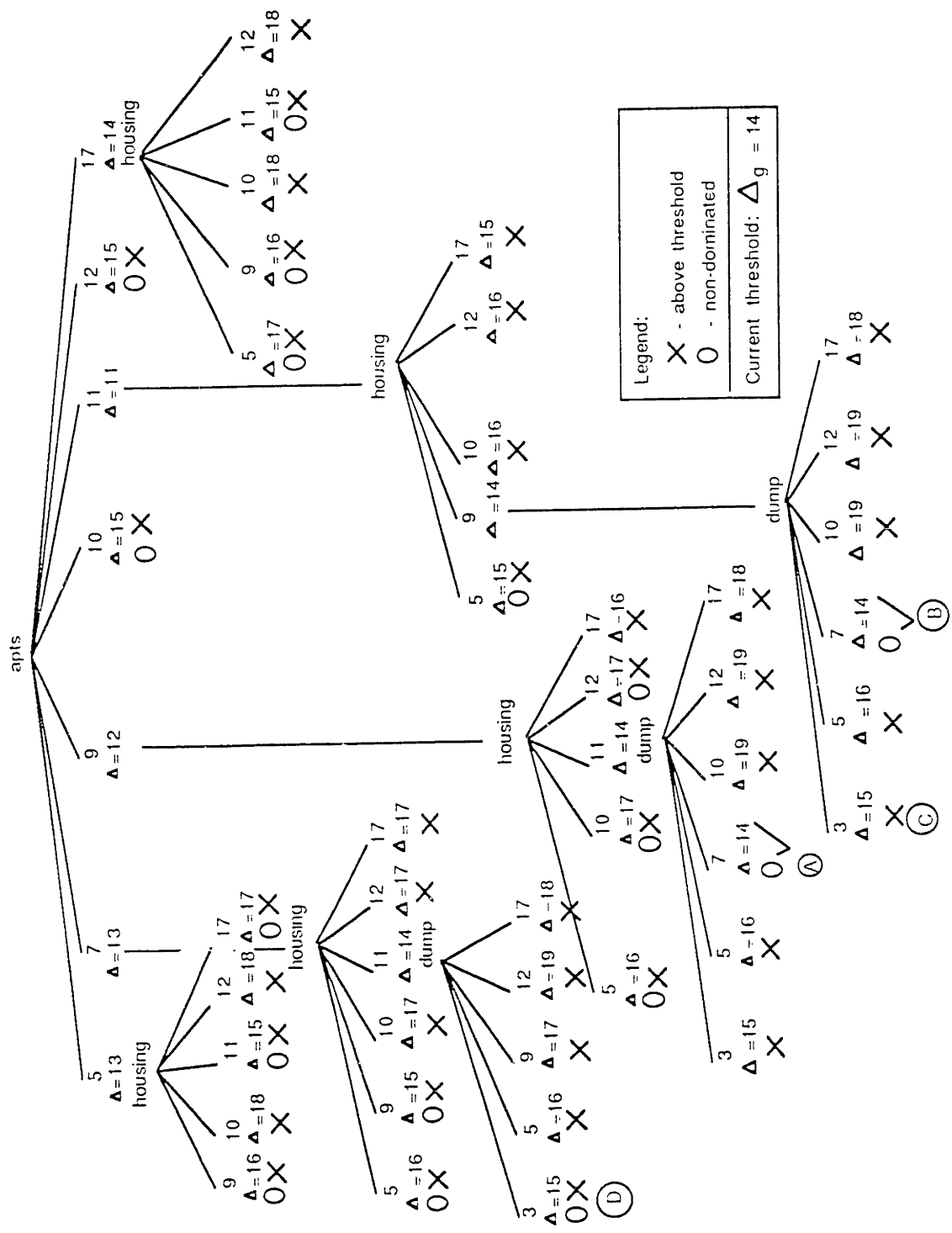

Figure 5-12: Tree at $\Delta = 14$

more expensive but is 6 times less noisy than the solution found without relaxation. Relaxing the slope constraint has given us an interesting design. The relaxation process, however, need not stop here. One need not stop relaxing criteria as soon as a solution is found. It is always possible that further relaxation might yield yet another good design. This is illustrated in the next two subsections.

5.2.4 Relax, and hope for the best

Assume for a moment, that we decide to relax further by increasing Δ_g arbitrarily to 15. This action surfaces many new designs. Among the designs surfaced, there are two new complete designs to consider. They are C and D (Figure 5-12). D is pareto optimal while C is dominated. The set of non-dominated complete solutions at $\Delta_g = 15$ is shown below:

	apts	housing	dump	Δ	$	NF
A	9	11	7	14	35	2
B	11	9	7	14	35	2
D	7	11	3	15	29	14

One of the new designs, D is actually $3M cheaper than the previous best design G (Figure 4-8, Page 69) which had a total cost of $32M and had an ENF of 24.

This has been an example of how constraint relaxation can unearth potentially good designs even after some solutions have been found[15]. Furthermore, criteria relaxation could generate alternatives which turn out to be interesting in some unexpected way. This is possible even though the new alternative compromises the specified criteria. A new alternative may have certain characteristics which causes the observer to be reminded of some favorable past experience, making the observed design favorable. The sudden recognition of a design as being favorable (or unfavorable) is equivalent to adding a new criterion to the design problem. This phenomenon is termed ***criteria emergence***.

---

[15]This phenomenon is repeated several times in the trace of the program CYCLOPS (refer Appendix B).

5.2.5 Criteria Emergence

Criteria can emerge during the design process. For example, while considering a given design alternative, a landscape designer might recognize that the design provides an excellent view. Assuming he had not considered view as a criterion earlier, he might decide to make "good view" one of his considerations for selecting a design. Such re-specifications can often change the focus of the designing process.

Let us now return to the continuing example and see how new criteria can emerge during design. In our last step, we were examining three designs A, B and D. Now assume that Δ_g is relaxed to 16 and that a new design, E is generated. The designs which are currently being considered are shown below:

	apts	housing	dump	Δ	$	NF	slope	soil
A	9	11	7	14	35	2	high	OK
D	7	11	3	15	29	14	high	OK
E	7	9	3	16	35	14	OK	OK

Assume that these alternatives are given to a designer for final selection. As none of the alternatives are dominated, a tradeoff has to be made. Now assume that the designer, while examining design D is suddenly reminded of his parent's home that was located high on a hill side that overlooked a beautiful wooded valley. He realizes that placing the houses on lot 11 will give rise to a situation similar to the one he had enjoyed as a youngster (Refer Figure 4-2, Page 61). A new criterion "good-view" is introduced into the basis. This new criterion adds a new dimension to the pareto optimality graph[16], which in effect makes all designs with "good-view" non-dominated. It is possible that some of these new design alternatives might lead to the final design. Had the designer not been confronted with this alternative (regardless of whether this particular alternative was ultimately selected or rejected) the new criterion "good-view" would not have emerged and a good opportunity would have been lost.

---

[16]We have seen this phenomenon before: Figure 5-3 on page 79.

Criteria emergence also plays an important role in exploration. It helps in the recognition of interesting designs among the alternatives generated as a result of exploration. Further, the new criteria which emerge during design need not always be favorable, they can also correspond to problems in a design. The recognition of problems plays an equally important role in exploration as it helps prune the space.

The question is, where do new criteria come from? One source of new criteria is the experience of the designer. When a designer recognizes an alternative as having some problem or opportunity, he might do so by recalling some past experience (precedent) which matches the characteristics of the observed design. In this context, we can use the ***OPO-A**** algorithm to generate alternatives which can then be either rejected or accepted by the user as when he recognizes a problem or an opportunity. A drawback of this approach is that it places too much burden on the user. An earlier implementation of CYCLOPS which had only the ***OPO-A**** algorithm (with an interface) was difficult to use as it produced dozens of alternatives which the user had to scrutinize in detail to find any interesting ones. The process is tiring and time consuming for people. Our experience with this early version of the program led to the idea that if the computer had the relevant knowledge, it could help the designer by trying to recognize problems or opportunities by itself and then report them to the user. An advantage of this approach is that the computer can be left to examine large numbers of alternatives leaving the designer free to work on other problems. It should be noted, however, that the program is only in a position to suggest alternatives and all decisions are left to the designer.

In the program, CYCLOPS, the ability to recognize interesting designs and emerge new criteria is implemented by providing it with a database of past experiences (precedents). The program matches given designs to the database. If a design has certain characteristics that match a precedent, then the precedent is retrieved and applied.

5.2.6 Precedents and Emergent Criteria

A precedent, in its simplest form, is a record of an experience or episode. More specifically, a precedent is a record of the conditions and the effects experienced. The effects may be either physical or emotional. All precedents are input by the programmer or knowledge engineer.

A precedent is represented as a frame [Minsky 75]. For example, in the scenario above, we saw how a designer might take a liking for a particular design because it reminds him of some past favorable experience. This precedent is coded in the system as shown below:

Precedent#4232

Location Acquired:	hometown, in the year 1962
conditions:	(home on high slope) AND (home over-looking valley) AND (valley is heavily wooded and has lakes)
effect:	(very beautiful view) AND (good-view rank = -1)

The application of the precedent is quite simple. If the condition of the precedent matches any design, then a new criterion is added. This new criterion is called "good-view" and is given a rank of -1 wherever it is satisfied. All other designs are assigned a rank of 1 (denoting no opportunity) by default[17]. In CYCLOPS, a condition can be either favorable or unfavorable. Favorable conditions are given negative ranks and unfavorable conditions are positive. Neutral conditions are given a rank of unity. The ranks, in criteria such as soil, slope etc, represent degrees of unfavorability. For example, in the slope constraint, a rank of 1 means there is no *problem* with the design. Ranks of 2 or 3, on the other hand, denote problems. Worse ranks represent increasing unfavorability. The "good-view" criterion, on the other hand, represents something favorable. The ranks used should not be measures of problems but should be measures of *opportunity*. As, an opportunity is the opposite of a problem, we use a ranking scheme opposite of that used for criteria

[17]Note that adding new criteria with rank = 1, does not change the Δ of a design.

representing problems. Consequently, in the "good-view" criterion, a rank of 1 will represent no opportunity, while, ranks of 0, -1, -2 and lower will represent increasing levels of favorability.

Precedents can also recognize problems. Some precedents that are triggered, might introduce an unfavorability into the design. Such criteria are assigned positive ranks. Examples of such precedents, and a better explanation of precedent based reasoning is provided in Chapter 6.

5.3 Summary and Conclusions

5.3.1 Summary

Exploration is the basis of innovation. The process involves generating a wide variety of alternatives and then throwing away the uninteresting ones.

Design exploration can be done in several ways. The technique we have used is based on criteria relaxation. The idea is to explore outside the solution space by relaxing the criteria that bound the space. This technique is a modified form of the *PO-A** algorithm. The first step in our algorithm is to convert all the constraints and objectives into relaxed criteria. The ways in which a given criterion can be relaxed is pre-determined. For example, the constraint: "Build homes only on South facing slopes" is converted into the following relaxed form: "Build homes facing a direction that has a bearing as close to South as possible". Objectives are also converted in this manner. For example, the objective: "Minimize the cost of the design" is manually converted into the following form: "Maximize the closeness of the actual cost to the theoretical minimum". The purpose of these conversions is to allow for solutions which are in the neighborhood of the optimal and which partially satisfy the constraints.

The next step is to search the space using relaxation to explore sub-optimal design alternatives that normally would not have been considered. The purpose for considering a wide variety of design alternatives is that some of these alternatives could, possibly, have some hidden opportunities or characteristics that solve the design problem in an unexpected way.

In one mode of operation, the program can be viewed as a generator of alternatives for the user who, based on his knowledge and experience, analyses each design for hidden problems or opportunities. However, as the number of alternatives can be very large, we propose that the program be provided with experiential knowledge that it can use to recognize interesting designs. The program, CYCLOPS, is provided with a database of experiences which we call precedents. A precedent, in it's simplest form, is represented as a frame which records the conditions of the experience and the effects of the conditions. While examining design alternatives, if CYCLOPS finds that a design's characteristics match a precedent, then the precedent is retrieved and applied. The effect of applying a precedent can either be favorable or unfavorable. For example, CYCLOPS might recognize that some design configuration provides a very good view, one which is not available in the other designs. When this happens, the program introduces a new criterion "view" into the design problem. This phenomenon of criteria being added during design, as a result of designing, is called criteria emergence. If CYCLOPS finds any such new opportunity, it reports the finding to the user.

It is through the combination of criteria relaxation and emergence that CYCLOPS performs design exploration.

5.3.2 Conclusions

In this chapter we saw how criteria relaxation can lead to better designs. It is through the process of relaxation-based exploration and precedent-based reasoning that new and innovative designs are generated and recognized.

*OPO-A** is the Ordered Pareto-Optimality we used in the example, where we started form zeroth order optimality and slowly increased the order so as to find solutions. It sometimes may be possible to find a final solution with Δ_g=0, that is, with all the criteria maximized simultaneously. In practice, however, we may be required to gradually increase Δ_g in order to find a basic first solution. The gradual increasing of Δ_g is representative of gradually relaxing one's expectations, where Δ_g becomes a cumulative measure of deviation from the ideal zeroth order optimal solution.

The first solution, although it satisfies the given specifications, may not necessarily be the one that is ultimately chosen. It is always possible that some further criteria relaxation may give raise to new criteria or unearth some good design that was "just on the fringe." This process is shown in Figure 5-13.

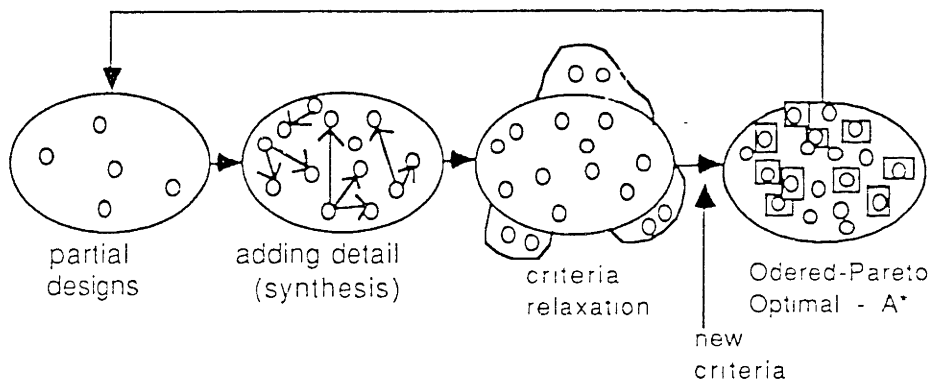

Figure 5-13: Criteria Relaxation

This chapter emphasizes our belief that innovation in design is based on one's ability to explore a wide variety of design alternatives. Innovative designs are not obtained through some deliberate attempt at producing them, but by exploring a wide variety of alternatives in search of a better understanding of the problem and potential opportunities available. The recognition of opportunities or problems in design alternatives is implemented in CYCLOPS using precedent knowledge. In this chapter we have discussed how such knowledge can be used by a program while examining an alternative. This form of precedent-based reasoning can be used not only for recognizing problems and opportunities but also for adapting designs which have promise but need a few corrections. Precedents can be used to solve design problems either directly or analogically. This idea is explored further in the next chapter.

Chapter 6

Design Adaptation

Design adaptation is the process of solving a design problem by reasoning from past experiences. It involves making innovative analogies to experiences drawn from a wide variety of sources. The design adaptation idea represents the third and final component of the thesis presented in this dissertation.

In Chapter 1 we argued that an innovative design system should have, at the least, the following three abilities:

- the ability to consider a wide variety of design alternatives (explore),
- the ability to evaluate a design, that is, to recognize opportunities and problems (criteria emergence), and,
- the ability to solve design problems by using past experiences drawn from within or without the current design domain (design adaptation).

The program CYCLOPS generates designs by searching the state-space of designs. It uses constraint relaxation to generate design alternatives in addition to those generated by the normal search process. The design alternatives are checked for opportunities or problems by reasoning from a database of precedents. The best designs are then chosen using dominance criteria. If the chosen designs have some problems, they are passed on to a design adapter. Designs are adapted by drawing strategies from past design experiences (precedents). The technique used for guiding the adaptation process is called ***demand posting***. This chapter presents the technique.

This chapter starts with an introduction to computer-based analogical reasoning and its relation to innovative problem solving (Section 6.1). The chapter presents an algorithm, demand posting, that CYCLOPS uses to perform precedent-based reasoning. The demand posting technique is presented in three stages. The first stage provides an intuitive introduction to how design problems can be solved by analogy (Section 6.2). This is followed by a detailed description of demand posting from an implementation point of view (Section 6.3). Finally, an extended example

of demand posting is presented. In the example, it is shown how the technique can draw multiple analogies to solve design problems (Section 6.4).

6.1 Background

Most design problems are not completely new. Often, parts of a design problem can be solved by reasoning from experience. If one has already seen the problem before, then one has to retrieve the appropriate precedent and apply the earlier solution. If, however, the given problem is new, it could be solved by reasoning analogically from a past experience similar to the problem at hand. Further, if the target problem is complex, then it could be broken down into subproblems and the subproblems may be solved by reasoning either directly or analogically from precedents. Solving a design problem by finding matching identical precedents is a fairly well understood task. Analogical matching, on the other hand, is a relatively new research topic in computer aided design. Research interest in this important topic has been growing in recent years: [Mostow 85, Ullman & Dietterich 87, Huhns 87, Carbonell 86, Navinchandra et.al. 87].

An Analogical Reasoning System (ARS) follows a multi-staged process consisting of the following steps: "**retrieving** a potentially analogous base, **elaborating** the base representation by additional inferences, **mapping** aspects of the base to the target, and **justifying** the mapping" [Kedar-Cabelli 85a]. In order to make an ARS retrieve and apply the appropriate base to the target problem, it has to be given a purpose with which it can search the database of precedents. Several techniques for directing an ARS have been developed. For example, the use of shared goals [Carbonell 83], the use of abstractions [Kolodner 85] and purpose-directed analogy [Kedar-Cabelli 85b].

The approach we have adopted for guiding our ARS is based on posing questions to it. For example, imagine yourself trying to open a screw in your terminal but you don't have a screwdriver. You pose a question to yourself: "Can I think of some other item that will serve as a screwdriver[18]?"

[18]You end up using your thumb nail.

Another example: You have just built a tree-house on the branch of a large tree. However the branch is sagging too much, you ask yourself the question: "Can I think of some way of making the branch stronger[19]?" Such questions are actually demands that are posted to your long term memory.

Posting demands on oneself, also called self-interrogation, has been studied by researchers interested in the psychology of creativity. Osborn's work on Brainstorming [Osborn 53] is particularly interesting. He suggests that self-interrogation can lead to creative ideas. In this book *Applied Imagination*, he suggests that one of the ways of brainstorming is to redefine the problem question by finding and addressing the causes and effects of the problem[20]. In other words, if you cannot think of a way to solve the given problem directly, try addressing its causes. CYCLOPS uses this technique to solve design problems. The technique is called ***demand posting***.

The demand posting technique helps CYCLOPS solve design problems by drawing analogies. Whenever CYCLOPS encounters a design problem which does not directly match any past experience, it tries reason by analogy. Drawing an analogy requires the ability to match a precedent and the target problem even though they have very different characteristics. For example, a landscape designer faced with the problem of locating a house on a steep slope might solve the problem by placing the house on stilts. He might get this idea by reasoning analogically from a precedent about how villagers in Thailand put their huts on stilts in order to avoid flooding. Notice that the purpose for using stilts in the base precedent is different from the purpose of the target problem. The matching is not based on surface features of base and target but on a deeper understanding of why the huts are put on stilts. CYCLOPS performs this kind of reasoning by matching against the subgoals in the explanation of the precedent. This technique is called ***subgoal matching***.

---

[19]You end up putting an angle brace under the branch.

[20]The book introduces several other self-interrogation techniques: (1) Can it be put to **other uses**? Are there new ways to use? (2) Can I **adapt**? What else is like this? (3) **Modify**? New twist? (4) **Magnify**? **Minify**? Longer? Larger? Condensed? (5) **Substitute**? Other ingredients? Other power source? (6) **Redefine**? What are the causes and effects? Can rearrange? (7) Can I **reverse**? Turn upside down? (8) **Combine**? Blend? Alloy?

The subgoal matching technique can successfully solve design problems only if it is provided with a good set of precedents to draw analogies from. The precedents, however, have to be identified from a large body of precedents in the knowledge base of the program. We need a method for finding good precedents. Techniques for achieving this have been suggested by researchers working on the psychological aspects of human creativity and problem solving. The developers of techniques such as Brainstorming [Osborn 53] and Synectics [Gordon 61] suggest that creative problem solving is predicated on retrieving and using a wide variety of precedents. The techniques they use to facilitate this process are based on asking many questions. Questions serve as cues into memory and help retrieve precedents which normally would not have come to mind. CYCLOPS performs this kind of reasoning by redefining the problem question given to it. For example, given a problem X the program first asks: "Is there a known way of eliminating X?" If it cannot find an appropriate precedent, it then asks: "What are the causes of X?", "If it is not known how to eliminate X, can its causes be eliminated?", "Can its effects be reduced or eliminated?". This questioning process is recursively applied until a solution is found. The implemented version of this technique is called ***dependency tracking***. The demand posting technique combines dependency tracking and subgoal matching into one problem solving algorithm. Dependency tracking helps identify relevant precedents and subgoal matching draws analogies to the retrieved precedents.

6.2 Demand Posting: Intuitive Ideas

In this section we will see how precedents can be used to recognize a problem in a design and how the problem can be solved by analogy. This section provides an intuitive introduction to how designs are adapted using the demand posting technique. The following section reviews the same example providing implementation details.

Consider the following scenario: A landscape designer is using CYCLOPS to find good places to locate a house. After some exploration, the program finds a site on the side of a steep hill. While the program recognizes that the site will provide a good view, it also infers that placing the

house on a steep slope will make the house tilted and unfavorable. The problem is recognized using the following precedent and rule:

Precedent#123

conditions:	a green block-1 placed on sloped ground
effects:	the block-1 becomes tilted
explanation:	the block-1 is tilted because it is on the ground and the ground is sloped

---

Rule#668

IF:	house is tilted
THEN:	unfavorable: "tilted-house"

The program matches the conditions of the problem to the precedent and infers that the house will be tilted if it is placed directly on the steep slope. This is done by matching against the explanation of the precedent. The explanation represents the real cause for tilt, and helps eliminate detractors such as color of the block. Further, the matching of "house" and "block-1" is based on the fact that they are both rectangular parallelipipeds.

Following the use of Precedent#123, the Rule#668 fires and the program infers the clause: *(unfavorable "tilted-house")*. Whenever CYCLOPS infers an "unfavorable" clause, it treats the clause as a problem that needs to be fixed. The problem is solved using the demand posting algorithm.

Dependency Tracking:

The process starts by asking the following question:

Q1: Is there some precedent that can achieve: *(not (unfavorable "tilted-house"))* **?**

If no precedent is found, the question is transformed by replacing the current problem by its cause. According to Rule#668, the cause of *(unfavorable "tilted-house")* is the clause *(house is tilted)*. The new question reads:

Q2: Is there some precedent that can achieve: *(not (house is tilted))* **?**

Assume no precedent is found. Once again, the cause of the current problem is retrieved. According to the explanation of Precedent#123, the house is tilted because it is placed on the ground and because the ground is sloped. There are two causes, negating either of the causes will solve the problem. Two new questions are posted:

Q3a: Is there some precedent that can achieve: *(not (house on the ground))* **?**

OR

Q3b: Is there some precedent that can achieve: *(not (ground is sloped))* **?**

Using Q3a, the program finds a precedent about how huts in Thailand are put on stilts to protect them from floods. Based on this precedent, the program decides to use stilts for achieving: *(not (house on ground))*. By using stilts, the program solves the original problem of putting a house on a steep slope. The program is able to match Q3a with the precedent by examining the subgoals in the explanation of the precedent. We will now see how precedents are represented to facilitate this kind of reasoning.

Subgoal Matching:

The precedent about how villagers in Thailand put their homes on stilts to prevent them from floods is shown below.

Precedent#1002

Initial Situation:

conditions:	the hut is on the ground and the ground is flooded
effects:	the hut is water-logged and is unfavorable
explanation:	(B1) the hut is unfavorable because it is water-logged (B2) the hut is water-logged because it is on the ground and ground is flooded

Solution:

action:	put the hut on stilts
effects:	the hut is not unfavorable
explanation:	(B3) the hut is not unfavorable because the hut is made not water-logged and (B1) is true (B4) the hut is not water-logged because the hut is not on the ground and (B2) is true[21] (B5) the hut is not on the ground because it was originally on the ground and the following action was taken: "put the hut on stilts".

The precedent has two parts: a situation and a solution. The first part describes the problem and provides an explanation of the problem. The second part describes how the problem was solved. The explanation consists of causal relations among conditions, effects and actions. The relations in the solution part of the precedent represent how subgoals are achieved. The main goal of the precedent it to achieve: "the hut is not unfavorable" and the subgoals are: "hut is not water-logged" and "hut is not on ground". As the second subgoal (B5) matches the demand Q3a (show in the previous section), the causal relation B5 is transferred from base to target. Hence, the problem is solved by placing the house on stilts.

Having taken an intuitive look at demand posting, let us now examine how precedents are represented in the computer and what steps are taken by the program when it draws analogies.

[21]In english, this relation would read as follows: "The hut is not water logged because it is not on the ground anymore, which is true because, the original reason for the hut being water logged was that it was on flooded ground (B1)."

6.3 Demand Posting: Details

In this section we will re-trace the steps of the scenario just presented while providing implementation details. The scenario started with the use of Precedent#123 (Page 104) to recognize that the house will be tilted. In a semantic network form, the precedent may be represented as shown below:

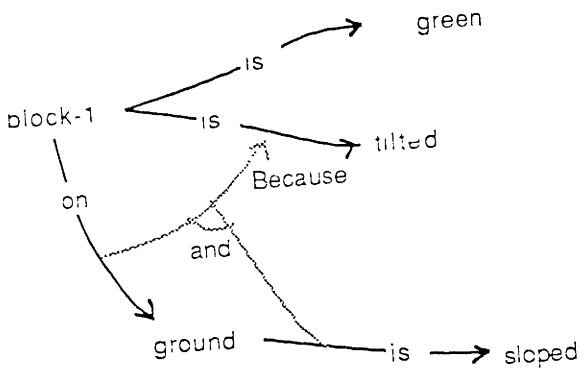

Figure 6-1: Block on a slope: semantic network representation

The figure shows three types of links: attribute links ("is"), an inter-object link ("on") and a causal relation ("B"[22]). The purpose of the causal relation is to specify the structure of the precedent [Gentner 83]. The causal relation in the precedent reads as follows: "The block is tilted because the block is on the ground and the ground is sloped." Attributes (e.g. color) which are not part of the causal structure are not as important [Gentner & Toupin 86]. Although we will continue to keep the fact that the block is green, we will ignore that property while using the precedent to infer causes for tilt. Let us now redraw the precedent, highlighting its causal structure.

[22]The letter "B" denotes the word "Because"

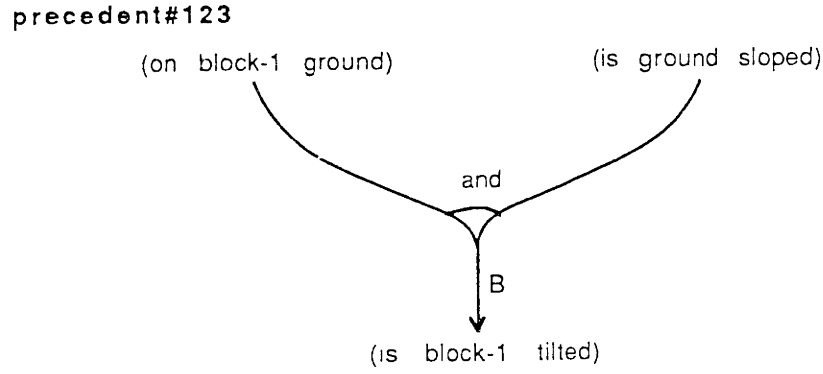

Figure 6-2: Block-1 is tilted Because block-1 is on the ground and the ground is sloped

Assume that the design at hand has the following clauses in its description: *(on house ground)* and *(is ground sloped)*. The program matches these clauses with Precedent#123 to infer that the house will be tilted. It matches "block-1" and "house" by walking up and down an object hierarchy. The program uses a hierarchy like the one shown below.

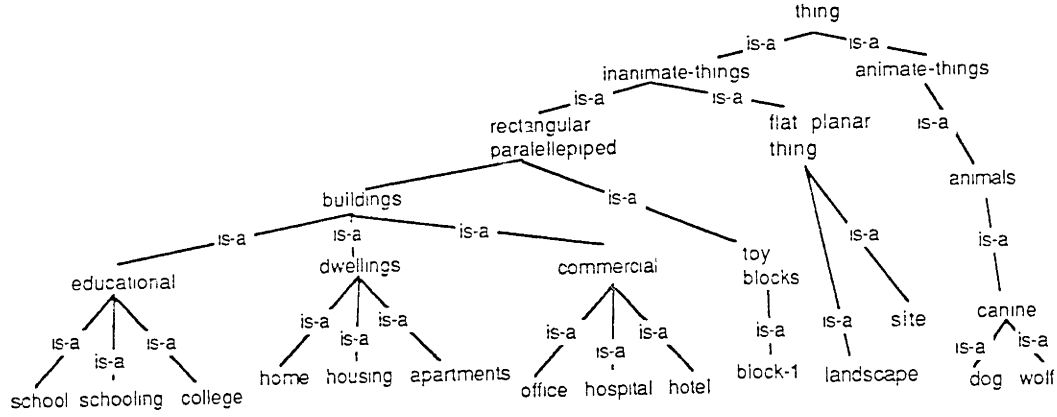

Figure 6-3: Object Hierarchy: used for relaxed matching only

The only purpose of this hierarchy is to match clauses indirectly. CYCLOPS will match any two objects (at the leaves) which are separated by up to six links[23]. This kind of matching is called *relaxed matching*. After performing the match and inferring from the Rule#668, the design's problem is represented as:

[23]There is no theoretical justification for the amount of relaxation allowed. All one can do is to prefer matches which require less number of links to be traversed. CYCLOPS leaves it up to the user to reject matches which seem outrageous.

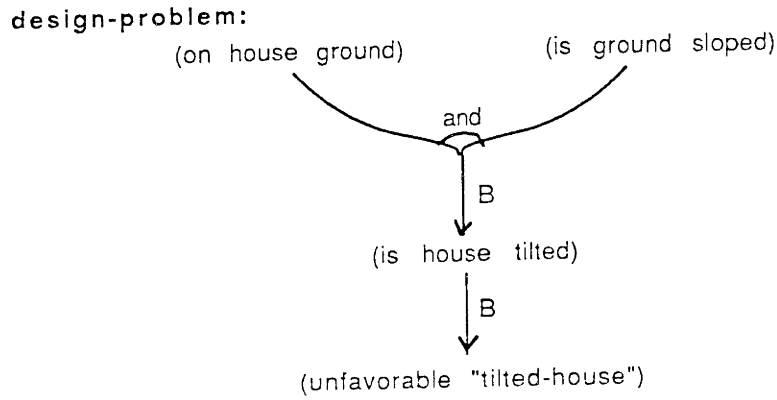

Figure 6-4: The design problem's causal structure

Whenever CYCLOPS infers an "unfavorable" clause, it reports the clause as a design problem that needs fixing. If the user decides to let CYCLOPS try fixing the problem, the program enters the demand posting process.

Let us assume, for now, that the only precedent in memory (other than Precedent#123) is one about huts in Thailand. Using a causal structure representation similar to that of Figure 6-2, the Precedent#1002 is represented as shown in Figure 6-5. The figure has two parts: a situation and a solution. These parts correspond to those shown in the text of the precedent (page 106).

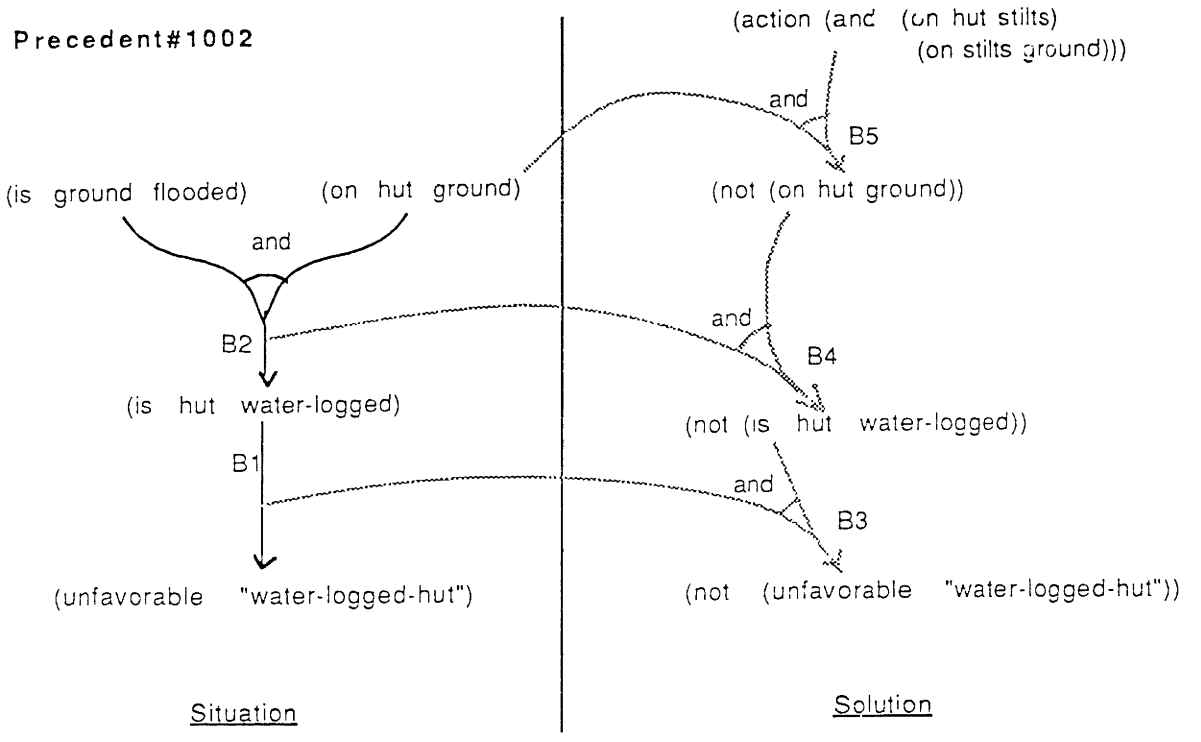

Figure 6-5: Thailand precedent's causal structure

The precedent above has five causal relations:

B1: (unfavorable "water-logged-hut") because (is hut water-logged)

B2: (is hut water-logged) because (and (is ground flooded) (on hut ground))

B3: (not (unfavorable "water-logged-hut")) because (and (B1 is true) (not (is hut water-logged)))

B4: (not (is hut water-logged)) because (and (B2 is true) (not (on hut ground)))

B5: (not (on hut ground)) because (and (on hut ground)
(action (and (on hut stilts) (on stilts ground))))

The demand posting process starts by posting the following question as the first demand:

Q1: *(not (unfavorable "tilted-house"))* ?

The program then tries to match the question against the causal relations in the precedent. As no match is found, the cause of the problem is retrieved and posed as a new demand (Q2). The idea behind this process is to derive a solution to the problem situation shown in Figure 6-4. The

solution to the situation will eventually look like the situation-solution pair that exists for Precedent#1002 (Figure 6-5).

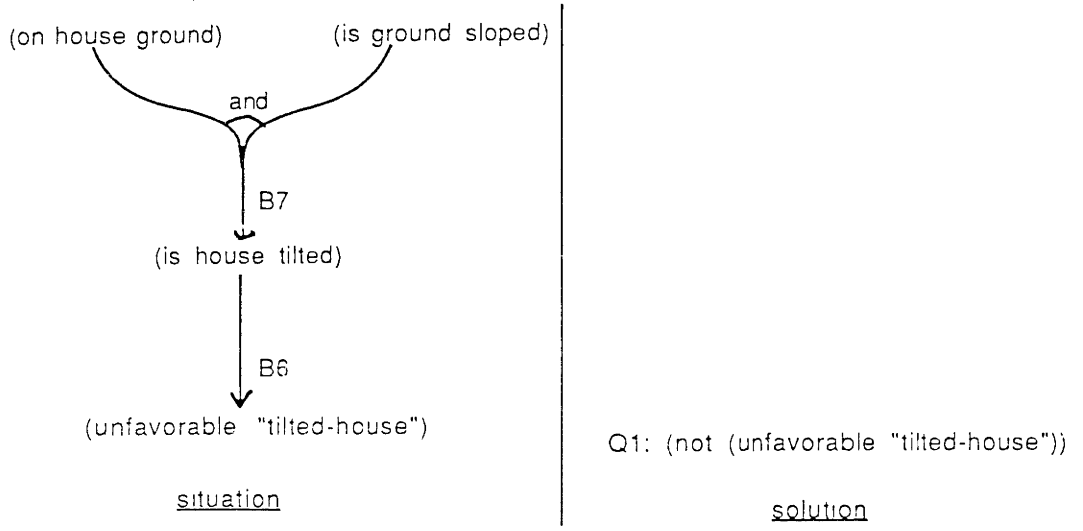

Figure 6-6: Posting Q1

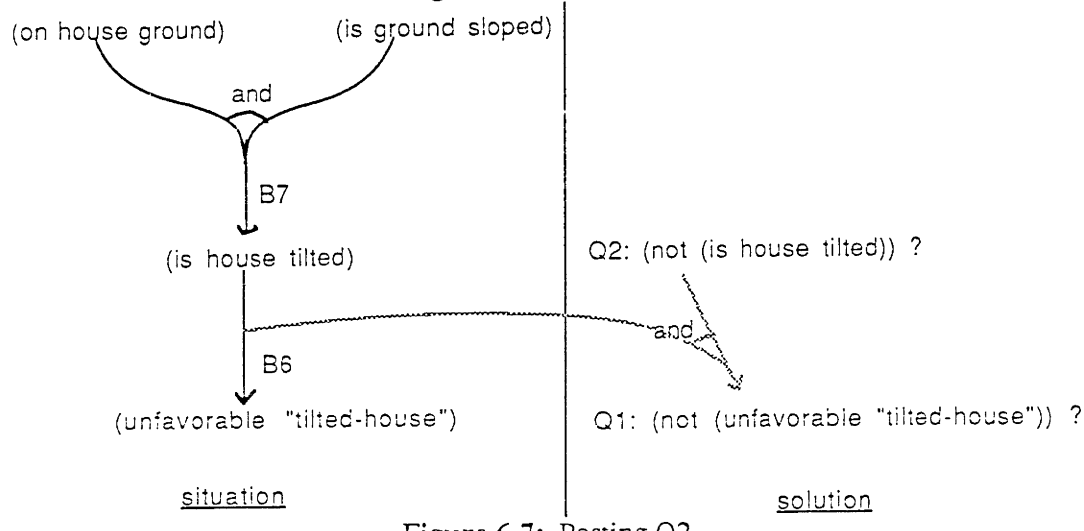

Figure 6-7: Posting Q2

Posting of the first question is shown in Figure 6-6. As the program cannot find an answer to Q1, it then posts Q2 which corresponds to the cause of Q1. The posting of the second question (Q2) is shown in Figure 6-7. The program justifies Q2 by a causal relation (gray line) which represents the following reasoning: "Q2 can substitute for Q1 because B6 is true." The demand in Q2 is tried against the causal relations of Precedent#1002. As no match is found, the question is transformed by retrieving the causes of the latest demand. This is shown in Figure 6-8. The figure shows that the

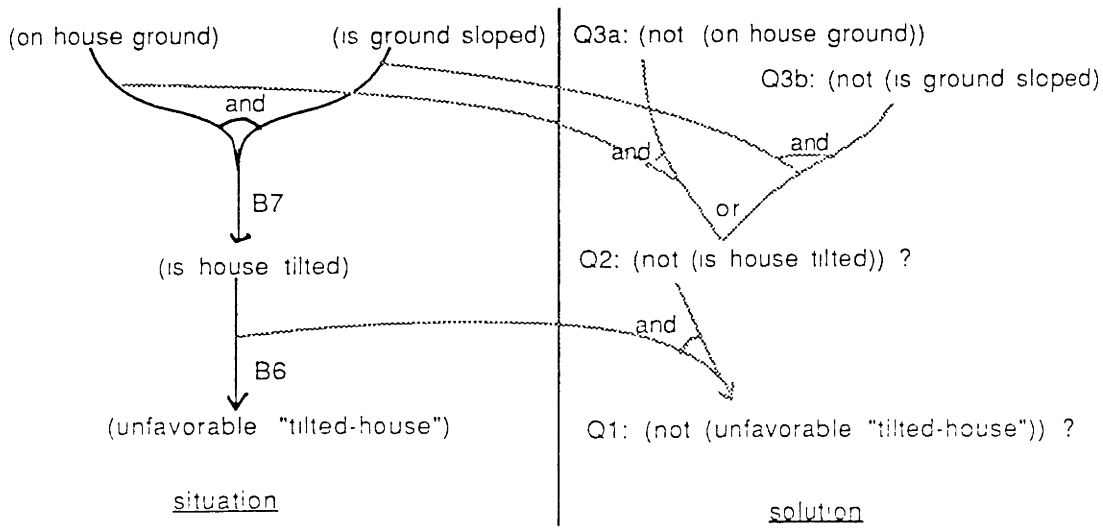

Figure 6-8: Posting Q3

demand in Q2 can be solved by answering either Q3a or Q3b. Once again, the demands are tried against the relations in the precedents. This time, the causal relation: B5 of Figure 6-5 is found to have a subgoal that matches the given demand. The matching is shown in the Figure 6-9.

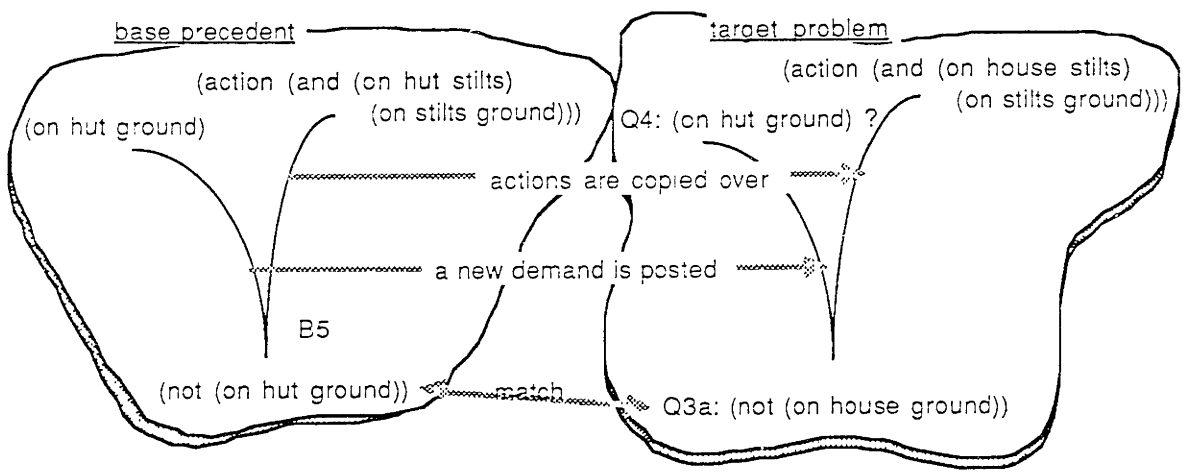

Figure 6-9: The matching of causal structure between base and target

As the clause in Q3a and the clause on the left hand side of B5 match, the causal structure of B5 is transferred to the target problem[24] The action is copied over directly and a new demand for

[24]This idea of matching and transferring of causal structure is borrowed from Winston [Winston et al. 83].

(on hut ground) is posted. It turns out that a matching clause *(on house ground)* is already true in the target problem's description. As there are no more demands to be answered, the problem is deemed solved.

It is in this way that demand posting is used to analogically solve a problem. The analogy lies in the fact that the purpose for using stilts in the precedent was different from the purpose it is used for in the target problem. The original purpose was for avoiding flooding and the target purpose is for avoiding steep slope. The analogy is drawn by matching one of the subgoals of the precedent with one of the causes of the problem.

The following section presents an extended example of demand posting. The example shows how CYCLOPS manages multiple solutions and draws multiple analogies to solve a problem. Before moving on to the next section, let us quickly characterize the demand posting process.

6.3.1 Notes on Demand Posting

Multiple support for a problem. What happens if there are two or more independent causes for a problem? When the demand posting algorithm finds a way of eliminating a problem, it unasserts the corresponding clause. If there are other independent causes for the clause, the program is fooled into believing that the problem is solved. This situation, however, is only temporary. When CYCLOPS enters the next inference cycle, the problem clause will be re-asserted if there is independent support for it.

Learning. When the demand posting process is used to solve a problem, the resulting situation-solution structure can be used as a precedent in future problems.

Completeness assumption for causal relations. The causal relations in a precedent are treated independently. For example, the relation B2 (page 110) which reads: "the hut is water-logged because the hut is on the ground and the ground is flooded" is a statement that stands alone[25]. However, some relations are dependent on others, for example B4 is dependent on B2. We assume that a causal relation is complete and that it carries the complete context in its body.

[25]This is similar to the explanations used in XPs [Schank 86]

Closed world assumption. CYCLOPS does not actively verify analogies. When an analogy is drawn, the modified design is posted to the precedents data-base. If none of the precedents find a problem, the analogy is deemed correct. We are making the assumption that if a clause cannot be inferred from the knowledge possessed, then it is false. This is a closed world assumption on the knowledge base.

Indexing many precedents. Based on the completeness assumption, all the causal relations of all the precedents are indexed independently. The index is based on the clauses that the relations assert. We currently use a flat indexing mechanism, only because it serves our immediate needs. For large databases, a discrimination-network based technique could be used for indexing [Kolodner 80].

6.4 Demand Posting: Extended Example

When there are several precedents to reason from, there may be several ways of solving a problem. CYCLOPS manages multiple partial solutions using a tree structure. In this section we will start with an intuitive introduction to how a problem may be solved in a larger context. Following this, a detailed explanation is provided.

6.4.1 Extended Example: Intuitive ideas

In this subsection we will see how a program could go about solving a design problem. The problem is the same as before: placing a home on a steep slope.

Following is a hypothetical protocol of how the problem might be solved. The purpose of the protocol is not to describe exactly how the program should solve the problem but is to provide the reader with an intuitive idea of the process. The protocol is written as if somebody is thinking aloud while solving the problem.

Let me see. The site is steep and placing the house at the site will make the house tilted. That's not good.

Can I think of some way of getting rid of the tilt?

Not really. Well then, I need to find some way of removing the ground's tilt. (... a short pause ...) Yes! I remember how people in Nepal terrace steep slopes to create flat pieces of ground for growing rice. I could also terrace the ground to site the house.

Wait! There is a problem here. I just remembered that this landscape receives a lot of rain. Terracing the site is not acceptable. I remember reading about how terracing sites in very rainy areas can lead to bad erosion and scarring of the land. Terracing will not work.

(after a break) What I really need to do is analyze the problem. The house is unfavorable because it is tilted. The house is tilted because the ground is sloped and because the house is on the ground. I tried getting rid of the slope by terracing, but failed. What else can I try? (...short pause...) If I could find some way of not having to put the house on the ground, I could solve problem.

(a few minutes later) Yes! I remember seeing pictures of village homes of Thailand. The villagers have a problem with flooding and dampness but have found a way of keeping their homes off the wet ground: they put their homes on stilts. I'll do the same.

As far as my memory serves, I don't see any problems with using stilts. OK, I will use stilts...."

6.4.2 Extended Example: Solution by CYCLOPS

In this section we will examine how the program, CYCLOPS actually solves the design problem in a way similar to the pseudo protocol presented in the previous section.

The process starts by first listing down the known facts:

---

1. (is landscape rainy)
2. (on house ground)
3. (is site sloped)
4. (part-of ground landscape)

This is shown in box A of Figure 6-10. At this stage CYCLOPS finds (using subgoal matching) a precedent about how: putting an object on a sloped surface makes the object tilted[26]. The precedent is represented as a frame:

Precedent#123

condition:	(is ground sloped) and (on block-1 ground) and (is block-1 green)
effect:	(is block-1 tilted)
explanation:	(is block-1 tilted) because (is ground sloped) and (on block-1 ground)

The effect of applying the precedent is shown in box B. In box B, several new propositions

[26]Such a precedent could have been acquired during one's childhood days when one played with blocks and balls.

have been inferred by reasoning from Precedent#123 and Rule#668 (Page 104). Clause 7 in box B states that the house is unfavorable. CYCLOPS responds to this by posting a demand:

Q1: *(not (unfavorable "tilted-house"))* ?

CYCLOPS tries matching the question against the database of precedents. As before, let's assume it does not find appropriate precedents and that question Q2 is transformed into Q3a and Q3b. This is shown as boxes E and H in the Figure. Picking up box E (at random), CYCLOPS starts looking for a precedent that has information on how to achieve *(not (is ground sloped))*. The program finds precedent#999. This precedent is about how people in Nepal, use terracing in order to grow rice on steep hillsides.

Precedent#999

Situation

conditions:	(is site sloped) and (is site paddy-fields) and (location site Nepal)
effect:	(unfavorable "cannot-grow-rice-on-hillside")
explanation:	(B91): (unfavorable "cannot-grow-rice-on-hillside") because (not (at site water)) and (is site paddy-fields)
	(B92): (not (at site water)) because (flow-away-from site water)
	(B93): (flow-away-from site water) because (is site sloped)

Solution

action:	(action (is site terraced))
effect:	(not (unfavorable "cannot-grow-rice-on-hillside"))
explanation:	(B94): (not (unfavorable "cannot-grow-rice-on-hillside")) because (at site water) and (true (B91))
	(B95): (at site water) because (not (flow-away-from site water)) and (true (B92))
	(B96): (not (flow-away-from site water)) because (not (is site sloped)) and (true (B93))
	(B97): (not (is site sloped)) because (is site sloped) and (action (is site terraced))

Precedent#999 has two parts. The situation part of the precedent states that: having paddy-fields on a sloped ground is unfavorable because there will be no water at the site, which is because

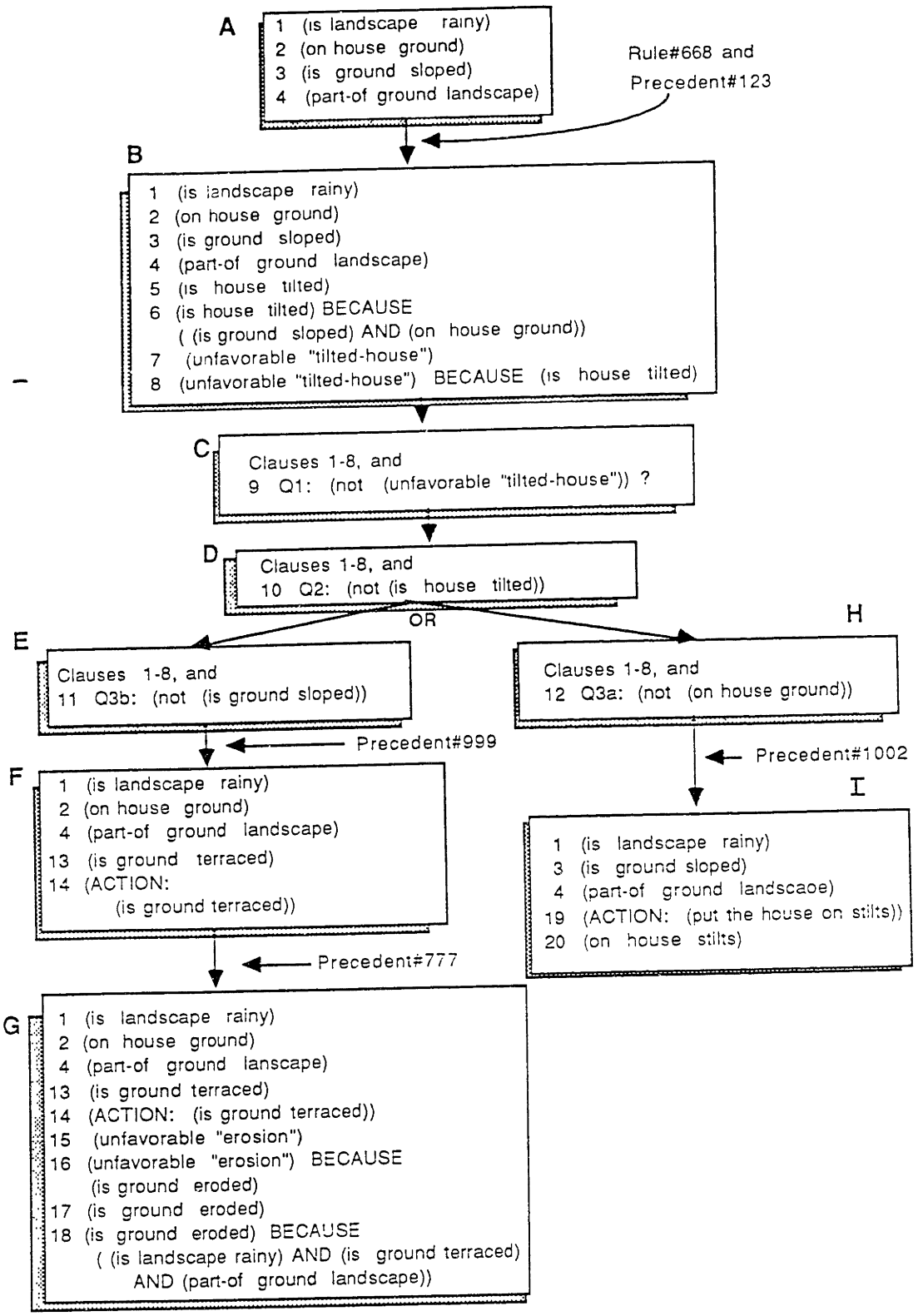

Figure 6-10: Demand Posting Tree

the water flows away from the site, which in turn, is because the ground is sloped. The solution part of the precedent says that the problem was solved by terracing, an action which makes the ground not-sloped.

CYCLOPS, which is now looking for some strategy to achieve *(not (is ground sloped))* uses Precedent#999 and applies the terracing action. This is shown in Box F of Figure 6-10. Notice that many propositions have disappeared. When it was decided to terrace the ground, then the proposition *(is ground sloped)* was removed, consequently propositions that lost support were also removed.

Soon after this, CYCLOPS starts looking for problems. This is CYCLOPS' way of verifying the analogy. In box G of the figure it is shown how CYCLOPS realizes that terracing in a rainy landscape will cause erosion problems. This is based on precedent#777.

Precedent#777

conditions:	(is site terraced) and (part-of site washington-state) and (location site washington-state) and (is washington-state rainy)
effect:	(unfavorable "erosion")
explanation:	(unfavorable "erosion") because (is site eroded)
	(is site eroded) because (is washington-state rainy) and (part-of site washington-state) and (is site terraced)

Box G leads to a new problem. By solving one problem, a new one has been generated. There are two options here. We can either post a new demand and try proceeding with G or examine box H. CYCLOPS prefers branches that have a less number of changes made to them. The program moves up to box H. In trying to satisfy the demand of box H, CYCLOPS finds and uses the precedent about huts in Thailand (Page 110). When CYCLOPS finds a way of achieving *(not (on house ground))* it uses the action and applies it to the target. This is shown in Box I. Finally, the program checks the propositions in box I for new problems (if any). If none are found, the design is accepted.

6.5 Summary and Conclusions

6.5.1 Summary

One of the characteristics of an innovative design system is its ability to use precedents to solve design problems. Problems can be solved by reasoning analogically from precedents retrieved from long term memory. Retrieval of the precedent, among other things, requires providing the Analogical Reasoning System (ARS) with a purpose. In the chapter we examined a technique, *demand posting* that is used to post questions to the ARS.

In a demand posting system, whenever a problem is detected a demand is posted to the ARS requesting that a precedent that can solve the problem be retrieved. If an appropriate precedent is not found, the causes of the problem are retrieved and new demands are posted. This process continues recursively till a solution is found. If not, the attempt to adapt the design is abandoned.

The demand posting technique consists of two parts: Dependency Tracking and Subgoal Matching.

Dependency-tracking involves the following steps: first, the problem is identified; second, the problem statement is posted as a demand on the database of precedents. This means that the database manager has to find precedents that match the demand (pattern).

If no suitable precedent is found, the program retrieves the causes of the problem. These causes are then posted as demands. This process of posting demands, retrieving causes of problems and posting the causes as new demands proceeds recursively till "suitable" precedents are found.

Subgoal-matching. A precedent is said to be "suitable" with respect to a posted demand (pattern) if it contains a design strategy which attains a goal (pattern) that matches the demand. If such a match is found, the design strategy in the precedent is transferred to the current design problem. If, on the other hand, a match between the demand and the goal of the precedent is not found, the program retrieves an explanation (pre-coded) of how the design strategy in the precedent attains the precedent's goal. An explanation is usually a trace of how the different parts of the overall strategy address the subgoals of the overall goal. After retrieving an explanation, the program matches the demand (pattern) against the subgoals of the precedent strategy in order to find a match. If no match is found, failure is announced.

An advantage of this technique is that it can match a base and target even if their main goals or surface features are radically different. It is for this reason that the program appears to reason analogically.

6.5.2 Conclusions

The program CYCLOPS, is able to innovatively solve design problems by three strategies: exploration, criteria emergence and precedent-based design adaptation. This chapter has been about how precedents can be used to fix design bugs. We examined a technique, ***demand posting***, which uses dependency tracking and subgoal matching to solve problems analogically. The demand posting technique is fairly general and could be extended to use strategies other than dependency tracking. This could lead to systems which could serve as brainstorming assistants to designers.

This has been the last chapter in the main body of the dissertation. The next three chapters are concluding chapters. The first of these provides an overview of CYCLOPS' architecture and shows how the ideas discussed in the last four chapters fit into into system. The second concluding chapter compares our research to other work in the field. The last presents the contributions of this dissertation and lists the assumptions and shortcomings of our current approach.

Chapter 7

Putting It All Together: A Detailed Architecture of CYCLOPS

In this chapter we will see how the techniques described in the last four chapters fit into one coherent system. CYCLOPS' architecture can be viewed in three layers. Each layer represents one of the modes the program operates in: the ***normal search*** mode, the ***exploration*** mode and the ***adaptation*** mode. Recall that in Chapter 1 we introduced the program's architecture in a simplified form (Figure 1-2, Page 19), we will now take a detailed look at it.

The chapter has five sections. The first three correspond to the three modes of operation. The last two sections describe how CYCLOPS is used and how it has been implemented.

7.1 Normal Search Mode

In the normal search mode, the program performs the ***PO-A**** algorithm (Chapter 4). In this basic mode, the program will find all non-dominated solutions (provided they exist). This is shown in Part-I of Figure 7-1. The search process uses two modules: a *synthesizer*, a *selector*. The synthesizer takes partial designs and adds detail to them by instantiating their variables. The selector checks for dominance and places the designs either in a dormant list or in an active one. The active designs constitute the non-dominated set and are returned to the search process.

7.2 Exploration Mode

In the exploration mode, CYCLOPS relaxes the governing criteria and searches alternatives outside the original solution space using the ***OPO-A**** algorithm. This is done by the ***explorer*** module (Part-II of Figure 7-1). The explorer examines designs that have been discarded by the selector. It tries to see if there are potential opportunities in the designs that have been dominated. This is done by emerging new criteria during design. The ***selector*** has a ***criteria emergence***

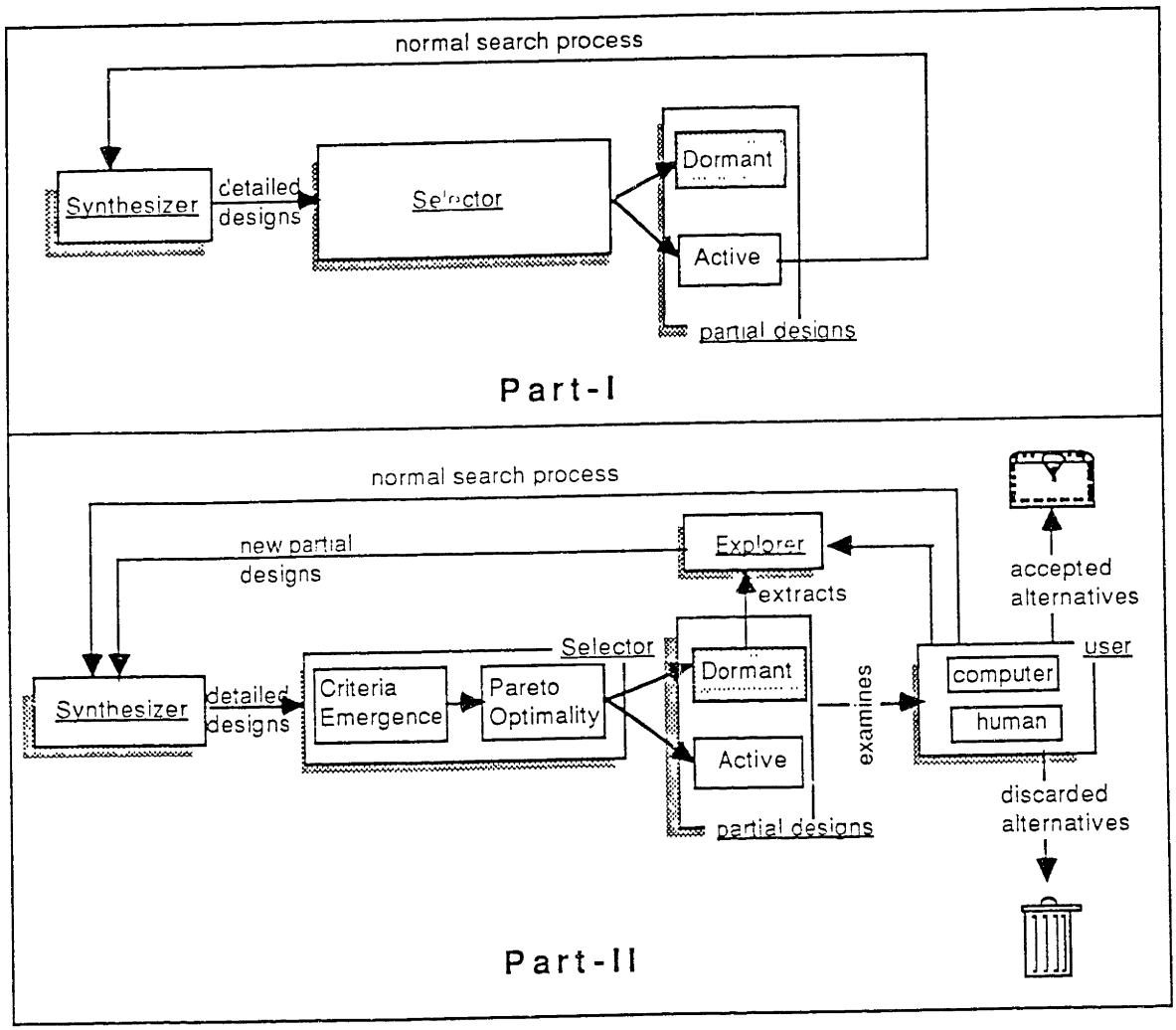

Figure 7-1: Normal search mode (Part-I), Exploration mode (Part-II)

submodule which matches design alternatives against precedents (this aspect is shown in the complete architecture, Figure 7-2).

The *user* plays an important role during exploration. The user examines designs and decides to either accept or discard them. If the user accepts a complete design, it is stored in the treasure chest. If it is incomplete, it is put back into the normal search process. The user can either be human or a program or both.

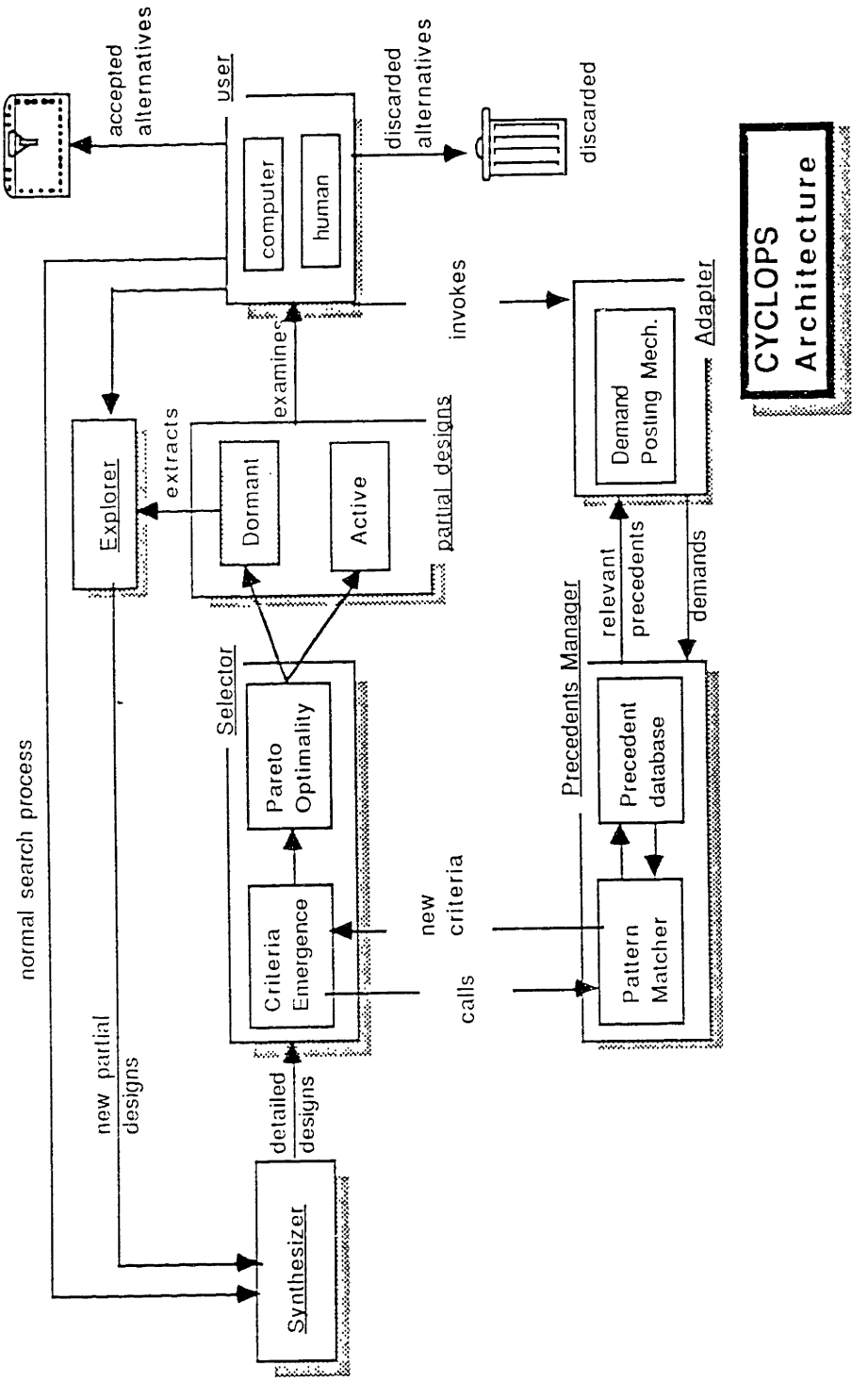

Figure 7-2: The complete CYCLOPS architecture

7.3 Adaptation Mode

In the adaptation mode, CYCLOPS uses precedent knowledge to solve design problems. This is done by the ***adapter*** (Figure 7-2). The adapter runs the demand posting algorithm. It posts demands on the precedents manager and accepts precedents to draw analogies from. As described above, the precedents manager is also used by the ***selector*** module. Just before the selector applies a dominance check (***pareto optimality***), it checks if the designs it is examining have some problems or opportunities.

7.4 Operating CYCLOPS

CYCLOPS is started by putting the root node in the "active" box. The program starts in the normal search mode. If no solutions are found, or if the user makes a request, the program goes into the exploration mode. The adapter is invoked only by the user. This is done because, adaptation is computationally expensive and hence it is not possible to try and adapt each and every design produced by the synthesizer.

A trace of one of the implementations of CYCLOPS is provided in Appendix B.

7.5 The Implementation

CYCLOPS is implemented in FranzLisp[27] and runs on VAX[28] machines running the UNIX[29] operating system. CYCLOPS' interface uses the X[30] window system. The program has 20 precedents and has been run on problems with 10 landuses and 15 lots.

---

[27]Trademark of Franz Inc., Alameda, CA

[28]Trademark of Digital Equipment Corp, Maynard, MA

[29]AT&T Bell Labs, NJ

[30]Developed at MIT's Project Athena

Chapter 8

Relationship to other work

8.1 Introduction

This chapter relates the ideas presented in this dissertation to other research efforts. We will compare the techniques used in CYCLOPS to those used in related systems. This chapter does not review the literature. Readers who wish to get a broader sense of the techniques used in Design Automation (DA) systems may refer to the following papers [Mostow 85, Sriram 86, Tong 86b].

In the dissertation we have seen how:

- some design problems can be represented as ***multi-criteria, configuration*** problems, and how search can be used to solve them,
- innovative designs can be generated by ***exploring*** a wide range of alternatives, and
- precedents can be used to ***adapt*** designs which have problems.

In this chapter, we will examine related research in the light of the three aspects listed above.

The chapter has three parts. The first is about optimization in configuration problems. We will relate CYCLOPS to other work in OR and AI-based design automation. The second part relates CYCLOPS' exploratory behavior to AI-based discovery systems. The third part discusses precedent-based reasoning and the origins of the demand posting algorithm.

8.2 Design as a Multi-Criteria Configuration Problem

Several conceptual design problems can be viewed as ***configuration*** tasks. Many of the AI-based design automation systems solve conceptual design problems using search and ***constraint-satisfaction*** techniques. Although these systems use different problem solving techniques, a majority of them treat design as a satisficing problem and assume that the solution space always exists. CYCLOPS builds on the ideas in these systems by requiring ***multi-criteria*** optimization in addition to consistency over constraints. Multi-criteria situations often tend to be

over-constrained requiring our system to have the ability to ***relax criteria*** and recognize critical ***tradeoffs***.

8.2.1 Constraint satisfaction and Configuration Problems.

The techniques used in most design automation systems are motivated by the work on constraint manipulation and satisfaction. The techniques include: generate and test [Buchanan & Feigenbaum 78, Lindsay 80]; network consistency algorithms [Mackworth 77]; combined backtracking and constraint-based reasoning [Fikes 70]; dependency directed backtracking [Stallman & Sussman 77]; the constraint posting idea [Stefik 80]; and forward-checking in consistent labeling problems [Nadel 85d].

Our decision to view landscape design as a configuration task is motivated by some of the early work on design automation. In particular AIR-CYL [Brown & Chandrasekaran 86] and PRIDE [Mittal 85]. Both these systems use a combination of design goals and constraints to guide the solution process. PRIDE views its task of designing paper transports for photocopiers as a configuration task involving the choice and location of paper handling mechanisms such as rolling-stations, pinch-rolls and paper-guides. Similarly, CYCLOPS views landscape design as a spatial configuration of landuses such as houses, churches and schools which satisfies given constraints. CYCLOPS differs from these and a majority of other conceptual design automation systems in that it requires designs to be optimal with respect to a given set of objectives.

8.2.2 Optimality and Tradeoffs

There are several techniques for multi-objective optimization [Goicoechea 82]. These techniques deal with continuous, monotonic functions and cannot be applied to landscape design problems. This is because landscape problems can have criteria which are non-linear, non-monotonic, non-continuous and discrete. The only way such problems can be solved is by search. Finding optimal solutions using search involves ***look ahead***. Further, finding solutions over multiple objectives almost always involves making ***tradeoffs***.

Look ahead. Optimization by searching requires some method for "looking ahead". The technique used in CYCLOPS is based on the notion of admissibility [Hart, Nilsson & Raphael 68]. The A^* algorithm is modified to handle multiple-criteria. This is done by using dominance as a measure of optimality. CYCLOPS achieves admissibility by choosing optimistic values from previously enumerated tables of possible outcomes. The program estimates the "cost to completion" of partial designs by making optimistic assumptions about how the rest of the design will be completed. Other approaches to providing look-ahead include: the use of abstract operators for VLSI design in the DONTE system [Tong 86a]; use of micro-structural properties of alloys as rough designs in the alloy design program ALADIN [Rychner et al. 86]; and use of rough parameter values in AIR-CYL [Brown & Chandrasekaran 86]. These systems, unlike CYCLOPS, are only attempting local optimization. CYCLOPS' ***PO-A**** algorithm is guaranteed to give global optimal solutions[31]

Tradeoffs. In multi-criteria problems, the space often tends to be over constrained. Criteria have to be relaxed and traded off against one another in order to find solutions. CYCLOPS handles this problem by using a representation for constraints and objectives which allow for relaxation when required. The program, however, leaves all final tradeoff decisions in the hands of the user. Some systems, on the other hand, have explicit tradeoff heuristics. For example the two VLSI design systems VEXED [Steinberg 87] and ULYSSES [Bushnell & Director 86] have explicit rules such as: "IF the goal is to minimize delay of an adder circuit, THEN prefer carry-lookahead over ripple-carry". Use of explicit knowledge to prefer one design decision over another is not done in CYCLOPS, however, it makes these kinds of decisions in another way - it emerges new criteria to select interesting alternatives. The program does this while it explores the design space.

[31]A proof of this result is provided in Appendix A

8.3 Design Exploration

Innovation in design is based on one's ability to ***explore*** a wide variety of alternatives. Innovative designs are not obtained through some deliberate attempt at producing them, but by generating lots of designs and by throwing away the bad ones. Consequently, the ability of a system to innovate depends on how well it can generate diverse alternatives that break away from the norms and the governing constraints. The idea is that new alternatives can sometimes ***serendipitously*** lead to interesting solutions.

The idea of exploration is motivated by Darwin's theory of evolution. In his book *On the Origin of Species* he notes:

> "As many more individuals of each species are born than can possibly survive; and as, consequently, there is a frequently recurring struggle for existence, it follows that any being, if it varies however slightly in any manner profitable to itself, under the complicated and sometimes varying condition of life, will have a better chance of surviving, and thus be naturally selected." - **Charles Darwin** [Darwin 59].

Restating the quotation, in a design domain, we get:

> As many more designs are generated than can possibly survive pruning; and as, consequently, there is a frequently recurring test of optimality, it follows that any design, if it varies however slightly in any manner profitable to itself or its variations, under the complex and sometimes varying set of constraints and objectives, will (together with its variations) have a better chance of surviving the pruning test, and thus be naturally selected.

CYCLOPS explores alternatives (variations) by relaxing criteria. It recognizes interesting designs by reasoning from a database of previous experiences.

8.3.1 Exploration and Discovery

There are several systems that discover solutions by exploring the state space. Examples of such systems are: DENDRAL, a program that searches for novel structural combinations of organic molecules [Buchanan & Feigenbaum 78]; DALTON, a system that discovers molecular compositions based on given reactions [Langley et al. 87]; SIDS, a bridge design system that discovers novel combinations of structural elements [Chehayeb 87]; EDISON, a program that invents new mechanical systems [Dyer et al. 86]; BACON, a quantitative discovery system that

finds relations among variables; and AM, a system that discovers new mathematical concepts [Lenat 76]. CYCLOPS is also a exploratory system, but in a different domain. Let's see how it differs:-

All the related systems cited above are based on the idea of searching the state space using special heuristics. As the state space tends to be very large, it is important to control the combinatrics with good exploration techniques. Examples of the techniques are: use of chemical knowledge in DENDRAL's generator; DALTON's use of domain knowledge such as laws of conservation and combining volumes to limit the search; BACON's use of quantitative heuristics such as inversion, multiplication etc.; and AM's use of mutations. The exploration technique used in CYCLOPS is criteria relaxation. The program explores new alternatives outside the solution space by relaxing the criteria that bound the space. Other approaches to exploration are based on making changes directly to the artifact being generated. For example, AM explores new mathematical concepts by applying mutation operators to the LISP code that represents the original concepts. Similarly, EDISON invents new mechanical systems by making mutations to the artifacts it deals with [Dyer et al. 86]. For example, it might take a door and chop it in half to discover a bar-room door, alternatively, it might randomly rearrange the hinges to produce a trap door. CYCLOPS is different in that it does not make mutations directly to the artifact but to the constraints and objectives that bound the artifact solutions space. In effect, criteria mutation/relaxation is the ***dual*** of artifact mutation. A problem with applying mutators directly to the artifacts is that it can produce too many useless alternatives [Lenat 84]. In our case, however, as criteria relaxations are gradual, we only look at those alternatives which are just dominated. This allows us to gradually move away from the solution space every time a relaxation is made. It should be noted, however, that although this technique at first, only looks at solutions close to each other on the pareto graph, it can generate artifacts that have very different physical characteristics. Just because two solutions are close to each other on the pareto graph does not mean they are structurally similar. Mutation operators, on the other hand, can only generate solutions that are structurally close, if the mutations are too radical

they run the danger of producing too much junk. In the dissertation, we saw how a threshold[32]-based algorithm (*OPO-A**) can be used for exploring the artifact space by making relaxations in the criteria space.

Another way CYCLOPS is different from the other systems is that it can extend the search space dynamically. This happens when CYCLOPS uses new objects drawn from precedents. For example, we saw how CYCLOPS used stilts to solve a design problem. The notion of stilts was not in the original representation and came from outside. A program's ability to opportunistically extend the state space is very useful. The idea is to not start the search process using a fully extended state space, but to extend it as and when required.

8.3.2 Recognizing Interesting Solutions

In order to innovate, it is important not only to be able to explore new alternatives but also to be able to recognize innovative alternatives when they are generated. The BACON program, for example, discovers new laws by detecting numerical trends. Using a depth-first control strategy, the program generates data that is checked for patterns of relationships among variables. The AM system uses heuristics of interestingness to move from one interesting concept to another. For example, AM considers extreme cases of set membership interesting, for instance, a prime number is an extreme case of having no divisors other than itself and unity. In the same spirit, CYCLOPS also recognizes interesting designs but it's mechanism is based on past experiences (precedents). If CYCLOPS finds a design to be interesting for some reason, it introduces a new corresponding criterion CYCLOPS emerges new criteria by matching design alternatives against precedents. The idea is not to start with a long list of criteria in the beginning of the search, but to add new ones as when they are relevant.

The criteria emergence idea is somewhat similar, in spirit, to the serendipity recognition

---

[32]Our use of thresholds is similar to those used in the ID-A* algorithm [Korf 85]. However, our purpose for using thresholds is completely different.

mechanism in the DAYDREAMER system [Muller 87]. DAYDREAMER takes descriptions of events in the world as input and produces a sequence of events it will perform in an imaginary (dream) world. The program plans its dreams by performing an intersection search in a network of rules. In order to find serendipity in a given state, it starts by finding a rule that corresponds to the state. It then tries to find a path between the state and its goals using intersection search. If it finds a connection, a serendipity is recognized.

8.4 Design Adaptation

Innovation, in a given design domain, comes from the ability to use knowledge drawn from sources inside or outside the current domain. In particular, a design will appear novel if it incorporates knowledge which come from sources that seem unrelated to the current design problem. Consequently, a innovative design system should be able to draw ***analogies*** between its target problem and base precedents drawn from a varied set of sources. In addition, the system should have some way of knowing which precedents it should use, it cannot try to draw analogies from all precedents in memory, a context-dependent method for ***retrieval*** is required.

In CYCLOPS we proposed a technique, ***demand posting***, for adapting designs by drawing analogies to previous cases. The demand posting technique consists of two algorithms, ***dependency tracking*** and ***subgoal matching***. Dependency tracking helps identify and retrieve precedents and subgoal matching tries to draw analogies to the precedents retrieved. The demand posting idea is motivated by Osborn's work on Brainstorming [Osborn 53].

8.4.1 Adapting Designs

CYCLOPS adapts designs by drawing analogies to precedents. It's subgoal matching technique draws directly from the Structure Mapping Principle [Gentner 83, Gentner & Toupin 86] and Winston's idea of matching and transferring causal structure from base to target [Winston 80, Winston et al. 83]. Subgoal matching is based on using the causal structure (explanations) of precedents to find analogies without being distracted by irrelevant features in the base precedent.

Another approach to design adaptation is to use modification operators. The innovative design system, PROMPT [Murthy & Addanki 87] debugs designs by performing a qualitative analysis of the problem. It uses modification operators which are either drawn from a pre-defined set or derived by analyzing the governing equations. PROMPT adapts by reasoning from first principles, CYCLOPS adapt by reasoning from past experiences. These approaches could complement each other.

8.4.2 Precedent Retrieval by Asking Questions

Techniques such as Brainstorming [Osborn 53] and Synectics [Gordon 61] are aimed at helping people be creative by inducing them to take different views of a problem and by drawing interesting analogies. In order to make such interesting analogies, one needs to retrieve the right precedents. This can be done by asking questions. Questions serve as cues into memory. It is through the process of posing the right questions and transforming them, that one can retrieve useful precedents. Several techniques for posing questions have been developed: Wishful and fanciful thinking [Rickards 74]; boundary examinations [Rickards 74]; Brainstorming; Synectics; Divergent Thinking [Guilford 59]; and Ideonomy (Wall st. Journal, June 1, 1987). CYCLOPS implements one of the brainstorming techniques in its demand posting algorithm. The program starts by posting a demand question and keeps reformulating the question by drawing upon the causes of the problem, each reformulation retrieves a different set of precedents, which are checked for relevance to the problem at hand. The process continues till such relevant precedents are found.

There are two other systems that use question transformation to retrieve episodes. They are, CYRUS [Kolodner 81] and SWALE [Schank 86]. CYRUS is a question answering program that uses domain knowledge to transform given questions to suit itself. CYRUS can convert a seemingly unanswerable question into a series of questions that are relevant to the data in its episodic memory. SWALE is an understanding system that creatively retrieves and uses precedents. Given a story, the program starts by checking to see if the story is consistent with memory. If not, it retrieves episodes

from memory and tweaks them to help explain the inconsistency. In his book *Explanation Patterns*, Schank provides a long list of heuristics that can be used for question transformation and episode tweaking. All these heuristics have not been incorporated in SWALE yet. CYCLOPS does one kind of question transformation and does not tweak precedents explicitly, it is different from the above systems in that it is a design problem solver.

8.5 Conclusions

CYCLOPS draws ideas from several research efforts from across disciplines. Many of the systems cited in the chapter seem unrelated to one-another when taken independently, but not in the context of design. Design is a complex enough activity which involves a wide range of human faculties and problem solving activities. Future research in design will draw ideas from an even wider range of disciplines and will serve as a forum for integrating diverse ideas.

Chapter 9

Conclusions

We have viewed innovative design as a combination of **exploration** and **adaptation** processes. We have seen how the gradual relaxation of criteria can help designers systematically explore design alternatives in large state spaces. We have also seen how precedents can be used for: recognizing interesting designs, identifying design problems and fixing problems. We examined a technique, demand-posting, for keeping track of design problems, their dependencies and adaptations.

In the previous chapter we reviewed the literature and placed the CYCLOPS framework in relation to the other work in design automation. This chapter starts with a list of the limitations of our current approach. This is followed by suggestions for future research. Later in the chapter a list of possible future applications is presented. The chapter ends with a broader summary of the issues presented in the dissertation.

9.1 Assumptions, Shortcomings and Future Research

The thesis makes several assumptions and the current implementation has several shortcomings. Here is a list of issues that need to be tackled in the future:

Hierarchical approach. CYCLOPS views designs as consistent labeling problems. The representation is flat and at only one level of detail. This makes it very difficult to tackle large problems. For example, a landscape problem which requires locating a hundred houses on a large landscape, the combinatorial explosion could grind any computer down to a halt. On the other hand, if we divided the hundred houses into clusters and grouped clusters into neighborhoods, we could design at different levels of abstraction. One of the biggest challenges in developing a hierarchical algorithm for Consistent Labeling Optimization Problems is to find some way of assuring non-

dominance of generated solutions. This is because, a non-dominated solution at one level of abstraction is not guaranteed to remain non-dominated at lower levels of abstraction.

Exploiting commonalities. When CYCLOPS produces design alternatives it checks them for opportunities and tries to adapt designs which have bugs. The problem is that only a very small percentage of the generated alternatives turn out to be interesting, and consequently, a lot of time and effort gets wasted. One obvious way of alleviating the problem is to have CYCLOPS examine alternatives in parallel. This approach would improve performance to some extent, however, the real problem is more subtle. Many of the alternatives generated by the program have common features, making it unnecessary for the system to examine all the alternatives in full detail. As CYCLOPS does not look for commonalities, it repeatedly discovers the same opportunity in several designs, expending the same amount of effort each time. There are two ways to approach this problem: (a) the program could be modified to learn as it examines designs. It could remember its findings and use them in the future, and thus, avoid duplication of effort or, (b) the program could examine the designs to identify common problems and solve the problems only once. The author feels that the second approach is better, as the first will require too much memory. We need some way of identifying commonalities and solving them at a level of abstraction higher than the individual designs. An added advantage of this approach is improved user interaction. CYCLOPS currently requires users to make tradeoffs among individual designs - this is a slow process. It would be better if the program extracted out commonalities and had the user make tradeoffs at higher levels of abstraction before examining individual designs.

Flat Indexing. CYCLOPS indexes all precedents on the patterns which correspond to the goals and sub-goals in the precedents. The patterns are stored in a flat database requiring excessive search each time a precedent is requested. A network based indexing could help alleviate this problem [Kolodner 80]. Further, we would like the indexing scheme to change in response to new information and new the problem solving contexts. In other words, we would like the memory organization to be dynamic [Schank 82]. One of the major problems is coming up with a dynamic

indexing scheme that can index not only on the attributes of the precedents but also on the relationships among attributes. This allow systematicity-based access to the precedent database

Adapting Precedents to fit problems. CYCLOPS can adapt designs by using precedents. It cannot adapt precedents to designs. We will have to watch the progress of the SWALE project for approaches to precedent adaptation (tweaking) [Schank 86].

Causal & Functional Demand Posting. CYCLOPS adapts designs by tracking the causal dependencies of design problems. It is possible to expand this framework to deal with design functions. A technique called functional demand posting could be developed to solve design problems by tracking design functions and sub-functions. Consequently, a design process may be allowed to switch between causal and functional demand posting as and when required.

Representation. CYCLOPS uses consistent labeling as a way of representing designs. We have found this representation to be limited for complex domains. The CLP representation was useful for representing relaxable criteria which made exploration processes much easier to implement. We will have to work towards a better representation strategy which will help perform complex qualitative simulations for checking designs.

9.2 Future Applications

Applications to other Consistent Labeling Optimization Problems. The CLOP is a fairly general formulation and is applicable to a large range of problems. For example:

- Site Planning. All constraint-based layout problems can be formulated as CLPs.
- Scheduling. In a job-shop situation one has to decide which job, goes where, on which machine, undergoes which operation and at what time period. Each schedule attribute can be treated as a variable in a CLP formulation. Criteria may include items such as resource availability, due dates, cost minimization, idle-time minimization etc.
- Facilities Maintenance Management. A building facility deteriorates over time. In coming up with a maintenance strategy one has to decide, when to maintain, what maintenance action to take, how often and what level of serviceability is acceptable. As the problem often involves non-linear, non-monotonic functions, standard mathematical programming techniques are inapplicable. A CLOP algorithm could be used to find optimal values for the decision variables.

Exploring Alternatives in a Decision Support System. CYCLOPS is able to generate alternatives and identify critical tradeoffs. If the program were used in a decision support environment it could help decision makers play what-if games or locate critical tradeoffs in complex multi-attribute situations.

Learning to design by Discovery. A design that is adapted by CYCLOPS is represented in the same way as precedents it uses. Consequently, each adaptation that CYCLOPS performs can be used as a precedent in future design problems. This capability has not been implemented yet. However, if the problem of precedent indexing and generalization is solved, one could use CYCLOPS to discover designs by setting it off on an unsupervised exploration.

9.3 Final Summary

The thesis draws on ideas from several disciplines and, consequently, makes contributions to many of them:

Engineering Optimization. The Consistent Labeling Optimization Problem Solving methodology (CLOPS) is an interactive, multi-attribute decision making (MADM) technique. As CLOPS uses a branch and bound technique, it allows for evaluation of partial solutions during the solution process. This makes CLOPS more efficient than traditional MADM techniques which generate complete solutions before testing them. Further, as CLOPS works in discrete spaces, it can handle constraints and objectives that are non-monotonic, non-linear or non-continuous. The CLOPS methodology is applicable to a large number of engineering optimization problems that could not be solved by standard mathematical programming techniques.

Artificial Intelligence. The demand posting technique is interesting for its ability to employ multiple precedents or parts of precedents to solve a problem either directly or analogically. It uses the precedents to identify problems, identify opportunities or solve problems analogically. It provides a technique for indexing into memory by posting demands and reformulating the demands when appropriate precedents are not found.

Design Automation/Computer-Aided Design. The idea of exploring design alternatives by criteria relaxation is a new one. It is interesting as it is a way of helping designers identify critical tradeoffs and promising alternatives.

Further, the use of precedents-based reasoning in design is a relatively new idea [Mostow 85]. The thesis develops a method for representing precedents that simultaneously allows for: problem identification, criteria emergence and analogical problem solving.

Evolutionary Epistemology. The study of any human activity by analogy to biological evolution is called evolutionary epistemology. Design can be viewed as an evolutionary process. The artifacts, or products of design we see around us can be thought of as instantiations of the ever-evolving process of design ideas. The synthesis, exploration and adaptation process introduced in this dissertation is a step towards an Evolutionary theory of design.

9.4 Epilogue

In order to innovate one must have an open mind. One must be willing to relax one's constraints in order to examine alternatives. Innovation comes from one's ability to break the rules, to look beyond the norms and to avoid mind-sets. In fact, many of the ideas about constraint relaxation presented in this dissertation are a result of constraint relaxation! As I recollect, the hardest task in applying constraint relaxation to a problem is not the relaxation of the constraints but the identification of the constraints in the first place. We often solve problems making too many assumptions about the world. Our training forces us to impose constraints without being aware of them. For example, consider the following question: "Why do flat mirrors flip images left to right and not up to down?". Many people have problems answering this question as they make assumptions about mirrors which are really not necessary. Computer models such as CYCLOPS, can handle constraints which are explicit in the representation. People, however, can solve problems by extending or changing their view of the problem by switching among representations. We have yet to understand this process.

In the dissertation we argued that relaxing constraints and taking different views of a problem can make us recollect experiences which help in solving problems. The question is, can a computer contribute to this process? Can computers be creativity tools? I believe we can build innovative design tools which will serve as brainstorming assistants. Such programs would work from a large databank of precedents drawn from many different fields.

Innovation is not as mysterious as it is made out to be. We are now beginning to understand some of the processes that underlie innovation and creativity. We have uncovered some components of the process which can now be incorporated in computer programs. It will be long before we can realize our aims, however, current technology is at a stage where useful creativity tools can be developed. This is only a beginning.

Appendix A
The Consistent Labeling Optimization Problem

In this appendix we will take a formal look at the ***PO-A**** and ***OPO-A**** algorithms. We will review other techniques and see how the ***PO-A**** algorithm is developed.

The appendix starts with a formulation of the Consistent Labeling Optimization Problem. This is followed by a description of a CLP algorithm called Forward Checking [Nadel 85a]. After describing the CLP algorithm, a new algorithm for CLOPs is developed. Finally a proof of optimality of the CLOP algorithm is provided.

A.1 Introduction

Over the past decade the consistent labeling problem has received considerable interest in Artificial Intelligence. In particular, machine vision, cryptography and theorem proving. Many algorithms have been developed for solving consistent labeling problems. A majority of these algorithms are based on heuristic search. Techniques such as constraint propagation and forward checking have been used to reduce search effort.

Typically, the *consistent labeling problem* is formulated thus: there is a set of discrete variables, each with a finite set of values from which one is to be chosen. The task is to find labelings for all the variables such that a given set of constraints are simultaneously satisfied. The consistent labeling problem places few requirements on the nature of the variables or the constraints: the constraints can be either symbolic or numeric, monotonic or non-monotonic, continuous or discontinuous, linear or non-linear.

In this appendix we propose a heuristic search technique for handling consistent labeling problems with the added requirement of generating a consistent labeling that optimizes a set of objectives. This new type of formulation is applicable to a wide variety of engineering problems.

This technique will be treated by considering the layout planning problem. The layout problem is formulated thus: there is a set of objects that have to be placed in a designated area where, each object has a finite set of locations (within the designated area) that it can be assigned to. The task is to find unique locations for all the objects such that the given set of constraints and objectives are simultaneously satisfied.

A layout problem involves the placing of objects on a two dimensional[33] space such that a set of inter-object relationships are satisfied. For example, in the Landscape desgin domain, a planner may deal with objects such as houses, parks, shopping-malls etc. While laying out these objects, the planner has to deal with constraints and objectives about soil conditions, visibility, noise, inter-object distance, access etc.

A simple landscape design problem can be formulated thus:

1. There is a set of landuses that have to be sited.
2. Each landuse is to be assigned to a lot of land, chosen from a given set of alternatives.
3. There is a set of constraints governing the labelings.
4. The problem is to find few (or all) ways of assigning landuses to lots such that the constraints are all simultaneously satisfied.

In the above formulation landuses are treated as variables and the lots are treated as values that can be assigned to variables. The act of assigning a value to a variable is known as "labeling" the variable. In Artificial Intelligence (AI) literature, the formulation is known as the Consistent Labeling Problem (CLP).

In general, a CLP is formulated thus: (Adapted from [Nadel 85a])

1. There is a set of variables.
2. Each variable has associated with it, a finite set of values. These sets are known as the domains of the variables.
3. There is a set of constraints on the values that various combinations of variables may compatibly take on, and
4. The goal is to find a few (or all) ways to assign to each variable, a value from its associated domain in such a way that all constraints are simultaneously satisfied.

---

[33]The CLOP formulation is applicable to n-domensinal spaces too. We have chosen a 2-D problem in order to simplify the examples.

A.2 Formulating the Consistent Labeling Optimization Problem

The consistent labeling optimization problem is a modified CLP with the added requirement of having to generate labelings which are optimal over a given set of objectives. The problem formulation introduced in this section is based on the notation introduced by Nadel in [Nadel 85a, Nadel 85b]. A CLOP has four basic components: variables, values, constraints and objectives.

A.2.1 Variables

The CLOP handles discrete variables only. Let Z be a set of n such variables, where $Z = \{ z_1, z_2, \cdots z_n \}$ and $n \in \{2,3,4 \cdots \}$.

A.2.2 Values

Each variable has associated with it, a set of possible values it can take. The set associated with the variable z_i is denoted by d_{z_i} and is called the domain (or range) of the variable. The set of domains, $\mathbf{d} = \{ d_{z_1}, d_{z_2} \cdots d_{z_n} \}$ is prespecified.

A.2.3 Constraints

The set of constraints is denoted by C. A given constraint c_j constrains a set of variables Z_j such that $Z_j \subseteq Z$. The constraint is a relation among the variables in Z_j. This relation is denoted by T_j. This means that T_j is the set of consistent labelings taken from all possible labelings of the variables in Z_j.

In other words, $T_j \subseteq D_j$ where, $D_j = \bigtimes_{z_i \in Z_j} d_{z_i}$, the set of all possible labelings of Z_j.

A constraint has a corresponding function called its Constraint Function (CF_j). The arguments to the constraint function is a labeling of the variables in Z_j. Any such arbitrary labeling of Z_j is denoted by $\overline{Z}_j$. The function call $CF_j(\overline{Z}_j)$ returns a value of either 1 or N, where N is some very-large number $\rightarrow \infty$. Consequently, any $\overline{Z}_j$ with $CF_j(\overline{Z}_j) = 1$ is said to be a consistent labeling on Z_j with respect to c_j. Hence:

$$T_j = \{ \overline{Z}_j \mid CF_j(\overline{Z}_j) = 1 \ \forall_{\overline{Z}_j \in D_j} \} \quad (A.1)$$

In other words, T_j is the set of $\overline{Z}_j$ which are consistent with respect to constraint c_j.

Let us now look at an example of the formulation we have this far. This example will be used to illustrate points throughout the paper. (All references to the example will appear between two horizontal lines as shown below.)

Example

There are three variables (n = 3): $Z = \{ z_1 z_2 z_3 \}$ with domains {1 2}, {1 2 3} and {a b} respectively. There are three constraints c_1 c_2 and c_3. The three constraints are shown below:

Constraint c_1 has CF_1 : $(2z_1)^2 + z_2^2 \geq 18$

As c_1 constrains z_1 & z_2, the range of the constraint is $Z_1=\{z_1,z_2\}$. Given the domains of z_1 and z_2, the value of the function can be calculated in tabular form:

actual	z2 = 1	z2 = 2	z2 = 3
z1 = 1	5	8	13
z1 = 2	17	20	25

The table is converted into CF values based on CF_1:

CF_1	z2 = 1	z2 = 2	z2 = 3
z1 = 1	N	N	N
z1 = 2	N	1	1

The CF_j returns values from the CF values table shown above. For example, the labeling $\overline{Z}_j$ = {1 2} has $CF = N$. The set of legal labelings for $c_j = T_j$ = {(2 2)(2 3)}.

Constraint c_2 has $CF_2 = 10e^{0.1z_1z_2} \geq 13$, the tables:

Actual values:

actual	z2 = 1	z2 = 2	z2 = 3
z1 = 1	11	12	13
z1 = 2	12	15	25

CF values:

CF_2	z2 = 1	z2 = 2	z2 = 3
z1 = 1	N	N	1
z1 = 2	N	1	1

Constraint c_3 has CF_3 expressed directly as a *CF* table:

CF_3	z2 = 1	z2 = 2	z2 = 3
z1 = 1	N	1	N
z1 = 2	1	N	N

A.2.4 Objectives

The set of objectives is denoted by $O=\{o_1,o_2 \cdots o_3\}$. A particular objective o_h has a set of variables that it acts on Z_h. Where $Z_h \subseteq Z$. The objective is characterized by a relation T_h on the variables in Z_h. Which means that T_h is the set of optimal labelings taken from D_h where $D_h=\bigtimes_{z_i \in Z_h} d_{z_i}$, the set of all possible labelings of Z_h. Objectives are represented in the same way as constraints[34]. The relation imposed by an objective on D_h is similar to that of a constraint, in that, T_h is *constrained* to be optimal.

[34]We will relax this definition later

Example

There is one objective, o_4 in our example:

Objective o_4 has CF_4: Maximize $Sine(40z_1+z_2)$. The corresponding actual value and CF value tables are:

Actual Values:

actual	z2 = 1	z2 = 2	z2 = 3
z1 = 1	.72	.72	.73
z1 = 2	.99	.99	.99

CF values:

CF_4	z2 = 1	z2 = 2	z2 = 3
z1 = 1	N	N	N
z1 = 2	1	1	1

A.2.5 A Characterization of T_j and T

Let $\overline{Z}$ denote any arbitrary labeling of the variables in Z. $\overline{Z}$ is a set of values for each variable z_i chosen from the variable's respective domain. Any n-tuple $\overline{Z} \in D = \bigtimes_{z_i \in Z} d_{z_i}$ can be either consistent or not. For a labeling $\overline{Z}$ to be consistent it should return value 1 from the Constraint Function of all the constraints and objectives. This is done by looping through all the CFs and passing them the right arguments taken from $\overline{Z}$. For example, if $\overline{Z} = \{\overline{z}_1,\overline{z}_2,\overline{z}_3,\overline{z}_4\}$ and Z_j of $c_j = \{z_2,z_4\}$ then we have to extract the second and fourth values from $\overline{Z}$ to serve as inputs to CF_j. This action is denoted by $\overline{Z}_j\langle\overline{Z}\rangle$, which is called the projection of $\overline{Z}$ onto $\overline{Z}_j$. In our example, $\overline{Z}_j\langle\overline{Z}\rangle = \{\overline{z}_2\,\overline{z}_4\}$.

Consequently, the global set of consistent labelings T is given by:

$$T = \{\, \overline{Z} \mid \overline{Z} \in D \wedge \overline{Z}_j\langle\overline{Z}\rangle \in T_j \quad \forall_{j \in J_1{}^c} \,\} \tag{A.2}$$

Where, c is the total number of constraints (m) and objectives (h) and is given by: $c = (m + h)$. J_1^c is the set of all integers from 1 to c, the total number of CF's in the CLOP.

A.3 Solving CLPs with the Forward Checking Algorithm

Before describing how CLOPS are solved we will first review a technique for solving CLPs. The technique is called Forward Checking and was suggested by Haralick and Elliot [Haralick and Elliot 80]. The notation used here is from [Nadel 85d].

We must first define the terms we will be using to perform and characterize the search for consistent, optimal labelings. For the sake of simplicity, it is assumed that the variables in Z are instantiated in numerical order. (The problem of finding a suitable instantiation ordering is in itself an NP-complete problem and is not examined here.)

A.3.1 Searching the State Space

Let X denote the global instantiation order, where $X=\{x_1,x_2,\cdots x_n\}$ which is the same order as in Z. This could be any random order, as there are !**n** possible orderings.

The process of searching the space starts by instantiating the variables in the order $x_1,x_2,\cdots$ till x_n. Each instantiation corresponds to a stage in the search tree. The stage number is denoted by k. The variables that have been instantiated at stage k are denoted by X_k, which is given by:

$$X_k=\{x_1,x_2,\cdots x_k\} \tag{A.3}$$

The variables in Z that have yet to be instantiated are called Future Variables and are denoted by F_k. The search progresses from stage to stage by concatenating the domains of variables to partial labelings generated by the previous stages. This is denoted by a concatenation operator: ||. For example if there is a list $A = (abcdef)$ then $A \| q$ gives the new list $(abcdefq)$. Correspondingly, $\overline{X}_k = \overline{X}_{k-1} \| d_{\overline{x}_k}$. Where $d_{\overline{x}_k}$ is the domain of variable x_k. This is best explained by an example.

Example:

In the example, $Z = \{z_1, z_2, z_3\}$. If we follow the same instantiation order in X, the search starts with the domain of z_1 which is {1 2}. The first branching produces two alternate partial labelings: { (1) (2) } which are derived from d_{z_1}

At this stage $X_{k=1} = \{z_1\}$ *and* $F_{k=1} = \{z_2\ z_3\}$. The next stage involves the instantiation of the second variable z_2 with domain {1 2 3}. These values are concatenated with the initial partial solutions shown above:

{ (1 1) (1 2) (1 3) (2 1) (2 2) (2 3) } $= d_{z_1} \times d_{z_2}$

with $X_2 = \{z_1 z_2\}$ and $F_2 = \{ z_3 \}$. The third stage of the search gives:

$d_{z_1} \times d_{z_2} \times d_{z_3} =$

{(1 1 *a*) (1 2 *a*) (1 3 *a*) (2 1 *a*) (2 2 *a*) (2 3 *a*)
(1 1 *b*) (1 2 *b*) (1 3 *b*) (2 1 *b*) (2 2 *b*) (2 3 *b*)}

where$X_3 = \{z_1\ z_2\ z_3\}$ and $F_2 = \varnothing$

A.3.2 Forward Checking and Search

In the example above, we saw how search can be used to enumerate $\overline{Z}$. It is possible to scan $\overline{Z}$ to find all the consistent labelings with respect to the constraints C and objectives O. This approach, however, is too inefficient. The forward checking algorithm propagates constraints during search. This is shown in the example below:

Example

Consider, for a moment, the search tree at stage k = 1, where $X_k = \{z_1\}$ and the tree has only two partial labelings {(1)(2)}.

Consider the first labeling: $\overline{X}_k^1 = (1)$. Setting variable $z_1 = 1$ gives us a clue as to what values

it's possible for z_2 can take. This is done by examining the constraints' CF value tables. Take for example the constraint c_1 which has $Z_1=\{z_1,z_2\}$ and $T_1=\{(22)(23)\}$. Clearly, setting z_1 to unity precludes z_2 from taking any of the values from its domain. This means that there is no consistent labeling with $z_1=1$. Consequently, the first labeling is dropped from the search tree, in effect, eliminating half the search tree.

The idea of using a constraint to constrain the values of future variables in F_k based on values chosen for variables in X_k is called Forward Checking (it can also be viewed as a kind of Constraint Propagation).

At any stage of the search k, there is a set of instantiated variables X_k and a set of future variables F_k. Forward checking is the process of filtering out inconsistent values from the domains of the future variables by checking the values of $x_i \in X_k$ against the constraints and objectives. In other words, the original domain d_f of some future variable $f \in F_k$ is subject to filtering at stage k. The filtering is done with respect to a branch of the search tree. Such a branch is denoted as $\overline{X}_k$. The filtering of future variables is done separately for each branch of the search tree. If $\overline{X}_k$ denotes a particular branch (partial labeling on X), then, the filtered domain of future variable f with respect to labeling $\overline{X}_k$ is denoted by $d_f^{\overline{X}_k}$, and is given by:

$$d_f^{\overline{X}_k} = \left\{ \overline{f} \mid \overline{f} \in d_f \quad \bigwedge \quad \overline{Z}_j \langle \overline{X}_k \| \overline{f} \rangle \in T_j \quad \forall_{j \in \Psi_{X_k \cup \{f\}}} \right\} \tag{A.4}$$

where $\overline{f}$ is a value from the domain of f and, where Ψ_A is defined as the set of all constraints that are applicable to the variables in set A. Where:

$$\Psi_A = \{ j \mid 1 \le j < (c+h) \text{ and } Z_j \subseteq A \} \tag{A.5}$$

In the definition of $d_f^{\overline{X}_k}$, Ψ is taken over all constraints and objectives $(c+h)$.

The definition of $d_f^{\overline{X}_k}$, reads thus: it is the set of all values in d_f that pass the filtering test. This means that if f was to be the next variable for instantiation, then only the members of $d_f^{\overline{X}_k}$ will

survive all the applicable constraints and objectives. When this filtering is applied to all f in F_k we get the filtered set of future domains:

$$\mathbf{d}^{\overline{X}_k} = \{ d_f^{\overline{X}_k} | \ \forall_{f \in F_k} \ \} \tag{A.6}$$

A.3.3 The Forward Checking Algorithm

The forward checking algorithm has two basic steps: instantiation and forward-checking of future variables.

The algorithm is denoted by FORW-CHECK and takes two arguments, a labeling and the current variable being instantiated. The algorithm starts with k = 0, $X_k = 0$ and $F_k = Z$:-

FORW-CHECK ($\overline{X}_k$, k)

1 IF k = n THEN return { $\overline{X}_k$ }

2 Compute $\mathbf{d}^{\overline{X}_k}$

3 IF (k ≤ n and ∅ ∈ $\mathbf{d}^{\overline{X}_k}$) THEN return ∅

ELSE return { UNION of FORW-CHECK($\overline{X}_k \| \overline{x}_{k+1}$ $k+1$) } (taken over all $\overline{x}_{k+1}$ in the filtered set $d_{x_{k+1}}^{\overline{X}}$)

The algorithm starts with the call FORW-CHECK({∅}, 0). The algorithm returns all the consistent and optimal labelings in the state space of labelings.

Example

Let us now return to the example to illustrate FORW-CHECK. Let S-TREEE denote the set of all partial and complete labelings being considered at ony stage of the search.

Stage 0 k = 0 and n = 3, X_n = { ∅ } $F_0 = Z$

S-TREE = {∅}

$d^{\overline{X}_k} = \{(1\,2)(1\,2\,3)(a\,b)\}$

Stage 1 k = 1, n = 3, $X_k = \{ z_1 \}$ $F_1 = \{ z_2\, z_3 \}$

S-TREE = $\{(1)(2)\}$

for $X_k = (1)$, $d^{\overline{X}_k} = \{ \varnothing\, (a\,b) \}$

for $X_k = (2)$, $d^{\overline{X}_k} = \{(2\,3)\,(a\,b)\}$[35]

Stage 2 k = 2, n = 3, $X_k = \{z_1\, z_2\}$ $F_2 = \{ z_3 \}$

S-TREE = $\{(2\,2)(2\,3)\}$

for $X_k = (2\,2)$, $d^{\overline{X}_k} = \{(a)\}$

for $X_k = (2\,3)$, $d^{\overline{X}_k} = \{\varnothing\}$

Stage 3 k = 3, n = 3, $X_k = Z$ and $F_n = \{\varnothing\}$

The labeling (2 2 a) is returned as the only consistent labeling in the state space of labelings.

This section has introduced and illustrated the forward checking algorithm for solving CLPs. We also saw how a CLOP can be represented with binary constraints and objectives and can then be solved by forward checking. The problem with using this approach is that very few CLOPs have perfect solutions. Multi-objective problems often tend to be over-constrained, there are usually no solutions that satisfy all constraints and optimize all objectives simultaneously. CLOPs often have to be solved by relaxing the constraints and by making tradeoffs among the objectives. In the follow sections we will examine an algorithm for solving CLOPs.

---

[35]We get the set (a b) because only arc consistency is maintained in this example, no path consistency is checked [Mackworth 77].

A.4 Solving the CLOP

In this section we will show how over-constrained problems are dealt with. Problems that have many constraints and many objectives are liable to be over constrained. The only way a solution can be found is through the relaxation of the constraints and objectives. Formally, an over-constrained situation is said to have occurred when at some some of the search k, such that k < n:

$$\bigcup_{f \in F_k} d_f^{\overline{X}_k} = \{ \varnothing \} \text{ for all } \overline{X}_k \text{ in } S\text{-}TREE \ .$$

Overconstrained problems are solved by relaxing the constraints and objectives. In order to be able to do so, we need a way of representing the constraints and objectives which will allow relaxation. This is discussed below.

A.4.1 Representing Relaxable Constraints and Objectives

From hereafter we will refer to both constraints and objectives as criteria. A criterion is denoted by ω_j and, the set of all criteria is Ω where $|\Omega| = (c+h)$, the total number of constraints and objectives. Each criterion ω_j has an associated function CF_j with a corresponding value matrix. As before, Z_j is the set of variables that ω_j acts on.

In the previous section we defined a consistent labeling on Z as a labeling which obtained CF value of 1 for all the criteria in Ω , we will relax this requirement now.

Definition. Let $S_{\overline{Z}}$ be a set that represents the CF values obtained for all the criterion by the labeling $\overline{Z}$.

$$S_{\overline{Z}} = \left\{ r_{\overline{Z},1}, r_{\overline{Z},2}, \cdots r_{\overline{Z},(c+h)} \right\} \tag{A.7}$$

Where, $r_{\overline{Z},j}$ is the CF value of the j^{th} criterion for the labeling $\overline{Z}$. The value $r_{\overline{Z},j}$ is called the rank of the j^{th} criterion and S is called the *spectrum* of $\overline{Z}$. A labeling is consistent and optimal if it is complete over Z and has all ranks = 1.

Relaxing CF_j. Each criterion ω_j has a corresponding function CF_J . The function call

$CF_j(\overline{Z}_j)$ returns a value in the set J_1^N, where N is some arbitrarily large number. This value is called the rank of the j^{th} criterion. Although $\overline{Z}_j$ which obtains ranks of 1 for all the criteria continues to be the best possible labeling, other $\overline{Z}_j$'s that obtain $r_{\overline{Z}_j,j}$ greater than 1 are considered to be incompatible with the original ω_j but compatible with relaxations of the criterion ω_j.

Optimality as non-dominance. We need some way of differentiating among the spectra of different $\overline{X}_k$ at some level k. If there are some spectra with all ranks = 1, then clearly, these partial solutions are the best possible and should be selected for future instantiation. However, we may have a situation where all the spectra are suboptimal. For this reason we need a measure of suboptimality. For example, if we had three spectra $S^1 = \{2\,3\,2\}$, $S^2 = \{3\,6\,5\}$ $S^3 = \{1\,5\,3\}$. It can be seen that S^1 dominates S^2 with respect to all the criteria and that's because it has better ranks in its spectrum. All non-dominated labelings are said to be Pareto Optimal.

Representing the search tree. At any stage k there may be several labelings of X_k. The set of all labelings at any level of the search is stored in a search tree and is denoted by S-TREE$_k$. The p^{th} labeling in this set is denoted by $\overline{X}_k^p$. Any particular labeling $\overline{X}_k^p$ is said to be pareto optimal if there is no other labeling that has all ranks in its spectrum better than the ranks in $S_{\overline{X}_k^p}$.

Differentiating among non-dominated solutions. While pareto-optimality can be used as a check for consistency it is not a good enough discriminator. For example, consider the following three spectra {1 2 3}, {19 1 14} and {2 2 4}. By pareto-optimality criterion alone, the first and second labelings are pareto optimal. It is clear however, that the second spectrum is much worse than the first one, a better measure of "goodness" among pareto optimal labelings is required.

It should be noted however, that there is no objective way of choosing among pareto-optimal solutions. This is because choosing one over the other requires making a tradeoff. For now we shall leave all such tradeoffs to an outside agent (user). We may, however, attempt to use some method of discriminating among tradeoff alternatives that are more middle-of-the-road, i.e. have less extreme ranks in their spectra.

This is be done by defining a measure of deviation (Δ) from the very best spectrum that has all ranks equal to unity. For any given labeling $\overline{Z}$ the total deviation is denoted by:

$$\Delta_{\overline{Z}} = \sum_{j=1}^{(c+h)} W_j \left(r_{\overline{x},j} - 1 \right) \tag{A.8}$$

Where W_j is the weight of the j^{th} criterion. For simplicity, we will assume equal weighting for all criterion.

The deviation measure just defined has these properties: A $\Delta_{\overline{Z}} = 0$ means that $\overline{Z}$ is a true optimal labeling. Any $\Delta_{\overline{Z}} > 0$ means that $\overline{Z}$ is not the best solution and that if it is accepted as a solution, in effect, means that the governing criteria are being relaxed.

We are now in a position to redefine the method for finding an optimal labeling. Given a set of complete labelings, the method used for selection of the optimal will now be two staged. First, the pareto-optimal set will be extracted and then their Δ's calculated. The solutions with lowest Δ are given maximum importance, which means they are considered as tradeoff alternatives before the other pareto-optimal solutions are considered. It is important to note, however, that non-dominated solutions with high Δ values are never discarded but are only examined later in the search.

A.4.2 Determining the Ranks for partial labelings

In the previous section we defined $\Delta_{\overline{Z}}$ in terms of the ranks in $\overline{Z}$'s spectrum. A particular rank $r_{\overline{Z},j}$ is given by the function call to CF_j with the argument $\overline{Z}_j \langle \overline{Z} \rangle$. The question is: How does one evaluate the rank of a partial labelings that has insufficient data for a given CF_j ? If, for example, $\exists z_i \; [z_i \in Z_j \wedge z_i \notin X_k]$ then it is not possible to get $\overline{Z}_j \langle \overline{X}_k \rangle$. For example, if $Z_j = \{z_1 z_3 z_5\}$ and $X_k = \{z_1 z_2 z_3\}$ with $F_k = \{z_4 z_5 z_6\}$, then evaluating CF_j requires that we know the value of z_5 which is actually a future variable.

We can, at best, make an estimate of the unknown ranks. One approach to estimating the rank of a partial labeling is to find the ranks of all possible completions of the partial labeling and choose the minimum. This can be done without actually searching space of labelings but by searching the

CF table for a minimum value. For example, if variable $f \in Z_j$ and $f \notin X_k$ then it's rank can be estimated as:

$$r_{\overline{X}_k j} = MIN \left[CF_j(\overline{Z}_j \langle \overline{X}_k \| \overline{f}_1 \rangle), CF_j(\overline{Z}_j \langle \overline{X}_k \| \overline{f}_2 \rangle), \cdots CF_j(\overline{Z}_j \langle \overline{X}_k \| \overline{f}_3 \rangle) \right]$$

where $d_f^{X_k} = \{\overline{f}_1 \overline{f}_1 \overline{f}_2 \cdots \overline{f}_i\}$ (A.9)

Note that the filtered (forward checked) domain is used for the future variable f. Further, the above equation assumes that X_k is short of Z_j by only one future variable.

At this point, we will go back to our continuing example. In the following section we will see why the estimating heuristic used above, is guaranteed to lead to optimal solutions. Readers who feel the example is unnecessary may safely skip it.

Example

In our example, we have a total of five criteria (three constraints and two objectives). The first step is to convert the criteria into a relaxable form

For example: CF_1 is : $(2z_1)^2 + z_2^2 \geq 18$

The values obtained for different combinations of z_1 and z_2 are:

actual	z2 = 1	z2 = 2	z2 = 3
z1 = 1	5	8	13
z1 = 2	17	20	25

This matrix can be ranked as shown below:

CF_1	z2 = 1	z2 = 2	z2 = 3
z1 = 1	N	N	N
z1 = 2	2	1	1

where, the labeling ($z_1 = 2$, $z_2 = 1$) is given a rank of 2 because the value 17 is only close to the best. The determination of the rank is a different problem. There are several informal ways of determining the rank of sub-optimal labelings. In this appendix we will assume that ranks are input at the discretion of the user. Using this technique, all the other CF matrices are changed as shown below:

Ranks for CF_2:

CF_2	z2 = 1	z2 = 2	z2 = 3
z1 = 1	N	2	1
z1 = 2	2	1	1

Ranks for CF_3 (chosen arbitrarily):

CF_3	z2 = 1	z2 = 2	z2 = 3
z1 = 1	N	3	N
z1 = 2	1	1	N

Ranks for CF_4:

CF_4	z2 = 1	z2 = 2	z2 = 3
z1 = 1	N	N	N
z1 = 2	1	1	1

Ranks for CF_5:

CF_5	z2 = 1	z2 = 2	z2 = 3
z1 = 1	N	N	N
z1 = 2	1	2	3

The spectrum of any labeling will have the ranks for CF_1 to CF_5

Let us work through the staged search process

k = 0, n = 3

$X_k = \{\varnothing\}$ $F_k = \{z_1\ z_2\ z_3\}$

S–TREE = $\{\varnothing\}$

k = 1, n = 3

$X_k = \{z_1\}$

S–TREE = $\{(1)(2)\}$ = $\{\overline{X}^1{}_k\ \overline{X}^2{}_k\}$

The first step is to forward check[36] with respect to each $\overline{X}_k$ in *S–TREE*. $\overline{X}^1{}_k$ and CF_1 lead

[36]Definition of forward checking is not changed

to a $\{\varnothing\}$ for the future variable z_2, hence $\overline{X}^1_k$ is eliminated. Turning to $\overline{X}^2_k$, we get no changes through forward checking, consequently:

S–TREE = $\{(2)\}$

The next step is to find the spectrum, which is:

$S_{\overline{X}_k{}^2} = \{11111\}$

The spectrum is calculated for each *CF* by looking for the lowest rank possible given the condition that z_1 is already set to 2.

<u>k = 2, n = 3</u>

$X_k = \{z_1 z_2\}$

S–TREE = $\{(21)(22)(23)\} = \{\overline{X}^1{}_k \overline{X}^1{}_k \overline{X}^1{}_k\}$

$d^{\overline{X}^1_k}_{z_3} = \{b\}$, $S_{\overline{X}^1_k} = \{22111\}$

$d^{\overline{X}^2_k}_{z_3} = \{ab\}$, $S_{\overline{X}^2_k} = \{11112\}$

$d^{\overline{X}^3_k}_{z_3} = \{b\}$, $S_{\overline{X}^3_k} = \{$no use calculating this$\}$

as $\overline{X}^1{}_k$ and $\overline{X}^2{}_k$ are both pareto-optimal, their deviations are calculated as $\Delta_{\overline{X}^1{}_k} = 2$ and $\Delta_{\overline{X}^2{}_k} = 1$

$\overline{X}^2_k$ for further expansion:

<u>k = 3, n = 3</u>

S–TREE = $\{(21)(22a)(22b)\} = \{\overline{X}^1_k\ \overline{X}^2_k\ \overline{X}^3_k\}$

$\overline{X}^1_k$ is the incomplete pareto optimal solution. As long as it is not instantiated further, it will remain pareto optimal (by Theorem1, below). The Δ values for the the three labelings in S-TREE are: 2, 3 and 1 respectively. With the second labeling turning out to be pareto inferior over S-TREE. Finally $\overline{X}^2_k$ emerges as the best labeling, it is pareto optimal and has the lowest Δ.

In the example above, we saw how certain partial solutions, even though they are pareto-optimal, do not get instantiated due to high Δ values. This however, does not eliminate these solutions from S-TREE. Such solutions are guaranteed to remain pareto-optimal as long as they are not instantiated further. This issue is examined below:

Lemma1 In a partial labeling $\overline{X}_k$ estimated rank of a criterion ω_i: $r_{\overline{X}_{k,j}}$ is liable to remain constant or increase in further instantiations of the partial labeling $\overline{X}_k$. If and only if $r_{\overline{X}_{k,j}}$ is estimated using equation (A.9).

Proof Consider a partial labeling over the variables X_k . If the rank of criterion ω_j for labelings $\overline{X}_k$ when $\exists\, [f | f \in Z_j \wedge f \notin X_k]$ then, an estimate is needed. As f is the only variable left, any arbitrary expansion of $\overline{X}_k$ is given by:

$$\overline{X}_{k+1} = (\overline{X}_k \| \overline{f}) \text{ where } \overline{f} \in d_f^{\overline{X}_k}$$

The set of all such labelings (children of $\overline{X}_k$) is denoted by $\{\overline{X}_{k+1}\} \subseteq S\text{–}TREE_{k+1}$. In ***Lemma1*** we assumed that the ranks will be calculated using equation (A.9). This means that $r_{\overline{X}_k}$ is the minimum of the ranks in all $\overline{X}^i_{k+1} \in \{\overline{X}_{k+1}\}$. Consequently,

$$\forall_{\overline{X}^i_{k+1} \in \{\overline{X}_{k+1}\}} \left[r_{\overline{X}_{k,j}} \leq r_{\overline{X}_{k+1,j}} \right]$$

where, $\overline{X}^i_{k+1}$ is a child of node of $\overline{X}^j_k$, and where i and j are any valid integer.

QED

Corollary to Lemma1: An estimated rank $r_{\overline{X}_{k,j}}$, once estimated using (A.9), can never reduce in value.

Theorem1 A partial labeling $\overline{X}_k$ that is pareto-optimal in the set *S–TREE* will remain pareto-optimal as long as it is not instantiated further. This is regardless of the number of instantiations carried out on the other $\overline{X}$ in the *S–TREE*.

Proof Let there be a partial labeling $\overline{X}_k^1 \in S\text{–}TREE$. Let $\overline{X}_k^1$ be among the pareto-optimal labelings.

Let us assume that $\overline{X}_k^1$ is kept unchanged while instantiations are made on other pareto-optimal labelings.

If $\overline{X}_k^1$ is to be dominated by some future instantiation: $\overline{X}_{k+\delta}^i \in S\text{–}TREE$ (where, $i \neq 1$ and δ is some arbitrary number of instantiations, such that $k + \delta \leq$ the total number of possible instantiations) then, the ranks in the spectrum of $\overline{X}_k^i$ have to reduce in value. This violates the Corollary to ***Lemma1*** and hence, by contradiction, $\overline{X}_k^1$ cannot be dominated by $\overline{X}_{k+\delta}^i \in S\text{–}TREE$.

QED

A.4.3 CLOPS: the CLOP Solver

This subsection formalizes the algorithm for solving CLOPs. The algorithm has two important characteristics:

- Use of pareto optimality and deviation Δ as the measure of optimality, and
- use of relaxable criteria.

The algorithm is called CLOPS and is given by:

1. Put the start node in a list called *S–TREE*
2. If *S–TREE* = $\varnothing$, exist with $\varnothing$, ELSE continue
3. Calculate the spectra of each labeling in S-TREE, using (A.9) to calculate ranks that need to be estimated
4. Select all Pareto_optimal Solutions in *S–TREE* and put them in the list *PO*. Put the rest in $\neg PO$.
5. Sort *PO* by Δ
6. Set Δ_{best} = lowest Δ in *PO*
7. Select all the labelings in *PO* with $\Delta = \Delta_{best}$ and put them in *PO′*. Put the rest in *PO″*
8. Set *S–TREE* = { $PO' \| PO'' \| \neg PO$ }
9. If *PO′* = $\varnothing$ GOTO Step 6
10. If $\exists\, \overline{x}^i \in PO'$ such that $\overline{x}^i$ is a complete labeling, return all such $\overline{x}^i$ and STOP.

11. For all labelings $\overline{PO}' \in PO'$ expand them one more level in their respective search trees. The new set of labelings is denoted by PO'_{new} .
12. Forward check all the future variables of all the $\overline{PO}'_{new} \in PO'_{new}$
13. IF for any $\overline{PO}'_{new}$, $\emptyset \in d^{\overline{PO}'_{new}}$, where $d^{\overline{PO}'_{new}}$ is the set of domains of the future variables of the labeling $\overline{PO}'_{new}$

    THEN eliminate $\overline{PO}'_{new}$ from PO'_{new}.
14. Set *S–TREE* = $\{ PO'_{new} \| PO'' \| \neg PO \}$
15. Go to Step 2

A.4.4 Proof of Optimality of a CLOPS-type algorithm

In this section we wil prove the optimality of the CLOPS algorithm. The proof is based on that provided for the A^* algorithm [Hart, Nilsson & Raphael 68, Pearl 84] The CLOPS algorithm is essentially composed of two parts:

1. An instantiator (INSTAN) that expands nodes which are given to it, and
2. A selector (SELECTOR) that selects from among all nodes in S-TREE and passes the selected nodes to INSTAN.

The process continues till a complete labeling gets selected by SELECTOR. The selection process in CLOPS uses pareto optimality coupled with a measure of deviation from ideal (Δ). Let SEL denote part of the selection function which knows how to select non-dominated solutions.

In this section we will prove why SEL, even though it is applied only to partial labelings, will lead to a global pareto-optimal solution. To simplify matters, we will examine a simple version of the CLOPS algorithm: CLOPS' . This new algorithm concentrates only on the SEL function and does not have forward checking and does not use deviations from the ideal as a selection criterion.

The CLOPS' algorithm:

1. Put the start node in *S–TREE*, set the list *ANSWERS* = $\emptyset$.
2. IF *S–TREE* = $\emptyset$ THEN exit with $\emptyset$, ELSE continue
3. Calculate the spectrum for each labeling in *S–TREE* using Equation (A.9) to calculate ranks that need to be estimated
4. Use SEL to select all pareto optimal labelings in *S–TREE* by comparing *S–TREE* over

the set { *S–TREE* || *ANSWERS* }. Put the selected labelings in *PO*. Put the rest in $\neg PO$

5. Select all the complete labelings in *PO* and transfer them to the list *ANSWERS*. IF any answers are found and IF others are not needed, THEN STOP, ELSE, IF more solutions are needed, THEN continue on to the next step.
6. IF $PO = \varnothing$, THEN return *ANSWERS* and STOP, ELSE continue
7. Expand the labelings in *PO* to give PO_{new}
8. Set *S–TREE* = { PO_{new} || $\neg PO$ }
9. GOTO step 2

Definition. Let *B* denote the beginning root node and let Γ denote the set of pareto-optimal complete labelings on *Z*.

Definition. Let *S–TREE* denote the set of all labelings (partial or complete) in the search tree.

Definition. Let *PO* denote the set of pareto optimal labelings in *S–TREE*.

Definition. Let the nodes (labelings) in the search tree are denoted by *N*. The i^{th} node in *N* is denoted by N_i. If node N_i is accessable from node N_j, the set of all paths between the two nodes is denoted by $PP_{N_i-N_j}$. Let $PP_{N-\Gamma}$ denote the set of paths going from any arbitrary node N to the goal set Γ.

Definition. Let $PO(T)$ denote a function that extracts the pareto optimal labeling from a set of labelings T.

Definition. Let $\uparrow$, $\sim$ and $\downarrow$ be indicators of pareto superiority, pareto non-dominance and pareto inferiority respectively. These three relations are defined below:

If there is a labeling α and a set of mutually pareto optimal labelings β, then:

IF $PO(\alpha \| \beta) = (\alpha \| \beta)$ THEN, α is said to be pareto equivalent over β. This is denoted as $\alpha \sim \{\beta\}$.

IF $PO(\alpha \| \beta) = \beta$ THEN, α is said to be pareto inferior over β. This is denoted as $\alpha \downarrow \{\beta\}$.

IF $PO(\alpha \| \beta) = \alpha$ THEN, $\alpha \uparrow \{\beta\}$.

IF α is either pareto non-dominated or superior over β, THEN the relation is denoted as: $\alpha \approx \{\beta\}$. Formally the definition is given by:

$$\{\alpha\} \approx \{\beta\} \rightarrow \{\alpha\} \sim \{\beta\} \vee \{\alpha\} \uparrow \{\beta\}$$

Here are some equivalences with will be useful later in this section.

$$\{\alpha\} \uparrow \{\beta\} \wedge \{\beta\} \uparrow \{\delta\} \rightarrow \{\alpha\} \uparrow \{\delta\} \quad \text{(A.10)}$$

$$\{\alpha\} \approx \{\beta\} \wedge \{\beta\} \approx \{\delta\} \rightarrow \{\alpha\} \approx \{\delta\} \quad \text{(A.11)}$$

$$\{\alpha\} \uparrow \{\beta\} \rightarrow \{\alpha\} \approx \{\delta\} \quad \text{(A.12)}$$

A.4.5 Proof of Optimality of CLOP'

Theorem2. The first complete solution that CLOPS' will terminate with is guaranteed to be pareto optimal over the entire state space of *Z*.

Proof Suppose CLOPS' terminates with a solution l such that $l \downarrow \{\Gamma\}$.

From the definition of CLOPS' we know that the algorithm terminates whenever a node it selects for expansion turns out to be complete. The algorithm only selects those nodes which are non-dominated over the entire *S–TREE*. Consequently, if $l \downarrow \{\Gamma\}$ and $l \sim \{S\text{–}TREE\}$, then $S\text{–}TREE \downarrow \{\Gamma\}$.

In other words, $\forall_{\overline{X} \in S\text{–}TREE} [\overline{X} \downarrow \Gamma]$.

However, as Γ is in the state space, it has to be derived by instantiating nodes in *S–TREE*, and as $S\text{–}TREE \downarrow \{\Gamma\}$, it follows that:

$\exists \overline{X}_k \in S\text{–}TREE$ such that one of its decendents $\overline{X}_n$ is a member of Γ, where, $n \geq k$

It then follows that: $\overline{X}_n \uparrow \overline{X}_k$ where, $n \geq k$

This violates ***Theorem1*** which states that none of the instantiations of a node can be pareto superior to the node. Hence, by contradition, CLOPS' will terminate with a solution which is pareto-optimal over Γ.

Q.E.D.

A.5 Conclusions

The CLOPS algorithm is applicable to a large number of engineering problems that involve a mix of constraints and objectives that may be non-linear, non-monotonic and even non-continuous. The algorithm is a modified form of the A^* algorithm. CLOPS is different in that, it is applicable to multi-objective problems and uses non-dominance as a measure of optimality.

Appendix B

A trace of CYCLOPS in action

In this chapter the workings of the program CYCLOPS is presented. CYCLOPS implements the techniques presented in the thesis.

This Appendix has two major parts. The first part starts with a description of CYCLOPS and shows how data and knowledge is input into the system. The second part (Page 178) is a trace of the program.

B.1 CYCLOPS Architecture

The CYCLOPS architecture is shown in Figure B-1.

The **user** plays a central role in the CYCLOPS framework. The **user** may be either a program or a human or both working together. The **user** watches designs as they are being detailed and developed and can guide the program through the design process.

The **user** can do several things:

- accept or discard a design, or
- invoke the adaptation process, or
- invoke the exploration module, or just
- let the normal search process continue.

The normal search process involves branching and selection. These two steps are carried out by the **synthesizer** and the **selector**. The synthesizer takes partial designs and adds detail to them by instantiating their variables. The selector performs two functions. The selector first hands off the design to a precedent manager. This is done to check for any new problems or opportunities in the partial design. Next, the selector performs a pareto optimality check. It decides to place a design either in the active list or the dormant list.

The user can also invoke the **explorer**. This module simply relaxes criteria and hands off, to

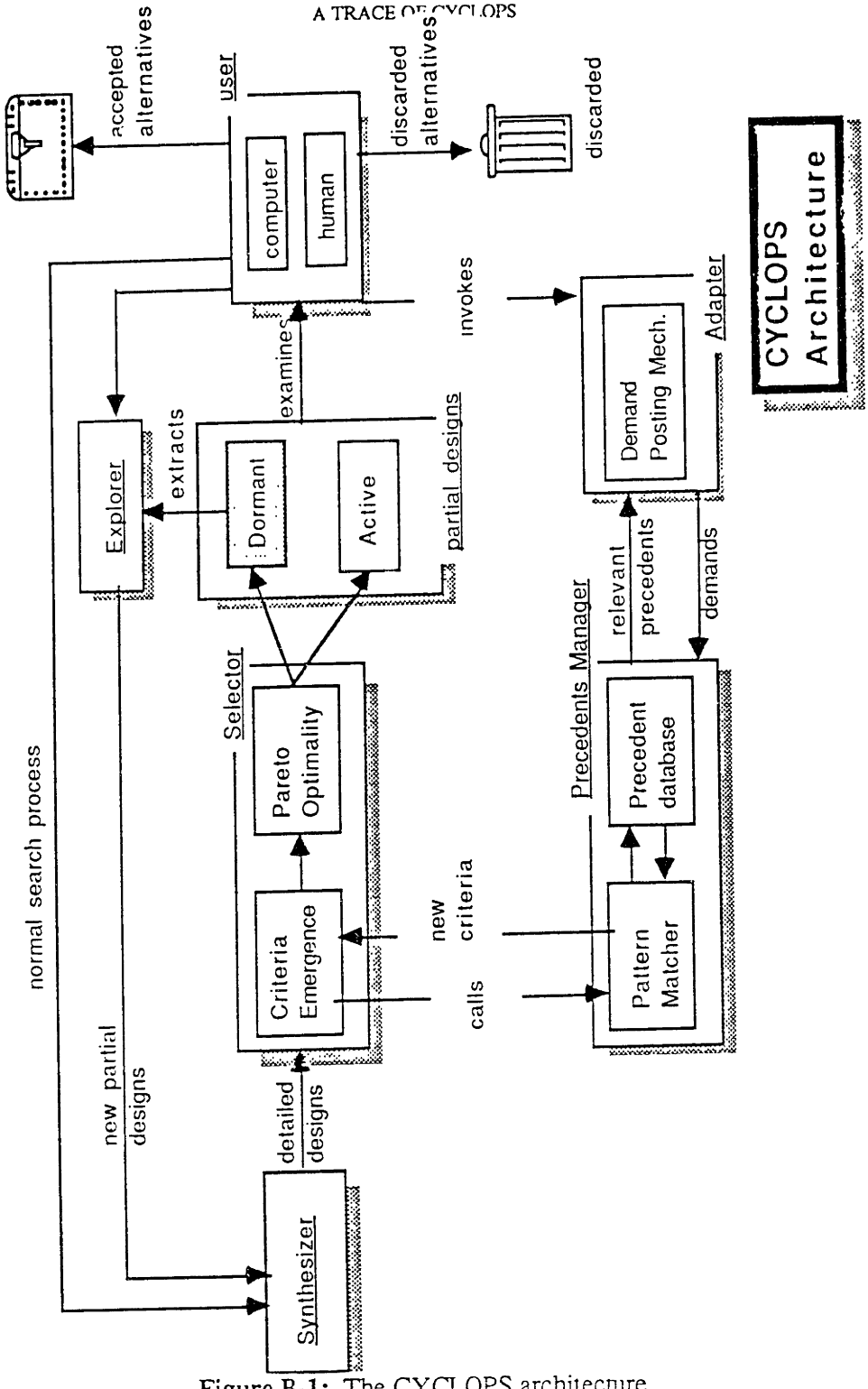

Figure B-1: The CYCLOPS architecture

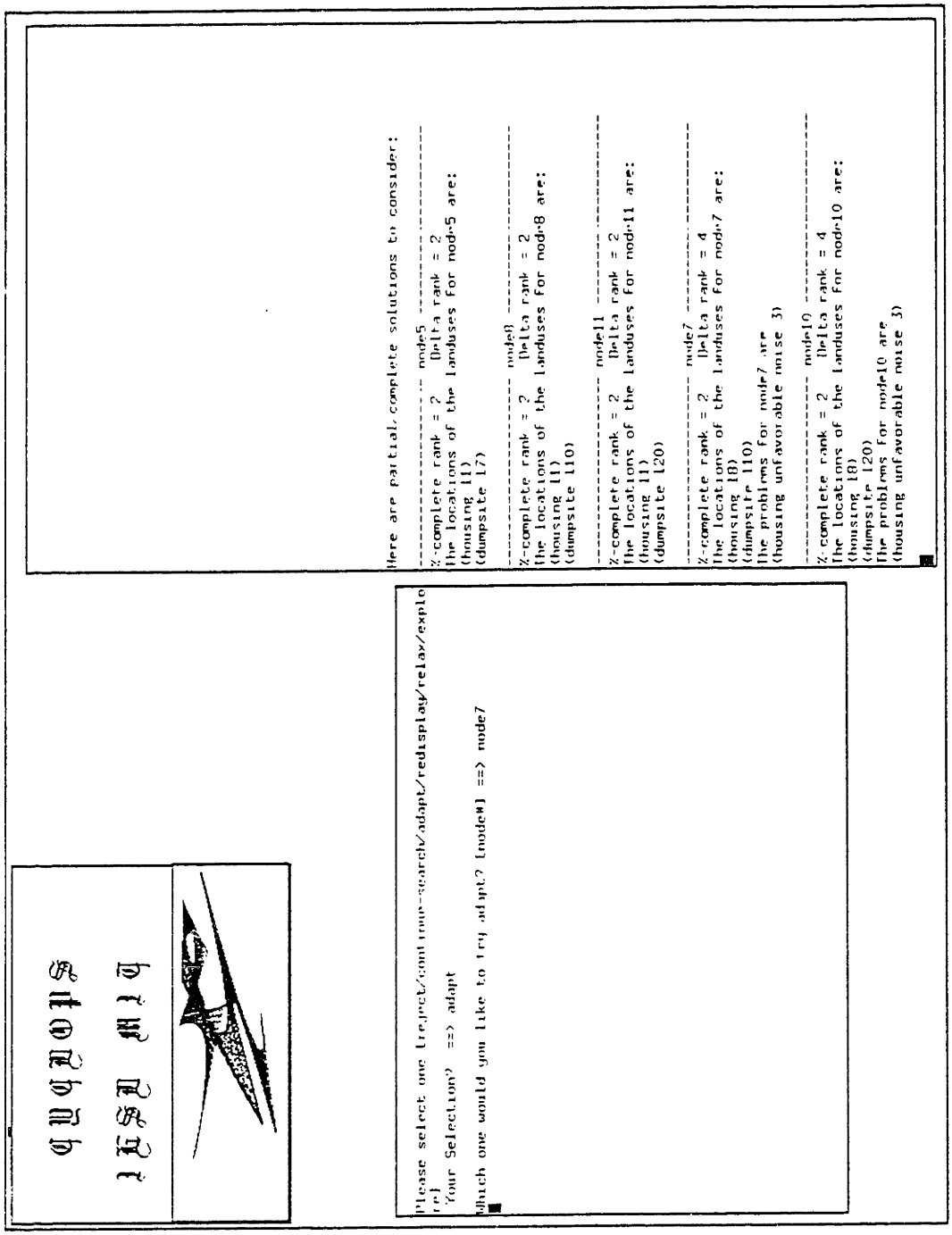

Figure B-2: The CYCLOPS interface

the synthesizer, any new partial designs surfaced as a result of the relaxation. The rest of the process is the same as that of the normal search process.

Finally, the user may decide to adapt a particular design. When the design adapter is invoked, it searches the partial design for problems and uses the ***demand posting*** mechanism to post demands on the **precedents manager**. If a relevant precedent is found, it is retrieved and applied to the design problem by the **adapter**.

The interface to CYCLOPS is shown in Figure B-2. The interface has two windows, a prompting and command input window (left) and a window for viewing alternatives (on the right).

B.2 An Example

In this section we will see how a design problem is posed and how precedents are represented. The example used here is a modified version of the example introduced in Chapter 2.

B.2.1 The problem statement.

The design involves the siting of a sector of housing, an apartment complex, a dumpsite, a recreational area and a cemetery. The program CYCLOPS needs this data to be input as LISP lists. All commentary provided in the trace are [boxed].

```
The atom *nodes* represents the root node.
Data about the landscape is stored at the root. This
information is inherited by all the branches.
```

```
(set '*nodes*
   '(( (name (root))
     (description ((root landscape is rainy)
                   (root sound propagates-in-waves)
                   (root light propagates-in-waves)
                   (root landscape in washington-state))))))
```

```
The variables are stored in  a list. Each variable is stored with
its domain and a description of the variable. In this case, the
variables are landuses that have to be sited.
```

```
(set '*variables*
```

```
   '(
;-----------------------------------------

((name (housing-sector-1))
 (domain ( lot7 lot10 lot12  lot9 lot11 ))
 (description ( (housing-sector-1 is-a landuse)
          (housing-sector-1 is-a low-rise-building)
          (housing-sector-1 is-a building) )))


((name (apartment-complex))
 (domain ( lot7 lot10 lot12  lot9 lot11 ))
 (description ( (apartment-complex is-a landuse)
          (apartment-complex is-a high-rise-building)
          (apartment-complex is-a building) )))

((name (dumpsite))
 (domain (lot3 lot5  ))
 (description ()))

((name (recreational-area))
 (domain ( lot7 lot10 lot12  lot9 lot11 ))
 (description ()))


((name (cemetery))
 (domain (lot3 lot5  ))
 (description ()))


;-----------------------------------------
))
```

Each of the values of the variable's domains are also described. The lots of land have to be described. The descriptions are stored in the form of a list. The description provides a X-Y location and properties local to the site.

```
(set '*values*
'(
;-----------------------------------------

( (name (lot3))
  (description ((location 250 550)
          (lot3 next-to sewage-treatment-plant))))

( (name (lot5))
  (description ((location 250 400)
          (lot5 next-to sewage-treatment-plant))))

( (name (lot4))
  (description ((location 350 400))))

( (name (lot8))
  (description ((location 350 300))))
```

```
( (name (lot7))
  (description ((location 450 450)
          (next-to highway))))

( (name (lot9))
  (description ((slope rank 2)
          (location 450 300)
          (ground has slope))))

( (name (lot11))
  (description ( (slope rank 6)
          (can-view lake)
          (can-view power-lines)
          (location 500 300)
          (ground has slope)) ))

( (name (lot17))
  (description ( (slope rank 6)
          (can-view lake)
          (can-view power-lines)
          (location 600 250)
          (ground has slope))) )

( (name (lot14))
  (description ( (location 450 200))))

( (name (lot16))
  (description ( (location 550 150)) ))

( (name (lot15))
  (description ( (location 450 100))))

( (name (lot10))
  (description ( (location 500 400)
          (lot10 next-to lake)
          (soil rank 6)) ))

( (name (lot12))
  (description ( (location 550 400)
          (lot12 next-to lake)
          (soil rank 6))))


;-----------------------------------------
))
```

The Δ_g is denoted in the system as *delta*.
The starting value of *delta* is 0 (zeroth order design)

```
(set '*delta* 0)  ; starting delta value
```

For help CYCLOPS to reason across precedents, a domain specific dictionary is to be provided. Currently, CYCLOP's method of handling similar words is crude, as the following code shows.

```
; the object hierarchy

;dwellings
(putprop 'dwelling 'buildings 'is-a)
(putprop 'house 'dwelling 'is-a)
(putprop 'housing 'dwelling 'is-a)
(putprop 'home 'dwelling 'is-a)
(putprop 'housing-sector-1 'dwelling 'is-a)
(putprop 'housing-sector-2 'dwelling 'is-a)
(putprop 'apartment-complex 'dwelling 'is-a)

;commercial
(putprop 'commercial 'buildings 'is-a)
(putprop 'shop 'commercial 'is-a)
(putprop 'office 'commercial 'is-a)
(putprop 'hospital 'commercial 'is-a)
(putprop 'hotel 'commercial 'is-a)

;educational
(putprop 'educational 'buildings 'is-a)
(putprop 'school 'educational 'is-a)
(putprop 'schooling 'educational 'is-a)
(putprop 'college 'educational 'is-a)

;land
(putprop 'planar-object 'inaminate-thing 'is-a)
(putprop 'land 'planar-object 'is-a)
(putprop 'landscape 'land 'is-a)
(putprop 'site 'land 'is-a)
(putprop 'ground 'land 'is-a)

;misc
(putprop 'buildings 'rectangular-prism 'is-a)
(putprop 'recantular-prism 'inanimate-thing 'is-a)
(putprop 'inanimate-thing 'thing)

;end of object hierarchy
;-----------------------------------------------
```

B.2.2 Domain Knowledge-Base: The Precedents

This section lists the precedents used by CYCLOPS. The list is very short and should be several thousand times larger in order to show true intelligent behavior. Currently CYCLOPS has a flat, one-level indexing mechanism which is slow.

Precedents are defined by the **defprecedent** function. These precedents are compiled into rules. This has been shown for the first precedent listed below. The rules are derived directly from the precedent. The rules are then fed into a production system[37].

```
; The PRECEDENTS
;

;*****************************************************************************
; A precedent about how people in thailand keep their homes form
; flooding


(defprecedent thailand-stilts
 (conditions1
  (and (on hut ground)
     (is ground flooded)))

 (effects1 (unfavorable house-is-flooded (rank 8)))

 (explanations1
  (because (unfavorable house-is-flooded (rank 8))
        (is building flooded))

  (because (is building flooded)
        (and (on building ground)
           (is ground flooded))))

 (action (on building stilts))

 (effects2 (not (unfavorable house-is-flooded (rank 8))))

 (explanations2
  (because (not (unfavorable house-is-flooded (rank 8)))
        (and (not (on ground flooded))
           (is ground flooded)))

  (because (not (on building ground))
```

---

[37]The production system used is **IMST** [Navinchandra 85]. The rules are represented as lists. A rule has a IF part and a THEN part separated by an arrow. Propositions are added and removed from the working memory with the **assert** and **unassert** commands. Variables in **IMST** are denoted by angle brackets. For example: <variable1>

```
        (<action>))))

; the translation into rules

;(rule thailand-stilts-1
;    (on hut ground)
;    (is ground flooded)
; --> (is hut flooded)
;    (assert because
;          (is hut flooded)
;          ((is ground flooded)
;           (on hut ground))))
;
;(rule thailand-stilts-2
;    (is hut flooded)
; --> (assert unfavorable house-is-flooded (rank 8))
;    (assert because
;          (unfavorable house-is-flooded (rank <rank>))
;          ((is hut flooded))))
;
;(rule thailand-stilts-3
;    (is-a hut landuse)
;    (wanted (not (unfavorable house-is-flooded (rank <anyrank>))))
; --> (assert action (on hut stilts))
;    (unassert (wanted
;           (not (unfavorable house-is-flooded (rank <anyrank>]
;
;(rule thailand-stilts-4
;    (wanted (not (on hut ground)))
;    (on hut ground)
; --> (assert action (on hut stilts))
;    (unassert wanted (not (on hut ground)))
;    (unassert on hut ground)
;    (assert because
;          (not (on hut ground))
;          ((on hut ground) (action (on hut stilts)))))

;*************************************************************************
; light transports itself from one point to another as long
; as there are no obstruction

(defprecedent transport-light
 (conditions
  (and (at person place1)
     (next-to place1 place2)
     (not (between place1 place2 anything))))

 (effects (unobstructed-path-between place1 place2))

 (explanations
  (because (unobstructed-path-between place1 place2)
        (and (at person place1)
           (next-to place1 place2)
           (not (between place1 place2 anything))))))
```

```
;*************************************************************************
; A precedent about how ugly highways can be.

(defprecedent ugly-sight
 (acquired "Many years ago, at my aunt's place")

 (conditions (and (at building site)
            (next-to site  highway)
            (not (between highway site anything))))

 (effects (unfavorable ugly-highway (rank 5))
       (say OH no The <building> is too close to the ugly highway))

 (explanations
  (because (unfavorable ugly-highway (rank 5))
        (unobstructed-path-between building highway))

  (because (unobstructed-path-between building highway)
        (and (precedent transport-light)
           (at building site)
           (next-to site highway)
           (not (between highway site anything))))))


;*************************************************************************
; A precedent about how noisy a highway can be

; highway noise precedent

(defprecedent ugly-highway-noise
 (acquired "Many years ago, at my aunt's place")

 (conditions (at building site)
          (next-to site highway)
          (not (between highway site anything)))

 (effects (unfavorable highway-noise (rank 5))
       (say Oh No The <house> is too close to the noisy highway))

 (explanations
  (because (unfavorable highway-noise (rank 5))
        (and (unobstructed-path-between site highway))))


  (because (unobstructed-path-between site highway)
        (and (precedent transport-light)
           (at building  site)
           (next-to site highway)
           (not (between highway site anything))))))


;*************************************************************************
; The power lines are ugly to look at
```

```
(defprecedent power-lines-ugly
 (conditions (and (at building site)
                  (can-view site power-lines)))

 (effect (unfavorable power-lines-in-view (rank 4)))

 (because (unfavorable power-lines-in-view (rank 4))
          (and (at building site)
               (can-view site power-lines))))


;*************************************************************************
; an object on a sloped surface is tilted

(defprecedent sloped-object
 (conditions (and (on object ground)
                  (is ground tilted )))
 (effects (is object tilted))

 (explanations
  (because (is object tilted)
           (and (on object ground)
                (is ground tilted)))))


;*************************************************************************
; A precedent about homes on steep hillsides

(defprecedent slope-rank
 (acquired "nepal, while trekking")

 (conditions (building is tilted)
             (slope <rank> >= 3))

 (effects (unfavorable slope (rank <rank>))
          (say The slope at the  site is too high.))

 (explanations
  (because (unfavorable slope (rank <rank>))
           (building is tilted)
           (slope <rank> >= 3)
           (precedent sloped-object))))


;*************************************************************************
; dumpsites are bad places. Being within 200 m of one is
; really bad!

(defprecedent avoid-dumpsites
 (conditions (distance-between building  dumpsite) <= 200m)

 (effects (unfavorable dumpsite-close-by (rank 6)))

 (explanations
  (because (unfavorable dumpsite-close-by (rank 6))
           (distance-between building dumpsite) <= 200m)))
```

```
;*************************************************************************
;sewage plants are like Ugh!
;


(defprecedent sewage-plants-are-bad
 (conditions (and (at building site)
             (next-to site sewage-treatment-plant)))

 (effects (unfavorable close-to-sewage-plant (rank 9))
       (say Ugh! I notice that the <building> is near a sewage plant.))

 (explanations
  (because   (unfavorable close-to-sewage-plant (rank 9))
          (and (building at site)
              (site next-to sewage-treatment-plant)))))


;*************************************************************************
; Avoid bad soil. A precedent picked up in a geo-technical class

(defprecedent bad-soil
 (conditions (and (building at site)
             (site soil <rank>  >= 3)))

 (effects (unfavorable soil-conditions (rank <rank>))
       (say Bad soil conditions for the <context has been detected.)))

 (explanations
  (because (unfavorable soil-conditions (rank <rank>))
        (and (building at site)
            (site soil <rank> >= 3)))))

;*************************************************************************
; A precedent about my parent's home that was next to a lake
;

(defprecedent childhood-home-next-to-lake
 (conditions
  (and (at home site)
     (next-to site lake)))

 (effects (favorable home-next-to-lake (rank -3)
       (say The home being next to a lake reminds
          me of my parent's home.)))

 (explanation
  (because (favorable home-next-to-lake (rank -3))
        (and (at home site)
            (next-to site lake)))))


;*************************************************************************
; a precedent about how army  uses camouflage in battle

(defprecedent camouflage-used-in-war
```

```
(conditions1 (can-view person barracks))

(effects1 (unfavorable barracks-can-be-seen (rank 10)))

(explanations1 (because (<effects1>) (<conditions1>)))

(action (camouflage the barracks with green paint and plant-matter))

(effects2 (not (unfavorable barracks-can-be-seen (rank 10))))

(explanations2
 (because (not (unfavorable barracks-can-be-seen (rank 10)))
       (not (can-view person barracks)))

 (because (not (can-view person barracks))
       (<action>))))


;*************************************************************************
;no two things can be at the same place at the same time

(defprecedent no-overlap
 (conditions (object1 at place1)
        (object2 at place1))

 (effect (unfavorable CONFLICT-overlap (rank 1000)))

 (explanations
  (because (unfavorable CONFLICT-overlap (rank 1000))
       (and (object1 at place1)
          (object2 at place1)))))

;*************************************************************************
; this is a simple constraint that is retrived from memory
; and posed to the problem


(defprecedent recreational-area-should-be-next-to-lake
 (conditions
     (and (at recreational-area site)
        (not (next-to site  lake))))

 (effects ( unfavorable because-no-lake-nearby (rank 5)))

 (explanations
  (because ( unfavorable because-no-lake-nearby (rank 5))
       (and (at recreational-area  site)
          (not (next-to site lake))))))

;*************************************************************************
; the cemetery should not be visible

(defprecedent cemetery-should-not-be-seen-easily
 (conditions
  (and (at cemetery site1)
     (at building site2)
```

```
    (is building high-rise-building)
    ((distance-between site1 site2) <= 300m)))

 (effects (unfavorable cemetery-visible (rank 7)
       (say I think the cemetery is visible
        to the high-rising <building>))))

 (explanations
  (because (unfavorable cemetery-visible (rank 7))
        (and (cemetery at site1)
           (building at site2)
           (building is high-rise-building)
           ((distance-between site1 site2) <= 300m)))))


;*************************************************************************
; it is nice to have a lake visible from the dwellings

(defprecedent lake-visible-from-dwellings-is-nice
 (conditions
  (and (housing at site)
     (can-view site  lake)))

 (effects (favorable lake-visible (rank -5)
       (say That is nice the <housing> can has a view of the lake))))

 (explanations
  (because (favorable lake-visible (rank -5))
        (and (housing at site)
           (can-view site lake)))))


;*************************************************************************
; A precedent about a noise protection strategy

(defprecedent noise-protection-in-washington-state
 (acquired "While driving from Sea-tac to Tacoma")

 (conditions1
  (and (building at site1)
     (site1 next-to highway)
     (unobstructed-path-between site1 highway)))

 (effects1 (unfavorable highway-noise (rank 5)))

 (explanation1
  (because (unfavorable highway-noise (rank 5))
        (and (precedent ugly-highway-noise)
           (building at site1)
           (site1 next-to highway)
           (unobstructed-path-between site1 highway))))

 (action (plant a wall of trees between building and highway))

 (effecs2 (not (unfavorable highway-noise (rank 5))))
```

```
(explanation2
 (because (not (unfavorable highway-noise (rank 5)))
      (not (unobstructed-path-between site1 highway)))

 (because (not (unobstructed-path-between site1 highway))
      (and ( <action> )
         ( <explanation1>)))))
```

B.3 The run

Here is a run the the program CYCLOPS. The run is annotated by boxed text and user input is in underlined bold **text**. The rest of the text is as it appeared in run, however, large parts of the trace has been edited out for brevity.

<hera-16> lisp

WELCOME TO CYCLOPS

Do you want audio prompt(ding-dong) ? [y/n] > **n**

Which data base? > **example**

Franz Lisp, Opus 42.16.3
(C) Copyright 1985, Franz Inc., Alameda Ca.

=> **(search-loop)**
The best delta value is: 10
This is value is liable to reduce/increase as we go along

The program first shows the user the root node.
This is only by convention.

Here are partial/complete solutions to consider:

---------- root ----------
%-complete rank = 10 Delta rank = 10
The locations of the landuses for root are:

Please select one:
[reject/automatic/continue-search/adapt/redisplay/relax/sort/quit/explore]

Your Selection? ==> **automatic**

The user selects automatic mode. A controlling program takes the place of the user for a specified number of synthesis steps

How many steps automatic? > **5**
Starting search ...
Looking for Problems .....

Working on: design0

Working on: design1

Starting search ...
Looking for Problems .....

Working on: design2
Oh No The housing-sector-1 is too close to the ugly highway

Working on: design3
The housing-sector-1 being next to a lake reminds me of my parent 's home.
Bad soil conditions for the housing-sector-1 has been detected.

Working on: design6
That is nice the housing-sector-1 can has a view of the lake
The slope at the housing-sector-1 site is too high.

Working on: design7
Oh No The housing-sector-1 is too close to the ugly highway

Working on: design10

Looking for Problems .....

Working on: design12
Oh No The apartment-complex is too close to the ugly highway

Notice, how the program repeatedly discovers the same problem in different designs. This problem was discussed in Chapter 9

Working on: design13

Working on: design21
That is nice the apartment-complex can has a view of the lake
The slope at the apartment-complex site is too high.

Working on: design42
I think the cemetery is visible to the high-rising apartment-complex
The apartment-complex being next to a lake reminds me of my parent 's home.
Bad soil conditions for the apartment-complex has been detected.

and so on.....

Finally the program finds a complete solution. The Current *delta* value is 1

Here are some COMPLETE solutions to ponder...

---------- design48 ----------
%-complete rank = 0 Delta rank = 1
The locations of the landuses for design48 are:
(cemetery lot3)
(recreational-area lot10)
(apartment-complex lot12)
(housing-sector-1 lot9)

(dumpsite lot5)
The problems for design48 are
(apartment-complex unfavorable soil-conditions 6)
the opportunities of design48 are
(apartment-complex favorable home-next-to-lake -3)

Please Choose any of [reject-accept/redisplay/ignore]
Your response ==> **reject-accept**

the user decides to accept the design

Give me a list of nodes you wish to accept ==> **(design48)**

The user is not satisfied with the design and wants to explore, even though he has found a decent design.

Are you satisfied with what you have [y/n] ==> **n**

Please select one:
[reject/automatic/continue-search/adapt/redisplay/relax/sort/quit/explore]

Your Selection? ==> **relax**

The user decides to call for a relaxation. Pushing down of the pareto surface

By how much [#] ==> **5**

Here are partial/complete solutions to consider:

---------- design16 ----------
%-complete rank = 3 Delta rank = 5
The locations of the landuses for design16 are:
(apartment-complex lot11)
(housing-sector-1 lot9)
(dumpsite lot3)
The problems for design16 are
(apartment-complex unfavorable slope 6)
(apartment-complex unfavorable power-lines-in-view 4)
the opportunities of design16 are
(apartment-complex favorable lake-visible -5)

---------- design21 ----------
%-complete rank = 3 Delta rank = 5
The locations of the landuses for design21 are:
(apartment-complex lot11)
(housing-sector-1 lot9)
(dumpsite lot5)
The problems for design21 are
(apartment-complex unfavorable slope 6)
(apartment-complex unfavorable power-lines-in-view 4)
the opportunities of design21 are
(apartment-complex favorable lake-visible -5)

---------- design22 ----------
%-complete rank = 1 Delta rank = 6
The locations of the landuses for design22 are:
(recreational-area lot7)
(apartment-complex lot10)
(housing-sector-1 lot9)
(dumpsite lot3)
The problems for design22 are
(recreational-area unfavorable because-no-lake-nearby 5)
(apartment-complex unfavorable soil-conditions 6)
the opportunities of design22 are
(apartment-complex favorable home-next-to-lake -3)

---------- design26 ----------
%-complete rank = 1 Delta rank = 6
The locations of the landuses for design26 are:
(recreational-area lot11)
(apartment-complex lot10)
(housing-sector-1 lot9)
(dumpsite lot3)
The problems for design26 are
(recreational-area unfavorable because-no-lake-nearby 5)
(apartment-complex unfavorable soil-conditions 6)
the opportunities of design26 are
(apartment-complex favorable home-next-to-lake -3)

---------- design27 ----------
%-complete rank = 1 Delta rank = 6
The locations of the landuses for design27 are:
(recreational-area lot7)
(apartment-complex lot12)
(housing-sector-1 lot9)
(dumpsite lot3)
The problems for design27 are
(recreational-area unfavorable because-no-lake-nearby 5)
(apartment-complex unfavorable soil-conditions 6)
the opportunities of design27 are
(apartment-complex favorable home-next-to-lake -3)

---------- design31 ----------
%-complete rank = 1 Delta rank = 6
The locations of the landuses for design31 are:
(recreational-area lot11)
(apartment-complex lot12)
(housing-sector-1 lot9)
(dumpsite lot3)
The problems for design31 are
(recreational-area unfavorable because-no-lake-nearby 5)
(apartment-complex unfavorable soil-conditions 6)
the opportunities of design31 are
(apartment-complex favorable home-next-to-lake -3)

---------- design32 ----------
%-complete rank = 1 Delta rank = 6
The locations of the landuses for design32 are:
(recreational-area lot7)

(apartment-complex lot10)
(housing-sector-1 lot9)
(dumpsite lot5)
The problems for design32 are
(recreational-area unfavorable because-no-lake-nearby 5)
(apartment-complex unfavorable soil-conditions 6)
the opportunities of design32 are
(apartment-complex favorable home-next-to-lake -3)

---------- design36 ----------
%-complete rank = 1 Delta rank = 6
The locations of the landuses for design36 are:
(recreational-area lot11)
(apartment-complex lot10)
(housing-sector-1 lot9)
(dumpsite lot5)
The problems for design36 are
(recreational-area unfavorable because-no-lake-nearby 5)
(apartment-complex unfavorable soil-conditions 6)
the opportunities of design36 are
(apartment-complex favorable home-next-to-lake -3)

---------- design37 ----------
%-complete rank = 1 Delta rank = 6
The locations of the landuses for design37 are:
(recreational-area lot7)
(apartment-complex lot12)
(housing-sector-1 lot9)
(dumpsite lot5)
The problems for design37 are
(recreational-area unfavorable because-no-lake-nearby 5)
(apartment-complex unfavorable soil-conditions 6)
the opportunities of design37 are
(apartment-complex favorable home-next-to-lake -3)

---------- design41 ----------
%-complete rank = 1 Delta rank = 6
The locations of the landuses for design41 are:
(recreational-area lot11)
(apartment-complex lot12)
(housing-sector-1 lot9)
(dumpsite lot5)
The problems for design41 are
(recreational-area unfavorable because-no-lake-nearby 5)
(apartment-complex unfavorable soil-conditions 6)
the opportunities of design41 are
(apartment-complex favorable home-next-to-lake -3)

Please select one
[reject/automatic/continue-search/adapt/redisplay/relax/sort/quit/explore]

Your Selection? ==> **reject**

The user decides to reject many designs, even though they are pareto optimal. (He feels he cannot handle poor soil conditions.)

Give me a list of nodes you want to reject ==> **(design22 design26 design27 desgin31 design32 design36 design37 design41)**

Please select one
[reject/automatic/continue-search/adapt/redisplay/relax/sort/quit/explore]
Your Selection? ==> **explore**

The explore command causes a relaxation of 1 to the *delta* and it then kicks off the search process. The new *delta* is set to 7.

Setting Delta to 7

**** LOOKING FOR PROBLEMS

Working on: design50
That is nice the apartment-complex can has a view of the lake
The slope at the apartment-complex site is too high.

Working on: design45
I think the cemetery is visible to the high-rising apartment-complex.
The apartment-complex being next to a lake reminds me of my parent 's home.
Bad soil conditions for the apartment-complex has been detected.

Working on: design60
The housing-sector-1 being next to a lake reminds me of my parent 's home.
Bad soil conditions for the housing-sector-1 has been detected.
Oh No The apartment-complex is too close to the ugly highway

Answers found !

Here are some COMPLETE solutions to ponder...

---------- design43 ----------
%-complete rank = 0 Delta rank = 7
The locations of the landuses for design43 are:
(cemetery lot5)
(recreational-area lot12)
(apartment-complex lot10)
(housing-sector-1 lot9)
(dumpsite lot3)
The problems for design43 are
(apartment-complex unfavorable soil-conditions 6)
(apartment-complex unfavorable cemetery-visible 7)
the opportunities of design43 are
(apartment-complex favorable home-next-to-lake -3)

---------- design45 ----------
%-complete rank = 0 Delta rank = 7
The locations of the landuses for design45 are:
(cemetery lot5)
(recreational-area lot10)
(apartment-complex lot12)
(housing-sector-1 lot9)
(dumpsite lot3)

The problems for design45 are
(apartment-complex unfavorable soil-conditions 6)
(apartment-complex unfavorable cemetery-visible 7)
the opportunities of design45 are
(apartment-complex favorable home-next-to-lake -3)

---------- design46 ----------
%-complete rank = 0 Delta rank = 7
The locations of the landuses for design46 are:
(cemetery lot3)
(recreational-area lot12)
(apartment-complex lot10)
(housing-sector-1 lot9)
(dumpsite lot5)
The problems for design46 are
(apartment-complex unfavorable soil-conditions 6)
(apartment-complex unfavorable cemetery-visible 7)
the opportunities of design46 are
(apartment-complex favorable home-next-to-lake -3)

Please Choose any of [reject-accept/redisplay/ignore]
Your response ==> **reject-accept**

Give me a list of nodes you wish to accept ==> ()

The user decides to reject all the designs he sees. This is because he hates bad soil conditions

Are you satisfied with what you have [y/n] ==> **n**

Give me a list of nodes you wish to reject ==> **(design43 design45 design46)**

The best delta value is: 3
This is value is liable to reduce/increase as we go along

Here are partial/complete solutions to consider:

---------- design51 ----------
%-complete rank = 1 Delta rank = 3
The locations of the landuses for design51 are:
(recreational-area lot10)
(apartment-complex lot11)
(housing-sector-1 lot9)
(dumpsite lot3)
The problems for design51 are
(apartment-complex unfavorable power-lines-in-view 4)
(apartment-complex unfavorable slope 6)
the opportunities of design51 are
(apartment-complex favorable lake-visible -5)

---------- design52 ----------
%-complete rank = 1 Delta rank = 3
The locations of the landuses for design52 are:
(recreational-area lot12)
(apartment-complex lot11)

(housing-sector-1 lot9)
(dumpsite lot3)
The problems for design52 are
(apartment-complex unfavorable power-lines-in-view 4)
(apartment-complex unfavorable slope 6)
the opportunities of design52 are
(apartment-complex favorable lake-visible -5)

---------- design56 ----------
%-complete rank = 1 Delta rank = 3
The locations of the landuses for design56 are:
(recreational-area lot10)
(apartment-complex lot11)
(housing-sector-1 lot9)
(dumpsite lot5)
The problems for design56 are
(apartment-complex unfavorable power-lines-in-view 4)
(apartment-complex unfavorable slope 6)
the opportunities of design56 are
(apartment-complex favorable lake-visible -5)

---------- design57 ----------
%-complete rank = 1 Delta rank = 3
The locations of the landuses for design57 are:
(recreational-area lot12)
(apartment-complex lot11)
(housing-sector-1 lot9)
(dumpsite lot5)
The problems for design57 are
(apartment-complex unfavorable power-lines-in-view 4)
(apartment-complex unfavorable slope 6)
the opportunities of design57 are
(apartment-complex favorable lake-visible -5)

Please select one
[reject/automatic/continue-search/adapt/redisplay/relax/sort/quit/explore]
Your Selection? ==> **adapt**

The user decides to try adapt design57. This is currently left in the user's hands because it is a very slow process. Under utopian conditions, the system should try adapt every design!

Which one would you like to try adapt? [node#] ==> **design57**

Cleaning up junk nodes, if any..

Here are partial/complete solutions to consider:
The adapted partial design

---------- design83 ----------
%-complete rank = 1 Delta rank = -5
The locations of the landuses for design83 are:
(recreational-area lot12)
(apartment-complex lot11)

(housing-sector-1 lot9)
(dumpsite lot5)
the opportunities of design83 are
(apartment-complex favorable lake-visible -5)
The adaptations of design83 are
(apartment-complex action (on building stilts))
(apartment-complex action
(camouflage the
power-lines
with
green
paint
and
plant-matter))

---------- design51 ----------
%-complete rank = 1 Delta rank = 3
The locations of the landuses for design51 are:
(recreational-area lot10)
(apartment-complex lot11)
(housing-sector-1 lot9)
(dumpsite lot3)
The problems for design51 are
(apartment-complex unfavorable power-lines-in-view 4)
(apartment-complex unfavorable slope 6)
the opportunities of design51 are
(apartment-complex favorable lake-visible -5)

---------- design52 ----------
%-complete rank = 1 Delta rank = 3
The locations of the landuses for design52 are:
(recreational-area lot12)
(apartment-complex lot11)
(housing-sector-1 lot9)
(dumpsite lot3)
The problems for design52 are
(apartment-complex unfavorable power-lines-in-view 4)
(apartment-complex unfavorable slope 6)
the opportunities of design52 are
(apartment-complex favorable lake-visible -5)

---------- design56 ----------
%-complete rank = 1 Delta rank = 3
The locations of the landuses for design56 are:
(recreational-area lot10)
(apartment-complex lot11)
(housing-sector-1 lot9)
(dumpsite lot5)
The problems for design56 are
(apartment-complex unfavorable power-lines-in-view 4)
(apartment-complex unfavorable slope 6)
the opportunities of design56 are
(apartment-complex favorable lake-visible -5)

Please select one:

[reject/automatic/continue-search/adapt/redisplay/relax/sort/quit/explore]

Selection? ==> **continue-search**

The user decides to continue searching. The system asks for the user's precerece, if any. The user can either give a node number of just state that he is indifferent among the parto-optimal partial designs.

Choose a node for that you like [node#/indifferent] ==> **design83**
The selection is : design83

Looking for Problems

.... extra output edited out ....

Working on: design86
That is nice the apartment-complex can has a view of the lake

The best delta value is: -6
This is value is liable to reduce/increase as we go along

Answers found !
---------- design86 ----------
%-complete rank = 0 Delta rank = -6
The locations of the landuses for design86 are:
(cemetery lot3)
(recreational-area lot12)
(apartment-complex lot11)
(housing-sector-1 lot9)
(dumpsite lot5)
the opportunities of design86 are
(apartment-complex favorable lake-visible -5)
The adaptations of design86 are
(apartment-complex action
(camouflage the
power-lines
with
green
paint
and
plant-matter))
(apartment-complex action (on building stilts))

Please Choose any of [reject-accept/redisplay/ignore]
Your response ==> **reject-accept**

Give me a list of nodes you wish to accept ==> **(design86)**

The user has made a selection. He will now try to see if he can find anything better. If not, he will stop.

Are you satisfied with what you have [y/n] ==> **n**

Give me a list of nodes you wish to reject ==> **nil**

Please select one:
[reject/automatic/continue-search/adapt/redisplay/relax/sort/quit/explore]

Your Selection? ==> **relax**

By how much [#] ==> **5**

---------- design51 ----------
%-complete rank = 1 Delta rank = 3
The locations of the landuses for design51 are:
(recreational-area lot10)
(apartment-complex lot11)
(housing-sector-1 lot9)
(dumpsite lot3)
The problems for design51 are
(apartment-complex unfavorable power-lines-in-view 4)
(apartment-complex unfavorable slope 6)
the opportunities of design51 are
(apartment-complex favorable lake-visible -5)

---------- design52 ----------
%-complete rank = 1 Delta rank = 3
The locations of the landuses for design52 are:
(recreational-area lot12)
(apartment-complex lot11)
(housing-sector-1 lot9)
(dumpsite lot3)
The problems for design52 are
(apartment-complex unfavorable power-lines-in-view 4)
(apartment-complex unfavorable slope 6)
the opportunities of design52 are
(apartment-complex favorable lake-visible -5)

---------- design56 ----------
%-complete rank = 1 Delta rank = 3
The locations of the landuses for design56 are:
(recreational-area lot10)
(apartment-complex lot11)
(housing-sector-1 lot9)
(dumpsite lot5)
The problems for design56 are
(apartment-complex unfavorable power-lines-in-view 4)
(apartment-complex unfavorable slope 6)
the opportunities of design56 are
(apartment-complex favorable lake-visible -5)

Please select one:
[reject/automatic/continue-search/adapt/redisplay/relax/sort/quit/explore]

Your Selection? ==> **quit**

The user is satisfied with what he selected on the previous page.
He decides to stop.

=> **(exit)**

hera<17> **logout**

References

[ADINA Engg. Inc. 75]
ADINA Engineering Inc., Watertown, MA.
ADINA, Finite Element Analysis Package.
1975

[Barrow & Tenenbaum 76]
Barrow H.G., J.M. Tannenbaum.
MYSYS: A system for reasoning about scenes.
Technical Report TR-121, SRI International, March, 1976.

[Brown & Chandrasekaran 86]
Brown D.C., B. Chandrasekaran.
Expert Systems for a class of Mechanical Design Activity.
In Sriram D., Adey B. (editor), *Proceedings of the First International Conference on AI applications in Engineering.* Computational Mechanics, U.K., 1986.

[Buchanan & Feigenbaum 78]
Buchanan, B.G., E.A. Fiegenbaum.
Dendral and Meta-Dendral: Their Applications Dimension.
Artificial Intelligence 11(1):5-24, 1978.

[Burstall 69] Burstall, R.M.
A program for solving word sum puzzles.
Computer Journal 12:48-51, 1969.

[Bushnell & Director 86]
Bushnell, M.L., S.W. Director.
VLSI CAD Tool Integration Using the Ulysses Environment.
In *Proceedings of the 23rd ACM/IEEE Design Automation Conference Proceedings*, pages 55-61. 1986.

[Carbonell 83] Carbonell, J. G.
Derivational Analogy and its role in Problem Solving.
In *Proceedings AAAI-83, Pittsburgh, PA*, pages 64-69. 1983.

[Carbonell 86] Carbonell, J. G.
Derivational Analogy: A Theory of Reconstructive Problem Solving and Expertise Acquisition.
In Michalski, R. S., J. G. Carbonell, T. M. Mitchell (editor), *Machine Learning: An Artificial Intelligence Approach Vol 2.* Morgan Kaufman, 1986.

[Chehayeb 87] Chehayeb, F.
A Framework for Engineering Knowledge Representation and Problem Solving.
PhD thesis, Dept. of Civil Engineering, M.I.T., Cambridge, MA, May, 1987.

[Darwin 59] Darwin, C.
On the Origin of Species by Means of Natural Selection
London: John Murray, 1859.

[Deutsch 66] Deutsch, J.P.A., J.P.A. Deutsch.
A short cut for certain combinatorial problems.
In *British Joint Computer Conference*. 1966.

[Dyer et al. 86] Dyer M.G., M. Flowers, J. Hodges.
EDISON: An Engineering Design Invention System Operating Naively.
In *Proceedings of the First International Conference on Applicatoins of AI to Engineering*. April, 1986.

[Fikes 70] Fikes, R.E.
REF-ARF: A system for Solving Problems Stated as Procedures.
Int. J. of Artificial Intelligence 1:27-120, 1970.

[Fox 83] Fox, M.S.
Constraint Directed Search: A case of Job Shop Scheduling.
PhD thesis, Carnegie-Mellon University, 1983.

[Gentner 83] Gentner, D.
Structure Mapping: A Theoretical Framework for Analogy.
Cognitive Science 7, 1983.

[Gentner & Toupin 86]
Gentner, D., C. Toupin.
Systematicity and Surface Similarity in the Development of Analogy.
Cognitive Science 10:277-300, 1986.

[Goicoechea 82] Goicoechea A., D.R. Hansen, L. Duckstein.
Multibobjective Decision Analysis With Engineering and Business Applications.
John Wiley & Sons, 1982.

[Golomb and Baumert 65a]
Golomb, S.W., L.D. Baumert.
Backtrack Programming.
J. of the ACM 12:516-524, 1965.

[Golomb and Baumert 65b]
Golomb, S.W., L.D. Baumert.
Backtrack Programming.
J. of the ACM 12:516-524, 1965.

[Gordon 61] Gordon W.J.
Synectics: The development of Creative Capacity.
Harper & Row, Publishers, NY, 1961.

[Gross 86] Gross, M.D.
Design as Exploring Constraints.
PhD thesis, M.I.T., 1986.

[Grossman 76] Grossman, R.W.
Some Database applications of Constraint Expressions, LCS-TR-158.
PhD thesis, MIT, 1976.

[Guilford 59] Guilford, J.P.
Creativity.
American Psychologist (5):444-454, 1959.

[Habraken 83] Habraken, J.N.
Writing Form, the Notation of Form Transformations in a Built Environment.
Technical Report, M.I.T., Cambridge, MA, August, 1983.

[Haralick 80] Haralick, R.M.
Scene matching methods.
In Haralick, R.M., J.C. Simon (editor), *Issues in Digital Image Processing*, pages 221-243. Sijthoff and Noordhoff, Alphen aan den Rijn, Netherlands, 1980.

[Haralick and Elliot 80]
Haralick R.M., G.L. Elliot.
Increasing tree search efficiency for constraint satisfaction problems.
Artificial Intelligence 14:263-313, 1980.

[Hart, Nilsson & Raphael 68]
Hart, P.E., N.J. Nilsson, and B. Raphael.
A formal basis for the heuristic determination of minimum cost paths.
IEEE Transactions on Systems Science and Cybernetics SSC-4(2):100-107, 1968.

[Huhns 87] Huhns M.H., R.D. Acosta.
Argo: An Analogical Reasoning System for Solving Design Problems.
Technical Report AI/CAD-092-87, Microelectronic and Computer Technology Corporation, March, 1987.

[Kedar-Cabelli 85a]
Kedar-Cabelli, S.T.
Analogy - From a unified perspective.
Technical Report ML-TR-3, Department of Computer Science, Rutgers University, December, 1985.

[Kedar-Cabelli 85b]
Kedar-Cabelli, S.T.
Purpose-Directed Analogy.
In *Proceedings of the Cognitive Science Society Conference.* Irvine, CA, August, 1985.

[Kolodner 80] Kolodner, J.L.
Retrieval and organizational strategies in conceptual memory: A computer model.
PhD thesis, Yale University, 1980.

[Kolodner 81] Kolodner, J.L.
Organization and Retrieval in a Conceptual Memory for Events.
In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence.* 1981.

[Kolodner 85] Kolodner, J. L., Simpson, R. L. Jr., Sycara-Cyransky, K.
A Process Model of Case-Based Reasoning in Problem-Solving.
In *Proceedings IJCAI-9.* Los Angeles, CA, August, 1985.

[Korf 85] Korf, R.K.
Depth-First Iterative-Deepening: A Optimal Admissible Tree Search.
Int. J. of Artificial Intelligence 27(1):97-109, 1985.

[Langley et al. 87] Langley, P., H.A. Simon, G.L. Bradshaw, J.M. Zytkow.
Scientific Discovery - Computational Explorations of the Creative Processes.
The MIT Press, Cambridge, MA, 1987.

[Lenat 76] Lenat, D.B.
AM: An artificial intelligence approach to discovery in mathematics as heuristic search.
PhD thesis, Stanford University, STAN-CS-76-570, 1976.

[Lenat 84] Lenat, D.B.
Why AM and EURISKO Appear to Work.
Artificial Intelligence 24:269-294, 1984.

[Lindsay 80] Lindsay, R., Buchanan, B., Feigenbaum, E. and Lederberg, J.
Applications of Artificial Intelligence for Chemical Inference : The Dendral Project.
McGraw-Hill Book Company, 1980.

[Mackenzie & Gero 87]
Mackenzie, C.A., J.S. Gero.
Learning Design Rules from Decisions and Performances.
Int. J. of AI in Engineering 2(1):2-10, 1987.

[Mackworth 77] Mackworth A.K.
Consistency in networks of relations.
Artificial Intelligence 8:99-118, 1977.

[McDonnel Douglas 84]
McDonnell Douglas Corporation.
Graphic Design System.
1984

[Minsky 75] Minsky, M.
A framework for representing knowledge.
In Winston P.H. (editor), *The Psychology of Computer Vision.* McGraw-Hill, New York, 1975.

[Mittal 85] Mittal, S., Dym, C. and Morjaria, M.
PRIDE: An Expert System for the Design of Paper Handling Systems.
In Dym, C. (editor), *Applications of Knowledge-Based Systems to Engineering Analysis and Design*, pages 99-116. American Society of Mechanical Engineers, 1985.

[Mostow 85] Mostow, J.
Toward Better Models Of The Design Process.
The AI Magazine , Spring, 1985.

[Muller 87] Muller, E.T.
Daydreaming and Computation: A Computer Model of Everyday Creativity, Learning, and Emotions in the Human Stream of Thought.
PhD thesis, Univ. of California, Los Angeles, UCLA-AI-87-8, February, 1987.

[Murthy & Addanki 87]
Murthy, S.S., S. Addanki.
PROMPT: An Innovative Design Tool.
In *Proceedings of the sixth national conference on artificial intelligence*, pages 637-642. 1987.

[Nadel 85a] Nadel, B.A.
The Consistent Labeling Problem: Backround and Problem Formulation.
Technical Report DCS-TR-164, University of Michigan, 1985.

[Nadel 85b] Nadel, B.A.
The Consistent Labeling Problem: Subproblems, Enumerations and Constraint Satisfiability.
Technical Report DCS-TR-165, University of Michigan, 1985.

[Nadel 85c] Nadel, B.A.
The Consistent Labeling Problem: The Generalized Backtracking Algorithm.
Technical Report DCS-TR-166, University of Michigan, 1985.

[Nadel 85d] Nadel, B.A.
The Consistent Labeling Problem: The Generalized Forward Checking and Word-Wise Forward Checking Algorithms.
Technical Report DCS-TR-167, University of Michigan, 1985.

[Navinchandra 85]
Navinchandra D.
IMST user's manual: A tool for building Rule-based Expert Systems.
Technical Report CCRE-85-6, Center for Construction Research and Education, M.I.T., 1985.

[Navinchandra 86]
Navinchandra D.
Intelligent Use of Constraints for Activity Scheduling.
Technical Report CERL N-86/15, US Army Corps of Engineers, Construction Engineering Research Laboratory, July, 1986.

[Navinchandra et.al. 87]
Navinchandra D., D. Sriram, S. T. Kedar-Cabelli.
Analogy-based Engineering Problem Solving: An Overview.
In Sriram D., R.A. Adey (editor), *Artificial Intelligence in Engineering: Tools and Techniques.* Computational Mechanics Publications, Boston, MA, 1987.

[Newell 69] Newell, A.
Heuristic Programming: Ill-Structured Problems.
In (editor), *Progress in OR*, pages 360-414. John Wiley & Sons, NY, NY, 1969.

[Nijenhius and Wilf 75]
Nijenhuis, A., H.S. Wilf.
Combinatorial Algorithms.
Academic Press, New York, 1975.

[Nilsson 71] Nilsson, N.J.
Problem solving methods in Artificial Intelligence.
McGraw Hill, N.Y., 1971.

[Nilsson 80] Nilsson, N.J.
Principles of Artificial Intelligence.
Tioga, Palo Alto, CA, 1980.

[Osborn 53] Osborn, A. F.
Applied Imagination.
Charles Scribner's Sons, New York, 1953.

[Pearl 84] Pearl, J.
Heuristics - Intelligent search strategies for computer problem solving.
Addison Wesley, Reading, MA, 1984.

[Pfefferkorn 75] Pfefferkorn, C.E.
The Design Problem Solver: A System for Designing Equipment or Furniture Layouts.
In Eastman, C.M. (editor), *Spatial Synthesis in Computer-Aided Building Design.* Wiley, 1975.

[Rickards 74] Rickards, T.
Problem Solving through Creative Analysis.
Wiley, NY, 1974.

[Roos 66] Roos, D.
ICES System Design.
MIT Press, Cambridge, MA, 1966.

[Rychner et al. 86] Rychner, M.D., Farinacci, M.L., Hulthage, I., Fox, M.S.
Integration of Multiple Knowledge Sources in ALADIN, an Alloy Design System.
Technical Report, Intelligent Systems Laboratory, Robotics Institute, Carnegie-Mellon University, 1986.

[Schank 82] Schank, R.C.
Dynamic Memory: A Theory of reminding and learning in computers and people.
Cambridge University Press, 1982.

[Schank 86] Schank, R.C.
Explanation Patterns: Understanding Mechanically and Creatively.
Lawrence Erlbaum Associates, Hillsdale, NJ, 1986.

[Soukup 81] Soukup, J.
Circuit Layout.
Proceedings of the IEEE 69(10), Oct, 1981.

[Sriram 86] Sriram, D.
Knowledge-Based Approaches for Structural Design.
PhD thesis, Carnegie Mellon University, 1986.

[Stallman & Sussman 77] Stallman, R., G.J. Sussman.
Forward Reasoning and Dependency Directed Backtracking in a system for computer aided circuit analysis.
Artificial Intelligence 9:135-196, 1977.

[Stefik 80] Stefik, M.
Planning with Constraints.
PhD thesis, Stanford University, STAN-CS-80-784, 1980.

[Steinberg 87] Steinberg, L.I.
Design as Refinement Plus Constraint Propagation: The VEXED Experience.
In *Proceedings of the sixth national conference on Artificial Intelligence*, pages 830-835. 1987.

[Steinberg et al. 86]
Steinberg L., N. Langrana, T. Mitchell, J. Mostow, C. Tong.
A Domain Independent Model of Knowledge-Based Design.
Technical Report AI/VLSI Project Working Paper No. 33, Rutgers Universtiy, March, 1986.

[Tomlin 86] Tomlin, D. Professor, Graduate School or Design, Harvard University.
1986
Personal Communication.

[Tong 86a] Tong, C.
Knowledge-Based Circuit Design.
PhD thesis, Stanford University, 1986.

[Tong 86b] Tong, C.
A framework for organizing and evaluating knowledge-based models of the design process.
Technical Report AI/VLSI Project Working Paper No. 21, Rutgers University, 1986.

[Ullman & Dietterich 87]
Ullman, D.G., T.A. Dietterich.
Mechanical Design Methodology: Implications on Future Developments of Computer-Aided Design and Knowledge-Based Systems.
Engineering with Computers 2:21-29, 1987.

[Walker 60] Walker, R.J.
An enumerative technique for a class of combinatorial problems.
In *Combinatorial Analysis (Proc. Symp. Applied Math., Vol X), American Mathematical Society.* 1960.

[Waltz 75] Waltz, D.
Understanding line drawings of scenes with shadows.
In Winston P.H. (editor), *The Psychology of Computer Vision.* McGraw-Hill New York, 1975.

[Winston 80] Winston, P. H.
Learning and Reasonig by Analogy.
Communications of the ACM 23(12), December, 1980.

[Winston et al. 83]
Winston, P. H., T.O. Binford, B. Katz, M. Lowry.
Learning Physical Descriptions from Functional Definitions, Examples and Precedents.
In *Proceedings of AAAI-83.* August, 1983.

Index