Planning in Design Synthesis: Abstraction-Based LOOS (ABLOOS)

by

Robert F. Coyne

EDRC 48-14-89

Planning in Design Synthesis:
Abstraction-Based LOOS (ABLOOS)

(Thesis Proposal)

by

Robert F. Coyne

EDRC - (1989 Report Series)

Abstract

Abstraction is the quintessential mechanism for dealing with complexity. Abstraction may take many forms and appears to underlie all of the important strategies for handling large search spaces in planning domains. Design, modeled as search, also depends heavily on the use of mechanisms of abstraction, and may benefit from the employment of strategies to leverage search that are similar to those developed in automated planning. In some design domains, such as automated layout, the fact that the objects manipulated have shape and take up space adds an interesting twist and certain complexities to the use of abstraction mechanisms for handling the operators, objects and goals of the problem space.

With respect to leveraging search in a design system, this proposal describes a salient topic and the research aimed at exploring it. The topic is the combination and interaction of various forms of abstraction in the context of the automated synthesis of layouts. This research issue converges with a unique opportunity to explore it in the context of an especially promising approach to space planning. The system embodying that approach, LOOS, has a representation and an overall system architecture which suitably address some of the problematic limitations and compromises of former approaches. However, in order to extend the generality and scalability of the approach, the problem of the combinatorial explosion in the search space has to be addressed. In this regard, a promising direction appears to be combining decompositional abstractions formed around the objects of design, with the primary representational abstractions used for delineation and generation of alternatives.

The remainder of the proposal describes the creation of a new frameworic to represent and control decompositional abstraction levels in layout design. This new abstraction mechanism will strongly restrict the number of the objects that the layout system has to deal with at a given time, while also providing an appropriate place to store additional knowledge and constraints for pruning. Therefore, it will not only help to tame the explosion in the search space but will also add flexibility and adaptability to the approach. This new abstraction framework is proposed as a general hierarchical extension to LOOS's domain-independent problem solving structure, accepting domain-dependent decompositional knowledge as content.

Table of Contents

1. Introduction 1
2. Organization 3
3. Background 5
3.1 Planning methods in AI problem-solving 5
3.1.1 Domain-independent strategic planning 5
3.1-2 The value of strategic planning 7
3.13 The strategies of planning 8
3.1.3.1 Subgoaling 8
3.1.3-2 Constraint reasoning 10
3.1.33 Meta-Ievel reasoning 12
3.1.3.4 Hierarchic levels 13
3.1.3S Learning 14
3.1.4 Assessing the importance of the strategies 15
3.1.4.1 Decision-making, certainty and commitment in search 15
3.1.4-2 Appropriate decomposition: a key planning issue 16
3.1.5 Summary 17
3.2 Planning in design 18
3.2.1 Uncertainty in planning and design 19
3.2-2 Goals and interactions in design 20
3.23 The strategies of design 21
3.2.4 Integrating insights about the strategies 24
3.2.5 The design of design systems 25
3.2.6 Summary 26
3.3 Abstraction in problem-solving 26
3.3.1 General discussion 27
3.3.2 Abstraction in Design systems 31
3.33 The use of abstraction in space planning 32
3J.3.1 Human strategies 35
3J.3.2 Automated system strategies 36
3333 Decomposition: a key abstraction strategy in layout 37
3.3.4 Summary 38
4. Problem Statement 41
4.1 Motivation for the proposed research 41
4.1.1 LOOS: an automated layout system 41
4.1.1.1 Significance of the approach 42
4.1.1-2 Appropriateness of the basic architecture 44
4.1.13 Leveraging search in LOOS 49
4.1.1.4 Assessing limitations of the approach 50
4.1.1.5 The combinatorics of systematic enumeration 54
4.1.1.6 Directions for improving performance 55
4.1.1.7 Making the approach scalable 58
4.2 Description of Research 59
4.2.1 Principal objectives 60
4.2.2 Leveraging synthesis in layout design 60
4.23 Decomposition through "goal-objects" 63
4-2.3.1 The role of "goal-objects"[9] in layout design 63
4.2.32 The source of decomposition hierarchies 65
4.2.4 Development of an extended system: (ABLOOS) 65
4-2.4.1 System characteristics using GOBs 66
4.2.4-2 Control options: Combining planning with G&T 66
4.2.5 Research results: additional leveraging of search 67
4-2.5.1 Relation to other capabilities 68
4-2.5-2 Summary: GOBs as a general reservoir for knowledge 70

4.3 Plan of Research 70
4.3.1 Specification of decompositional abstractions 70
4.3.1.1 Interesting technical issues 71
4.3.2 Gradual development of the new abstraction mechanism 72
4 J.2.1 A simple rectangular example using goal-objects 72
43J2J2 A non-rectangular example using goal-objects 75
***43.23* Additional possibilities for GOBs** 79
4.4 Environment and Implementation 80
4.5 Schedule of research 82
4.5.0.1 Sequence of tasks to be completed 82
4.5.0.2 Schedule of completion of tasks 82
5. Significance and Contribution 85
I. A brief review of AI planning research 87
1.1 The formal roots of planning 87
1.2 A brief chronology of the developments in planning 92

1. Introduction

Abstraction is the quintessential mechanism for dealing with complexity. Abstraction may take many forms and appears to underlie all of the important strategies for handling large search spaces in planning domains. Among the most important bases for forming useful abstractions levels are dropping of detail (generalization), and hierarchic decomposition (aggregation). Design, modeled as search, also depends heavily on the use of mechanisms of abstraction, and may benefit from the employment of strategies to leverage search that are similar to those developed in automated planning. In some design domains, such as automated layout, the fact that the objects manipulated have shape and take up space adds an interesting twist and certain complexities to the use of abstraction mechanisms for handling the operators, objects and goals of the problem space.

The automated design of layouts has been a research area for approximately 20 years, in part because it was realized eaiiy on that spatial reasoning would have to play an important role in many types of automated problem-solving.[1] The positioning and dimensioning of objects in two-dimensional space is an interesting problem which cuts across domains, involves geometry and some of the difficult issues of representing and reasoning about it, but is restricted enough that progress has been made in its formalization.. Progress in developing systems to perform spatial reasoning, including the more restricted problems of composition in two dimensions (space planning or layout problems) has not been steady due to the much greater difficulty involved than originally anticipated.

With respect to leveraging search in a design system, this proposal focuses on the combination and interaction of various forms of abstraction in the context of the automated synthesis of layouts. This research issue converges with a unique opportunity to explore it in the context of an especially promising approach to space planning. That approach, called LOOS,[2] has a representation and an overall system architecture which formally model layout design as search in a well-defined space, and suitably addresses some of the problematic limitations and compromises of former approaches. Research on the LOOS approach has reached the point of "demonstration of concept" on realistic problems of limited size and complexity. A whole host of issues, problems and possible elaborations of the approach naturally arise. The background section of this proposal recounts the results of a survey of the planning literature, which I conducted in a search for systems, concepts or mechanisms which could be combined with or merged into the LOOS approach and leverage its capabilities. This background serves as the basis for reviewing the current limitations and possibilities for extending the generality and efficiency of the approach.

Of these possibilities, I present the case that incorporation of an additional mechanism of abstraction based on hierarchic decomposition will most directly address the fundamental problem of the combinatorial explosion in the search space. The remainder of the proposal describes the creation of a new framework to represent and control decompositional abstraction levels in layout design. This new

---

[1] **"Since** much **of design,** particularly architectural **and** engineering design, is concerned with objects or arrangements in real Euclidean two-dimensional or three-dimensional space, the representation of space and of things in space will necessarily be a central topic in a science of design" [Simon 81,p 132].

[2] In [Hemming 86a] a restricted pilot version of a generator for the automated layout approach was called 'LOOS'. Since then the overall approach has been extensively developed but has continued to be called 'LOOS' as a whole by its developers. Therefore, with little risk of confusion, in the future we will use the name 'LOOS' to refer to the approach which expresses a paradigm for automated space planning. The paradigm combines the theoretical underpinnings for representation and generation with an overall system architecture based on incremental generate-and-test within a knowledge-based framework. The theory and basic system architecture were developed by Ulrich Hemming; see the section on the current LOOS system for details and references.

abstraction mechanism will strongly restrict the number of the objects that the layout system has to deal with at a given time, while also providing an appropriate place to store additional knowledge and constraints for pruning. Therefore, it will not only help to tame the explosion in the search space but will also add flexibility and adaptability to the approach. This new abstraction framework is proposed as a general hierarchical extension to LOOS's domain-independent problem solving structure, accepting domain-dependent decompositional knowledge as content.

Creation of this decompositional abstraction framework will involve fundamental issues (such as the *representation* of useful goal/object abstractions in layout design) which may support incorporating other powerful behavior into the system, such as the use of prototypical plans and learning. Another most immediate and essential direction, the creation of a language for layout - facilitating non-programmer entry of problem description and constraints into the system and allowing for constraint-rejected generation and more efficient evaluation of partial layouts - will be undertaken concurrently by another researcher. Taken together, these two extensions will effectively enhance the heuristic and epistemological adequacy of the approach.

In summary, this paper proposes research with the general goal of exploring the combination of hierarchic decomposition with other required forms of abstraction in the modeling of the design process as search. The context of space planning is chosen because it is important in its own right, it reflects in a restricted form fundamental issues regarding the manipulation of physical objects in a design process, and finally, because there exists a promising automated approach to space planning (LOOS) with a formally developed representation and overall architecture that support a search process in a well-defined space. As a more specific goal, the proposed research will extend the capabilities of the LOOS paradigm in one of the most obviously promising ways in order to create a more powerful automated space planner than those currently available.

2. Organization

In this paper I proceed from the general background of building intelligent systems to increasingly more specific considerations regarding the use of abstraction to leverage search in design systems. Then, from a description of the current status of the LOOS project (demonstrating a representation and design process architecture for accomplishing automated layout design) to a proposed system, ABLOOS, which will make use of additional abstraction mechanisms to address the limitations of the approach while preserving its unique assets. Thus the first half of the paper is an attempt to provide the background and general motivation for exploring abstraction by decomposition in the context of an automated design system. The second half describes a specific approach to automated layout, LOOS, evaluates its current development, and justifies its extension as an appropriate context within which to explore the larger issue.

This organization, converging from the general strategies involving abstraction to its use in a specific context, helps to motivate a research agenda which has both general and specific objectives. Of general interest is the examination of certain issues that arise in the design of automated design systems that have to deal with the shape and geometry of objects. Of particular interest is the goal of building a more capable automated space planning system than those currently available.

Since the LOOS approach is a critical foundation to the proposed research, it might seem logical to describe and discuss it first However, I do not discuss it at all until after the background section for two important reasons. First, it is much easier to describe critical aspects of the architecture of the approach after setting forth some of the best known general strategies of intelligent problem solving and planning[3] The LOOS approach already uses many of these ideas in a clever and unique way which is more easily explicated with the previous discussion and terminology in place. Second, the research goal here is as much a general exploration of the role of abstraction techniques for leveraging search in intelligent design systems, as it is the specific goal of extending a promising automated space planning approach.

The remainder of the paper describes the proposed research in terms which build on and elaborate the architecture of the current approach and concludes with a plan and schedule for accomplishing the research. With this general outline in mind, the division of the paper into the Chapters, "Background", and its subsections dealing with planning, design and abstraction, and "Problem Statement", and its subsections dealing with motivation, description and technical plan should be clear and self-evident.

With the inclusion of the extensive background section, this paper may appear to be intimidatingly long for a proposal. However, the second half of the paper covering the proposed research can stand alone so that the reader who wishes to skip the "Background" section can easily do so. In that case, perhaps a quick scan of the summaries at the end of the major subsections will help to place the proposed research in clearer perspective.

---

[3]The research proposed here builds squarely on core components of the LOOS approach and may not be clearly understood if the essential elements of that approach are not familiar and understood by the reader. Space does not permit repeating an explanation of the approach in the thorough enough manner in this paper. So I recommend that the reader consult either the brief overviews of the LOOS approach to be found in [Hemming 88a] or [Hemming 88b], or the extensive final report completely documenting the research on the LOOS approach up to this point [Hemming 88c].

3. Background

3.1 Planning methods in AI problem-solving

In everyday terms planning is prearranging a sequence of actions to accomplish a goal. The essence and value of planning is that it not only makes the achievement of tasks more timely and efficient, but that in many cases it makes possible the achievement of some goals which otherwise could not be attained due to unforeseen complications. In the explicit analysis and modeling of human problem-solving, which is an important branch of research in Artificial Intelligence, it has been noted that planning is an important capability of any intelligent agent [Rosenbloom 86]. At a recent conference [Linden 88] the reflections of some of the most experienced researchers in the field of automated planning reiterate that *plans* are representations for proposed courses of action which can be reasoned about and evaluated before the action is undertaken, and that *planning* is the most important technique available for building systems that can act in flexible and intelligent ways. Thus, at a certain level of expected performance, the idea of combining planning capability with a problem solving system not only seems very natural and intuitive, but essential.

An early precedent in this regard is the Dendral system(~1968-1978) [Buchanan 71][4]. It demonstrated an effective combination of a planning mechanism, to satisfy performance standards, with a complete and correct, but otherwise hopelessly inefficient problem-solving architecture which included an exhaustive generator of molecular structures. What was generalizable from this experience was the combination of planning and problem-solving, not the particular planner whose effectiveness was totally dependent on the knowledge of organic chemistry structure.[5]

3.1.1 Domain-independent strategic planning

Since the early 1970's, planning itself has been studied very intensively in the form of AI planning systems. The type and style of planning that became the focus of this research grew out of the attempt to model simple tasks and guide robots in their execution (in a certain limited facsimile of the real world). Out of the attempts in this direction there quickly emerged a paradigm and a quest for a type of planning capability which was coined *domain independent conjunctive planning*(DICP). Domain independence was desired so as not to have to build a new planning component for every new task in each different domain. Conjunctive planning meant that a planner ought to be able to take into account and solve several goals simultaneously, especially if planning was to take advantage of the "divide and conquer" approach of problem-solving as pioneered and exemplified in automated problem solvers by Newell and Simon's **GPS** [Newell 72](See AppendixI for further discussion.) This approach, variously termed *problem reduction*, *subtasking*, or *subgoaling*, decomposes a problem into relatively independent subtasks and considers it solved if the conjunction of each of the subtasks is solved.

Al planning has met only limited success in achieving the goal of practical DICP (that is, a standard methodology that can be applied to real problems from a variety of domains.) The lack of success is due to certain fundamental structural limitations built into this approach to planning in order to preserve its generality, minimal efficiency and essential completeness; see [Chapman 85]. Unfortunately, the same

---

[4]See appendixI for a summary of Dendral's problem-solving architecture and planner

[5]Although the mechanism employed for planning, *constraint generation and satisfaction* has since been utilized in many other planning domains and subsequently studied and abstracted into a well known general concept.

set of assumptions and limitations does not apply effectively to all domains in which planning capability is required. Notwithstanding this crucial applicability criteria, much of the research on planning that has been done over the last fifteen years, and that yielded an ordered sequence of incrementally more capable planning systems - STRIPS, NOAH, NONLIN, SIPE, DEVISOR - assumes a great deal of predictability about the domain at hand. These traditional AI planners, even the most recent state of the art models, assumed *a static state of the world* in which nothing changes unless changed by *a sole acting agent(cg:* a robot) whose *actions are* dfccrcteQnstantaneous), *non-overlapping(orly* one primitive action at a time) and *deterministic(ccttain* and fully predictable effects). Because of their assumptions about complete information and about the lack of time limitations for decision-making these planners have been termed *strategic* planners; see Figure 3-1. These assumptions facilitated some sophisticated representations and mechanisms for reasoning about restricted types of actions under a relative degree of certainty.

However, few real world domains meet the strong predictability requirements necessary for the methods of the strategic planners. Therefore, lately, much of the emphasis in planning research has shifted to reactive *tactical(scc* Figure 3-1) planning in which the available information is limited and uncertain, and the amount of time for the system to respond may be extremely limited. This type of planning, also called *situated activity,* is concerned with deciding what to do and acting properly based on what is currently known, where what is known is assumed to be significantly less than what is true [Dean 88]. This type of planning, essential to sensor-directed robots, requires modeling or handling directly many factors, such as: multiple agents, and/or uncertainty in the effects of the actions of agent(s); continuous, non-discrete actions by agent(s); scheduled or non-scheduled events, other than the agent's actions occurring in the world; actions being constrained by real-time intervals or windows and not just the precedences established to avoid the interactions of goals which otherwise decompose into discrete, instantaneous actions, etc.

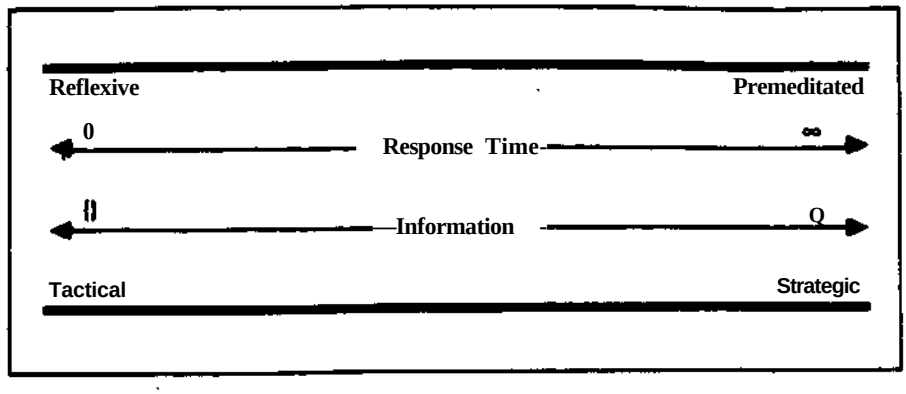


Figure 3-1: Two Extremes in a Continuum of Problems, from [Dean 88]

Though the methods of the strategic planners are no longer viewed as entirely suitable to problems such as robotic planning, they encompasse a major portion of all the "domain independent" search-based problem-solving research done in AI over the last twenty years. Therefore the strategic planning methods may still have much to contribute as knowledge sources for guiding the design and implementation of any problem solver modeled as a search process. Even with the restrictive assumptions about certainty there remain many difficult issues of strategic planning that are not completely resolved and are of theoretical interest: "the frame problem, temporal persistence, search control of a potentially huge search space, tmth maintenance, interaction detection and resolution, temporal representation and reasoning(that is both expressive and efficient), goal representation (including trade-offs between goals), constraint handling, and so on" [Cheeseman 88]. Despite these remaining issues, there has been some progress in developing

useful strategies for planning in relatively static domains **and those strategies are discussed below. All of** them, in one way or another, depend on solving a conjunction of subgoals, or dividing a problem into parts, producing plans for those parts and integrating and reconciling the parts. However, in terms of achieving true domain independence, even the basic subgoaling technique has difficulties in both representation and methodology as indicated by an important conclusion of the most recent DARPA workshop on planning: "Thus, just as there seems to be no practical general planning method, it appears also there is no useful, ultimately general plan representation that allows goals to be decomposed in domain-independent ways" [Linden 88].

3.1.2 The value of strategic planning

With the above points in mind, some pertinent questions arise when contemplating the addition of planning capability to a design system: What is the relationship of planning to design?; What factors would determine or guide the effective integration of planning with an automated design system?; and, What concepts can be taken from this vast body of planning research to support automated design? In order to answer these questions I conducted a thorough literature search of the strategic planning research. I found that merely reviewing and attempting to classify the planning systems was a non-trivial task.

It would be helpful in understanding the significant methods of the AI planning research if it were straightforward to classify the AI planners with respect to the different issues involved in the planning process. Reasonably good overviews or histories (some outdated as per their time of publication) are provided in [Chamiak 86, Chapman 85, Rich 83, Cohen 82, Sacerdoti 80, Mcdermott 78, Waldinger 77], a useful annotated bibliography is provided in [Christiansen 85], and a very good concise attempt at categorization of the AI planners along six dimensions is provided by [Tate 85]. All of these sources contain worthwhile views on the useful methods of planning, and though they share much common ground, each also offers unique insights and slightly different views on the meaningful developments. A recent thesis on construction planning also contains a review of planning methods and illustrated comparisons of various planning algorithms in solving the same simple blocks-world problem [Zozaya 88]. Zozaya concurs with the difficulty I mentioned above, that of coming up with an adequate classification scheme for reviewing the AI planners. This is because the planners use different combinations of search, representation and plan manipulation strategies. A recent DARPA conference on planning [Swartout 88] also confirms the fact that AI planning research has never coalesced into a coherent paradigm.[6]

This proposal will not discuss the literature overview in any detail, but the interested reader may consult Appendixl for an attempt at classifying the issues, mechanisms, and components of the various planners encountered in a brief chronological survey of the field. On a general level some important conclusions emerge from this search:

1. There does not (yet) exist a domain-independent planning system or algorithm that one could likely adopt whole cloth, or feel secure in employing as a "black box" part of a problem solver in most domains, especially design (though particular domains may be exceptions). Nor is this situation likely to change due to fundamental restrictions required in the representation in order to achieve DICP. Again, the interested reader is referred to Appendixl, or [Chapman 85] for more details.

---

[6]A possible exception to this lack of coherence is Chapman's work, cited above, which is remarkable in that it formalizes the combination of representation and methodology used in die bulk of the strategic AI planners over an extended "scruffy" phase of approximately 15 years.

2. There is a small set of potentially extremely effective mechanisms or *strategies* which have emerged form the general planning research and may be abstracted away from the planning paradigm as general *concepts* for leveraging problem solvers. When these concepts are *instantiated* in particular domains as knowledge sources - dependent-on and intricately utilizing **the** particular structure, processes and entities of a domain - *they may serve as* ***effective means for reducing the work-load of problem solving in that domain.*** **In essence,** the planning research has contributed a great proportion of the currently best known concepts to serve as the basis for building effective *domain-dependent problem-solving* ***capabilities.***

Point (2) above reveals the true contribution and value of the general planning research to automating problem-solving in other domains and disciplines, such as design. Of course, these concepts were not invented by AI planning research. Most have been employed for ages in some fashion in many disciplines, notably design, as will be discussed later. Polya's "How to Solve It" introduced many of these concepts to the modem problem solving literature and inspired their use in many AI systems [Polya 57]. The contribution of the AI planners has been to pave the way for operationalizing and implementing those concepts as effective strategies in *automated problem-solving.*

3.1.3 The strategies of planning

Planning can be viewed as *search* using additional, more powerful "knowledge sources" [Korf 87] or *strategies,* beyond combinations of the heuristic search devices known as the weak methods. Figure3-2(from [Tate 85] shows an excellent family "tree"(allowing for some incest!) of planning systems and strategies. The remainder of this section discusses the significant methods, or *strategies* of the planners, on which there is much greater, though still not complete, agreement. By and large my selection and informal classification of the important strategies follows the consensus of most other reviewers but it is not meant to be taken as complete or rigid. Below is a listing of the most important followed by a discussion of some of the advantages and possible limitations of each.

3.1.3.1 Subgoaling

It is known that the appropriate use of subgoals can greatly reduce the amount of search, because decomposing an exponential problem into two or more simpler problems tends to divide the exponent, and hence drastically reduces the total problem-solving effort [Minsky 63] as cited in [Korf 87]. Most planners have incorporated reasoning backward from goals based on a problem reduction approach and therefore most plans have a nested subgoal structure. These subgoal structures are inherently hierarchical and the search spaces in which plans are developed are hierarchical because the problem-solving operators have preconditions that are subproblems with preconditions of their own, etc. Thus, even the non-hierarchical planners typically develop a hierarchy of subgoals, but since they have only a single level representation of a plan, the subgoals are all at the same level of abstraction.

Despite its widespread use in reducing complexity, subgoaling, or problem reduction, has an associated liability, namely, that subproblems are rarely independent. Korf [Korf 85, Korf 87] analyzes subgoaling as a strategy and categorizes types of subgoals in terms of their levels of interactions, difficulty of handling, and impact on the complexity of search. The categories he presents are:

- ***Independent subgoals***
  They do not interact and therefore they divide both the base and the exponent of the complexity function by the number of subgoals. Many everyday tasks are independent except when they are constrained by global resource limits such as time or money, which only introduce dependence among subgoals when one begins to approach the limits they

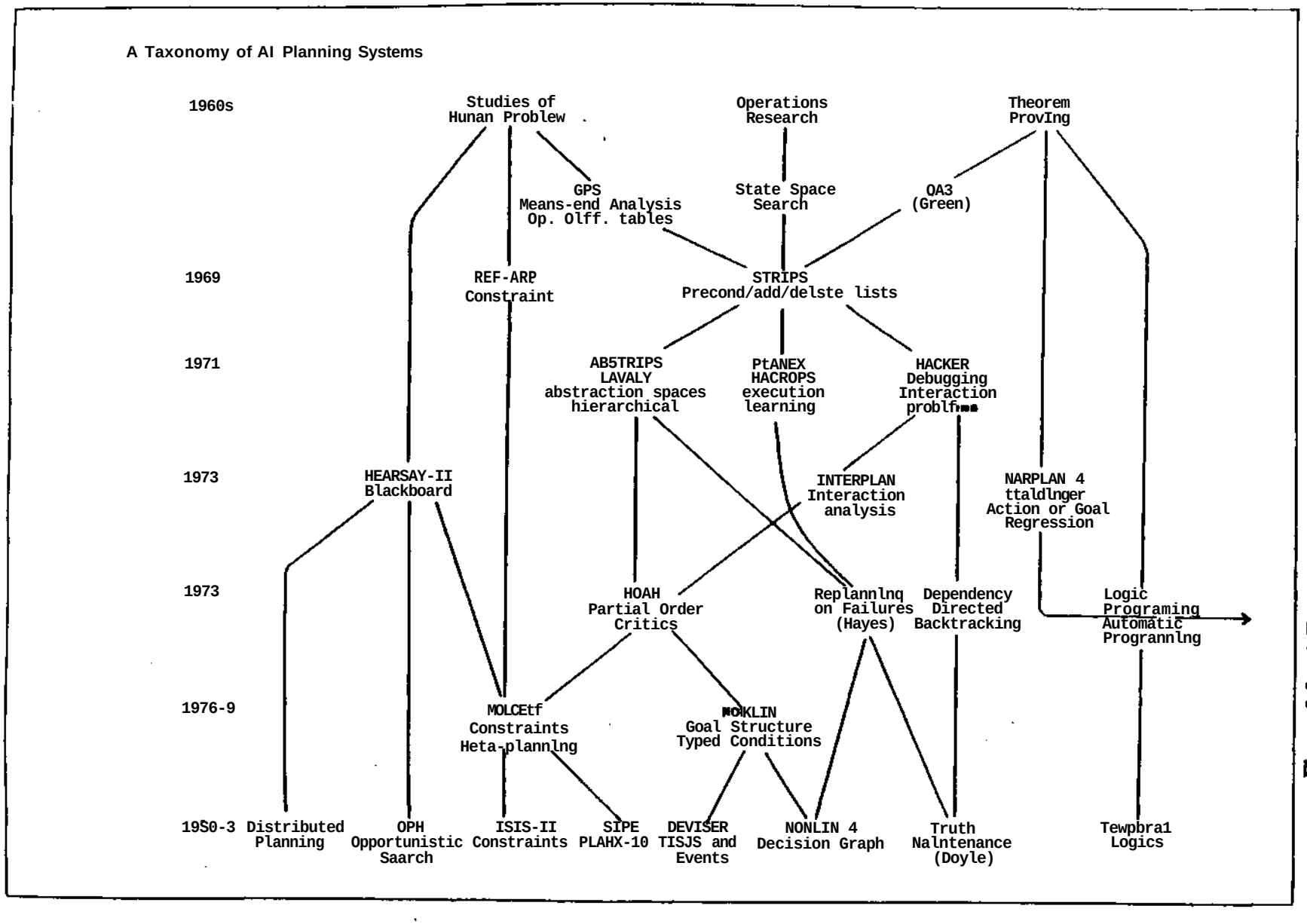


Figure 3-2: A Taxonomy of AI Planning Systems, from [Tate 85, p 6]

impose.

- *Serializable subgoals*
  Serializable subgoals are those with the property that there exists an ordering among them such that once a subgoal is satisfied it need never be violated in order to satisfy the remaining subgoals. Their value is not completely obvious since protecting previously solved subgoals may increase the overall solution length since certain operators cannot be applied at certain points. However, they are a viable strategy (where applicable) because they reduce the branching factor of the space since the number of legal moves is decreased by operators that violate the subgoals being ruled out.

- *Non-serializable subgoals*
  this is a set of subgoals such that all known algorithms to solve the set require that previously satisfied subgoals be violated later in the solution path. Non-serializable subgoals not only do not decrease the branching factor, but have the effect of adding to the solution length as do all subgoals. The possibility that decomposition may result in non-serializable subgoals indicates that decomposition of goals into subgoals must be done carefully in a domain dependent, if not problem dependent, manner. In fact, the subgoals represent abstraction states that must be tailored to each problem such as the precompiled subgoal hierarchies required by the planners such as NOAH.

- *Block-serializable subgoals*
  Sometimes serializability is a function of the granularity of the set of subgoals; that is, a sequence of non-serializable subgoals can be made serializable by simply grouping together multiple subgoals into a single subgoal. Korf calls such a sequence of subgoals block-serializable. The trick is that a block-serializable subgoal may then be matched to a *macro-operator,* which leaves all previously satisfied subgoals intact while satisfying an additional subgoal as well. Within the body of the macro, previous subgoals may be disturbed, but by the end of the macro all previously satisfied subgoals must be restored. New instances of problems may then be solved in a "macro network" space, rather than in the original problem space with great computational advantage. He states that the advantage is, in fact, exponential and the idea is known as *abstraction.*

In summary, Korfs quantitative analysis of the knowledge sources suitable as a basis for planning suggests that there is an intimate relationship between subgoals, macro-operators and abstraction spaces. It also suggests how these strategies may be utilized to begin to deal with less well-structured problems through subgoaling and techniques to deal with the more difficult types of subgoal interactions.

3.1.3*2 Constraint reasoning

Employing a *least-commitment* strategy means that decisions are deferred as long as possible to minimize arbitrary choices - in other words, avoid making an uninformed guess that might result in backtracking. The intimate connection between constraint-reasoning and the least-commitment approach to decision making is based on the use of constraints as an explicit way to move information around between subproblems and thereby alleviate interactions and *opportunistically* focus the attention of the problem solver on the most *certain* part of the problem. Constraints are a type of abstraction in that they can be viewed as partial descriptions of entities. Constraint *formulation* or "posting" guides the hierarchical creation of problems in contrast to the former use of constraints for *constraint satisfaction* which is a technique to determine the values of variables in a completely specified problem. Other operations on constraints are *propagation* (for passing information between nearly independent subproblems), and *satisfaction,* (refining abstract entities into specific objects by choosing values for the variables.)

Least-commitment is also closely related to abstraction. Abstraction ignores some aspects of a problem

as detail, and least-commitment postpones decisions[7] not forced by the problem, thereby leaving as much freedom as possible for achieving subsequent goals (without backtracking). Despite this positive leveraging aspect, the least-commitment strategy has two important flaws that must be noted. First, it is a heuristic that trades-off the deferring of decisions against not having to backtrack. In doing so it allows many partial solutions to stay active in the search space, which in some domains might just prove to be more costly than risking commitment with the possible less costly penalty of backtracking from mistakes [Dean 88]. Second, a more subtle, but possibly even more serious drawback to least-commitment is that when a problem solver runs out of decisions it can make immediately, it must make a *guess* in order to continue. This inherent difficulty with *pure* least-commitment approaches has been called the phenomenon of *least-commitment deadlock*. Thus, though least-commitment attempts to avoid wrong choices and *backtracking*, in some problem solvers it can lead to an impasse where a relatively uninformed choice is made just to keep the process moving. These problem-solvers then make provisions for elaborate and expensive backtracking mechanisms as a fall-back.

Least-commitment is an attempt to extend the problem-solvers reasoning under certainty as far as possible, and it is hoped that the "multiple coverage" on constraints will ensure that those which remain "undecided" at any one cycle in the problem solving process will be able to be decided with certainty later through accumulated decisions on other constraints [Mittal 86a]. However, least-commitment is not a guarantee of certainty, it is only a heuristic and not an easy solution to fundamentally difficult problem. *Monotonic reasoning* is defined as "a reasoning system based on the assumption that once a fact is determined it cannot be altered during the course of the reasoning process" [Rosenberg 86], Monotonicity in search guarantees that any significant results produced through execution of an operator will not be violated by new operations. It involves *complete commitment* on some aspect of the problem (careful abstraction of the representation of the problem may isolate critical aspects on which monotonic choices can be made) and thus involves no guessing or backtracking on decisions taken in terms of those aspects.

In most systems backtracking is not so much a strategy, as it is a default failure mechanism. It has sometimes been confused as a strategy when, in addition to providing for failure handling, it has been equated with a multiple worlds ability or the ability to find all the solutions. However, even guided backtracking augmented with an advice mechanism will be inadequate in many cases since the viewpoint from the choice point in question is largely localized and the failure which caused the backtracking may be global in character or have collective sources. An example is the consumption of a global resource such as space, where if too little is left at some point, it is not clear where to assign blame.

The work on using constraints to guide problem-solving has been steadily elaborated since MOLGEN [Stefik 81a, Stefik 81b] and extended to other complex domains involving search, such as factory scheduling [Fox 83] and design [Mittal 86a]. Fox [Fox 86] describes the basis of a theory for constraint-directed search by elaborating the semantics of constraints to guide search. The work of Mittal and others at Xerox PARC has elaborated the use of plans in design and demonstrated strategies for extending the certainty of least-commitment and the use of constraints as an incremental approximation scheme.

---

[7] Choices which expand exponentially, that is, decomposition operations which create subcomponents, are deferred in favor of deducing all "details" that expand linearly. The refinement of components at a certain level through detailing operators or propagation of constraints are preferred because they may actually help to tighten constraints thereby causing the search to converge.

3.1.3J Meta-level reasoning

It has long been realized that the performance of problem-solvers can be improved by supplying various sorts of **knowledge about** the **knowledge** in the system; knowledge about knowledge is often called "meta-knowledge". **Just as** there arc many possible types of knowledge there arc many possible types and levels of meta-knowledge. It has been pointed out that experts have a great deal of perspective on their own problem-solving behavior and programs ought to have some of that reflexive capacity also [Hayes-Roth 83]. Explicitly represented meta-knowledge may help greatly to define and justify the architecture -the organization and control structure - of a system.

The tenn *metaplanning* has been used to describe systems that plan about their own planning processes. Since planning is a 'reflection' on acting, the plan level is a meta-level above the action level. Also, before a system takes a decision(acts) at any given level it can in principle push up to another level for reflection about the pending decision "... the choice of whether to act or reflect applies at all abstraction levels" [Linden 88]. The earliest and perhaps most important use of meta-level knowledge in problem-solvers is to perform "conflict resolution". Conflict resolution - choosing among the many rules that may be relevant at the moment - is an instance of the more general *control problem:* "Which of its potential actions should an AI system perform at each point in the problem-solving process? [Hayes-Roth 85]."

Meta-level reasoning can be organized in many ways including a fully recursive structure in which the same inference engine reasons about rules at all levels: rule, meta-rules, meta-meta-rules, etc. Or it can be organized as *separate layers* for metaproblems at each level until the infonnation in the top-most layer is trivial. General proposals for expressing control knowledge in terms of metaproblems have been advanced by [de Kleer 79, Weyhrauch 80, Wilensky 81], general meta-level architectures are discussed in [Genesereth 83, Davis 80a, Davis 80b], and examples of systems with explicit meta-rules for control are [Takewaki 85, Orelup 87].

There are some persistent doubts about the overall value of metaplanning: [Biikel 86]

> All planners metaplan; few do so explicitly. Many planners find very simple control mechanisms sufficient; the added overhead of a metaplanner outweighs any apparent advantages. Whether implementing an explicit metaplanner increases the capabilities of of the resulting system is unknown.

It may be interesting but seems dangerous to assume that resolution of issues that a system faces can always be pushed up to some next meta-level where there will be some kind of knowledge available to do the job. Tong has noted that the importance of distinguishing qualitatively different types of inference engines was blurred by referring to all such distinctions as 'metarule' [Tong 87].

Meta-level control reasoning, as an approach to solving the general control problem, ultimately has to do with focusing the attention of the problem solver in an *opportunistic* way. Opportunism, or "island driving" is a strategy for controlling the focus of attention of the problem solver that differs from fixed predetermined strategies such as top-down or best first. It dynamically and tactically focuses the problem-solving effort in highly constrained or more certain areas of the search space in order to minimize the combinatorial explosion. It enables highly certain knowledge, at whatever level of abstraction it is found, to focus the search with the expectation that focusing on these "islands of certainty" will allow the extrapolation of further constraints in less certain areas.

Opportunistic problem-solving was introduced along with the blackboard paradigm by the HEARSAY systems, and became a method of planning through adoption of the blackboard architecture into planning in the 0PM system [Hayes-Roth 79]. The strategy of opportunistic reasoning is to reduce the size of the space that must be searched by identifying and using islands of certainty as anchors for search. The

blackboard and opportunistic commitment approaches were invented partially to address the problems of reasoning in situations involving uncertain or noisy data from multiple sources, large solution spaces with no plausible generator, etc., while maintaining as high a degree of certainty as possible in the decision processes. To most effectively handle the scheduling of the focus of attention of the problem solver the blackboard model has been extended to include *multiple blackboards*, some of which are concerned with control, others which are concerned with the problem domain [Hayes-Roth 85]. Apparently, the opportunistic model for control decisions in problem-solving is excellent for incorporating many diverse planning methods [Linden 88].

3.1.3.4 Hierarchic levels

Abstraction levels and hierarchies are mentioned frequently, and the terms are often confusingly intermixed in planning research. It is important to distinguish different hierarchies within planning systems. Hierarchical levels can be based on subgoaling, abstraction levels, or reflection, and these principles are largely orthogonal and interact in complex ways [Linden 88]. I noted above that subtasking is inherently hierarchical. However, only *hierarchical or abstraction level based planners* generate a hierarchy of *representations* of a plan in which the highest is a simplification, or *abstraction* of the plan and the lowest is a detailed plan, sufficient to solve the problem [Cohen 82].

ABSTRIPS [Sacerdoti 74] used a simple and efficient approach for representing abstractions (assigning criticality levels to preconditions of operators) and is an instance of the simple *top-down refinement*(see [Hayes-Roth 83, p 106] approach to *abstraction by levels*. In the top-down refinement approach the problem solver proceeds from the top downward through fixed and predetermined abstraction levels, and *solutions to the problem are completed at one level before moving down to the next more specific level.*

In NOAH [Sacerdoti 75] the planning process also proceeds through abstraction levels via a hierarchic decomposition (subgoaling) of the operators present in the plan in each cycle. Though the levels of operators are fixed by their subgoal hierarchy rather than criticality levels, the abstraction levels are proceeded through in a fixed order and completely (all operators) at each level. The difference in NOAH is that it also uses the *least-commitment* principle for "detailing" decisions (constraints on the ordering of operators and identity and parameters of objects). However, NOAH does not use least-commitment for selecting the next operator to decompose; that *choice* is still done under a top-down refinement regime. Therefore, in principle, it would be possible to capture the abstraction levels of ABSTRIPS by setting up a NOAH-style subgoal decomposition hierarchy that captured the same information as criticality levels; however, the representation in that case would likely be more cumbersome and less efficient. The big difference comes in the MOLGEN system which utilizes least-commitment, wherever it can, to guide all of its decision making processes - including the choice of which operator to decompose next. There, operator decomposition choices as well as details in the plan can be incorporated at multiple levels of abstraction rather than uniform levels. This is accomplished through constraint reasoning and a more complicated (meta-level) controller guided by least-commitment.

It is clear, as Tong [Tong 87] points out, that hierarchical problem decomposition and problem factoring via abstraction levels are both 'divide-and-conquer' techniques, but they also differ and complement each other in important ways. Hierarchical problem decomposition generally cleaves the solution space in a single way that cuts through levels, e.g., decomposition with respect to geometric location in floorplanning problems. Abstraction by levels, on the other hand, often assigns a concern to each level, which cleaves the space in a different way at each level. These two techniques may interact and be combined in many complex ways which provides the basis for an interesting research issue in many domains. The communication and control between these different techniques of abstraction leads to

reasoning about the focus of attention of the problem solver which is intertwined with the third hierarchical issue - meta-level control reasoning.

To complete this extremely brief discussion of hierarchic levels it is necessary to make the connection between meta-level reasoning and reflection hierarchies mentioned above. The following quote should serve this end: [Linden 88]

> *Reflection* is the principle that distinguishes between acting and thinking about acting. If we take action as a basic..then planning is reflection about acting, and metaplanning is reflection about planning. Thus, we can view planning as metaactivity; it is also reasonable to assume that there should be an order of magnitude or more of activity than there is of metaactivity. A reflection hierarchy is orthogonal to an abstraction hierarchy. This distinction was made clear in Molgen, where there was reasoning about domain objects at multiple levels of abstraction as well as metaplanning. In Molgen, one could view the reflection hierarchy as primary, with the domain objects organized in an abstraction hierarchy. For robot control, one might rather view abstraction levels as primary, leading to an opportunity for reflection (planning) in the control decisions at each level of abstraction.

To summarize, hierarchical planning is perhaps the best known but most misunderstood of the approaches to planning. In all forms, it is in essence a motivation for abstraction: away from the primitive operators of the plan; away from commitments to a precise order in which to cany out those operations. In retrospect, perhaps the most important use of hierarchical abstraction is concerned with the relation between tasks and subtasks [Dean 88]. A planner can decide to perform action A and later decide to do A by doing Al...An. Usually, when a researcher states that a planner is hierarchical, he means that system uses this sort of abstraction(or *aggregation hierarchies of operators).* The basic ideas behind hierarchical planning have come to be recognized not so much as an approach to planning but rather as part of the problem statement. The planning of a problem largely involves *& problem-structuring* phase wherein the correct formulation of the problem occurs and allows it to be handled by the *problem-solving* part of the system. That is, *the goal or operator decomposition has to be precompiled based on the knowledge of the domain*[8] Thus, in an important and fundamental sense, planning *is* hierarchical and abstract.

3.1.3.5 Learning

Learning in the planning research is related to the following: skeletal planning, macros, and case-based planning. These concepts represent a set of important ideas in problem-solving which were integrated into planning research very early. These are the intertwined ideas of *learning,* and the use of preformed plan segments, sometimes called *skeletal plans* or *macros.* They are used to enable planners to avoid having to repeatedly plan typical tasks from scratch.

Two important ideas surface from the research on learning in planning [SUSSMAN75]: first, planning should consist of using (re-using) known plans wherever possible (and debugging them in the current context as required); second, once a planner has constructed a useful plan, it should store the plan for later use. The most trivial and direct re-use of a plan segment might involve the retrieval of an exact sequence of primitive operations. However generally the attempt has been to retrieve an *abstracted* version of a plan that contains the basic steps and *instantiate* each of the plan steps within the environment of the particular problem by weaving together the experience implicit in the stored plan and a heavy dose of domain knowledge about the current context The idea of abstracted plans is found as early as the STRIPS planner which parameterized successful plans in order to generalize them [Fikes 72]. The generalized plans were stored in structured, indexed tables for retrieval and were called MACROPS (for

[8] **As discussed later, dynamic decomposition is a possibility, but remains quite difficult in most domains at this time.**

macro-operators). Central to the utility of the skeletal planning approach is knowledge for competent plan selection **and** plan-step refinement. These tasks require an approach on how experiences are retrieved and exploited during planning.

More recently this style of planning is re-emerging as a new paradigm called *case-based planning(CBP)* which takes as its premise that the organization of experience is paramount in formulating new plans and re-using old ones [Dean 88]. Thus, storage and retrieval are vital, and most case-based approaches involve some theory of learning and memory organization. The concept of skeletal-planning or CBP has roots and direct precedent in Schank and Abelson's work with scripts; in their view, plans are generalized scripts that explain events related to, but not exactly like those the user has seen before [Schank 77].

Learning in automated problem-solving is a vast area of research in its own right, going far beyond the simple mechanisms introduced in the planning research. I will say no more here about it other than to reiterate the importance of abstraction (as generalization) to learning, and the synergistic relationship that may exist between a problem-solving performance system and a learning system; see the SOAR architecture [Rosenbloom 86]

3.1.4 Assessing the importance of the strategies

It is clear that the strategies do not exist in pure form or in isolation; rather, they seem interlinked interlinked if not inseparably combined. In particular, the theme of abstraction recurs and seems to be an inextricable part of nearly every one of the strategies. It seems of primary importance that a search-based problem solver incorporate as many methods of effectively abstracting its search space as possible. Therefore, the first planning strategies to be applied should make maximum use of abstraction levels and hierarchical decompositioa Next in importance, the idea of when to make commitment to decisions, whether early or late(least) commitment is fundamentally linked to the use of hierarchy, constraints, etc. All of these strategies must be utilized judiciously and tailored to the characteristics of the problem domain; otherwise, they may result in more cost than benefit in terms of efficiency. For instance, in the employment of concepts of abstraction and commitment it has been "...left to the implementors of planners and schedulers to provide sufficient knowledge to their systems to be able to decide what degree of commitment is warranted by a given level of informatioa The point is that it is a difficult decision" [KEMPF?].

3.1.4.1 Decision-making, certainty and commitment in search

Important strategies involving constraint and meta-level reasoning are related to control and the timing and level of commitment of decisions within the problem-solving process [Stefik 81a, Cohen 82, Hayes-Roth 83, Fox 86, Nfittal 86b,Tong 87, Dean 88]. Problem-solving decisions involve the potentially combinatoric branching or decomposition *choices* as well as a great variety of detailing *deductions.* These strategies involving constraint reasoning depend, more or less, on the idea that constraints may be used to limit search. Variations of these strategies, which fall on a spectrum from less certain to more certain, are early commitment(with patching and debugging), opportunistic commitment(focus of attention with as much certainty as possible), and least-commitment(deferring decisions until certain or else forced to guess). Any of these three may be used separately for either the branching choices (decomposition), or deduction decisions (detailing) of problem solving. Which of these three strategies is most effective in reducing search depends intimately on both the characteristics of the problem, and related to this, the suitability of the overall architecture of the problem-solver.

Early commitment is simple, direct, requires the least overhead but is possibly the least effective for

realistic problems. Opportunistic commitment is the most complex in terms of both machinery and overhead required; its cost has to be carefully weighed against results (empirically) and it should only be employed in problem domains that dictate it, not to make up for an unsuitable problem solving architecture. Least-commitment is the clearest of these three in concept requiring more of an attitude than machinery. But it is only a heuristic and has certain drawbacks, mainly that it may break down at a certain point requiring guessing, which can be costly in terms of backtracking. [Dean 88, Cohen 82]

Another alternative, monotonicity or complete commitment, may be the most powerful strategy for problems to which it can be applied. It is usually the most demanding in terms of the theoretical formulation and representation of a problem, but once this is accomplished it is probably the least costly in machinery and the most effective in reduction of overall effort

3.1.4.2 Appropriate decomposition: a key planning issue

[Simon 81,p 148]

> The basic idea is that the several components in any complex system will perform particular subfunctions that contribute to the over-all function. Just as the "inner environment" of the whole system may be defined by describing its functions, without detailed specification of its mechanisms, so the "inner environment" of each of the subsystems may be defined by describing the functions of that subsystem, without detailed specification of *its* submechanisms. To design such a complex structure, one powerful technique is to discover viable ways of decomposing it into semi-independent parts. The design of each component can then be carried out with some degree of independence .of the design of others, since each will affect the others largely through its function and independently of the details of the mechanisms that accomplish the function.

It appears that having an appropriate problem decomposition, which may be viewed as part of problem formulation, is one of the most vitally important *planning* results that can be provided to the problem-solving part of a system. To date there has been remarkably little woik that addresses automated problem formulation and decomposition; evidently because of the difficulty involved. Nevertheless there is great interest in the (automated) compilation of task hierarchies, because as AI moves into concurrent and distributed problem-solving, task decomposition is affected not only by dependence among tasks and communication issues, but by *dynamically* changing patterns of dependency. So decomposition decisions may need to be revised with contextual changes, and resources available for allocation of tasks may affect distribution and aggregation choices - that is, decomposition alternatives. Therefore, "...the work of making these decisions is itself a task that autonomous (collections of) agents must address; in the extreme, they will need to reason about their representations and formulations of problems and the decomposition/allocation resources they use" [Bond 88 p, 10]. Alternative decompositions have been conventionally provided by alternative problem-reduction operators in an AND/OR graph representation, such as in the planners. As noted, these have been typically precompiled, and actually depend on designer's forethought in operator construction and description.

The following extended quote reiterates the importance of appropriate decomposition and also points out the many issues and difficulties involved:[9] [Bond 88, p 11-13].

> Conflicts over incompatible actions and shared resources may place ordering constraints on activities, restricting decomposition choices and forcing a need to consider decomposition in a *temporal* (or other resource) dimension as well as the more usual dimensions of knowledge, location or abstraction dimensions...In summary, intelligent approaches to task decomposition need to consider the representation of tasks, several dimensions of decomposition, available operators that can be applied to perform subtasks.

---

*In the following extended quotation all italicized points of emphasis are mine, except the word *temporal.*

available resources, and dependencies among tasks.

...three dimensions of problem decomposition. Problems could be decomposed along lines of *location(e.g., spatial,* temporal, logical), by information level or *degree of abstraction,* and by ..."interest areas/ which included the given partitioning of skill among knowledge sources.

But what general basis do we have for making decisions on how to decompose tasks? *The basis of all task-decomposition approaches is to find dependencies and logical groupings in problem tasks and knowledge.* Several bases for cleaving dependencies and for decomposing tasks have appeared in the literature: Abstraction level: ...Although it is tempting to think we may be able to rely on decomposition by abstraction level as a potential for automated problem decomposition, *there remains the great difficulty of how to define, recognize, or to construct these different abstraction levels in a problem.* ...[Several other bases for decomposition are also discussed].

General classes of solutions have been proposed for the problem of task decomposition, although there appear to be few principles, methods, or experimentally validated techniques, for automatic decomposition of tasks... Some classes of solutions[for decomposing problems] are as follows: [Several means of employing decomposition are discussed, of which I pick out only hierarchical planning] ... Hierarchical planning:... A hierarchical planner does genuine task decomposition. It generates tasks that are goals to work on. However, this provides only a partial solution, because the decomposition depends on the prior descriptions of the available execution operators, which are usually static.

In summary, appropriate decompositions have been found and demonstrated for certain domains, and there has even been some progress in automating this process in a few. But decomposition is still a difficult and key issue in harder problem domains (such as design) in which automated problem-solving has yet to yield much success on real problems. The decomposition problem may be intimately tied to appropriate representation of the task, which may involve levels of abstraction from other bases. In many areas of design, appropriate representations are yet to come, or just now emerging. Therefore, in certain domains, the combination of a decomposition hierarchy based on appropriate abstractions with well-suited representation remains an open research issue. The combination of decomposition and abstraction levels will also involve, to some degree, constraint reasoning and reasoning about control at various (meta) levels. Creating a workable decomposition of tasks in a domain, whether automated or not, is a large part of "planning" for efficient and effective problem solving.

3.1.5 Summary

In an overall sense, planning research is large, diverse and unresolved. Key points regarding the value of strategic planning methods to problem-solving are:

- Strategic planning involves problem-solving in a static model of the world.
- Strategic planning research has produced several strategies effective in reducing search in problem-solving. Planning bridges a span between blind search and highly informed search. Problem-solving modeled as search serves as a framework for embedding knowledge of different sorts.
- The strategies are interrelated and abstraction plays a fundamental role in all of them.
- The timing and level of commitment of decisions within the problem-solving process depend on the amount of certainty possible. This in turn, is partially dependent on the characteristics of the problem, and partially on how the problem is modeled - the characteristics of its formal representation, etc. Enabling as much commitment with certainty as early as possible helps to leverage search by allowing the problem solver to ignore major portions of the search space.
- The most basic, but perhaps the most important strategy is subgoaling, or decomposition.

Some types of subgoals reduce search, while others may not. Therefore, how and when a problem is decomposed into subproblems is critical.

- In *essence,* planning is goal-directed behavior and goals provide a frameworic in which to place a great deal of knowledge regarding a task and the problem-solving processes appropriate to solving it. The decomposition of goals through hierarchic levels is a fundamental mechanism for adding the leverage of abstraction levels to a search based problem solver. This mechanism for cleaving the solution space into smaller subspaces may be combined and interact in powerful ways with other mechanisms which introduce abstraction levels into the problem-solving process.

3.2 Planning in design

[Coyne 85a]

> For artifacts that are created in the 'ground' level of abstraction the cost of backtracking tends to be high (although it is less for sand castles than for marble sculptures). It is obviously cheaper to design buildings by considering only certain attributes of walls and spaces, on paper, before committing the design to bricks and concrete, but it may also be appropriate to consider higher levels of abstraction again...It will be noted that this type of exploration (working backwards from goals) is one that can be carried out only in an abstraction of the real world....This is the process of planning: determining sequences of actions that enable goals to be satisfied.

As the above quote suggests, in an important sense, *design* is a planning activity. Just as planning, it is a "reflection on acting", the "acting" being that of building construction in the case of architectural design. In fact, the result or product of design generally constitutes a plan for specifying a sequence of actions to construct an artifact. The relationship of planning and design is complex and largely dependent on the domain and goals in question. Planning and design activities may be very different in terms of certain assumptions and limitations involved. In other ways they are very similar, particularly when both are modeled as structured problem-solving activities involving search [Coyne 85a, Tong 87, Simon 81]. To a certain extent, they both may employ the same strategies mentioned above to good effect in reducing their search load.

Thus far, in attempting to automate design processes, the primary way of modeling design has been as search-based problem-solving.[10] I agree with this viewpoint and its implications as summarized by the following [Chandrasekaran 88]:

> The design problem is formally a search problem in a very large space for objects that satisfy multiple constraints. Only a vanishingly small number of objects in this space constitute even "satisfying", not to speak of optimal, solutions. What is needed to make design practical are powerful strategies that radically shrink the search space.

The effectiveness of problem-solving strategies depends both on the characteristics of the problem, and on how the design or planning problem is modeled - on choosing the appropriate representation and set of abstractions for handling the problem. In comparing the characteristics of experts' tasks, design has many of the same requirements as planning [Hayes-Roth 83 , p 84-5]. The following is a list of some of the key issues: design problems are large and complex with no clear theoretical basis; have a large number of potential solutions; often have no one 'correct' formulation; involve a mixture of local and global constraints and a complex many-to-many relationship between design choices and performance constraints; impose a heavy load on memory if a systematic exploration of alternatives is attempted; and

[10] Some designers, for instance [Archea 86] take a very different point of view.

frequently require reasoning about geometry and spatial relations which demands considerable computational resources and resists approximative or qualitative reasoning methods.[11]

The main similarities between planning and design tasks are the following: large search spaces which must be factored into subproblems to deal with the complexity, and the notion of interactions between the subproblems(but not necessarily the same types of interactions). Certain important differences may exist also, depending on the domain of design involved. These differences revolve mainly around the expected types of interactions and issues of uncertainty involved in design problems as opposed to planning problems.

3.2.1 Uncertainty in planning and design

Planning for real-time action in a dynamically changing environment with temporal and resource constraints generally has very different characteristics than designing an artifact. "The operation of planning a sequence of actions can be viewed as a design process, in which the result has more dynamic variation than is ordinarily considered in design." [Rychener 83].

Uncertainty is obviously a large and crucial issue in problem solving. In general terms, uncertainty in the task environment can be defined as the difference between the information available and the information necessary to make the best decision [Bond 88]. Fox [Fox 81] discusses several types of uncertainty **including *information uncertainty{\xnctminty* of the correctness of data), *decision algorithm uncertainty(\xncenȧxnty* due to lack of knowledge of outcomes of decision), *environmental uncertainty* (a** changing environment places prediction outside the control of the program making modeling difficult if not impossible), and *behavioral uncertainty* (individual agents may not deliver on their commitments). For many design tasks, much uncertainty can be excluded or severely restricted by making assumptions[12]such that it does not play a large role if the appropriate representation and approach are employed. Such design and configuration tasks (e.g., space planning or layout tasks) are able to be ***statically modeled.***

However, a large source for uncertainty in most design processes is the *problem formulation* itself, including the goals and constraints to be applied. The need for problem re-structuring is usually indicated by lack of convergence due to insufficient constraints, or a failure to meet all the constraints currently specified. Design systems sometimes fail to deal effectively with a set of specified constraints because they the do not properly take into account the types of interactions involved (local vs. global constraints) and because they cannot 'trade-off constraints (or goals) but simply fail if not all constraints can be met.

The interaction of global and local concerns implies an *uncertainty of outcome* until the design is complete. Such interactions have an important impact on the architecture of the design system in terms of when decisions are made (focus of control), with what degree of commitment (amount of certainty), and whether the process can proceed monotonically or must provide for backtracking. Much of design does not have to deal with noisy or uncertain data, so a key uncertainty to deal with is uncertainty of outcome due to number and types of constraints.

---

[u]**But see [Woodbury 87] for a beginning in this area,**

[12]**These assumptions are similar to those made by the strategic planners and are very reasonable in many design domains.**

3.2.2 Goals and interactions in design

The design of artifacts involves the composition of parts or objects, as opposed to planning which may be viewed as the composing (or designing) of operations or processes. The composing of either actions or objects involves the problem of functional interactions. But in other ways the interactions between actions (or the operators that represent them) may tend to have different characteristics than those between objects. Aside from sharing functional interactions, objects have the property of taking up space, which is a global resource. For instance, in a layout problem, each of the objects to be placed implicitly represents a subgoal, with the overall goal being to place all the objects. The objects (goals in terms of placement) can be partially viewed as relatively *independent* (in Korf's terms) and only interacting when the limits of the resource they have to share (space) is reached. With this type of interaction it is impossible to guarantee satisfaction and protection of any intermediate performance goal because an unanticipated interaction (such as lack of space) may show up later and undo it. Furthermore, if this does occur, then it is not at all clear how to correct the situation, since the problem of dimensional fit is the collective result of potentially all objects entered, not just the last or most recent.

The above problem exists not only for the global resource - space - but for other types of shared resources and non-local performance requirements. It is not clear that the overall goal-directed approach and methodology of strategic planning is suitable for dealing with these global resource issues in the most effective way. The later generation of more sophisticated planners (non-linear, least-commitment)were based heavily on detection and avoidance of interactions between goals. In design this is possible to some degree, but in large measure the very essence of design is not in anticipating and avoiding all interactions, since this is virtually impossible with the mapping of design to performance variables being many to many. But rather, design is very often concerned with the exploration the trade-offs among design goals and the finding of an acceptable level of interaction. In fact, the capability for exploration of trade-offs may be viewed in large part as re-formulation of the problem as the design (or plan) proceeds:

> [Cheeseman 88]In particular, the AI use of goals needs to be extended to utilities, so that trade-offs between competing goals can be arrived at. This approach allows planning to produce useful plans, even when the given list of goals cannot be satisfied.

Just as in planning, the crude division of strategies to deal with interactions have been the following:

- Wait for violation, then backtrack or revise the design accordingly. (HACKER, DAA)
- Propagate constraints to warn of interactions and avoid interactions wherever possible. (MOLGEN, VEXED)

Neither of these two strategies is efficient in handling the types of interactions resulting from uncertainty of outcome, and the complex interactions and trade-offs that occur in many types of design problems.

More recently the idea of the systematic exploration of *all* solution alternatives has been raised again[13] as a valid strategy in design domains; this is particularly true of configuration and placement problems involving spatial representation and reasoning. If the right representation and architecture can be found to make such an approach effective, then it has the dual advantages of providing a systematic platform on which to explore trade-offs between alternatives, and guaranteeing that the full *set of valid solutions* to the problem are found, not just a single satisficing solution whose global 'goodness' is unknown. The next section takes a brief look at how the strategies of planning have been used in design and how they are being extended to better deal with the types of uncertainties and interactions typical in design.

---

[13]Simon and Newell's original and potentially exhaustive general search strategy, or G&T, earned a bad reputation and was dismissed for most realistic problems because of the combinatorics involved

3.2.3 The strategies of design

Space **does not permit** anything like a general review of the issues involved in automated design problem-solving. **What follows** is cursory listing of some research that has incorporated or extended variations of the **planning strategies** in an explicit way.

Design as Knowledge-Based Planning

Coyne and Gero propose design processes modeled directly after various paradigms of strategic planning, demonstrating planning as a method of implementing control over the design process [Coyne 85a, Coyne 85b, Coyne 86, Gero 85a, Gero 85b]. They contrast design, modeled as planning, with forward search (generate and test or heuristic search with backtracking) which are viewed as far too inefficient due to the number of components and the resultant combinatorics typically involved.

The model combines the notions of a design grammar and a hierarchical/opportunistic planning approach to control the satisfaction of goals by direct application of the rules of the grammar. The process can have an arbitrary number of hierarchical levels, but they discuss an example - in the area of automated layout - where a language of forms is applied by a language of actions which is controlled by a language of plans. Their hope, in this particular model, is that the dynamic characteristics of design processes can be accommodated by metalanguages and the grammars of such languages can be made explicit through the articulated knowledge of designers.

Incorporating the strategies of planning into design

Tong[Tong 87] presents a framework for organizing, evaluating, and developing knowledge-based models of the design process. He develops the beginning of a taxonomy of design problem and domain characteristics, and suggested appropriate strategies and system architectures to deal with those characteristics. He describes an important sequence of alternative design processes of increasing elaboration and sophistication.[14] He emphasizes that in using these strategies, in the engineering of a knowledge-based design system, it is very important to fully exploit domain knowledge. That is, domain-independent design process models cannot be made operationally effective without using domain specific features.

Tong additionally describes a method for 'goal directed planning of the design process* through extending the least-commitment phase of the process. This extension to least-commitment - called 'bottleneck-recognition' - leads to more informed choices(branching) by turning an *assumption* (decomposition or refinement choice) into a *deduction.* This is accomplished by forming abstracted operators which produce just those effects that are common to all individual operators at a choice point.

Design plans, or skeletal plans to guide design

Automated design system researchers in some domains have found that a useful way of structuring design knowledge is in **the** form of *design plans.* These plans integrate the types of knowledge needed during the design process into knowledge structures. Examples of systems built this way are AIR-CYL [Brown 86], and PRIDE [Mittal 86c, Mittal 86b] which refined and extended the approach. The knowledge-base of the design system is structured as a *generalized design plan(dn* abstract or skeletal plan) and the elements of the plan are: steps to be carried out, information about ordering the steps, how to perform each step, etc. Such design plans are meant to define a search space as well as, to provide ways of

[14]Tong uses a different terminology, but it is clear that these processes are adapted from and exactly parallel the developmental paradigms of planning in terms of what motivates them and how they are useful strategies for search reduction.

efficiently searching in that space.

The different elements of a plan are structured around *goals.* The goals represent the top-down structure of the design plan and decompose the process of design into simple steps. Each goal is responsible for designing a small set of design parameters that describe some part or aspect of the artifact. Subgoals are represented the same way, allowing a recursively nested plan. All alternative ways for making a decision about a design parameter are attached to the same goal. Attached to goals are the following knowledge-structures: *Methods* (encoding decision making knowledge); *Constraints:* (encoding performance requirements); *Advice:* (for backtracking and modification).

In the PRIDE system, plans were *hand-coded* from an analysis of design processes canied out by expert engineers. Therefore, these plans, as in the hierarchical decomposition of goals in the planning systems, represent *precompiled knowledge for problem structuring,* or the definition of the problem so that it can be solved in the next stage by some suitable problem solver. Because of their overriding importance and the difficulty of hand-building design plans for a variety of design problems, Araya and Mittal [Araya 87] are attempting the hard task of developing a method for automatically compiling design plans from descriptions of artifacts and problem-solving heuristics. They start with the skeleton of an efficient design plan and it is mapped in stages, by tailoring it to the requirements of the specific design problem, into a full-fledged plan interpretable by the problem solver.

The elaboration of constraints and extension of monotonic commitment in design

In dealing with large design search spaces it is important to reduce both the costs of *finding* and *evaluating* candidate solutions (which is another way of stating the general control problem). Constraint-directed or knowledge-guided search strategies propose to mitigate this cost by a number of measures for ordering generation and evaluation. The characteristics of design search spaces which create such order for decisions are not well understood [Mostow 85]. As cited above, both Fox's and MittaTs work are attempts to understand and utilize those characteristics in terms of *properties of constraints,* in the belief that the way constraints are used is crucial in determining how efficiently the design process operates.

The work of Fox [Fox 83, Fox 86] attempts to greatly extend the use of constraints to aid the search process by elaborating their semantics, and Baykan [Baykan 87] is testing the merit of that approach in the design domain of automated layout.

Mittal, et al, [Mittal 86a, Mittal 87] have also done woric on elaborating constraint reasoning to extend least-commitment in order to enable the analyzing of partial designs as early as possible, instead of waiting for the complete design. They describe two closely related strategies called *partial commitment* and *constraint compaction,* both of which extend the problem-solvers' ability to make additional decisions with certainty and less cost of processing. Both approaches enable the explicit maintainance of an inclusive, approximative description of the candidate solution set. In this way they both involve *monotonic inference,* that is, decisions made via these strategies never have to be retracted. Additionally, both approaches carry forward all solutions to the constraint-satisfaction problem; thus the search is *complete* in the restricted sense that the strategy can be used to generate all solutions for the initial set of requirements.

The Blackboard model as suitable for design problem-solving.

Nii [Nii 86] among others, has suggested that the Blackboard model might be suitable for design problem-solving, but also warns that it is complex, costly and may not always be appropriate (though even then, she suggests that it may be good for an initial problem formulation phase - until the problem-

solving process is well enough understood to be more simply encoded.)

An information processing analysis of design

Chandresekaran [Chandrasekaran 88] is developing a conceptual structuring of automated design-systems through an analysis of the classification of the processes of design. He states that design consists of traversing a search space through successive generation of alternatives and discrimination and choice among these alternatives that results in a goal state. He identifies design subprocesses with well-defined information processing responsibilities and suggests a functional architecture for design with these subprocesses as building blocks. The analysis divides design into two groups of processes: those that are responsible for proposing or making design commitments of some sort, and those that serve an "auxiliary" role, that is, they generate information needed for the proposers, and help test the proposed design. He asserts that many of the processes in the "generate" portion are quite specific to design as a problem solving process, and these are: *decomposition, design plan instantiation and expansion, modification of similar designs, constraint satisfaction.* Each of these comes with its own set of control problems that can be more or less complex, and needs knowledge in certain forms.

The auxiliary processes deal with testing and information production in the service of the main processes. These processes, which may be arbitrarily complex, are not particularly specific to design:

- *subproblem constraint generation:* Information about how the goals/constraints for a problem D are translated into goals/constraints for the subproblems D1, D2, .. Dn; usually part of decomposition knowledge.
- *Recomposition:* how to glue the the solutions of the subproblems back into a solution for the original problem.
- *Design verification:* the "test" component of the design activity.
- *Design Criticism:* basically, advice for modification on failure.

Returning to the "generation" processes, I have paraphrased some of Chandresekaran's observations:

- *Decomposition:* It is a ubiquitous strategy in AI work in design. Alternate decompositions may be available so an additional search in a space of decompositions may be involved. In well known domains, effective decompositions are known and little search at that level needs to be conducted. Repeated application of the decomposition knowledge produces *design hierarchies.* Dependable decomposition knowledge is extremely effective in controlling search by reducing the total search space. Decompositions represent a precompiled solution to part of the design problem. In dealing with decomposition two sets of inference processes are necessary, one dealing with which sets of decompositions to choose, and the other concerned with the order in which the subproblems within a given subproblem are attacked. The potential combined cost of these processes can be very expensive so it is recommended to avoid the search problem by using domain knowledge about decomposition - possibly through human machine interaction. The default control process for investigating within a given design hierarchy is top-down, but the actual sequence in which design problems are solved may occur in any combination of top-down and bottom-up manner.
- *Design plans:*[15] They are another pervasive form of design knowledge, that represents precompiled partial design solutions. A design plan specifies a sequence of design actions to take for producing a piece of design. The notion *that plans* constitute a very basic knowledge structure has been with us since before 1960 [Miller 60]; see Appendix I. Since plans may have steps that point to other plans, *design plans can subsume decomposition knowledge*

[15]These are as I described above, but I include a few more key observations from Chandresekaran.

Design plans contribute to complexity reduction by abstracting from experience the previous successful exploration of a problem space.

- ***Modifying almost correct designs:*** **The same as discussed under *learning* in the planning** section(Sussman's HACKER, etc.). this process is based on the retrieval from a storehouse of actual successful designs indexed by the goals and constraints they were designed to satisfy.
- *Design by constraint processes:* A process of simultaneous constraint satisfaction by constraint propagation, it is usually only computationally effective if the structure of the artifact is known and design consists of selecting parameters for the components. It can be viewed as a form of problem space exploration through incremental convergence. More complex processes such as constraint posting can be used to generate additional constraints as a result of earlier parameter choices. These constraints are used for remaining parameter choices. Chandresekaran cautions that although all design can be formally thought of as constraint satisfaction, global constraint satisfaction should not be considered as a universal solution for design, because these methods still can constitute a fairly expensive way to search the space. He advises that other methods of complexity reduction, such as decomposition, are still very important in general.

This lengthy sketch of Chandresekaran's classification of design processes is motivated by the fact that it is most salient and clear in regard to the leveraging of search in design. In this regard, it is easy to see the relation of his process categories to the strategies of planning I presented. A final quote confirms my **conclusion in the previous section, that *decomposition is the primary planning strategy, at the core of the most powerful leveraging of search* [Chandrasekaran 88 , p 9]:**

> The framework suggests that design by decomposition, i.e., breaking problems into subproblems, by plan synthesis where necessary, and by plan selection where possible, are the core processes in knowledge-based design, i.e., it gives importance to the first two processes in the above list as the major engines of complexity reduction in design.

3.2.4 Integrating insights about the strategies

The discussion of insights from the various researchers who have employed and extended strategies of planning into design systems leads to some general concepts that can be abstracted away from individual systems and domains. The most important overall strategy is the creation of multiple levels of abstraction for partitioning the search space. This is accomplished by means of properties of the representation and/or knowledge of the search space, providing for *early pruning* of entire branches via partial choices and partial evaluations with the greatest degree of certainty and commitment possible. The common thread in strategies allowing partial choices is use of description of classes of solutions as opposed to actual solutions. Such descriptions form a type of 'intermediate abstraction* which can help reduce search as well as enable finding multiple solutions. Especially important in achieving the above are strategies which aim at the separation of decisions about constraints (detailing, setting constraint values), from decisions which create constraints, or branching choices (ie: which generation or decomposition operators to apply).

It is also important to recognize the significance and difficulty of problem structuring. It is possible to extend the problem-structuring phase through the problem-solving phase via dynamic re-structuring, or having an environment where new knowledge is stimulated or evoked via the problem-solving cycle and easily added. The latter approach forms a tight loop for rounds of incremental formulation and solving, and though simpler than dynamic restructuring, it is effective. It has been observed by many designers of design systems that the knowledge-based systems approach can be very helpful in this regard: [Coyne 86,p 88]

> The intention is to create a flexible environment in which contextual information, knowledge, and system processes are visible and amenable to modification and development

For many more complex design problems it appears desirable to have the capability of producing multiple candidate solutions, or in some cases all candidate solutions so as to produce the globally best or optimal solution,[16] This is particularly true for classes of problems involving an uncertainty of outcome. Goal-directed or constraint-directed search by itself does not have that capability, and therefore is inherently not a *complete* search. It aims at finding a satisfying solution, or a certain set of solutions of 'reasonable quality', in as direct a manner as possible.

An overall planning approach can be made complete by augmenting it with sophisticated (backtracking) mechanisms to keep track of the state of the search and all choice points. However, such mechanisms usually add elaborate and potentially costly components and processes, which should only be employed if justified by the problem characteristics, such as, the types of uncertainty and interactions involved in the design problem. There are alternative architectures, e.g., variations of generate-and-test (G&T), which can be complete, depending how the generators are constructed. G&T is not itself a planning based architecture, however, it can be combined with many of the strategies of planning, particularly: levels of abstraction, constraint reasoning, control reasoning.

The best way to encourage the convergence of search is by carefully choosing and tuning system characteristics to create problem-solving processes most appropriate to individual problem and domain characteristics, thereby ensuring both epistemological and heuristic adequacy; that is, managing computational resources for *effective* and *efficient* search.

3.2.5 The design of design systems

> [Tong 87, footnote#28]. Using a design framework to characterize and evaluate design frameworks may seem 'meta-circular' — and it is! But it looks as though characterizing design problem-solvers is actually leading to convergence, rather than spiraling away into the ether.

The activity of building knowledge based systems has been repeatedly characterized as an *art* by notable practitioners in the field[17] At the same time emergent principles from existing systems reflect current understanding of the best way to design structures that support intelligent problem-solving. A small set of basic concepts for leveraging problem-solving, combined with the complexity of the application task and the type of knowledge available determine the most appropriate overall organization or *architecture.* In this context the term *architecture* refers to the emerging science and method of design that determine the structure of the system. Though system design and building may be moving from an art to a science (or a least an engineering discipline) system builders continue to operate against a background of competing ideas and controversies and continue to employ a large portion of creativity and heuristics gathered from experience to guide the construction of a knowledge-based systems.

---

[16]For instance, see [Frayman 87] regarding configuration tasks, where he states that it is desirable for the system to be able to provide multiple configurations because the evaluation criteria often work against each other, and the combination of different evaluation criteria is usually highly subjective.

[17]In a 1986 paper Mark Fox stated that the design of search architectures is an art and that AI engineers approach a problem with a set of techniques, which by example, have been shown to be useful in other tasks. The essence of the "art" is acquiring techniques through experience says Fox: "In fact, the training of AI engineers is similar to the medieval guild system where apprentices work with masters for some period of time" [Fox 86].

It is not surprising then, that in the last several years there has been a great deal of effort to identify the key issues in constructing design systems, particularly knowledge-based design systems (KBDS) and to clarify the relationship between AI and design. A number of researchers and groups have put forth overviews dealing with models of design, representation and reasoning issues, use and reuse of knowledge in design, learning, interactions in design goals, etc. And based on some of these issues, tentative taxonomies of design domains, or characterizations of design processes from routine to creative, and classification of system architectures and their utility have been proposed; see [Mitchell 85, Mostow 85, Tong 87, Brown 85, Mittal 86b, Chandrasekaran 88] among many others. Additionally the area has been deemed important enough for NSF to establish a center, the "Engineering Design Research Center" at CMU, and for the American Association of Artificial Intelligence to offer a tutorial course, "Artificial Intelligence and Design" at its annual convention for the past two years, AAAI87 and AAAI88.

The effort to "abstract" the useful concepts for automated design systems from the extant "successful" AI systems is worthwhile in the same way that we as learning creatures are always refining and augmenting our knowledge. But research into automated design, at its current state of development, must reflect the fundamental experimental and empirical nature of this woric. The following statements from Newell, express this point of view: [Newell 83] "AI as a whole is founded on some striking methodological innovations, namely, using programs, program designs, and programming languages as experimental vehicles. ... As we strive to implement complex processes, such as design in various domains, we are evolving our understanding of the theory and mechanisms of these processes. ... Each researcher who achieves a more "intelligent" program is contributing to an empirical foundation for a true science of design."

3.2.6 Summary

- Many design problems can be formulated as search problems. The challenge is then to find solutions efficiently through careful management of the space and time resources of the search process. Planning strategies apparently have much to contribute to the efficiency of design modeled as search.
- Of utmost importance, is again, the creation of multiple levels of abstraction for partitioning the search space. Decomposition is a fundamental and especially important strategy in reducing the complexity of design problems. Design plans can provide for decomposition knowledge, and precompiled solutions to parts of design problems.
- Constraint manipulation is an important computational technique for portions of the design problem, but it can be severely expensive to use as the main solving process.
- Efforts are under way to conceptually structure the enterprise of designing knowledge-based design systems. Currently however, building automated design-system architectures remains largely a combination of artfulness and empirical tests.

3.3 Abstraction in problem-solving

The use of abstraction thoroughly penneates human experience not only in the processes of solving problems but in our very perception and structuring of our knowledge of the universe. In the scientific disciplines, a hierarchy of entity descriptions is an integral part of the scientific view of the universe [Jacob 77]. Also, a common way to represent physical objects is within a taxonomic hierarchy which allows us to make assertions about a class of objects that we need not repeat for all of its subclasses [Tenenberg 86]. The instantiation of a very common hierarchical abstraction mechanism is employed regularly, in science, as well as everyday situations, to solve representation problems [Darden

87]: "An example of a common abstraction is a tree structure; instantiations include the Linnaean hierarchy in biological taxonomy and the organizational chart of a corporatioa" One of the best discussions of the importance of abstractions and hierarchy in dealing with complexity, whether in natural or artificial (man-made) systems, is contained in Herb Simon's "The Sciences of the Artificial", particularly the chapters on the "The science of design" and "The Architecture of complexity" [Simon 81].

3.3.1 General discussion

Though abstraction in human problem-solving and automated problem- solving is a common and apparently vital mechanism, its use has not been particularly well analyzed. In AI the use of abstraction up to now has been as a "fuzzy", heuristic type of concept in a productive, creative phase allowing for the exploration of its potential areas of application. Its power in leveraging problem-solving has been noted by many researchers, and is virtually assumed as an invaluable tool, but it has largely been documented only in a qualitative anecdotal manner. Korf points out that abstraction, as well as the other strategies of planning, remain at the stage of qualitative wisdom, such as: "problem-solving using a hierarchy of abstraction spaces greatly improves search efficiency" [Korf 87,p 65]. The situation is beginning to change through the efforts of researchers like Korf who presents an analysis of the use of abstraction in problem-solving which he characterizes as the first steps toward a quantitative theory of planning. He summarizes the value of abstraction, which he considers to be one of the three important "knowledge sources" or strategies of planning (the other two, mentioned in the section on planning, were subgoaling and macro-operators) [Korf 87]

> The value of abstraction is well known in artificial intelligence. The basic idea is that in order to efficiently solve a complex problem, a problem solver should at first ignore low-level details and concentrate on the essential features of the problem, filling in the details later. The idea readily generalizes to multiple hierarchical levels of abstraction, each focused on a different level of detail. Empirically, the technique has proven to be very effective in reducing the complexity of large problems.

Darden cites *abstraction* and an intimately related concept *instantiation* as two of the most powerful ideas developed by AI researchers in a computationally useful way. He also provides an analysis of the concept of abstraction in order to clarify its meaning and distinguish it from other similar concepts such as *generalization* and *simplification.* Much of the discussion of the concept of abstraction as generalization which follows is based on the analysis in [Darden 87]. Confirming the importance of the generalization aspect of abstraction but adding to it another basic type or category of abstraction - abstraction as *aggregation* - are the conclusions of many others, for instance [Eastman 81] and [SMITH77].

Abstraction is sometimes viewed as a process (abstracting) or as a product of that process (an abstraction). In either case, the fundamental issue is the *loss of content.* Abstracting from a given source involves dropping information, making the abstraction formed simpler in some way. Thus, abstraction is one way of *simplifying.* Because of this loss of information an abstraction may also apply to a larger class of objects of which its original source was a member, and in this sense abstraction is like *generalization.* This follows closely the set theoretic sense of abstraction where given any property there is a set whose members are just those objects having that property. The use of abstraction in this sense in AI, involving the omitting of individual differences to get the class concept, shows why abstraction and generalization are sometimes used synonymously. Like many of the other ideas in AI, the use of abstraction can be traced back to Polya's work, in particular the way in which he defined generalization [Polya 57]. But not all forms of generalization involve loss of content or information, for instance, universal quantification or inductive generalization.

The type of generalization involving an abstracting step is very familiar in AI. Early theorem proving and planning or problem-solving systems formed *abstractions* by dropping conditions(see the ABSTRIPS system in [Sacerdoti 74]), or replacing constants with variables(see the discussion of GPS in [COHEN82, "p 518"]), or the building of Macrops tables as an extension to the STRIPS planner (in [FIKES72]). Later this generalization type of abstraction was expanded to provide the basis for the familiar *schema, frame,* or *object-oriented* representation systems allowing for the flexible and creative invention of new semantic concepts through instantiation of an "is-a" or "type-of" hierarchy. These types of knowledge representation frameworks allow for multiple hierarchical levels of abstraction, facilitating *inheritance* or *specialization* of attributes; basically in moving to each next higher level of abstraction the system substitutes the class concept for the individual *by dropping content.* Thus, through generalization, abstractions may be formed from instances by at least the following methods: dropping conditions, replacing constants with variables, creating concepts at a higher level of abstraction that capture the meaning (in a certain limited sense) of lower level concepts. No doubt there are other ways of forming abstractions through generalization. Abstractions must be carefully "designed" in the sense that what guides the dropping of detail involves the purpose for which the abstractions are being formed and on the problem-solving context in which they will be used. Along these lines Danden describes another type of abstraction he envisions being useful in recording "compiled hindsight" in applications from the history of science: "... a schema that has resulted from dropping detail and preserving underlying structural or functional relations among component parts" [Darden 87,p 38].[18]

Many AI systems have also experimented with the opposite of forming abstractions from instances. They have explored the problem-solving leverage of retrieving available abstractions and *instantiating* them in in a given context of problem-solving. Systems which have used skeletal plans, or scripts, or prototypes are all in a sense employing abstractions based on former problem-solving experience to help solve a similar or analogous problem. Such abstractions represent compiled problem-solving experience in one fashion or another and it is hoped that they can be stored, retrieved and instantiated in an efficient enough manner so as to reduce the overall burden of the problem solver. Where such useful abstractions come from in the first place, that is how they are created, must involve even more complicated mechanisms in a problem solver and amounts to *learning.* Informally, the essence of learning is finding *abstracted* patterns and processes that through analogical reasoning and problem-solving can be implemented in other cases when appropriate problem types arise.[19]

Besides generalization the other major type of abstraction is *aggregation.* This type of abstraction is commonly seen as a hierarchical decomposition of an object or an operator into its component parts. As mentioned in the discussion of planning above this type of abstraction by decomposition is also known as *reduction* or *sub-tasking.* It has been utilized in many planning and problem solving systems where higher level goals or operators are precompiled aggregations of lower level subgoals and operators. The hierarchical planners, such as Noah [Sacerdoti 77] used this type of abstraction where higher level operators are *refined* during problem-solving into their component parts. Based on this concept many of the automated engineering design systems in the last few years have been based on *hierarchical refinement,* which takes advantage wherever possible of the basic "divide and conquer" approach of top-down design [Tong 87]. If the component parts are only weakly interacting then the approach of

[18]As we will sec later, this type of abstraction also plays an important part in the representation used in the LOOS system for space planning.

[19]The relationship of abstractions and learning to space planning systems such as LOOS will be touched on again later.

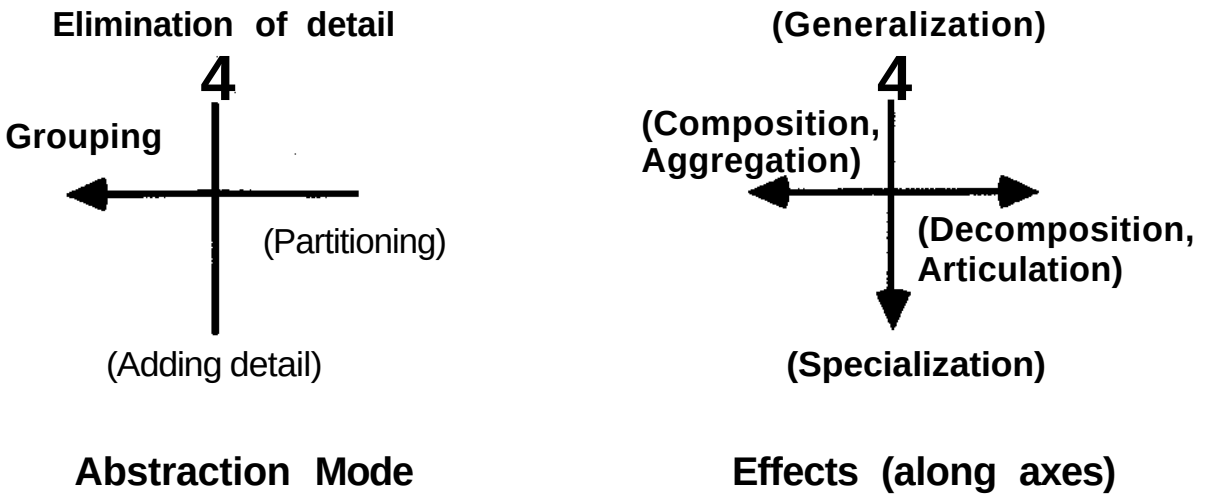


Figure 3-3: Crude division along orthogonal axes of the major abstraction modes.

decomposition through progressive levels allows for the interactions to be recognized and handled effectively. The dropping of information that occurs in this type of abstraction allows one to handle an aggregation or assemblage as a single unit and ignore, for the moment, the *unimportant* details of the component parts. This type of abstraction relationship is captured in some AI frame-based or schema-based representation systems as a "Part-of' relationship. It cannot be modeled directly in an object-oriented system where the primary definitional relationship is a "type-of' or "is-a" relationship, but it can be simulated by placing objects in slots of other objects. Eastman has stated that aggregation abstractions are the dominant ones in design [Eastman 81], a subject which I elaborate on in the section below on abstraction in spaceplanning.

So far I have shown that two of the major categories for classifying abstractions are those of generalization and aggregation. Confirming these two important categories in abstraction is a more recent researcher in planning, Tenenberg, who has proposed a general technique of abstracting the operators in a planner. He describes a generalization abstraction as an *inheritance abstraction* and an aggregation abstraction as a *decompositional* abstraction [Tenenberg 86]. Figure 3-3 shows these two methods of abstraction as basically orthogonal to one another, but both have been employed in various ways and sometimes in combination in problem solving systems.

The fact that these two types of abstraction can sometimes interact in the formation of object hierarchies is indicated by the following: [Mackworth 85]

> An important technique for handling large domains is to exploit their internal structure. Indeed, for many real world problems the domain elements often cluster into sets with common properties and relations. Those sets, in turn, group to form higher level sets. This clustering or categorization into "natural kinds" can be represented as a specialization/generalization (is-a) hierarchy(Havens and Mackworth 1983) [Havens 83].

For example, in a space planning problem, through generalization-abstraction, some objects might be grouped into a set such as all the offices that require placement next to windows. This set might then be viewed as a physical cluster which is abstracted via an aggregation abstraction into a single higher level unit for rough placement in a layout. The "natural kinds" mentioned above suggests that, depending on the domain, useful groupings of objects are often determined based on domain knowledge. A sophisticated method for doing this is called *conceptual clustering*, which may be described as arranging objects into classes corresponding to specific descriptive concepts instead of classes of objects that are

similar according only to a mathematical measure [Michalski 83].

Both of the main forms of abstraction discussed and necessarily *all* forms must involve simplification, or the suppression of information in order to be useful. Currently other forms of abstractions are being classified and their management and utility studied; for instance, in planning and design, see [Korf 87, Eastman 85, Galle 86, Woodbury 87] among numerous others. Korfs analysis of single and multiple levels of abstraction in problem spaces suggests many important facts about the design of abstraction levels:

- The optimum abstraction hierarchy for a problem space with *n* states consists of Inn levels of abstraction and that the ratio of the sizes of successive levels of abstraction is e. Use of such an abstraction hierarchy can reduce the expected search time form *0(n)\oO(\ogn)*. This improvement makes combinatorial problems tractable, for instance, if *n* is an exponential function of problem size, then logn is linear. Thus, the optimal hierarchy has a logarithmic number of levels with a constant ratio between their sizes and this hierarchy reduces search from linear in the number of states in the problem space to logarithmic.
- In practice, there are a number of different factors that constrain the design of abstraction spaces, so an optimal abstraction space in the above sense will not be achievable in general. The above analysis shows how much abstraction can reduce search complexity in the ideal or best case. However, the result does suggest that in general a deep abstraction hierarchy, in terms of number of levels, will reduce search more than a shallow one.

As I noted above in the hierarchical planners, the decomposition (abstraction) levels were typically tailored to the requirements of a specific problem. Researchers in automated design have suggested that there exist "natural taxonomies" of objects, or processes in most design domains, discovered by humans problem solvers over a long period of time, which appear to be the most effective decompositional hierarchies for efficient problem solving [Mittal 86c, Baykan 87, Chandrasekaran 88]. In fact, these hierarchic levels represent some of the most important domain knowledge, or expertise, with which to "plan" the efficient solving of problems within the respective domain. However, these levels are necessarily based on the semantics and requirements of the domain, and may not approach an *ideal* partitioning in the terms of Korfs quantitative analysis. But, in general they probably reflect the more qualitative wisdom of the second point above - where possible, a deep abstraction hierarchy is usually preferable.

In short, there remains great theoretical difficulty in assessing a quantitative prediction of the utility of abstractions in domains such as design, where every problem is unique since each must be interpreted, structured and solved in their own context. Therefore, I concur with another point made by Tenenberg, that the ultimate test of the utility of a set of abstractions must be *empirical* in that they must be cost-effective (in terms of some resource measure), but *only as compared with other problem solvers (human or machine) for a given domain.* [Tenenberg 86]. Currently, it seems that a great deal more experimentation with the careful use of abstractions within design systems of various architectures and in various problem domains must be undertaken before any generalizations or strong statements may be made regarding the ideal form of abstractions for any particular design problems. In any case, the high utility of abstractions suggests that they are essential to all but the most trivial types of automated design problems, and the choice of representation and set of abstractions used in problem-solving will largely determine the system architecture.

3.3.2 Abstraction in Design systems

An obvious but important point is that the very act of design is perhaps man's most fundamental act of abstraction in as **much as** he deals with a *model* of the woiid in order to achieve effects in the world. However, this first, **most** profound step of abstraction is not nearly enough in most design problems. The next important **observation is that** even with a **model** it is impossible to deal with all the issues and information important to the problem, at the same level of resolution, that is, in a single "flat" level of abstraction. Human designers, as problem solvers with well known fundamental limitations of memory, have found myriad ways to abstract the issues, processes and objects of design, and in doing so they are building hierarchies of many kinds - [Simon 81].

Hierarchies aid problem solvers in at least three distinct ways: first, and foremost, they reduce the information load for decision making at any given moment; second, they allow for the efficient introduction or recognition of partial results that are known to be on the way to the goal[20];third, if the domain permits and hierarchies are structured carefully[21] they allow us to bound and reason about the interactions of subparts so that they can be handled both effectively and efficiently. The following quote from Simon elaborates these points, especially the third:

> The dynamic properties of nearly decomposable systems are such that interaction among subsystems is generally much weaker than the interactions within subsystems. Thus comparatively little information is lost by representing them as hierarchies. "Subparts belonging to different parts only interact in an aggregative fashion -the detail of their interaction can be ignored." If we attempt to observe or comprehend complex systems in the absence of hierarchic organization then analysis of their behavior would involve such detailed knowledge and calculation of the interactions of their elementary parts that it would be beyond our capacities of memory or computation [Simon 81, p 219].

In the literature of automated problem solving, one of the earliest examples of a hierarchical design/planning process precedes the research on automated planning by several years [Manheim 64]. The example deals with a phase of 'highway design' called the 'highway location process', and Manheim structured the process hierarchically so that the costs of designing would be considered in guiding the design process. The process is an example of strict top-down refinement through fixed levels of abstraction with the levels being:

- bands of interest (within which to find a good route)
- locations (one or more are selected for closer examination)
- specific designs (for particular locations)

Figure 3-4 from Manheim's thesis (cited above) shows in parts a) and b), the hierarchical spatial nesting of a location within a location band within a band of interest, and in part c) the resultant 'tree-like* hierarchical structuring of the search space. Simon [Simon 81] refers to Manheim's work as one of the earliest clear examples of the use of hierarchical abstraction levels to guide search. He points out two main ideas of this "hierarchically structured sequential decision process":

1. the specification of a design progressively from higher levels of abstraction (general plans) down to the details of actual construction.
2. attaching values to plans at higher levels as a basis for deciding which plans to pursue to

---

^Recall that Minsky [Minsky 63] called these "planning islands"; see Appendix I for more details. Simon [Simon 81] has referred to them as "stable subassemblies" that represent recognizable progress toward the goal.

[21] In Korfs terminology the subgoals cannot be strictly non-serializable(that is, not even transformable into block-serializable subgoals through discovery of appropriate macro-operators). In Simon's terminology the goals must be nearly decomposable.

levels of greater specificity. (Manheim's three levels may be generalized in other contexts.)

What is most significant[22] is that this structured decision process is guided by a cleaving of the search space using an intuitive abstraction of spatial location, and allows for effective pruning of alternatives at a high level by evaluating which alternatives to pursue from one level to the next. Highway construction costs are calculated as a prediction of what the cost of an alternative would be -- if a plan for it were specified to completion through lower level design activities. Of these alternative plans those are pursued to completion that look most promising after design costs have been offset against them. Simon notes that this evaluation of higher level plans provides both a "steering mechanism for the search and a satisficing criterion for terminating the search."

Noting that the highway *location* phase of highway design may be generally analogous to the location of components, or spatial layout, involved in the specification of artifacts, I include the following selected quotes from Manheim's paper, suggesting the importance of a process based on hierarchical abstraction by physical aggregation: [Manheim 64]

> When we take locations as our basic elements, we find that every other action - band of interest, for example - can be visualized as a collection of locations. ...Even though we cannot in fact count total numbers of locations, we use this idea to construct an abstract representation of the location process in terms of sets. This representation is based upon two observations: first, that every action of higher level than a location can be interpreted as a set of locations; second, that the same location belongs to a band of interest as well as a location band, for example. ...First, we show an element to represent the set of all possible locations which might be solutions to the particular problem; this would be obtained by not specifying any characteristic of locations, yielding an action of the very highest level. At the next lower level, we show the two bands of interest, then the location bands at lower level, and finally the selected location itself, at the lowest level. ... Because of the multi-level nature of the route location problem,[footnote](Of course, the location problem is only one of a large class of problems with this same characteristic.) we must distinguish this as a "hierarchically-structured" sequential decision process. ...The basic idea of level is the difference between the preliminary, or planning, stage and the final design stage of an engineering process.

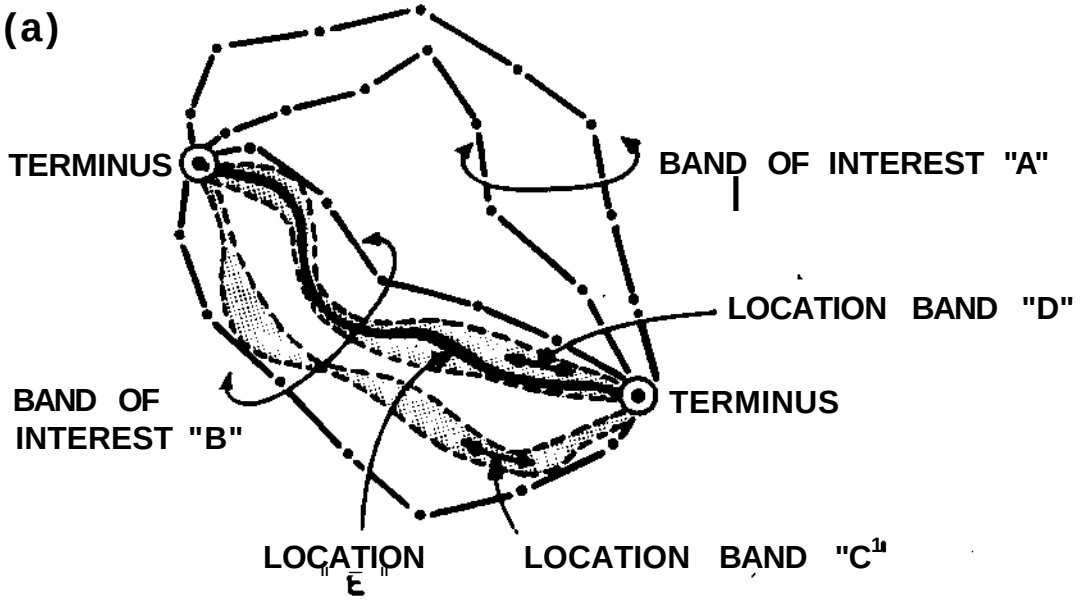
The main point in the above is that hierarchical structure is one of the fundamental requirements for the comprehension of complex systems. Furthermore, in many domains of design, physical decomposition, as well as functional decomposition, may play a vital role in this structuring. It is observed that in hierarchies only the aggregative properties of parts enter into the description of the interactions of those parts. The trick is to translate and propagate the most important properties of the components to the surface or boundary of the sub-assembly so that one can reason about the sub-assembly as a whole unit. If one can do this one can go on to construct a description of the whole system from a restricted alphabet of elementary terms corresponding to the basic set of elementary subsystems from which the complex system is generated.

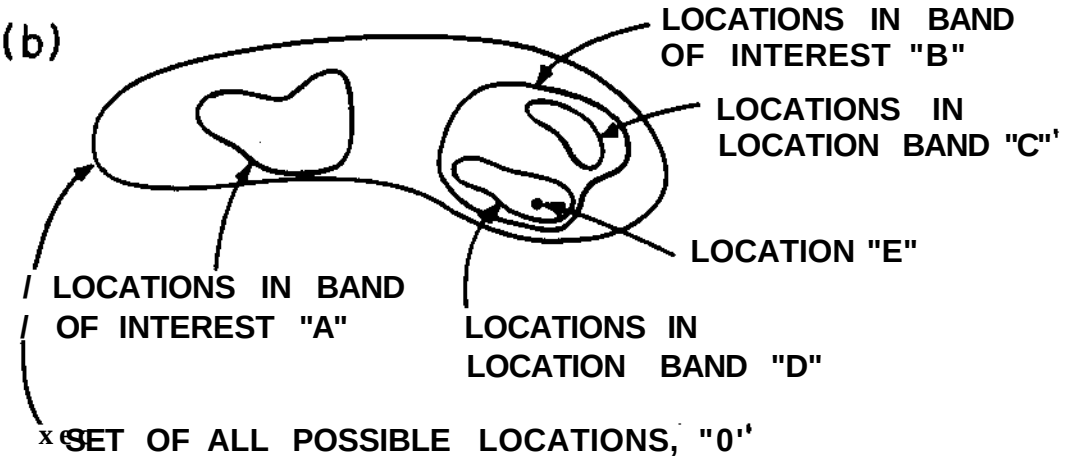
3.3.3 The use of abstraction in space planning

Space planning or layout design deals with many of the complex issues that typically arise in the design of artifacts that have to satisfy specified constraints and are composed of parts that have shape and take up space. It involves the design of two-dimensional layouts for floor plans, site plans, the placement of furniture in rooms or equipment in factories, etc. Through work that has been going on for almost twenty years it is better understood than other aspects of building design, or geometric reasoning tasks in general.

---

[22]More controversial and of lesser importance to this proposal is the model based on Bayesian decision theory that Manheim used for evaluating higher level alternatives.





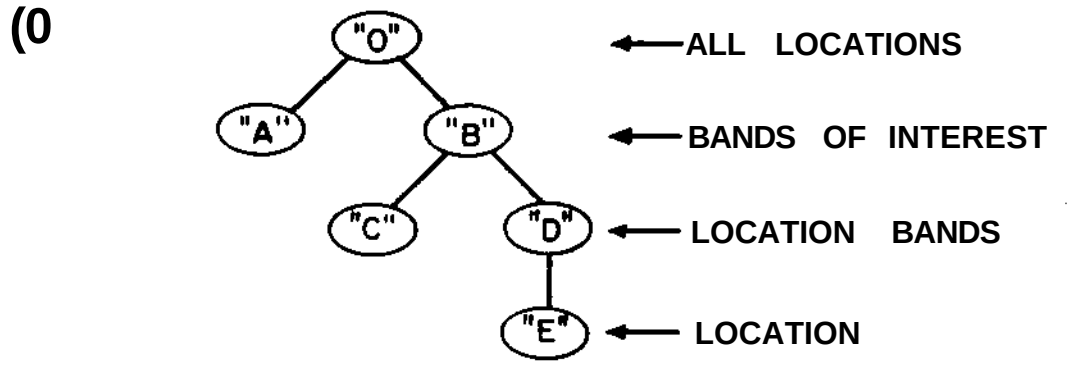


Figure 3-4: Hierarchical structuring of a highway location process, from [Manheim 64].

It permeates building design at most of its stages and therefore provides a rich context for the investigation of domains at various levels of complexity (or abstraction!). Also, there are many disciplines that have to deal with the equivalent of layout design in one form or another. Thus, though traditionally space planning has not included related domains -tiling window managers, layout of newspaper or magazine pages, and floorplanning in VLSI chip design- these also fall under the pervue of "automated layout". A given system may be capable of dealing with many or all of these domains if its representation and overall architecture are general enough.

In such problems topological and spatial relations such as adjacency, shape, distance, dimension and other functions of spatial arrangement are of principal concern [Eastman 73]. Alignments, groupings, aesthetic considerations as well as myriad functional requirements may be involved in the quality or performance of the layout. So as other complex design tasks, space planning involves multiple goals that cannot be be compared or tested for directly. Space-planning designers woik with models that capture the critical *design variables* for the task, which reflect the physical properties or form of the artifact and determine the "realizability" (see [Mostow 85]) of the layout. In architectural layout the design variables which are manipulated are the selection, position and dimensions of physical and spatial elements. These are the independent variables in the layout task and when translated into the representations) and abstractions used in the design method employed, they have been called by Simon [Simon 75] the 'autonomous constraints' of the method. Autonomous constraints are not supplied by the user of the method but are inherent in the design method itself, and to some degree they determine the 'style' of the design. These independent design variables or autonomous constraints are inevitable in as much as every design method must employ a representation which is an abstraction of elements of the real world. They also can be highly desirable if carefully chosen since they largely determine the generality, flexibility and effectiveness of the method.

The performance variables are those properties derived from, or dependent on, combinations of the design variables as fixed by the designer, and by which the multiple goals for a layout are evaluated. Achieving the goals of layout design is complicated because there is a many to many relationship between design and performance variables [Hemming 87]. Requirements or constraints on performance variables are an expression of knowledge of good design in the domain, coupled with problem specific and designer imposed requirements. An important part of structuring the problem is determining the performance variables and the constraints on them. Design has been characterized as a mapping of the constraints on performance to appropriate values for the design variables. It requires design knowledge (domain rules of good design) and skill (a good representation and method).

Space planning is inherently difficult because of the large (potentially infinite) number of location and orientation combinations available for placing any single design-object. The problem of locating a single design-object is compounded by the interdependencies among design-objects imposed by their shapes as well as spatial-relationships required to meet multiple performance requirements. No sequential sets of subgoals based on design-objects are likely to exist which allow partitioning of the problems with complete independence. Therefore, whether in human or automated approaches, search is required because there is no known method which is guaranteed to produce feasible solutions without trial and error.

33.3-1 Human strategies

Human designers develop efficient implicit search strategies based on problem decompositions that reduce cognitive load and suppress detail in order to handle complexity. For instance, much of the detail which can be **drawn** at the scale: 1/4 *inch = I foot* literally cannot be drawn at the scale 1 *inch* = 1 *foot.* Architects descend from grosser scale to finer scale in a relatively systematic process in developing solutions to architectural (layout) problems[23]. This transition through different scales is similar to the hierarchically structured sequential decision process for highway location planning of Manheim as cited above.[24] It includes the top-down refinement approach of hierarchical planning, and incorporates the recursive clustering of tasks and subtasks in the form of aggregation of the objects of design.

Other researchers in the field of systematic automated layout have also noted how a convention of "scales" enforces a discipline of abstraction in training young architects in the process of making a sequence of well-founded decisions: [Galle 86]

> When a young student of design is warned by his or her teacher not to ornament early drafts with premature details or a degree of accuracy not reflecting well-founded decisions, he or she is -perhaps unwittingly- taught an important lesson on *the art of abstraction.*

That the objects of layout are viewed by architects in recursively clustered hierarchies is also noted: [Kundu 88] "...we believe that the notion of subregion closely resembles the way an architect visualizes a design space. ...a dissection that represents a residential floor-plan may be perceived in many ways as "clusters" (and clusters of clusters and so on) of the basic regions."

Recurring in the literature on space planning is the observation that due to cognitive limitations (size of short term memory, etc., see [Simon 81, p 76]), human designers do not have the capability of making systematic explorations of alternative arrangements. To overcome this limitation, Hemming [Hemming 86b] and others stress the possibility of a symbiotic relationship between human designers and automated methods. In such partially automated methods an automated plan generator would enable the designer to explore the solution set systematically and consequently make more well-founded and rational decisions. The following quotes reinforce this idea, and the need for the capability of systematic exploration:

> The combinatorial complexity of most floor plan design problems makes it practically impossible to obtain a systematic knowledge of possible solutions using pencil and paper....Even ordinary design problems are so complex that an architect will probably find and contemplate very few solutions, hidden as they are like needles in the haystack of geometrical forms not satisfying all constraints [Galle 86, p 81].

> While it is relatively easy for a skilled human designer to produce reasonably good spatial arrangements, it is not so easy to make explicit the trade-offs and assumptions that form the bases of such arrangements, to understand how they might reasonably be varied, or see how well they might compare with potential alternatives [Liggett 8 1 , p 287].

It is just this capability that is sought through combining human expertise and automated systematic generation of alternatives in partially or wholly automated space planning systems.

---

[23]The suggestion of the importance of this practice and its relevance to the extensive use of abstraction in know ledge-based design systems was made in personal conversation by Ulrich Flemming - architect, PhD. and Professor of Computer Aided Design at CMU.

^It is also related in a general way to Hobbs's theory of granularity [Hobbs 85], which argues that viewing the world at different grain-sizes, and the ability to switch between granularities is a fundamental ability of humans.

33.32 Automated system strategies

Finding a good representation to support automated layout has been a difficult undertaking. The representation must express crucial aspects of the design method while suppressing other extraneous informatioa A representation is an abstract language for discussing and reasoning about the design at a certain level of detail. Therefore, it should incorporate exactly the information relevant to that level - too little information will not support the required reasoning while too much will obscure essential ideas and complicate processing unnecessarily. The most critical conceptual tool of the field has been the evolution of representations which can serve as the most appropriate and efficient 'intermediate abstractions' [Hemming 79, Galle 86]. Intermediate abstractions are nodes in the search tree representing partial solutions. No widely accepted meta-theory of design-method design yet exists, hence this remains an 'art of abstraction'.[25]

Critical reviews in the literature reflecting the sequence of development of abstract representations and systems in automated floor plan design can be found in [Francis 74, Eastman 75, Mitchell 77, Henrion 78, Flemming 79, Galle 86, Baykan 87, Flemming 88a]. These developments reflect the variety of *representations, overall system (search) architectures,* and *strategies* that have been attempted.

Representations:

- Grid representations or innumerable variations of numerical optimization methods based on the Quadratic Assignment Problem (QAP) [Koopmans 57, Liggett 81]: The drawbacks to this representation are that it is difficult to capture all the relevant requirements in a mathematical formulation of a spatial allocation problem. It may be suitable where flows (of material) are the overriding concern, but considerations of adjacency, shape, size, alignment, relative location, etc. are very difficult or impossible to model.

- Drawing based (dimensional) approaches; for the most notable systems using this type of representation see [Eastman 73, Pfeffericom 75]. These approaches had many interesting insights and strategies but suffered severe shortcomings in their representations and generators. In order to make the search space finite, this type of representation forces an arbitrary discretization of continuous space yielding heuristically based generators which were not systematic or complete. The order of insertion of objects affects the outcome of search, and there is no consideration of objects of variable dimensioiis.

- Graph or relational representations: These are the most sophisticated and they separate the topological issues of layout from the metrical ones. In graph representations the nodes represent elements in configurations and arcs represent adjacencies or incidence relations between those elements. Relational representations are the most suitable for addressing issues of interest in space planning because they allow for the most appropriate intermediate abstractions - generative mechanisms producing topological solutions as input candidates to dimensioning mechanisms for producing metrical layout schemes. They may also incorporate the desirable completeness property - where every conceivable solution will be generated. Galle [Galle 86] expressed concern with the strict separation of concerns approach because topological and dimensional constraints interact. He suggested that neither purely topological or dimensional bases for generation were adequate, but rather the appropriate intermediate-abstractions should produce integration of topology and dimensions in the same candidate solutions. This would make it possible to use both types of constraints for pruning the search tree as early as possible. The separation of concerns for complete and systematic generation and their reintegration for evaluation have been achieved in the LOOS

[25] As mentioned in the discussion on the design of automated systems, many researchers concur that the design of search architectures remains an art

system [Hemming 86c, Hemming 86a, Hemming 88a].

Overall architectures:

The distinctive feature of space planning problems is that decisions must simultaneously satisfy global requirements (such as, usage of space) and local or detail requirements (adjacency); a method must embody this [Liggett 81]. Also, an acceptable spatial arrangement results from a complex pattern of trade-offs. So ideally, there should be a structured method for making those trade-offs, such that the arrangement is justified and understood, location decisions are made explicitly and possible directions for variation indicated. To satisfy these requirements the overall architecture of the system should provide not just a 'satisfying solution[1] nor a single 'best solution' but a useful characterization of a class of solutions from which numerous acceptable variants may be constructed. Multiple alternative layout solutions are desirable. Current approaches, all of which use variations of relational representations, differ over whether it is most efficient and suitable to produce only the heuristically sound set of solution alternatives, or the complete set through a framework which separates enumeration and evaluation.

Auxiliary Strategies:

The overall search strategy is largely determined by the representation used: QAP or grid - hill climbing improvement strategy; drawing based - heuristic generation and evaluation strategies; graph based - constraint-directed search, or incremental G&T. Beyond that, a number of secondary, reinforcing strategies have been suggested in a number of sources, regardless of the overall approach:

- Priority solution. Assigns priorities to certain constraints by creating macro-objs (configs) to satisfy those constraints in a separate unbounded space. The overall layout constraints are then simplified by dropping those internal to the macro-objs. Macro-objs may be formed by weak clustering strategies based on distance or other constraints, or other strategies for forming disjoint sets. Pfefferkom's [Pfefferkom 75] use of macro-objs is related to Korfs [Korf 87] strategy of using macrops to solve non-serializable subgoal interactions.
- Planning the order of insertion of objects. Sorting by area and forcing the placement of the largest spaces first, or those with more rigid shape requirements (proportion ratio), or inserting the design object next that is most constrained by elements already located.
- Planning the order of testing of constraints. Constraints are applied to design-objects in order of the probability of failure.
- Use of pre- and post-processors. A knowledge-based front-end 'planner' is used to help structure or formulate the problem for the space planner.
- A sequence of iterations of the search and problem-formulation cycle as a solution to the issue of the over-constrained or under-constrained nature of space planning problems. Formulation is assisted by knowledge-acquisition from designers to glean their methods, approach and specific heuristics they use for abstracting and pruning the solution space. Empirical rules suggested by practical experience are incorporated as a basis for location decision making (different rules appropriate to particular types of problems).
- decomposition of the objects of layout on the basis of aggregation or clustering.

33.33 Decomposition: a key abstraction strategy in layout

Many former automated-layout approaches, which were not practically successful due to fundamental difficulties with their representations, nevertheless had identified similar strategies as critically important. A hierarchical approach to design problems in general and layout problems in particular has been suggested by myriad researchers and echoed repeatedly in human practice. A hierarchical representation and decomposition strategy for the *design objects* in layout was recognized very early in space planning

research [Eastman 73, Pfefferkorn 75], For example, in the above cited paper, Eastman initiates the long-standing intent to model hierarchical problem domains in automated space planning: "GSP is also being extended so as to handle hierarchical problem domains , e.g., 'arrange the equipment in the rooms, which are arranged and shaped within a building, which is arranged and shaped on a site.' *Such formulations allow precise study of problem decomposition in an interesting context[tmphasis* mine]."

Subsequent researchers up to the current time continue this emphasis, in systems with differing representation models of varying degrees of sophistication, and in a wide range of application areas: architectural and building layouts [Liggett 81, Galle 86, Baykan 87]; factory and facilities layout [Liggett 81, Fisher 86]; and VLSI chip layout [Kundu 88]. These researchers generally emphasize not only how a problem is divided into subproblems within a given representation but also stress that the solution procedure should be recursive - that the *subtosks* be of the same type as overall *tasks* so that a uniform algorithm, or solution procedure of the system, can be applied recursively at all levels of abstraction. Note the close correspondence between the following quotes from current researchers in space planning and the hierarchical approach of Manheim in highway location as discussed above:

> ...it has been our experience that human factory designers tend to work this way, successively refining rough designs into finished designs. Each task is viewed at different levels of an abstract solution space. The top level of this solution space is the factory, the next level departments, a third level work cells, a fourth, workstations, and a fifth workstation components [Fisher 86].

> Abstractions are important as a means for generating a dissection by the method of successive refinements, which is also called hierarchical placement.. Initially, one forms a decomposition of the set of basic regions into groups, and determines the possible placements of these groups as a whole within the design space *R.* As the refinement process progresses, each group is broken down into smaller subgroups and the positioning of these subgroups are determined within the space occupied by that group, and so on. The process stops when each group consists of a single basic region and the layout of the dissection is completely determined [Kundu 88].

In the above quotes, the first expresses the hierarchical decomposition of the layout task in terms of the entities of the domain of layout, while the second expresses nearly the same idea, but in terms of levels of abstraction of a particular representation for layout Neither of these ideas is new and certainly there is a long developmental history for both. As I have mentioned, the aggregation and decomposition of objects has seen much exemplification and justification in terms of human strategies and methods, but little clear implementational success in automated layout The refinement of representational abstractions for layout has been evolved with considerable effort through a series of automated system approaches.[26] Also, there is certainly nothing new about the power of abstraction in hierarchical planning and design by refinement. What is interesting is that these concepts have never yet been conjoined in a practically successful space planning system.

3.3.4 Summary

[Korf77]

> The design process can be viewed as a search through this tree for a desired design product at the bottom. The search can best be accomplished by a partnership of man and machine. The machine can systematically enumerate all the branches at a given level. This exploits the power of the machine as the enumeration could not be done by hand [Mitchell 77]. Since the tree is large, branches must be pruned at each level. This *selective* machine generation requires other information, such as orientation and enclosure constraints, in order to present only appropriate solutions. Since a branch represents an entire equivalence

[26] And has come to something of a fruition in the LOOS research, as described later in this paper.

class of solutions, the user can *interactively* explore other members of that class.

The following points summarize the important relationship of abstraction to a design process such as space planning:

- Abstraction is the main and prerequisite tool for handling complexity in problem-solving and design.
- Two main forms of abstraction are abstraction by generalization and abstraction by aggregation; other forms of abstraction no doubt exist. It has been observed that abstraction by aggregation is perfiaps the main form of abstraction utilized in design.
- The ideal number of levels in the abstraction hierarchy for a search space has been analyzed quantitatively. In practice there are a great many factors that may prevent that ideal from being achieved. Therefore, heuristically it is best to follow the guidance of the compiled experience of domain experts in constructing abstraction levels (e.g., decomposition hierarchies), and beyond that, to have as deep a hierarchy as possible.
- In space planning, finding the appropriate representation for modeling the problem is the most crucial form of abstraction (with many challenging properties required). The representation should support a process for systematic exploration of candidate solutions. It should also provide for intermediate abstractions that allow evaluation of partial solutions. Abstraction continues to play a critical role in the way in which a problem is divided into subproblems within a given representation, for instance, a decomposition hierarchy for the objects or design components.
- In both human and automated space planning systems, some strategies for planning the layout process appear repeatedly. Of these, pertiaps the most important strategy is the nearly universal requirement for an abstraction hierarchy for the *objects* of design. The objects of design in layout can be aggregated into clusters and these clusters may be viewed as a hierarchy of abstraction levels at different levels of detail. The relationship between the overall representation for performing the layout process and representation for the objects of design is critical to the architecture of the system in terms of these key properties: generality, flexibility, effectiveness, and efficiency.

4. Problem Statement

4.1 Motivation for the proposed research

From the discussions on the use of abstraction in planning and design systems, the issue of the combination and interaction of different forms of abstraction emerges as an interesting and complex issue. The integration of techniques of abstraction by levels with hierarchical decomposition is of particular interest, and takes on added dimensions in design problems where geometry - the shape and location of objects - is important. As with most reasoning about geometry, research in this area is likely to present some intricate and difficult issues. Therefore, in order to progress it has to be done in some realistic but manageable domain.

The research area of automated layout design provides an opportunity for reasoning about shape and geometry in a restricted two dimensional manner. It is also rich enough to include realistic problems in multiple domains, and has seen sufficient development to reach the point where representational abstractions and approaches exist such that search spaces can be defined and effectively traversed for some problems. This current state of development focuses further research on additional mechanisms to inform and leverage search. In this regard, a very important focus is the combination of decompositional abstractions through the objects of design, with the primary representational abstractions used for delineation and generation of alternatives.

Interest in issues of design abstractions converges, in the context of space planning, with the opportunity of extending a successfully developing system. The current status of this especially promising space planning approach is briefly reviewed in the following section. It presents a unique opportunity to investigate the integration of different forms of abstraction in a design system. The research promises results that should be of strong generalinterest, as well as the very real and concrete goal of producing a more powerful space planner.

4.1.1 LOOS: an automated layout system

The LOOS system has an architecture designed around a graph-based relational representation which facilitates the systematic generation of all candidate solution layouts and a separate tester which supports evaluation of candidate solutions over a wide range of criteria. The generator and tester, combined with an appropriate controller yield an overall architecture which functions as an efficient variant of generate-and-test (G&T).

The generator utilizes a domain-independent, purely syntactic grammar based on spatial relations between objects. These spatial relations formulate a type of intermediate abstraction in delineating layouts that reflect the separation of concerns[27] - topological from metric - for the purpose of generation. Each representation is an abstraction in that it suppresses the *continuous* dimensional properties of candidate solutions, and thus the possibly infinite set of solutions is divided into a finite set of subsets. By parsimoniously defining differences between alternative layouts and also governing the enumeration of alternatives, the representation (in terms of *discrete* spatial relations) enables the spatial-layout problem to **be framed as one of *search in a well-defined space.***

[27] **As described in the section above on abstraction in space-planning.**

Given such a well-defined space, the challenge is to evaluate and discriminate among alternatives such that the space is efficiently traversed. Hopefully, for the typical problem, this traversal will yield a manageably small solution set of equivalently good, but distinct alternatives. Here the spatial relations also help in providing an explicit structure among the objects in the layout which facilitates their evaluation over multiple criteria, including the re-integration of metric with topological concerns. For discriminating **among** alternatives, the LOOS system architecture integrates a domain-dependent *knowledge-based approach for evaluation* with the capability for systematic generation. The knowledge-based tester permits both evaluation over multiple criteria, and incremental formulation through knowledge-acquisition.

Communicating between the generator and tester is a controller that uses the results of evaluation and a simple branch-and-bound(BNB) strategy to steer the traversal of the space away from less promising alternatives (deferring their development until it is certain that nothing better can be found). This is handled through an evaluation record which facilitates the lexicographic ordering of solutions with respect to failing constraints. Through the evaluation record, the results of testing can guide the search, hopefully ignoring major portions of the space, without a priori prejudicing the resultant set of candidate solutions that emerge at the bottom.

Space does not permit a more detailed description of how the system really functions, let alone a full discussion of the justification of the G&T approach in space-planning, or of all the interesting features and possibilities that the LOOS system possesses. A full description of the current status of the approach can be found in [Hemming 88c] and a brief overviews in [Hemming 88b] or [Hemming 88a]. The history of its development through several versions can be found in [Hemming 87, Hemming 86a], and of the evolution of the theoretical basis for its representation and generator in [Hemming 88c, Hemming 86a, Hemming 80, Hemming 79, Hemming 78]. For the purposes of this proposal, the most important thing to note is the viability of the approach thus far in handling real, albeit small-scale problems, and to focus on the most severe potential obstacles to extending the generality of the approach - both to larger problems, and additional domains.

4.1.1.1 Significance of the approach

Taken together, the representation, generator and overall architecture of the LOOS system effectively address some of the long-standing fundamental problems of automated space planning.[28] Its representation suitably partitions the solution space so that no more candidates than necessary are described while all desirable candidates are represented. It combines this with a knowledge-based tester which can be incrementally constructed, and a control structure that allows for efficient pruning and for making tradeoffs among performance goals. Because it can represent all possible alternative layouts for a given problem, and separates their generation from evaluation, the approach deals efficiently with the *uncertainty of outcome* that is present in layout problems due to the interaction of global constraints (such as space consumption and dimensional fit) with more local constraints (such as a specific required adjacency between objects). The following points summarize how the approach addresses some of the most difficult issues of automated layout:

- The strict separation of the generator from the tester allows the design of a domain-independent generator defined in purely 'syntactic' terms, established *a priori* by a theory guaranteeing useful formal properties, and not complicated by contingencies of a particular domain or by unforeseen discoveries.

^For a good description of some of the difficult problems and limitations of former approaches see [Henrion 78].

- The 'semantic' concerns are located in the tester within a well-defined context (which can be analyzed by means of the explicit structure provided by the representation) for evaluation. The unencumbered generation of partial alternative solutions stimulates domain experts to make explicit their knowledge when confronted with a concrete solution, as opposed to formulating their knowledge in general terms unrelated to a concrete case. The stimulus that brings expertise to the surface is being confronted with flawed solutions in familiar graphical display[29], which the underlying relational representation is readily translated into.[30]
- The framework allows for tradeoffs between alternatives, with contradictory demands reflected in distinct alternatives, as opposed to other approaches in which contradictory demands usually cause the system to fail.
- It also allows for effective handling of the many to many relations that exist between design and performance variables - neither is evaluated in isolation. In order to compare alternatives, the set of alternatives is complete and can be evaluated over the entire range of criteria that applies. LOOS employs a 'generator of constraints' rather than a 'constrained generator' for producing candidate solutions. Each potential candidate solution is an intermediate abstraction generated in terms of the primary design variables (spatial relations) and each has associated with it the full set of performance constraints which can be evaluated in terms of those relations.[31]

Because of its capability for exhaustive enumeration, and concommitantly, its separation of producing candidates from evaluating them, LOOS is able to manage decision making and commitment in a unique and efficient way (from the point of view of commitment of resources in search). It divides the job of producing candidate solutions into two parts - their structure (or syntax), and their interpretation in meeting performance requirements (or semantics). By doing so it is able to assert the structural definition of candidates *monotonically* to produce new points in the search space with certainty. And it avoids making uncertain commitments(or guessing), in the form of predictive decisions about performance, by maintaining a "currently globally best" set of candidates through *monotonic* negative evaluation, and always expanding that set in the minimal way in the next round of generation. This management of resources is effective in the face of multiple, complex interactions among the goals of layout, the global uncertainty of outcome until all objects arc placed, and the need to systematically explore tradeoffs among multiple candidate solutions.

This is in contrast to approaches which attempt to combine both syntactic correctness(well-formedness) with semantic correctness - all in the generator. The problems with these approaches arc not having essential completeness in the operators of the generator, and related, attempting to foresee all interactions up front. Even with the invention of more subtle decision making processes to forestall uncertainty - such as extending least-commitment through partial commitment - uncertain decisions (guessing) may be required just to keep processes of generation moving at times (deadlock). Since commitment in the face of uncertainty can be narrowly local or wrong, it may result in expensive backtracking.

---

^Pictorial resemblance to solutions has been pointed out as an important quality of intermediate abstractions in space planning [Galle 86]. In the LOOS system this allows the user to modify the problem definition as soon as sufficient insight into the solution space has been obtained, which is essential for underconstrained problems. On overconstrained problems, this also facilitates a step-wise constraint relaxation.

[30] See [Mittal 86c] for confirmation that in many domains of design problem-solving knowledge-acquisition requires exercising the experts on representative problems, instead of asking them how to solve the problems.

[31] LOOS is (in Galle's terms [Galle 86]) an approximative approach, which continuously takes all dimensional and topological constraints into account, but without detail.

I will return to discussion contrasting these two approaches for space planning in the following section. But first, I conclude the brief overview of the LOOS approach by explaining in slightly more detail its monotonic approach to evaluation through failure.

Monotonicity and commitment

LOOS employs monotonic operators in dealing with its primary design variables -*spatial relations.* As a result, it is certain that if a necessary condition for a good floor plan is violated in a partial plan it remains violated in all extensions constructed by applying additional operators to it. Another way of saying this is that a partial solution can only get worse, never better, through placing additional objects into the layout. Mostow calls such conditions *monotonically* necessary and violations of such necessary conditions *monotonic failures* [Mostow 87]. Because space is a global resource, and interacts with all of the other performance constraints involved in space planning, LOOS stands commitment on its head in terms of the goals or performance requirements of layout design. It does this by utilizing *greatest commitment* to the design variables (spatial relations) and ensuring that they can not be altered on the same branch of the search tree through subsequent object insertions. Since all performance variable are evaluated in terms of these design variables, this guarantees that once a partial solution fails in terms of some performance test, it can never get better or pass the test through further development.

When a new object is inserted into a parent layout(one of the set of "current globally best"), we have certainty about the performance of that layout up to that level of its relevant domain tests. A new object carries its own set of performance requirements and may interact with other performance requirements (local or global). Therefore, in maintaining an open position regarding tradeoffs, there is no way to know the best way to add the new object So the parent is expanded in all ways. In the absence of certainty it is best to defer deciding, so the LOOS system does until it has more information on all those children. In the constraint-directed approaches, when least-commitment breaks down, they still have to commit to a choice (and later backtrack if it is wrong.) LOOS does not commit to choices based on prediction. It commits resources to exploring the "current globally best" set in all possible ways.

Thus the LOOS approach is intrinsically a monotonic process, with no backtracking, rather than a complex set of operators for producing only sound candidates, coupled with knowledge and mechanisms for modifying a design when it fails some acceptability condition. LOOS makes full commitment in terms of primary design variables - spatial relations - but it also provides an inexpensive way[32] to evaluate and rank all possibilities produced in terms of these spatial relations (proposed constraints).[33]. It makes no commitment in choosing which of the current globally best's children should be expanded - it expands them all! It is not attempting to predict performance, but uses *least-commitment of resources* to evaluate that set monotonically so it can again commit with certainty to the new set of currently globally best candidate solutions.

4.1.1.2 Appropriateness of the basic architecture

In modeling design problems as search, the distribution of design constraints between generators and testers is pragmatic and depends upon the characteristics of the problem domain, and what constraints are

[32] **See the section below on levels of abstraction in LOOS.**

[33] **Obviously this is putting aside the issue of the combinatorics involve, which I will take up shortly below**

easily embedded in generators and what ones are not [Simon 75, p 294, Hayes-Roth 83,p 72].[34]

The architecture of **LOOS** may be called *hierarchical-G&T.*[35] According to the evaluation of the LOOS problem features **and** matching problem-solving prescriptions in [Hayes-Roth 83, p 91] this G&T architecture is appropriate for solving a wide variety of layout problems because of the following characteristics:

- There now exists a plausible generator of the solution space (and of partial solutions)
- There is a tester for evaluating partial solutions.
- There is no uncontrollable uncertainty or noisy data, etc. as problem characteristics.

LOOS is very similar in architecture to the DENDRAL system, in which problem solving knowledge is represented as a special piece of code for the molecular structure generator, and as rules for the data-driven component and evaluator. Similar to DENDRAL the structure generator, which is essential to the approach, is a computationally effective algorithm that enumerates all possible [chemical] structures, that is, it computes the size of the solution space rather than the solution [Hayes-Roth 83, p39].

G&T has been cited as appropriate method to consider when it is important to find all of the solutions to a problem. [Hayes-Roth 83, p 89]. This is because it provides the capability for maintaining multiple design variations simultaneously between different partial designs. So, the system can explore different options in parallel [Mittal 86c]. For the method to be workable, however, the generator must partition the solution space in ways that allow early pruning. As I will discuss below, in a number of powerful ways LOOS already does this! The need and potential for additional means - decomposition - are the focus of this proposal.

Other possible approaches

Tong [Tong 87] describes an important division in the choice whether to build a design system based on a complete generator or one which produces only a heuristically restricted subset of sound solutions; the decision depends on which simplifying assumption has the most merit in the domain. Differing problems and domains may require different architectures and different tradeoffs with respect to this division. The dichotomy is motivated by how and when interactions in the design goals are handled:

1. ***Generate solution candidates without consideration of interactions and have a strategy for*** *dealing with them later.* Make no assumptions up-front, but generate and test in such a way that all possible candidates are considered and the ones which evaluate poorly (using knowledge) are thrown out or pruned. This approach relies on a complete generator to produce solutions with all possible tradeoffs of design goal interactions, along with strategies in the generator and evaluator to avoid the potential combinatorics of exhaustive enumeration.

2. ***Generate only those solution candidates in which design goals are realized and interactions***

---

[34] A generator does not even have to guarantee well-formedness, it can be left to the tester if it is a viably efficient approach in the domain. LOOS guarantees well-formedness in terms of spatial relations, and includes a dimensional feasibility precondition on generation operations.

[35] Hayes-Roth, et al [Hayes-Roth 83, p 72] classify *hierarchical* G&T as follows: "An important form of generate-and-test. this formulation permits pruning of candidate solutions that are only partially specified (only the first half may need to be specified say). When a partial description is pruned, an entire class of solutions corresponding to the description is eliminated from the generation process. Hierarchical generate-and-test applies powerful pruning rules early in the generation process. ... The ability to prune effectively (reasoning by class elimination in hierarchical generate-and-test) is appropriate for optimizing because it admits a complete search if the pruning rules are sufficiently powerful"

> *are resolved.* **Such systems apply knowledge directly, up front, in the form of their 'rules' for generation, so as to anticipate and avoid interactions. They satisfy constraints directly, and propagate constraints continuously to warn of interactions and avoid violations. Such systems create a search space of possible solutions in a constructive, opportunistic manner through constraint directed, or grammar based, or semantic-rule based synthesis of only the "sound" alternatives. They attempt to avoid the cost of search and backtracking as much as possible.**

It is interesting to note that both of these approaches may make use of all the strategies of planning described above in different ways, and they both need to use some of these strategies in order to be minimally functional. The use, in some form, of abstraction, hierarchy and constraint propagation seem common to all while they vary in terms of making eariy, opportunistic, or late-commitment. The specific architecture of a system dictates how successfully some of these strategies may be embedded. But each has different strengths and potential weaknesses which need to be examined further in the context of space planning. I have already pointed out the strengths the first approach, which is that of the LOOS system, so the following is a brief discussion of the merits of the second.

Constraint-directed or knowledge-driven search in layout design.

> **[Simon 81, p 139]An earmark of all these situations where we satisfice[36] for inability to optimize is that, although the set of available alternatives is "given" in a certain abstract sense (we can define a generator guaranteed to generate all of them eventually), it is not "given" in the only sense that is practically relevant We cannot within practicable computational limits generate all the admissible alternatives and compare their respective merits. Nor can we recognize the best alternatives and compare their respective merits. We satisfice by looking for alternatives in such a way that we can generally find an acceptable one after only moderate search.**

The biggest advantage claimed by this type of search is the use of more powerful problem-solving models to steer search through a vast space of possible designs in an efficient manner. Variations on this approach all attempt the application of design knowledge directly to produce candidate solutions and avoid the combinatorics associated with exhaustive enumeration. Whether a goal-directed planning, constraint-satisfaction, or blackboard approach they all attempt, in a combination of top-down or opportunistic 'middle-out* strategies, to connect the design goals directly to the sequence of generative operations which will achieve those goals.

Though these approaches allow for sophisticated structuring of the problem-solving process (they have the flexibility to include many kinds of knowledge and places to attach and invoke it) they are relatively large and complex architectures.[37] Though they are especially suited to certain types of problems, they may prove to be costly and ineffective when applied to problems which do not necessarily have the characteristics demanding such complexity.

I have already stated that the relationship of design variables and performance requirements is complex and many to many in space planning problems. Thus, negative interactions involved in achieving the many performance requirements are difficult to foresee and will likely require complex and costly processing to anticipate and avoid. Additionally, layout design definitely involves the problem of the

---

[36] **In the unlikely case that anyone is unfamiliar with the term "satisficing", it was coined by Simon some years ago for decision methods thai look for good or satisfactory solutions instead of optimal ones**

[37] **For instance, the blackboard approach was invented primarily to handle problems which other models were unable 10 treat effectively due to characteristics such as noisy or uncertain data, no plausible generators for candidate solutions, etc. [Nii 86].**

intermixing of both global and local constraints discussed earlier, which results in the complication of *uncertainty of outcome.* If these constrained generation approaches are to handle space planning problems effectively then they must either assume: 1) that all interactions and consequences are known up front, or 2) there is a general way to deal with them when they do arise (for instance, invoke some some meta-level rule which has direct knowledge of that type of interaction, or meta-meta-level, etc.). Sophisticated backtracking will not suffice as the general mechanism, for reasons I have noted about the difficulty of assigning blame when a global resource constraint is involved. Tong summarizes some of the shortcomings of using constraint-reasoning as the primary generator of candidate solutions in some design problems: [Tong 87]

> (1) constraint reasoning only avoids currently known and well-characterized classes of bugs; (2) constraint reasoning usually only avoids classes of bugs that are representable as interactions that are *local* with respect to some underlying structure; (3) constraint reasoning can pay a high overhead in memory ...(4) the *solution density* may be high, and guessing may be worthwhile, as the problem is underconstrained.

Of course, as noted above, the extensions to constraint-directed search that allow partial commitment or an explicit 'inclusive approximative description[9] of ranges for candidate solutions (see [Mittal 86b]) allow for both multiple candidate solutions to be carried along (which helps with tradeoffs between constraints) and for the certainty of decisions to be extended somewhat(which helps with the interactions between local and global constraints.) Both Tommelein [Tommelein 87a, Tommelein 87b] and Baykan [Baykan 87], in combination with opportunistic 'focus of attention' strategies, appear to be using some variation of this incremental range restriction approach in current space planning systems. Tommelein justifies using a *blackboard architecture for control* approach (based on BBI [Hayes-Roth 85]) for construction site layout because of the time-varying uncertainty of the data (objects to place). Baykan is exploring the effectiveness of the elaborate constraint-directed search architecture of Fox [Fox 86](originally motivated and developed for 'job-shop scheduling' problems, which also involve time-varying and uncertain data), in domains of layout which are easily statically modeled, e.g., residential kitchens.

Even if those systems are able to demonstrate efficiency in finding multiple solutions to given layout problems, they have no overall measure of the 'goodness' of the solutions they have found. This is because they base their generation of alternatives on known classes of constraints and constraint interactions' and have no formal basis for generating the complete space of alternatives without *a priori* reference to performance requirements; this also impinges on the capability for exploration of tradeoffs. They are inherently 'satisficing'[38] and encompass no notion of the 'globally best* or optimal set of solutions. Of course, the knowledge-based approach, with incremental knowledge acquisition in an environment supportive of a tight loop between problem-formulation (adding new constraints as generators) and problem-solving (constraint-satisfaction), will help in making the approach more complete in any given domain over time. But there is no guarantee that the process is likely to stimulate formulation of additional constraints from domain experts whose knowledge of 'good layout design' is largely implicit The process is self-circumscribing and has no mechanism to go outside of the current set of generation operators and thereby confront these experts with other alternatives, perhaps examples of 'bad layouts', which would trigger further knowledge in a language that they are familiar with. Additionally, there is no explicit structure in the candidate solutions which they generate, so further evaluations after generation are not easily supported.

---

[38]Baykan extends the notion of satisficing to cover cases where not all constraints can be met This allows for some degree of 'tradeoffs' where former satisfaction schemes would just fail. But these tradeoffs are still limited to known classes of constraints in terms of producing alternatives.

Therefore, there is some question as to whether constraint based or semantic rule based generation schemes (controlled by a sophisticated search frameworks, such as knowledge-based hierarchical planning or opportunistic constraint-directed search, or a blackboard model) are suitable for automated layout problems. In a sense they are both too much and too little machinery. The are too elaborate architectures for dealing with problems which do not have noisy or time varying data, but they are insufficient to deal with the uncertainty of outcome inherent in space planning problems. It is probable that in attempting to prescribe the goodness of proposed solutions, their heuristic generation schemes cannot deal as effectively with unforeseen interactions which occur. It is not clear what mechanisms those approaches would then employ.

An appropriate search architecture for layout design

In summary, it can be said that there exists a spectrum of knowledge based design approaches which are currently being applied to automated layout. The approaches at one end of the spectrum risk the daunting complexity of attempting to anticipate interactions and constructively meet the design goals so as to avoid the combinatorics of exploring all possible alternatives, most of which will be defective if not outright rubbish! On the other extreme is the LOOS approach which risks the combinatorics of exhaustive generation, which it is believed is controllable with the appropriate devices, in favor of explicitly exploring the tradeoffs among the design goals, which it is believed cannot all be anticipated nor resolved up front. Both types of systems advocate the modular architecture familiar in knowledge-based approaches so that the necessary knowledge of "good" design in a given domain can be built up incrementally. This requires an environment where the system can be run on problems in the domain, its performance observed and knowledge facilitating additional discrimination or refinement among design alternatives is "triggered" in the domain expert and conveniently added to the system. In this regard, a clear advantage of LOOS' generative approach is that it autonomously presents alternatives to these experts to react to (albeit some are ridiculous) thus drawing out their *implicit* knowledge of better design, which they may not have been able to explicitly articulate otherwise. This phenomenon has been well documented in cognitive psychology research where the greater difficulty of *recall* over *recognition* is noted.

However, both types of automated design systems are worth exploring and merit further research and development. It is only the empirical evidence of the performance of these systems on real world problems in varying domains that will reveal their comparative merit; there currently is no theoretical basis by which to evaluate and predict this. Of course this is perfectly normal for AI enterprises. In large part it is by your set of heuristics that you design an automated design system -- "You pays your money and you takes your chances!" In the end, with design being a rich, flexible, adaptable, and hard to encode process, it is likely that we are going to need *all* of these mechanisms and many more to build truly effective and intelligent CAD systems.

Formulation as a numerical optimization problem

Another possible approach to solving layouts is modeling them as a constraint problems using Mixed Integer Linear (or Non-linear) Programming(MILP or MINLP). But typically, not enough is known upfront about the constraints to *formulate* layout problems for this type of solver. So I believe that knowledge acquisition must take place first in an incremental, diagnostic manner. Also the knowledge base in typical layout domain is not likely ever to be stable. Design problems such as layout require interpretation in context even when the knowledge for problems in the domain is complete. Therefore the user may have to activate or deactivate rules (or tests) in the domain, and must be able also to enter

problem specific constraints. Use of an efficient[39] solver such as MILP may be possible in principle but its utility is obviated by the difficulty of the formulation of problems, and knowledge sufficient to solve them in the domain.

Conclusion about the appropriateness LOOS's G&T approach. [Bernstein 88]

> He [Charles kettering, inventor of the automobile starter, diesel locomotives, etc...] talked about research and education, and as he neared 80, he spoke increasingly of failure. *He was for* //.[emphasis mine]. 'I think it was the Brookings Institution that made a study that said the more education you had the less likely you wore to become an inventor. The reason why is: from the time a kid starts kindergarten to the time he graduates from college, he will be examined two or three or four times a year, and if he flunks once, he's out Now, an inventor fails 999 times, and if he succeeds once he's in. An inventor treats his failures simply as practice shots.'

If we may stretch things a bit and talk of the 'invention' of solutions to space planning problems, this quote goes a long way toward describing the philosophy of the the LOOS approach and its difference with the constraint-directed search approaches. They look where and how they already know how to look....LOOS looks everywhere and makes progress through failure! The conclusion is that the approach appears to be especially clever in its *epistemological adequacy* - its capability to accumulate and apply knowledge of good layout design. The looming issue is whether its viability in terms of *heuristic adequacy* (practical efficiency) can be demonstrated on larger problems.

4.1.U Leveraging search in LOOS

Abstraction levels in LOOS

In several important ways LOOS is already hierarchical and abstract. It incorporates abstraction levels in its architecture in least the following four ways. Each is listed preceded by a reference to that part of the system architecture whose properties primarily make the particular abstraction mechanism possible.

1. *Representation:* Each configuration represents a class of feasibly dimensionable solutions.
2. *Generator:* Each object added yields a partial configuration which may be evaluated with certainty to its level of development. The monotonic character of LOOS's generation scheme allows the addition of each object (adding another level to the search space) to be viewed as stable partial solutions and to be treated with certainty.[40]

3. *B-N'B control framework and tester*

- The BNB evaluation record partitions the order of importance of tests in ascribing each a "utility" by assigning it a slot for scoring - constraints, hard criteria, soft criteria. These levels of importance, which are extensible for finer grained discrimination, effectively contribute to partitioning the search space into levels in terms of deferred development.[41]

---

[39] **whether it would in fact be more efficient than the knowledge-based hierarchical G&T approach of LOOS is an interesting empirical research question.**

[40] **This is in counterpoint to GaUe's statement [Galle 86] that although design methods which build-up solutions one element at a time may be viewed as having partial solutions which are 'aspectual abstractions*, he regards this type of abstraction as not interesting in that it just reflects the trivial fact that a solution is composed of elements and intermediate abstractions of interest must be more than merely branching nodes!**

[41] **This results in essentially the same effect as in the constraint-directed search schemes where, it is claimed, the importance of a constraint can be used to determine which variables are to be bound first in a manner similar to that of ABSTRIPS [Fox 86].**

• Incremental testing of each partial solution just to the point of discrimination.[42]

Additional leverage through heuristic devices

Where possible we have employed additional heuristic search devices within the BNB control framework of LOOS. The first three slots in the evaluation record for a partial solution represent confirmed judgment of the domain tester regarding the "badness" of that partial design. This is not heuristic and it is guaranteed not to improve (because of the monotonicity of the generator). But additional slots can be added which reflect a heuristic judgment about the prospects for that partial solution, and which affect its scheduling for further development For instance, we added a slot which records the relative completeness of an partial layout solution in terms of the number of objects already included. This enables us to express and employ the typical heuristic that *all things being equal we prefer to expand partial solutions which are closest to complete* (in terms of number of objects already located).[43]

Under BNB this does not ultimately prevent the controller from returning to a deferred partial solution if other branches turn out to be worse in the long run. However, once we reach the first leaf (a layout with all objects in) then all those partial solutions that have a worse evaluation record can be pruned; this is when the heuristic of deferring some partial solutions (by putting them toward the back of the list for expansion) bears fruit. With the addition of this heuristic, which is similar to adding an estimate of the distance to the goal, the search strategy becomes more like A*.

In the above discussion, whether the classification of the system architecture as "hierarchical generate and test" and our search control as "A*-like" is completely accurate is overly important. It is important that we use an efficient variant of generate-and-test which allows certain evaluation of partial solutions, and we are employing whatever heuristic methods we can to effectively reduce search; that is, the methods do not cost more in overhead than they save in effort. We have convincing empirical evidence that this is the case in regard to those described and will continue to add whatever such devices occur to us in an incremental fashion. What is most essential is to assess the limitations of the current system to judge what the most critical extensions to the approach are in dealing with real problems of greater complexity.

4.1.1.4 Assessing limitations **of the approach**

A list of the potentially most significant limitations of the LOOS approach are as follows.

Restriction to rectilinearity.

At this point in the development of automated space planning restriction to rectilinearity appears to remain a necessary simplifying assumption. For now it is an acceptable limitation because it makes the problem of algorithm design manageable without excluding the most commonly used class of layout plans.

Overconstrained and underconstrained problems

With design problems in general it is often difficult to determine at the outset whether the problem may be over or underconstrained as formulated. Space planning problems are no exception, in fact they may be generally worse in this regard [Baykan 87]. Even an incremental generate-and-test framework capable

---

[42]Design and implementation of this strategy are attributed to Tim Glavin.

[43]This is similar to viewing the number of subgoals remaining to be satisfied as an evaluation function in means-ends analysis. Li space planning, the number of subgoals to be satisfied is often associated with the number of objects to be placed, as it is in LOOS. The use of this strategy in LOOS was also developed by Tim Glavin.

of maintaining tradeoffs with a BNB directed search could be problematic on either overconstrained or underconstrained problems since too many partial solutions would have equivalent evaluations (regardless of goodness or badness). This failure of discrimination would thus allow too many potential solutions to remain alive in the search space which would combinatorially explode. This limitation is partly offset by the tight loop in LOOS between running problems, and adding knowledge to the tester (which effectively re-fonnulates the problem by adding or refining constraints). In LOOS rapid problem reformulation is afforded by the strict separation of generation and testing, and the combination of G&T with the knowledge-based systems approach.

In future refinements of the LOOS approach, another mechanism to deal with this problem may be having additional levels of knowledge for evaluation, (or constraints) in testers representing a finer grained discrimination of alternatives. These levels would be reflected in additional slots in the evaluation record. For instance, a possible extended hierarchy of constraint types for architectural problems is:

1. well-formedness tests, such as dimensional fit
2. code tests
3. good practice
4. client preference
5. office style or policy
6. designer preference

The criticality of tests in these groups, in relation to the performance of the overall design, is hierarchic starting with the most important at 1. The groups 1,2,3,4 and 5,6 seem to have approximately equivalent weighting within their own group and so performances according to tests within these groups could be traded-off against one another. The significant thing about the elaboration of levels of possible tests that might be applied is that those that reflect somewhat arbitrary constraints or criteria (designer preference) could contribute to discrimination on problems which would otherwise be over or underconstrained. The LOOS approach will readily accommodate such customization, and its applicability will depend on the domain in question.[44]

Additionally, a knowledge-based front end planner could do much of the appropriate structuring of the problem by heuristically transforming user requirements into an initial set of 'components'as in configuration problems [Frayman 87],[45] This coordinates the idea of allowing user preferences in the performance requirements, with user preferences and system guidance in domain object selection. Frayman states that there is an important separation between the tasks of understanding clients needs, producing the specifications of their needs, and the layout task itself. That separation mirrors the separation of problem structuring and problem-solving because they require differing inferencing mechanisms. The input to the layout task itself should be in some *well defined language,* specifying requirements for individual design objects, and requirements for the overall layout.

In general, achieving effective problem formulation in conjunction with the LOOS approach would be most powerfully assisted by the creation of a *language for layout.* Such a language would support many of the above ideas - incremental knowledge acquisition, refinement of knowledge for increased

[44] For many architectural problems (which are frequently underconstrained), this idea does indeed reflect the reality of the types of concerns brought to bear on problem formulation (and re-formulation) in architectural practice and contributes a great deal to the uniqueness and individualistic style of solutions produced by designers.

[45] An early primitive version of such a planner for kitchen design was already built by Coyne as a project in the course "Expert Systems in Civil Engineering" Spring 1985 at CMU. The planner used a simple rule-based approach to help in the formulation of problems for the LOOS G&T solver. It attempted domain-specific, knowledge-based planning of the context, objects, constraints, tests, etc. to be used in the generation cycle.

discrimination, overall planning of problems - in a much clearer and more powerful way by allowing experts in various domains of layout to interact directly with the system in building up formulation and evaluation knowledge for their specific problems.

G&T architecture combinatorics.

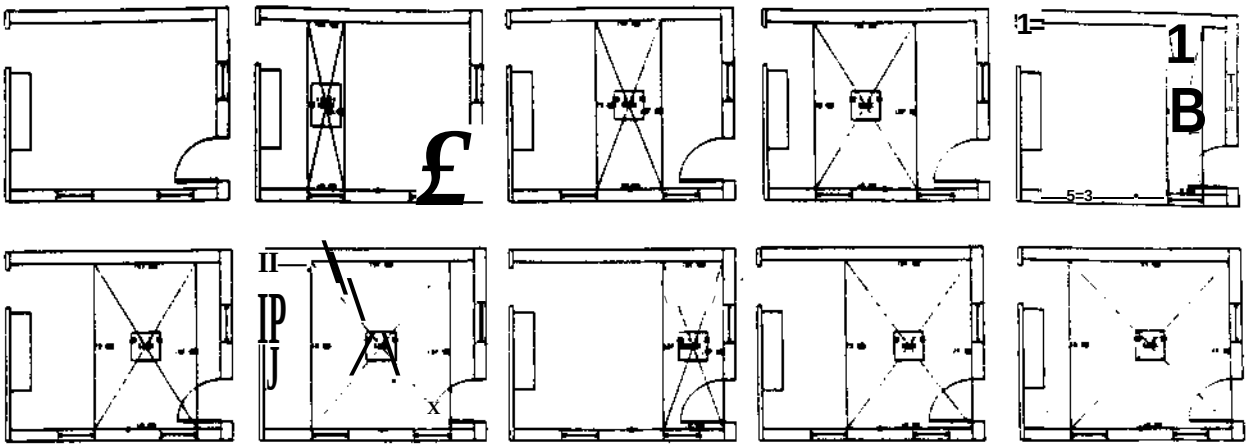
LOOS operates in an unnecessarily 'flattened' object space in terms of the domain design objects attached to nodes. Similar to the combinatorics of component selection in configuration problems, the lack of employing taxonomies for component aggregation natural to the domain affects search efficiency in two ways: [Mittal 87]

1. Flattening the components causes the search space to grow exponentially.
2. Flattening introduces constraints and interactions between subcomponents which could have been compiled away, increasing the time complexity at least linearly.

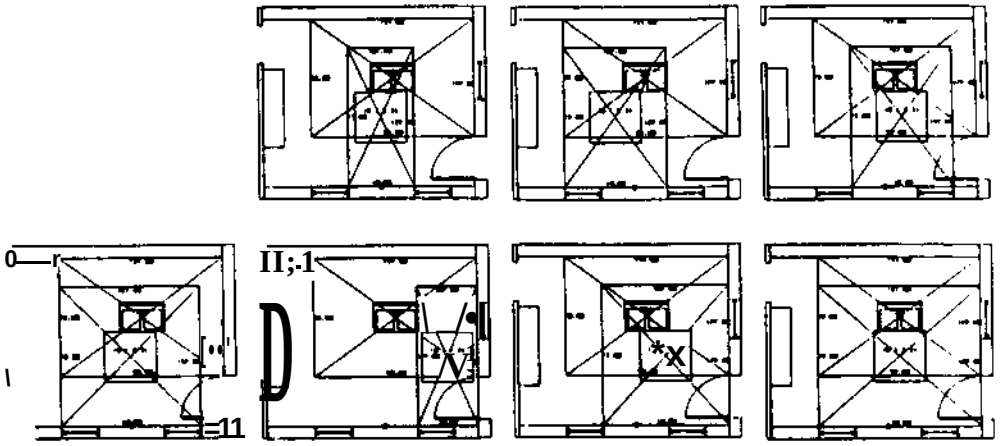
For a simple direct example of this 'flattening effect' parts a), b), and c) of Figure 4-1 show a problem with the current LOOS system having only one "flat" level of abstraction in representing its object space. Each grouping in the figure shows a generation sequence produced because of the articulation of the window wall at the bottom. In the current system each wall segment and window in the wall is represented by a separate node in the graph of the configuration. This is necessary because there currently exists no other mechanism for attaching and accessing the appropriate information and tests for constraints, such as: "the sink should be next to a window if possible." However, because of the formalism used for generation, a number of alternative configurations are generated beside the segmented wall at each generation level (with the insertion of each design object). Most of these alternatives are extraneous because the range of locations for the object placed are all subsumed by one particular placement which includes the ranges of the other placement alternatives within its limits. These must subsequently be eliminated by a 'redundancy test', after generation. This effect is multiplied throughout the layout process and may involve any adjacent group of objects. Obviously, the cost of this processing could be reduced if the appropriate alternatives were generated and the redundant ones suppressed (never generated). The representation of the wall segment, or any other appropriate aggregation of objects, as a single "abstract object" attributed to a single node in the graph will permit this.

Conclusion about current limitations.

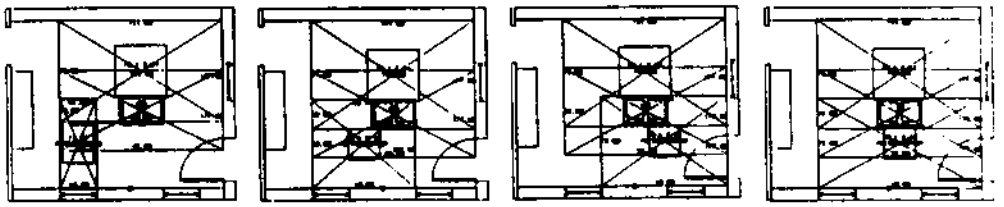
It is apparent that the most serious fundamental limitation of the current LOOS system is the combinatorics in the search space and question of the scalability of the approach for problems involving larger numbers of pieces. The next section shows just how serious the combinatorial growth of that space can be.



a) 1st level effect showing a sequence of generations of alternative placements for the first object • 'sink'. All are subsumed by the last and will be scored as 'redundant'.



b) 2nd. level effect showing 'redundant' alternative placements for the second objea. Interestingly, all of these will be scored and eventually rejeaed because of the sink's position. Nevertheless, die current system must generate them all.



c) 3rd. level efTea showing "redundant" alternative placements for the third object. This is only a sampling, there are 9 positions in all. Again, all will be rejected, but had to be generated first.

Figure 4-1: Shows the combinatoric cost of not suppressing the articulated details of a wall segment; the effect propagates down several levels.

4.1.1.5 The combinatorics of systematic enumeration

> [Hayes-Roth 83, p 92]If the search is exhaustive, the maximum size of the solution space depends on the time it takes to consider a single solution. A useful number to keep in mind for this maximum is ten factorial (10!). If 25 milliseconds are required to consider a solution, then 10! solutions can be considered sequentially during a full 24-hour day. This provides a practical upper limit for exhaustive search. The surprise is that the ceiling is so low.

Possible representations for generating designs in space planning, such as rectangular dissections and planar graphs result in numbers of alternatives that grow exponentially with the number of elements. For instance, Table 4-1 shows the number of alternative candidate structures (graphs or *orthogonal structures)*[46] with up to 10 internal vertices produced by the LOOS generator. Each orthogonal structure G_n represents a partial layout, each of which may be a candidate on the path to the solution in a layout problem involving *n* pieces.

n	number of **candidate structures** G_n	*n*\
1	1	1
2	**4**	2
3	36	6
4	576	24
5	13,920	120
6	462,240	720
7	19,847,520	5040
8	1,056,142,080	40,302
9	67,510,920,960	362,880
10	5,062,200,000,000	3,628,800

Table 4-1: Enumeration of alternative orthogonal structures in LOOS

Using the figures in the table for the number of structures produced with 10 internal vertices, divided by 10!, the number of days required to enumerate these structures at the speed conjectured above - 25 milliseconds per solution considered - would be 1,395,008, or ~ 3,922 years![47]

It is obvious that the combinatorics of the search space grows rapidly as *n* increases. The fundamental question raised by this observation is the 'scalability* of the approach. The LOOS system demonstrates a general technique applicable in multiple layout domains, and is, in principle, expendable to three dimensions. But can the technique be pushed to solve real problems with a large number of pieces?

[46] **Based on figures provided by Ulrich Flemming**

[47] **Of course, there would be no point in exhaustively enumerating them in a brute force manner using no knowledge or strategies for selecting or pruning alternatives since no tractable solution set would result The point is to show how the exponential explosion of the space will demand powerful search leveraging if convergence is to result, even with a lot of knowledge and strategies thrown in.**

4.1*1.6 **Directions for improving performance**

Additional heuristic search devices

I have already discussed the use of a heuristic in the system that makes the BNB search plunge toward a leaf. I believe that it is possible to 'hang' additional heuristics within the BNB framework in a similar fashion by adding additional slots [Lawler 66, Hall 71, Chakrabarti 86]. None of these slots would reflect a confinned judgment about the "goodness" of the partial design but they can serve to steer the search process in an effective way without ultimately excluding any alternatives. But enhancing the power of heuristic search in the system, while positive and worthwhile, is not likely to effect a meaningful reduction in the combinatorics of the search space, given exhaustive enumeration and real world problems of sufficient complexity. More powerful methods are necessary.

Parallel processing and G&T

The generation and evaluation cycle which is at the heart of the LOOS approach will be easily distributed since it is inherently parallel.[48] The decision process is extremely modular, therefore, each partial candidate solution in the expansion of a parent can be generated and evaluated independently with the exchange between processors of only a very small amount of information. It would be very interesting to pursue this for many reasons, but its advantage in leveraging the combinatorics of systematic enumeration would only be linear in the number of processors available in the distributed network. For example, if 100 processors were engaged, then the search could be at most be that much faster. This order of increase might make a significant qualitative difference in performance on a range of problems, and render some class of problems feasible which are out of range for the current system. However, in view of the combinatorics of Table 4-1 as *n* grows large, division of effort by a large constant (number of processors) will make no difference for sufficiently large problems.

Constraint-directed search

The following paragraph is paraphrased from [Hayes-Roth 83, p 89]. It suggests the possibility of moving the cleanly separated G&T approach of LOOS somewhat in the direction of constraint-directed search. DENDRAL reasons by elimination and is successful because pruning is incorporated early in the generate-and-test cycle. The key to the effective use of G&T is to prune classes of inconsistent candidates as early as possible. When such pruning rules exist, the solution space is said to be factorable. In principle it is possible to add more pruning rules to a generator so that only the consistent solutions remain. The rules needed to do this become increasingly complex and specialized, however. It then becomes difficult to prove the correctness of such rules (so that solutions will not be missed) and to ensure that the rules are faithfully represented in the program. Also a point of diminishing returns is reached when each new specialized rule covers a smaller number of cases, but adds to the cost of searching for the appropriate knowledge in the knowledge base.

I believe that the use of constraint-directed generation in LOOS should be limited to tightly controlled, well-defined, local subproblems where the knowledge is sufficient to guarantee that only a single solution (or perhaps a small number of alternative candidates) is possible. In other words, constrained generation should be considered as a viable auxiliary tool in layout design, but not the main proposer of alternative points in the search space. Otherwise, one of the greatest strengths of LOOS architecture - that the knowledge of good design Gayouts) in a domain can be accumulated incrementally in the style of a ***diagnostic expert system*** **- would be compromised. Its current *unconstrained generator* is constantly**

[48] **Tim Glavin originally suggested this possibility and he, Joe Peckne, and Robert Coyne assessed its relative value in leveraging search in LOOS.**

creating (developing) new configurations (in new contexts) for the expert designer to react to, thus providing a dynamic method of extracting that implicit knowledge of design which is so difficult to directly articulate.

Constraint rejected generation

My position on constrained-generation notwithstanding, certain classes of *negative rules* to restrict generation may be very cost effective. The practice of moving rules or "tests" from the position of evaluation after generation to that of constraining generation has been called "promotion of tests to preconditions" and has been studied by as least one researcher as a device for incorporating learning behavior into a system in order to improve its performance [Mostow 87]. Mostow calls this behavior "intelligent adaptive search" and it depends on finding properties that make a generated solution unacceptable and, in the future, avoiding proposing solutions with these properties. He has attempted to use "Explanation Based Generalization" in a system to explain its failures in order to get sufficient conditions for them.[49] These sufficient conditions, when negated, give necessary conditions for success which can be used to prune the search. The usefulness of such *negative heuristics* has been shown in the STRIPS blocks-world planning domain, where it was easier to avoid being *stupid* by using some negative heuristics than to try to be smart [Kibler 81]. Other systems have stressed the importance of negative heuristics, and how they can be learned: 1) statistically, through preprocessing of operators to discover conditions that can not simultaneously hold in any legal state [Siklossy 77], or 2) dynamically through adaptive search [Mostow 87]. In this regard, Mostow refines a G&T search procedure into a heuristic search by using monotonically necessary conditions to prune partial paths and by using monotonically sufficient conditions to reorder the search [Mostow 87].

Ideal preconditions have to be both infallible (applying an operator that satisfies such a precondition is guaranteed not to cause failure) and efficient (trivially infallible preconditions such as "this step lies on an acceptable solution path" are as expensive as blind search). A problem solver that has infallible preconditions for every operator can solve any problem without backtracking; although it may have to test various operator instantiations until it finds one that satisfies the precondition, and such a step is guaranteed to lead one step closer to the solution [Mostow 87]. In other words, classes of constraints may be "safe" to promote to preconditions on generation in the sense that they never prevent the problem-solver from reaching any correct solution, only from generating incorrect solutions. In general, further analysis of classes that are safe in this respect is needed (see the approach in [Mostow 87].

In LOOS, such safety becomes an easily gauged matter because of the monotonicity of the generation operators, and the possibility of rating a certain class of constraints as absolutely necessary (those satisfying physical reality such as dimensional fit and perhaps others required by code, etc.). In most problems, solutions which violate these classes of constraints will be seriously sub-optimal in comparison to other viable solutions so they can be pruned pre-emptively as "failures". Minton [Minton 88] also states that the notion of failure can be extended to certain search paths on the grounds of non-optimality, and that this is particularly true in a monotonic search space. Furthermore, the failure to meet constraints of the type described above is sufficiently serious to conclude that partial solutions involving them should not be preserved for "tradeoffs" even in the case where no other solutions are found. LOOS already employs, in a limited way, the use of negative preconditions on generation - what I call *"constraint rejected"* generation. The system already pretests for dimensional fit before generation. More generally,

---

[49]The failure of a search path is generalized into a "general condition for failure", that is, each such generalization gives a necessary condition on *legal* solutions.

we have experimented with a device to communicate other non-desired generation paths to the generator in a purely syntactic way. This device, called an "unbreakable arc", marks certain relations between two objects, or sets of objects with required adjacencies, so that the generator will not "break" that direct relation by insertion of another object. The semantics needed for concluding that the disallowed generation state should be marked can be confined to the knowledge-based evaluator. The generator need not know why the particular generation is *disallowed.*

The possibility of generalizing the means of communicating disallowed generation states in a purely syntactic way suggests a truly powerful way to begin to control combinatorics without compromising any of LOOS' strengths. However, the cost of testing for and maiking preconditions on generation operators could, in some cases be greater than the derived benefit. Related to this issue, Minton [Minton 88] has shown that efficiency is a fundamental issue in Explanation Based Learning, as it becomes counter-productive when so many macro-operators are learned that testing their preconditions costs more than rediscovering them by search (see also [Mostow 87]). Related to this, in planning for generation, is the fact that not all necessary conditions are equally important tests. Importance depends on both frequency of failure and the cost of failure, and a system that tests preconditions selectively should be sensitive to such factors [Mostow 87]. LOOS could also use a type of statistical failure analysis [Minton 88, Mostow 87] to identify conditions that lead to "failure" and these might be effectively promoted to preconditions dictating constraint-rejected generation. This is similar to the establishment of "NOGOODS" in systems utilizing dependency-directed backtracking [Stallman 77].

Imposing stages on layout development

A very crude but direct way to control the combinatorics of that LOOS has to contend with would be to limit the number of objects to be placed in each *stage* to a manageable number, and divide a problem into a suitable number of stages. At each stage the system could ask the user to choose from among the solution set a preferred solution to serve as the context for further developing the design at the next phase. This approach might work for some class of problems but is arbitrarily limiting of the power of systematic enumeratioa It detracts from the capability for exploration of tradeoffs and for leading a designer into parts of the search space that he otherwise might not imagine. On the other hand, limiting of the number of objects to be handled could be more powerfully achieved through another means. This would involve taking advantage of knowledge of hierarchies or clustering of objects in a layout domain, and coupling this with a new framework, in conjunction with LOOS, for representing and employing these more complex objects in the layout process

Interactive modes, learning, the use of prestored solutions, etc

The LOOS architecture supports a design system that can be used in a variety of ways, for instance as both a generative system or as a diagnostic, evaluative system. The interest and importance of using a design system in both problem-solving and diagnostic modes has been pointed out by other researchers [Mittal 86c, Oxman 87]; for instance, Mittal, et al's, PRIDE system was modified to accept an existing design and then test it Similarly, an existing design[50] in a domain for which LOOS has evaluation knowledge, is easily read-in and converted by the system into an orthogonal structure based on the spatial relations inferred from the arrangement. This configuration can be immediately submitted to the tester for evaluation and a listing of the flaws or constraints and criteria which it fails to meet. It would also be possible to remove a design object or objects deemed most responsible for the failure, and re-submit the reduced configuration, along with the object(s) to be inserted to the generative system.

[50] Which can be adequately represented as a composition of loosely packed arrangements of rectangles.

Additional alternatives or options to achieve the design goals would then be generated and it is possible that a better overall composition than the original would emerge.

Along these lines, it is easy to imagine how an interactive environment for using the LOOS system in multiple ways could be constructed. The device of removing a (critical) piece from the layout and re-inserting it in all possible ways may be related to *prototype adaptation* [Gero 88], and bears a strong relationship to the heuristic planning mechanism of *debugging an almost correct plan.* It is possible to imagine storing skeleton or prototypical layouts and using these as the basis for creating or completing a desired layout in terms of the current context. However, such a method is critically dependent on having the right representation for storing prototypical layouts as "canned or skeleton plans". It seems clear that ***these capabilities are dependent on resolving issues involved with the use of abstractions for*** *aggregations of objects.* As far as the actual storing and retrieval of an abstraction of a layout in terms of spatial relations, this could be done elegantly and quite compactly in LOOS. A compact listing with groups of values representing *generation states* would be stored. When coupled with an ordered list of the objects to be included in the layout it would efficiently reproduce the exact layout when given to the generative system by the correct controller.

Adding abstractions through hierarchical decomposition

I have noted that the current LOOS system includes levels of abstraction in its representation for space planning operations, and in its knowledge-base and control framework for testing candidate solutions. What it does not currently have is means of abstracting the objects of layout, or the *design-objects* that LOOS manipulates in performing layout. The design-objects are attributed to nodes in the graph-based representation employed by LOOS. If these objects would incorporate levels of abstraction (by aggregation or containment) then the granularity of the layout in process, as reflected in the cardinality of the graph which represents it, could be controlled and reduced in conjunction with these same levels of abstraction.

If one 'abstract' object can represent the composition of several 'subobjects' then using decompositional abstraction could counter in a direct syntactic way the severe tendency toward combinatorial explosion encountered in typical problems already engaged by the LOOS system. Furthermore, it appears that such *domain-object abstractions* would allow a transfer or translation of a great deal more of the domain knowledge of layout into the representation manipulated by the LOOS system. This would facilitate more flexibility and inferencing power for decision making and early pruning, and multiply the leverage already built-in to the LOOS approach through abstraction by levels.

Hierarchical decomposition promises to be a more powerful method for extending the LOOS approach. Combining it with automated layout revolves around additional ways of abstracting the search space and structuring an even more hierarchical design process within the basic architecture of the system. Fortunately, at least at a conceptual level, it appears not only to be an appropriate and timely addition to the system, but seems manifestly possible to accomplish in a relatively straightforward way.

4.1.1.7 Making the approach scalable

I have discussed the current limitations of the LOOS approach and identified the most crucial limitation as the combinatorial problem of generation. I have also discussed a number of possible extensions to the system which might bear on its generality, flexibility and efficiency. Of these, the only extensions that currently have promise in reducing the search load in a way that might control the combinatorial explosion are:

- A language for layout, allowing domain and problem specific knowledge to be directly and

easily added. Such a language would also make more feasible the extension of constraint-rejected search (or limited constraint-directed search in a specific context).

- Adding hierarchic levels (another form of abstraction), in the form of decomposition, in order to incorporate planning knowledge into the process of placing objects.

Both of these are necessary additions to the system if it is going to solve larger problems and solve a variety of problems. Though both of these directions are broad and encompass many issues in the layout process, the first deals more squarely with getting enough knowledge into the system, or *epistemological adequacy,* while the second deals with structuring knowledge for efficiency, or *heuristic adequacy.* The latter will motivate the research that I propose, while concurrently, the former will be the focus of another research effort.

In answer to the question of whether the LOOS approach is scalable, I believe that it is possible, and that addressing the question is the appropriate and relevant basis for the proposed research. Furthermore, if the proposed research is effective in extending the LOOS approach to handle larger two dimensional layout problems of realistic complexity, it also will serve as a necessary *conceptual base to support extending the total system to three dimensional placement problems in multiple domains.* Already built-in to the LOOS approach through its representation, is the strategy of *incremental refinement* through abstraction levels. I propose adding the strategy of *hierarchical refinement* through decomposition of the domain objects. Both of these are "divide and conquer" techniques. They also differ and complement each other in important ways which I plan to explore. I foresee the net result of combining these concepts to be research results of general interest in design, as well as an effective reduction of the search effort to find the "best" design alternatives in automated layout

4.2 Description of Research

In general terms, the proposed research will explore the effectiveness of combining different forms of abstraction to leverage search in the context of automated space planning - a design domain that must deal with the shape and geometry of objects. Specifically, to offset the potential combinatorial explosion in the LOOS approach, additional structuring of the problem solving process will be required, allowing more domain knowledge to be brought to bear. I propose to develop a generic mechanism to facilitate a very important aspect of this structuring - *a new abstraction framework to represent and process decompositions of the design objects of the layout domain.* This framework will mesh with the overall G&T architecture of the LOOS approach and will be domain-independent in structure, accepting domain-dependent decompositional knowledge as content

The decomposition hierarchy for a placement task will represent a high-level, knowledge-based "layout-plan" for achieving the tasks.[51] The overall task will be specified as a single top-level goal. The decomposition hierarchy will partition the layout-task into abstraction levels based on a taxonomy of objects suitable for achieving the task. In the proposed research, the decomposition hierarchies utilized to test the new abstraction framework will be hand-built and precompiled based on domain knowledge. The primary focus of the proposed research will not be the creation of useful decompositions but the structuring of their representation and use by the system.

[51] See [Chandrasekaran 88] for a discussion of why a decomposition hierarchy implicitly represents a "plan" for achieving a design task.

I assume at the outset that this decompositional approach will serve as an applicable and effective mechanism for scaling-up the LOOS paradigm of layout to realistic problems in a variety of domains, e.g., VLSI as well as building layout. Therefore the important characteristics of the decompositional abstraction mechanism that arc proposed must be generic to multiple layout domains. I propose to develop the mechanism as a general hierarchical extension to LOOS's domain-independent problem solving architecture. Decompositional abstractions will serve as high level pruning devices, as 'containers' for domain-dependent knowledge and methods,[52] and as a place to locate knowledge for locally *controlling* the problem-solving process and extending its complexity as required.

4.2.1 Principal objectives

The principal objectives of the research are:

- To add a form of planning knowledge to an automated layout system by building a new abstraction framework that provides substantial additional leverage in the form of search reduction.
    - by reducing representational complexity
    - by contributing to early discrimination and pruning
- To have this new capability merge seamlessly with the representational abstraction(s) and the overall system. It is essential to add capability to the system without compromising its overall strengths. In other words, I am not arguing for the creation of a wholly new system, only a new type of object that the current system can manipulate.[53].
- This decompositional abstraction framework is to be domain-independent in concept and structure, and domain-specific in content
- The framework should also provide the basis for concepts that would apply in extension of the approach to 3-D placement.

4.2.2 Leveraging synthesis in layout design

[Baykan 87]

> In this research, abstraction by omitting details and abstraction by aggregation are the two types of abstractions used in carrying out design. Abstraction by omitting details creates an operator hierarchy. Abstraction by aggregation creates an object hierarchy, where multiple objects at some level are combined into one object at the level above it There are natural levels of aggregation in space planning, such as buildings, rooms, equipment and furniture... Multiple abstractions with respect to design units result from abstraction by aggregation. With respect to the elements used, design can be carried out at the level of buildings, rooms, or arranging furniture and equipment within the rooms...The units at the one level guide the location of elements at the other.

In the section on planning strategies I pointed out that factoring a search space into abstraction levels by

---

[52]In this way the proposed extension also forms the basis for an *adaptive* approach by providing the expedient place to store additional knowledge and methods. This allows the possibility of good tools for solving subparts of layout problems (perhaps specialized to certain problem types or domains) to be readily absorbed into the architecture of an improved layout system based on the LOOS approach.

[53]Tong suggests that a good engineering principle in the design of design systems is to establish a functioning basic architecture for a problem-solving system, and then "... discover where it is heuristically and epistlemologically inadequate and modify it accordingly in a controlled minimal fashion." The modifications can usefully be driven by insights provided by human experts in the domain. [Tong 87]. This precisely describes the methodology that underlies extending the LOOS approach to use decompositional knowledge of layout.

focusing on a single concern at each level, or elimination of detail, will combine and interact in complex ways with the abstraction levels created via hierarchical decomposition of the problem. Later, in agreement with this, the general discussion of abstraction revealed that there are many kinds of mechanisms for cleaving a problem into abstraction levels, but that a crude division into the two major kinds is *generalization* vs. *aggregation;* see (Figure 3-3. There is no direct relation between these modes of abstraction - they are basically orthogonal methods of cutting through a space - but they are both techniques for reducing complexity and focusing. Also, they both involve, in different ways, the dropping of information to suppress detail. If handled carefully they may contribute levels of abstraction in problem solving that work in complimentary ways.

The previous section identified several existing mechanisms (related to focusing on a single concern or through class partitioning) which create abstraction levels in the LOOS system. It also pointed out the possibility of utilizing hierarchical decomposition to achieve additional levels of abstraction in the layout process in LOOS. The major question for this research is, what is the nature of the abstraction levels that decomposition can provide, and how can they be created so that they will they work in a cooperative way with the other abstraction mechanisms, especially the relational representation abstraction, of the LOOS approach.[54]

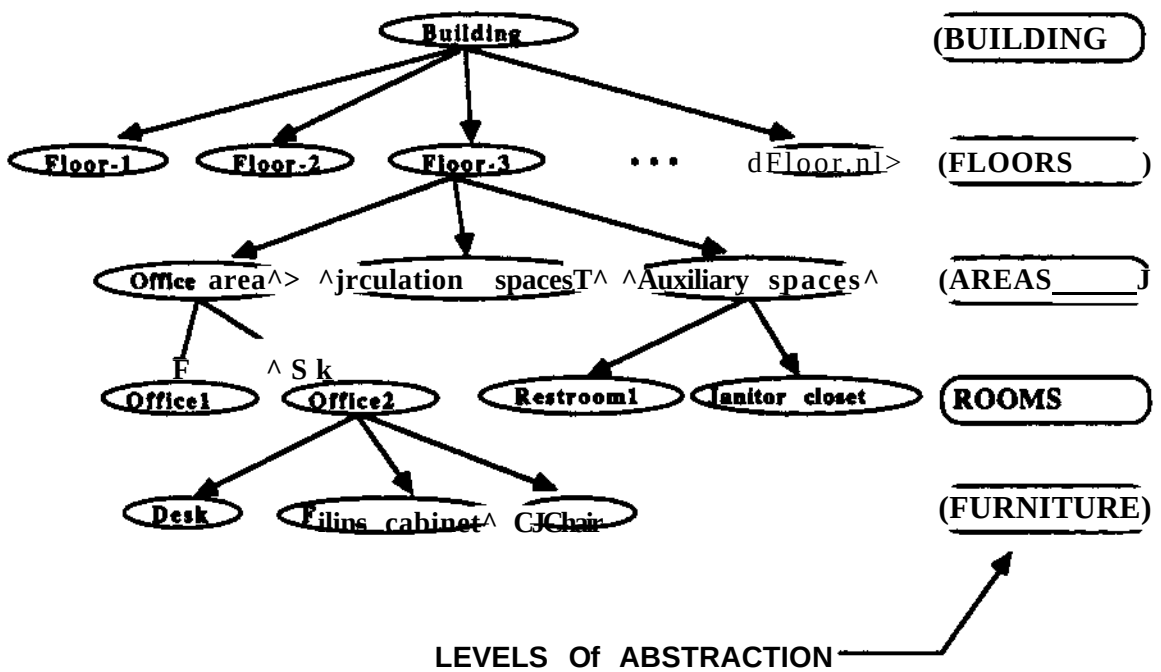


Figure 4-2: Typical levels of abstraction and decomposition for layout design

Figure 4-2 shows some decompositional levels that are typical in the design of an office building. Interestingly, the decomposition levels for layout in a typical architectural design problem such as this emerge naturally from the discipline of moving through "scales" or iterative design sequences at different

[54]Recall the conclusion of the section on abstraction in space planning that planning layout through hierarchical decomposition has not yet been successfully joined with an effective representational abstraction.

granularities.[55] Abstraction through "scales" would be characterized as on an orthogonal axis to abstraction by decomposition. But, in architectural layout, the levels of decompositional hierarchy emerge by moving down through the scales, and not by explicit decisions about aggregation and decomposition. Nevertheless, the decompositional hierarchy that emerges reflects useful clusters or aggregations that can be identified with the "objects"[56] of layout Therefore, decomposition through the objects of layout can be seen as smoothly integrated and compatible with the levels of abstraction created by moving through scales. And, significantly, both may be realized simultaneously through the hierarchical representation of the "objects" of layout.

This is extremely fortuitous, because it is easy the possibility for attributing objects of arbitrary complexity to the nodes in the graph in the LOOS representation for layout. Thus, dividing an overall layout task into subproblems within the primary representation can be accomplished directly through the semantics of the objects attached to the rectangles (nodes) that LOOS is manipulating at any given time. The focus and complexity of the layout task can managed and controlled through the decomposition of these objects, as if control were being manipulated directly. The same effect can be achieved as that of a top-down control regime associated with an abstraction by levels strategy such as moving from coarser to finer scale. But abstraction by decomposition through the objects of layout has the additional advantage of allowing for more flexible localized control that might allow some limited "bottom-up" processing to occur within the overall top-down control. For instance, this could happen "opportunistically" by pushing down a level of abstraction and dealing with the articulated components of layout object in order to retrieve infonnation or resolve an interaction as the higher level.[57] Furthermore, if the objects of layout are also viewed as "goals" then the knowledge for controlling their decomposition and placement can be stored with them as *domain specific planning knowledge for accomplishing the layout task).*

Many other researchers in space planning (see quote from Baykan above, and also [Tong 87]) have noted how goal or task decomposition in layout seems to have an obvious expression in aggregation abstractions and a taxonomy of domain objects. Object decomposition hierarchies in layout typically "nest", e.g. the task "Layout house" will involve the subtask "Layout kitchen" which will in turn involve the subtask "Layout sink". The nesting within the decomposition hierarchy may be arbitrarily deep. The nesting characteristic of the design-goals in layout implies that the system should have an overall recursive quality. This can be readily achieved in conjunction with the LOOS generative approach by simply keeping the representation and generation rules unchanged while attributing more complex, sophisticated objects/goals to the nodes or rectangles in the representation. Then, by decomposing the layout task through the hierarchic levels built into the objects, the same generative operators in the LOOS approach will be applicable to objects at all levels of abstraction.

I have come full circle in sketching a decompositional abstraction strategy that appears to meet the principal objectives of reducing representational complexity and adding more knowledge for early discrimination of alternatives in a layout task, while combining cooperatively with the LOOS approach. With this high-level sketch, what remains is to specify how to structure and represent these decomposition

---

[55]The relation of this discipline to top-down design, and the clustering of the tasks or objects of layout was noted in the section concluding that decomposition is a key strategy in layout.

[56]Note that the design levels shown in Figure 4-2 can be viewed alternatively as "goals" or objects, e.g., Layout areas, layout rooms, layout furniture, etc.

[57]This is explained further in a forthcoming section on control options.

abstractions. However, the representation of decomposition in space planning is difficult (and therefore requires research) for the same reasons that the overall problem of layout is difficult. It is harder than the decomposition tasks that involve merely the *selection* of components and their functional interrelation since it involves issues of geometric reasoning and locations of objects in space. Decompositions in space planning will have to accommodate component selection, help to facilitate (or at least not impede) the correct functional interrelation of components, all the while serving as a high level guidance (or pruning devices) for the *location* of the components. And, if a decomposition is not yet at the primitive object level, it will have to manage all of these requirements for objects interior to itself and in relation to objects exterior to itself simultaneously. The remainder of this section is a specification of the addition of such a decompositional planning framework in conjunction with the essential elements of the LOOS architecture.

4.2.3 Decomposition through "goal-objects"

I have noted that, in its essence planning, is "goal-directed". The formation of a goal-hierarchy based on knowledge of a domain is one of the most straightforward and powerful methods of leveraging problem-solving tasks in a domain. Therefore, the capability I propose to create will utilize a *goal-hierarchy* composed of tasks and subtasks based on aggregation of the objects of layout. To achieve this I will establish an equivalence between layout goals and abstracted aggregate objects of layout. Design-objects to be placed, at the highest level of abstraction, will be "goal-objects". In many respects they will be similar to the goals utilized in systems based on "design-plans" in that they will serve as "both a description of some part of the overall design, as well as a concept around which knowledge is organized." [Mittal 86c]. Associating goals with physical structure aggregation enables the knowledge of experience to be compiled in that it associates some of the design requirements to parts of the artifact [Chandrasekaran 88][58] It also provides a location for specialized control knowledge of how to configure independent parts of an artifact to achieve overall desired behavior.

A *goal* is defined in a standard dictionary as "Something toward which effort or movement is directed; an end or objective" [Funk 80], and in the "Dictionary of Artificial Intelligence" [Rosenberg 86]" as "the end to which problem solving aspires ... synonymous with task." Combining goal with object we get a **"goal-object" or GOB.** A gob is defined in the same standard dictionary as "a piece or lump as of a soft substance." The name may seem odd at first, but captures somewhat, the malleable quality of abstracted objects existing first as high-level goals or descriptions, and molded, constrained and otherwise processed into the geometry of an arrangement of dimensionable objects as they are placed into a layout context, as part of a given layout task.

4.2.3.1 The role of "goal-objects" in layout design

At the outset of this research I foresee that GOBs may have a multitude of roles to play in the guidance of an automated layout system.

High level abstractions for pruning

First and foremost, their main contribution to knowledge-based planning of the generation process must be - that placed by high-level abstractions in most problem solving processes - as coarse pruning devices in the search process by contributing to the representation of abstract candidate solutions. Their boundaries may be somewhat tentative at some phase but can still effectively serve to force the

[58] In this case the artifact is a layout design composed of objects.

recognition of 'bottlenecks' in the consumption of resources, especially the global resource of space, on alternative branches of the search space.

Moving information between abstraction levels

GOBs will allow the propagation of the effects of decisions made in one abstraction level to other levels. In this way they will 'span abstraction levels' in the sense of Tong's [Tong 87] goal-directed bottleneck recognition strategy. They can accomplish this to the degree that they inject constraints from their component parts (which are at lower levels of abstraction) into the context of the GOB, which is at higher level in the search space. This will help to guide, at a high level, the deferring or rejection of certain paths of the search space in favor of those the design system should choose for its more detailed phase.

Extending commitment to achieve earlier pruning decisions

As in the factoring of a search space by macro-operators in planning systems, these 'macro-objects'[59] may facilitate 'safe commitment' in terms of abstracted minimal properties(abstracted as a GOB), and early (and certain) pruning of vast amounts of the search space. Performing a comparison of the alternatives for a given abstracted-object (GOB) may yield "common requirements" (e.g. maximum dimensional range, or other least-common denominators of resource consumption) on which partial commitment and incremental definition can be made. This is analogous in certain respects with the use of a 'structured component hierarchy' in the COSSACK configuration system [Frayman 87] where components are abstracted in the superclasses of the hierarchy. This allows for a strategy of partial choice in terms of component selection:

> When a component choice has to be made, one way to process it is to add to the configuration a special object the "description" [sic] which is not a fully specified component with constraints describing differences between alternatives. Resolution of these constraints leads to full instantiation of the description.

GOBs will allow the strategy of partial commitment to be extended to component *placement* as well as selection.

Suppressing certain subgoal interactions

In terms of subgoal interactions, GOBs will have an effect similar to the forming of block-serializable subgoals (in Korf's) terms [Korf 87]. They will allow the finessing of subgoal interactions of their component parts, both internally and in relation to the other objects in the design space. This strategy has precedent in automated layout, see [Pfefferkorn 75]. It is generally important in any design dealing with functional and/or physical component aggregations [Chandrasekaran 88]: "In essence, components represent solutions to a set of constraints which can then be ignored because they are already implicit in the components". In this regard the notions of component aggregation and specialization hierarchies (or taxonomies) play a vital role in determining effective decompositions for particular domains [Frayman 87]. However, as noted in the background discussion, finding the ideal, or even practically useful, decompositions for a domain isn't always easy and may not be guaranteed.[60]

---

[59]In a certain sense an interesting analogy may be made between macro-operators(MACROPS) in planning and macro-objects("MACROBS") in space planning. Since in LOOS, the placement of macro-objects will involve recursive use of the same operators as at the primitive object level, their placement may be viewed as a *macro-operation*, which is a sequence of individual plan steps.

[60]See the discussion in [MITTAL87 " p 635"] on the difficulty of arriving at ideal decompositions, and consequently, the typical use of those 'compiled' through experience by domain experts.

42.32 The source of decomposition hierarchies

I will assume the existence of appropriate decompositions in a domain. Where they come from is an interesting and complex issue, but will not be the subject of this research except in an ad hoc way to create suitable test problems. The focus of the research is building a framework or shell that allows decompositions to be used as structured abstractions in conjunction with a generative layout system based on the LOOS approach.

Because the formation of appropriate decompositions is a difficult issue, many other planning and design systems which have employed decomposition have taken a similar position in assuming that appropriate decompositions are provided to the system. In many domains there may be a number of alternate decompositions at various points in the space (implying choices and possible backtracking) requiring search in a space of possible decompositions. This search could be very expensive and is often avoided by relying upon interaction with human experts to provide/choose decompositions: [Chandrasekaran 88]

> To avoid the search problem but to use the domain knowledge about decomposition, human-machine interaction between human experts and machine processing can be arranged so that the machine proposed [sic] alternative decompositions, and the human chooses the most plausible ones.

If the choice of effective decompositions during problem-solving is complex, then certainly automating the process of forming these decompositions (part of problem structuring) must involve even greater difficulty. Fortunately, in many domains, it is possible to rely on the example of stable decompositions representing the compiled knowledge from experimentation of domain experts [Chandrasekaran 88, p 3]. I believe that this is the case in many domains of layout planning, and GOBs which suppress subcomponent interaction to a useful degree may be formed along the lines of those used by human designers in those domains.

Some interesting alternate ways of forming decompositions exist, for example numeric and conceptual clustering approaches [Michalski 83]; such techniques have been used in prestructuring architectural layout problems [Sato 80] and VLSI layout [Khokhani 85]. These processes might serve as a useful source of aggregations which could be modeled as GOBs provide an effective basis for decomposing problems in some some layout domains. The formation of decompositions by such means may be used in some test problems in this research, and may prove to be worthwhile side issue. But again, such mechanisms underlying the formation of decompositions will not be a direct focus of the current research.

4.2.4 Development of an extended system: (ABLOOS)

> [Linden 88]...good reasons exist to look for a single wide-spectrum representation scheme that can represent intentions at all levels, from abstract goals to the lowest level of executable plans. A wide-spread representation encourages generality in the planning methods by allowing each method to be applied at any level of abstraction.

The principal concern of the proposed research will be the implementation of GOBs to support the *recursive, hierarchical*[61] decomposition and achievement of layout goals through the placement of the

---

[61] **The main emphasis will be to support and explore the utility of a *recursive* decomposition^ extension to the LOOS architecture. However, I am aware that in attempting problems in a variety of domains, specialized knowledge, subroutines, methods, etc. are likely to be encountered which may be required and/or much more efficient than the G&T approach of LOOS. GOBs, in their role as knowledge-based goal structures, will also provide a place to store and invoke such non-homogeneous methods when appropriate and effective. Similarly, the PRIDE [Mittal 86c] system elected to embed the subgoals (of goals) inside a method, making it possible, with the same protocol, to specify alternate sets of goals, or alternate ways(non-decompositional) of performing the goal. This is a robust organization for representing, experimenting and integrating different styles of performing design.**

objects. For instance, the layout task hierarchy shown in Figure 4-2 would involve the recursive application **of the LOOS** generative capability at four hierarchical levels of abstraction. This means that the design objects **for** layout, as typically attributed to the nodes of the graphs (orthogonal structures) which **LOOS generates** to solve layout problems, must be *statically indistinguishable* from goal-objects (GOBs) **at whatever level** of abstraction - from primitive objects to the highest level of the hierarchy. The characteristics **and leverage of the** new system will rest heavily on establishing a protocol for these composite objects which will satisfy their dual personality as "abstractions" which structure knowledge to help manage complexity, and as spatial objects fully capable of being attributed to LOOS's representation and manipulated by its generative system just as it manipulates primitive objects. Thus the proposed system will be referee to hereafter as **'Abstraction-Based LOOS' - or, ABLOOS,** with a nod to its dual lineage of planning and space planning.

4*2.4.1 System characteristics using GOBs

The main ideas behind the proposed ABLOOS system yield the following characterization of the system:

- That the layout process cycle of the system is triggered by goals in the form of GOBs and guided by their decomposition. A goal is activated when it is selected for placement by the ABLOOS controller.
- That it be recursive, that is, the same generation operators may apply to GOBs as to primitive level objects. Therefore, primitive level objects and composite objects, or GOBs, are statically indistinguishable to the system. GOBs are responsible for their own behavior and how they achieve this is hidden from the layout generative system (the former LOOS).
- That any shifts in control be handled implicitly by informing GOBs to keep the overall control framework simple. This implies that the knowledge sources for these shifts in control, reflecting concerns and decisions on different levels of abstraction, are the objects themselves and the operations occur as a result of "testing" or "querying" those objects. This will enable constraint propagation between levels of abstraction.
- That the goal-objects themselves contain the knowledge to facilitate each of the above. Therefore, knowledge for achieving a goal is stored with the goal itself.

4.2.4 JS **Control options: Combining planning with G&T**

[Simon 81p, 150]

> Alternatives are also open, in organizing the design process, as to how far development of possible subsystems will be carried before the over-all design should be carried before various components, or possible components, are developed. These alternatives of design are familiar to architects.

Decomposition, as a process that "proposes design choices" comes with its own set of control problems. This issue can be more or less complex depending on how it is handled, and requires knowledge in certain forms [Chandrasekaran 88]. As decompositions, GOBs will interact with the mechanism that proposes design choices (generation) in ABLOOS, and there are many questions and options involved with the control of **the** system. One of the major issues of control with decomposition is associated with possible storing of alternative decompositions (OR choices in an AND/OR reduction tree). Failure at a lower level can result in backtracking to a search point which amounts to a potentially combinatoric search in the space of decompositions. In this research, that issue will be avoided by having at most only two alternatives for a GOB at any level - a decomposition-plan, or a fallback to generation with all "primitive" components from that point.

Breadth first descent of layout in ABLOOS

Another major issue with decomposition is whether the design will proceed in a systematic way through

levels of the hierarchy, or opportunistically, such that different portions of the design may be decomposed and developed at multiple, simultaneous levels of abstraction. Considerable experience in layout design supports an overall control strategy of breadth-first, hierarchical decomposition combined with the advantages of some bottom-up opportunities.[62] This is a classical top-down strategy in which, at each successive stage, a complete solution at higher level of resolution of the details is produced. Conversely, *within* each stage, a bottom-up approach to a solution by solving a sequence of small sub-problems is utilized [Liggett 81]. Therefore, within the basically top-down control of generation cycles in ABLOOS, I propose to allow for a certain controlled amount of event-driven, bottom-up processing to occur [Chandrasekaran 88]:

> The default control process for investigating within a given design hierarchy is then *top-down*. While the control is top down, the actual sequence in which design problems are solved may occur in any combination of top down and bottom up flavor."

This mixture of a top-down framework, with localized descents into greater detail, or ascents into less detail by freezing a portion of the design, is modeled after an effective human strategy in architectural layout. This strategy is frequently expressed by the heuristic of carefully developing a design at a certain level of abstraction by focusing attention at that level. But occasionally attention is shifted "up a level" or "down a level" when the need for information or checking consistency requires it, or when it is possible to suppress information because decisions requiring it have been made. This heuristic allows for a certain amount of opportunistic processing, while at the same time bounding the extent to which it may occur. Attention will soon be returned to resolving the design at the current level within the overall framework of top-down development.

What this flexible control adds to the architecture of LOOS is the ability to dramatically reduce the search space combinatorics (by reducing the syntactic complexity of the graphs handled by the generator), while implementing a certain limited ability to span abstraction levels in order to resolve interactions. The shift in level of attention of the system would be accomplished by a combination of matching on problem feature and context knowledge (a query on a GOB) built into the representation of the objects, and a bit of declarative control knowledge passed to the object communicating whether there is enough importance to allow it to push down a level in development.

4.2.5 Research results: additional leveraging of search

If GOBs are developed in ABLOOS in the manner described above, they will enable the layout system to defer expenditure of computational resources in conducting search. They will contribute to the scalability and domain-independent mechanisms of the LOOS approach in the following ways:

1. They will reduce the representational complexity of the graphs which the system has to handle at any one time. They will do this by controlling the number of nodes present, while at the same time guaranteeing that enough knowledge will be brought to bear to ensure efficient high-level progress on the overall layout task.

2. They will factor the search space by means of representing decompositional 'planning' knowledge for the layout task at hand. GOBs will operationalize the planning of the layout task by:

    - Facilitating and guiding their own placement into the current context, while wherever possible remaining abstractions (of aggregations.) This key property provides the invaluable pruning power typical of hierarchical abstractions.

[62] Recall the example of Manheim's hierarchical sequential decision process, and again the discipline of scales.

- Providing for the location of their subcomponents in alternative ways. The most important strategy will be to remain 'abstractions' in the current context while facilitating the location of their components in a separate (sub)problem context of reduced complexity. Other alternatives are possible and will be discussed.

3. Provide for alternative component/subcomponent *selection* from a component library, e.g., in kitchen design there are three typical sizes for stoves(ranges): deluxe, standard, and apartment size.

4. Provide for many other interesting and potentially valuable types of behavior to be explored as the research develops.

4.2.5.1 Relation to other capabilities

Below is a brief elaboration of the potentially important relationship of these object-abstraction mechanisms to even more powerful ideas for intelligent behavior of an automated space planner, which could possibly be extensions of the proposed research. I expect that other desirable behavior of an automated layout system will be the focus of future research along the lines taking place in some other areas of design, e.g., reasoning by analogy, and learning. I believe that such explorations will also be supported by the creation of a general framework to represent and process the objects of layout in a hierarchical, goal-directed fashioa

Abstraction and the use of prototypes

[Simon 75, p 298]

> A second set of autonomous elements in most design programs are 'prefabricated' solutions to subproblems that arise repeatedly in different contexts. By using such assemblies as components in the design instead of synthesizing it form simpler elements, the program is able to operate at a more aggregate level and reduce its search effort... In conceptualizing a design program that makes use of prefabricated subproblem solutions and supplementary autonomous constraints, it is useful to distinguish between the program proper (the generators and test processes) and a memory in which the autonomous supplements are stared. Only a small subset of the latter may come into active use in the course of designing any single object They will be evoked by particular situations that arise in the course of the design, retrieved from memory and applied.

Another method of dealing with aggregations would involve their direct recognition as "prototypes" within the domain of the problem. There are many (probably difficult) issues regarding storing and matching on such prototypes based on problem features. One could envision the possibility of the system building up its library of prototypes based on its experience and history of problem runs - which amounts to learning. The abstractions (GOBs) that I aim to build would be able to serve as an important link (intermediating datastructure) between the LOOS problem solver (G&T) and a "memory" of prototypical solutions for layout problems. Therefore, I believe there is an important relationship between the representation of hierarchical abstractions in LOOS to *prototype instantiation, adaptation, creation;* see [Gero 88] for a discussion of the great usefulness of the concept of prototypes in design. The issues of handling prototypes in layout and finding a level of abstraction that is general enough to allow them to be stored but with adequate knowledge to enable their utility in specific problems seems to be a difficult and challenging issue in automated layout because layout prototypes would have to be instantiated into a unique geometry for each problem context

Related to the use of prototypes is the idea of the automated reuse of design plans (see [MOSTOW87] on the replay of a design sequences). This could easily be done in LOOS given a sufficiently well-specified context, an insertion sequence could be replayed by referring to a stored sequence of "states" passed to the

generator. In general macro-objects, or solutions to prototypical situations could be stored and regenerated in this very compact, precise and efficient way.

Abstraction, generalization, analogy and learning

> There is a basic fascination with learning processes, and the belief that they hold the key to intelligence. [Newell 82,p 30]
>
> "Sources of selectivity for search: feedback of information from the environment and search paths tried; previous experience, trying again paths that led to earlier solutions, or their analogues. [Simon 81]

A G&T problem solver obviously has the interesting possibility not only of saving effort by using prestored or prototypical solutions where possible, but of playing a role in creating, *learning* and storing a knowledge base of such useful solutions. In this regard a useful hierarchical and abstracted representation for the knowledge, objects and tasks of layout seems essential to higher level processes such as reasoning by analogy and learning. More recent work on the use of abstraction in planning has emphasized its relationship to the generalized representation of previous solutions [Tenenberg 86]. Tenenberg's use of abstraction is similar to the idea of ABSTRIPS, but whereas the main aim of ABSTRBPS was to reduce search by reasoning about the plan in decreasing levels of abstraction in order to *guide* planning at the more detailed levels, Tenenberg's primary motivation for using abstraction is so that the search for a solution to new problems can be improved by using solutions to old problems. The idea is to try matching abstract plans to problems that have the same goals, an approach which he believes is feasible *in any domains in which objects are distinguishable at various levels of detail.* This is yet another indication of the relation between abstractions and learning, and the retrieval of abstractions to enhance the performance of a program.

The hierarchical representation that I propose to build for the tasks of layout will enable the "objects" of layout to be distinguished and employed at "various levels of detail". As noted in the brief discussion of learning in the background section, in "Case-based planning" the organization of experience is paramount in formulating new plans and debugging old ones [Dean 88]. Competing approaches differ on how experience is stored in memory and how experiences are retrieved and exploited during planning. Given that storage and retrieval are so important, most case-based approaches involve some theory of learning and memory organization. The hierarchical framework that I am proposing to build could be central to the storage and retrieval of knowledge about the goals of a layout task, and interlinked with the central difficulties of analogical problem solving: retrieval from memory of analogous episodes upon encountering a problem , the analogical mapping process in applying this information to a target problem,, and the relation to other, nonanalogical, ways to solve such problems.

While acknowledging the interesting relationship of my proposed goal-objects and the potential for learning and analogical reasoning, I agree that learning research in a domain is in general better postponed until the the major issues in obtaining a minimally effective performance program in that domain are understood and implemented. The detailed structures for problem solving that the performance program employs are likely to be largely the same ones that the learning program will require at a minimum. In this regard it has been noted that the major advances in the psychology of problem solving during the next several years will involve relationships between problem solving and learning. The analyses of the processes of transfer and training necessary to learning need to be developed with the same rigor and attention to detailed structure as have been applied to processes of problem solving. However, for the most part our knowledge of problem solving (especially design) is incomplete and some critical aspects of learning may have to await further explication of the knowledge and

structures (knowledge representation) involved in problem solving [Greeno 78]:

> On the other hand, at present we lack adequate understanding of how problem solving occurs when conditions permit it We do not understand the kinds and organization of the knowledge that persons have that enable them to solve problems, and it does not seem optimal to engage in detailed study of the mechanisms that are used to receive and store information during problem solving, until we have better knowledge about what that information is.

4.2.52 Summary: GOBs as a general reservoir for knowledge

In summary, in addition to their main role in factoring the search space, GOBs may serve as a reservoir for may kinds of knowledge useful in space planning, such as:

- assisting in the control of problem solving through focusing attention by 'negotiating' between adjacent levels of abstraction;
- facilitating 'partial choices' on objects from a component hierarchy;
- by serving as a place to store alternative decompositions, or other 'design methods' to achieve the goal they represent;
- to store or reference a library of prototypical layouts or layout knowledge for the goal they represent;
- to assist in learning processes in automated layout by closing the loop in terms of storage and retrieval of "generalized" layout solutions.

4.3 Plan of Research

So far I have motivated the concept of an extension to the LOOS architecture via its handling of the objects of layout. There are numerous complex issues involved in the specification of the behavior of those abstracted objects (GOBs). Also of interest is their interaction with the control of the layout process. The purpose of this section will be to present a technical overview of issues as a kind of map of the territory to be explored in the research. Certain aspects of the capabilities desired of GOBs are clear at the outset, and are also of fundamental importance as the main research focus. Therefore these can serve as a starting point for addressing and implementing the required behavior. Other aspects are less certain and perhaps more complex. Their importance and readability will emerge only as the research progresses and perspective is incrementally gained on the tradeoffs between leverage achieved vs. the clarity and complexity of mechanisms required.

The preference will be to push as far as possible mechanisms which are simplest, clearest and of highest generality. As in the research progress of the LOOS system, an exploratory style to development will be adopted with the appropriate level of commitment to intermediate versions of the proposed system. This is in accordance with the engineering of knowledge-based systems in general which require the intermixing of specification and implementation in a manner which has been defined as *Exploratory Programming,* or "a conscious intertwining of system design and implementation." [Sheil 83] as cited in [Nii 86].

4.3.1 Specification of decompositional abstractions

Each GOB, in addition to whatever complexity and knowledge it has as a goal, must have the minimum attributes needed to be viewed as a rectangular object which can be constrained by spatial relations as defined by LOOS, and placed into a configuration by the recursive rewrite rules of LOOS for inserting a new node in a graph. The dual nature of GOBs will require that they be able to satisfy the protocol and

behavior for each of their separate personalities — as layout goals, as layout objects.[63] At the same time there must be a fundamental equivalence between goals and objects to be placed in ABLOOS. The development of a type of abstraction to merge the required behavior of these two entities is the main focus of the proposed research. The following points will be important characteristics of the proposed abstractions (GOBs):

- A "design-goal" for the system will become a "design-object" of arbitrary complexity plus planning knowledge. Objects in the LOOS system already function as repositories of self-descriptive and acquired knowledge (e.g., constraints, such as direction of orientation).
- A "design-object" to be placed (even a primitive level object) will be at the same time a goal of arbitrary complexity.
- A goal-object, or GOB, has certain abstracted or approximate attributes to reflect the properties of the subcomponents (objects) it represents. A GOB will allow for and handle alternate views of itself: *opaque* in which all internal detail is suppressed ; *transparent,* in which the internal articulation and/or development of a GOB will be expressed to a certain degree. An important example of these alternate views is reflected in the property of dimensional bounds for a GOB:
  - viewed from outside it has a maximum size given by the range of the node to which it is attributed.
  - viewed from the inside it has a minimum size given by the minimum boundary required for the objects represented. This may be given explicitly by the subgraph that it represents (if it has been instantiated), or it may be given by declarative knowledge (which may initially, under least-commitment, default to be the smallest dimension of the smallest object it contains).

4J.1.1 Interesting technical issues

In achieving the combined goal-object behavior for GOBs, one technical complication arises immediately. ABLOOS, based on LOOS, accepts the limitation that the objects of design, attributed to nodes in the graphs generated, are rectangles placed parallel to the axes.[64] But one of the main achievements of the LOOS paradigm is the capability of modeling and generating loosely-packed arrangements of rectangles which are <u>not</u> required to have an overall rectangular boundary. There appears to be some contradiction in the fact that a node in a layout graph which is restricted to be a rectangular range, may be attributed by a GOB, which itself may encompass a loosely-packed arrangement of design-objects (or subGOBs). For recursive generality, these should be allowed to have a non-rectangular boundary (the loosely-packed aspect does not itself pose any problem). This will be an interesting issue to explore and I discuss it in further detail in a later section. In response to this, as well as many other issues that will no doubt arise, the protocol for layout goals as GOBs must be developed experimentally and incrementally as a major part of the proposed research.

---

[63] The minimum protocol required for layout objects to be attributed to the graphs of LOOS is reasonably well known and may be expected to continue to be developed along the lines of its realization in the first two versions of the LOOS system.

[64] As described above under the description of LOOS this assumption is necessary to the formulation of the theoretical foundation for the representation utilized

4.3.2 Gradual development of the new abstraction mechanism

The main role of GOBs in layout will be to control the number of active nodes in the layout process by focusing it at any given time on resolving layout at a certain level in the abstraction hierarchy. An example of that level of behavior is not shown, but it will reflect structuring of the layout process as shown in Figure 4-2. Before they will be able to support that full recursive, hierarchical decomposition role some of the basic properties of GOBs will have to be explored and developed. To the end of their gradual design and refinement, interesting technical issues involved in employing GOBs are illustrated by means of two examples that were solved by the current LOOS system. These examples demonstrate many of the basic issues and some of the leveraging effect of the capability that I am proposing to add. They also suggest other less central possibilities that may merit exploration.

43.2.1 A simple rectangular example using goal-objects

This section demonstrates layout generation incorporating a GOB abstraction at about the simplest level possible. It utilizes an example which has already been encoded on top of the current LOOS system in an ad hoc fashion as an initial exploration of processing advantages gained and behavior and mechanisms required. This is the first small step of the many toward the ABLOOS system.

a) b)

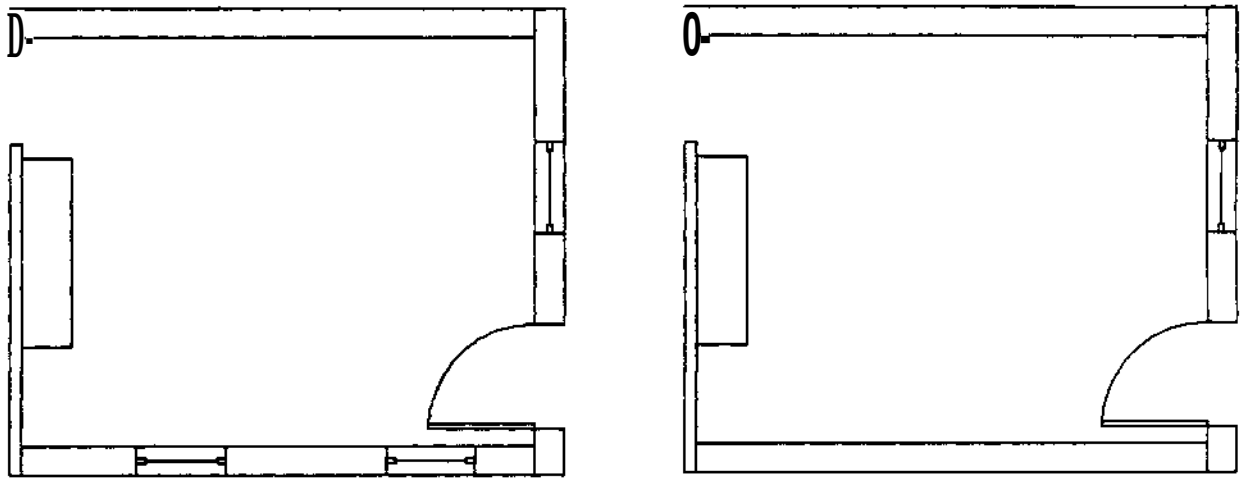


Figure 4-3: Kitchen context with bottom wall represented: a) as primitive objects(wall pieces and windows); b) as an abstracted 'wall-segment*.

Figure 4-3a) shows one of the kitchen problem contexts[65] on which the LOOS system was developed and tested. The bottom wall in that context is represented by five separate nodes in the underlying graph. As shown in Figure 4-1, the system is forced to do extraneous work in eliminating redundancies caused by the granularity in the representation. In Figure 4-3b) the bottom wall of the context is replaced with a single node attributed by an abstracted wall-segment. Figure 4-4 shows the four solutions[66] generated by

[65]**The problem description is supplied by the user as an arrangement of rectangles and this is automatically converted into an orthogonal structure (underlying graph representation) by the LOOS preprocessor. The initial orthogonal structure (which may be empty initially) serves as the context for layout to proceed.**

[66]**One of which should be eliminated by our door clearance rule, but is not due to an unsolved conflict resolution problem related to order of testing and re-testing.**

using the context in Figure 4-3a). The system generates and tests a total of 166 configurations to arrive at these four solutions. By using the context of Figure 4-3b) the same four solutions are generated after only 69 configurations are generated and tested! The savings of effort is dramatic, but of course must be bought by the required elaboration of the abstracted object 'wall-segment' so that its behavior invisibly provides the same level of information for queries and tests as was provided by the individual wall and window nodes that it subsumes.

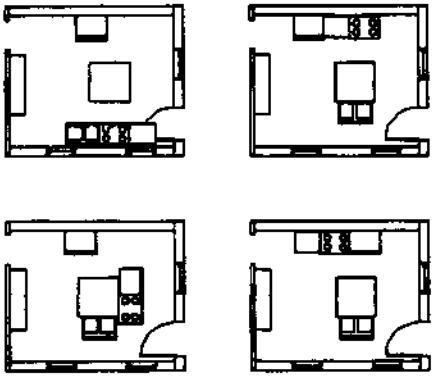


Figure 4-4: Solutions to Kitchen contexts in Figures 4-3a), b)

The requirements for the behavior of the 'wall-segment' as an abstracted object in the kitchen layout problem bring out several interesting issues immediately. These issues begin to establish the basic specification and protocol for a GOB capable of representing the abstracted object and providing the required behavior.

Reduction of the complexity of the graph representing a given context

This example shows clearly how the combinatorics of generation are greatly reduced by placing the knowledge of and responsibility for a group of objects in a single higher level object Though this reduction in *proposed points* in the search space from 166 to 69 is dramatic it is not an order of magnitude effect This may lead to initial questioning of the significance of the mechanism for sufficiently large combinatorial spaces. But I must point out that the effect is multiplicative by the number of abstracted nodes in a given configuration. For instance, in Figure 4-3a) the right wall, containing a window, door and wall pieces could also be replaced by a suitable abstracted wall-segment which would contribute an additional reduction of effort; the actual reduction is unknown but would likely bring the total reduction from the original 166 configurations to nearly an order of magnitude. Furthermore, this example shows only in a limited sense the 'goal* behavior of the abstracted object because it is already located as part of the context given for the problem. Another example that I will illustrate shortly will show the kind of leverage that a GOB may buy when it introduces real *strategic planning* knowledge into the layout process. This planning knowledge will replace significant generative effort in a well-known layout context with precompiled knowledge of the few possible solution alternatives. In that case, as well as the current 'wall-segment' example, the planning knowledge is that of composition rather than decomposition, but these are two sides of the same coin in terms of the utility of the abstractions proposed. For this test case, the composition of objects represented by the wall-segment were precompiled, but the possible automatic composition of groups of elements is discussed later.

Handling interior and exterior constraints and queries on GOBs

The following quote discusses the incremental definition of a component-subcomponent structure for a design through the interleaving of functional requirements at adjacent levels of abstraction. The ideas can be generalized to spatial as well as functional requirements and are related to the question of how the communication of a GOB with its interior and exterior environments can be handled: [Mitchell 85]

> The design philosophy...follows a least commitment strategy in defining the interface between submodules in the design. When a decision is made to implement a module in terms of certain submodules, these submodules may impose requirements on the interface of the parent module...These required constraints on the interface further constrain the possible implementations of neighboring modules. Furthermore, as choices are made regarding how to implement the submodules, the top-level module interface will become still more fully specified. In this way, the interface between modules is defined incrementally, interleaved with and driven by decisions regarding how to implement the submodules.

In terms of GOBs this raises an interesting technical problem. Goal objects, which are non-primitive, represent *clusters* of objects. The performance criteria for the clusters has to arise (automatically) from criteria for its component pieces; [Kundu 88] and [Galle 86] also point out this issue. It requires coordination of the design variables (alternative spatial relations allocated) between clusters as a whole and the design variables between their subcomponents. This raises an interesting 'chicken or egg' type of ordering question: What takes priority, the development of the interior of a GOB (in general involving alternatives) or the development of the exterior context of which it is a part? Information from either direction may afford earlier discrimination on alternatives from the other direction.

The quote above seems to suggest a least-commitment, constraint-posting/satisfaction approach from the inside out, or from the lower level of abstraction to the higher. This may be in keeping with an overall constraint-directed search approach to design, appropriate to some domains. However, in regard to space planning, I have already noted how the LOOS paradigm differs by taking an exhaustive G&T approach combined with a least-commitment-of-resources strategy for efficiency. As a first pass, I believe this attitude should be carried over to usage of GOBs as efficient, search-reducing, pruning devices with their interiors developed only to a sufficient level to answer the most significant higher level queries;[67] that is, queries at the level of feasibility constraints, such as dimensional, or other types that may be discovered, as noted in the discussion of extensions to LOOS. The hope is that since these GOBs represent significant aggregate-objects or clusters in the domain (otherwise why were they created?), they possess enough knowledge at their abstract level to assist in discrimination as layout proceeds in a largely top-down, breadth first manner. This implies, as much as possible, solving the outside problem without solving the inside problem. It reflects ordering the problems to *pay minimum cost* with the realization that at the outside level of abstraction most alternatives involving a GOB will ultimately be thrown away. Therefore it would not have been efficient to pay the cost of development of their interiors in terms of that particular outside context.

With these general principles in mind, I recognize that some projection or communication of constraints will be required between levels of abstraction, if only to take action or post data regarding feasibility constraints. Exactly how this is to be realized, or how difficult it will prove to be in varying domains, is not known at present, but some suggestions follow.

---

[67]Insights from Woodbury's work [Woodbury 87] on reasoning about the partial and incremental definition of geometric objects (lazy evaluation), and how to efficiently handle information processing (storage or calculation) may be applicable here to GOBs. Their properties (abstracted) must be incrementally developed, but they must also answer queries on demand.

Designing GOBs to yield desired behavior

1. Generally the abstraction of an aggregation's attributes will involve a transformation function, mapping knowledge that the aggregation has in terms of its components, to a set of attributes or evaluations to be performed on the aggregation itself (abstraction); these arc in some sense a generalization or weakest preconditions **for** the aggregate object At the very least, one can know dimensional bounds on the maximum size of the object from the outside, and on its minimum size from the inside. As another example, one may apply a more abstract notion of adjacency. For instance, at the most general level this could involve a trivial elimination. An example of such this is: if an adjacency to a certain type of object is requested of the GOB, and that object is not a member of the GOB, then there is no way the adjacency could be satisfied.

Figure 4-5a) illustrates a partial, pseudo-language listing of the implementation of the 'wall-segment' as a sort of frame/object with slots for: structure, min. and max. size, component-list (for trivial elimination on certain queries, etc.). Figure 4-5b) shows how this object will have multiple inheritance of properties to allow it to have the behavior of a goal-object, bounding-object, physical-object, wall-segment-object, etc. Through these 'type-of' chains, its behavior will be efficiently specialized.

2. In some cases, the interior articulation(s) of a GOB may be prestored solutions, or may have already been dynamically created (see below), or the query/evaluation from the outside may carry enough priority (a feasibility concern) to justify sufficient commitment of resources to force generating and traversing the subgraph of a GOB to the level required. In these cases the structure or subgraph (or alternative subgraphs), for a GOB will raise issues such as: When does a GOB have a subgraph? Can it have a partial subgraph? How are subgraphs stored and retrieved ? How are they oriented in relation to the outside context of the GOB? All of these are at least interesting technical issues that must be addressed by the research in order to provide utilities for the behavior of GOBs in the layout process; some of these questions could easily grow to a greater significance than technicalities.

Figure 4-5a) shows a 'structure' slot in an instance of a GOB where a list of pointers to alternative orthogonal structures for its interior arrangement may be stored. Figure 4-6a) shows part of the graph defining the spatial relationships of the abstract wall-segment GOB to the kitchen context of which it is a part Since it is fixed in position its orientation is known and defined in terms of the cardinal directions filled by the single distinguished node *E*. Figure 4-6b) shows the internal graph that would be stored for the wall-segment GOB. Just as a primitive object in LOOS, its orientation at any given time in the development of the layout may be completely undetermined, partially determined, or completely determined. If it is determined then the absolute directions of its internal arcs can be computed through its relative exterior interface *E*. Likewise, the actual arcs that would exist if the internal graph were to be intersticed into the outside context would be determined by graph operations, guided by the arcs that connect to *E* from the interior, and from *E* to objects in the outside context

43.2~2 A non-rectangular example using goal-objects

In this section the potential planning power inherent in the hierarchical approach is illustrated by way of a more complex example. Here the objects to be abstracted are the design-objects which must be placed and the focus is on what is to be gained by their potential modeling as a GOB.

Generating the solution for the layout of the service core of a high-rise office building using the LOOS system involves an elevator banking problem and requires the G & T of 118 configurations in order to arrive at the two solutions with a 3 - 2 split of the elevators on either side of the lobby. Figure 4-7 shows these solutions. Figure 4-8 a) and b) show how the current system laboriously generates the two solutions

a)

```
(defclass  wall-segment
             (goal-obj
              bounding-obj)
    ((structure    '( . . . )
      (component-list  '(wall,  window...)
      (min-size-x'     _ )
      (max-size-x'     _ )
      (min-size-y'     _ )
      (max-size-y'     _ )
           .
           .
           .
    )
)
```

b)

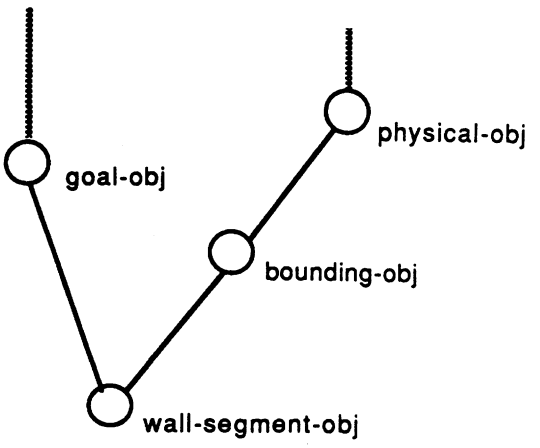


Figure 4-5: Illustrating ideas about the properties and specification of GOBs

by placing the lobby and each of the five elevators as primitive objects, one at a time. Figure 4-8 b) also shows another incarnation of possible spurious generations beside an articulated sequence of elevators which should be abstracted into a block. Additional objects may all be tried in numerous redundant positions, with a unnecessary cost of generation proportional to that found in Figure 4-1. Producing the solution set of Figure 4-7 a GOB(s) would be much more direct and immediate with tremendous savings of effort.

Structuring the layout process through the decompositional 'planning' knowledge implicit in GOBs
It is known that in the layout of building service cores there are generally only two ways of placing the elevators adjacent to the lobby: all on one side, or on both sides with as even a distribution for each side as possible (e.g., if the total number of elevators is an odd number, then one side will be assigned one more elevator than the other.) Therefore, aggregating the objects in the elevator bank shown in Figure 4-7 - a lobby and two groups of elevators on either side - into a abstracted cluster represented by a single GOB as a single GOB would be a very concise way of structuring the problem. In the problem shown in Figure 4-7 it is not possible to fit all 5 elevators on one side, so it would be very easy for it to be 'known' at the encompassing GOB that only two solutions are possible, those shown in Figure 4-7. These two solutions could be placed directly in the queue of partial solutions in the overall core layout problem. Just as in the current LOOS system, they would serve as the starting place for the continuation of the layout process, but at a reduction of effort from 118 configurations generated to the cost of retrieval of the prestored alternatives.

Before discussing further the possibilities for strategic planning behavior implicit in GOBs, I note an important related issue. The aggregation of objects that make up an appropriate GOB for representing the solutions of Figure 4-7 do not have an overall rectangular boundary. Furthermore, it becomes important in the further development of this layout problem to allow the insertion of additional objects adjacent to the lobby, thus requiring that they 'overlap' the corner of a GOB abstraction that represents the combined aggregation of lobby and elevators; see Figure 4-9. This case illustrates a very important general issue in creating abstraction levels in space planners that can work in conjunction with the primary representation (graph).

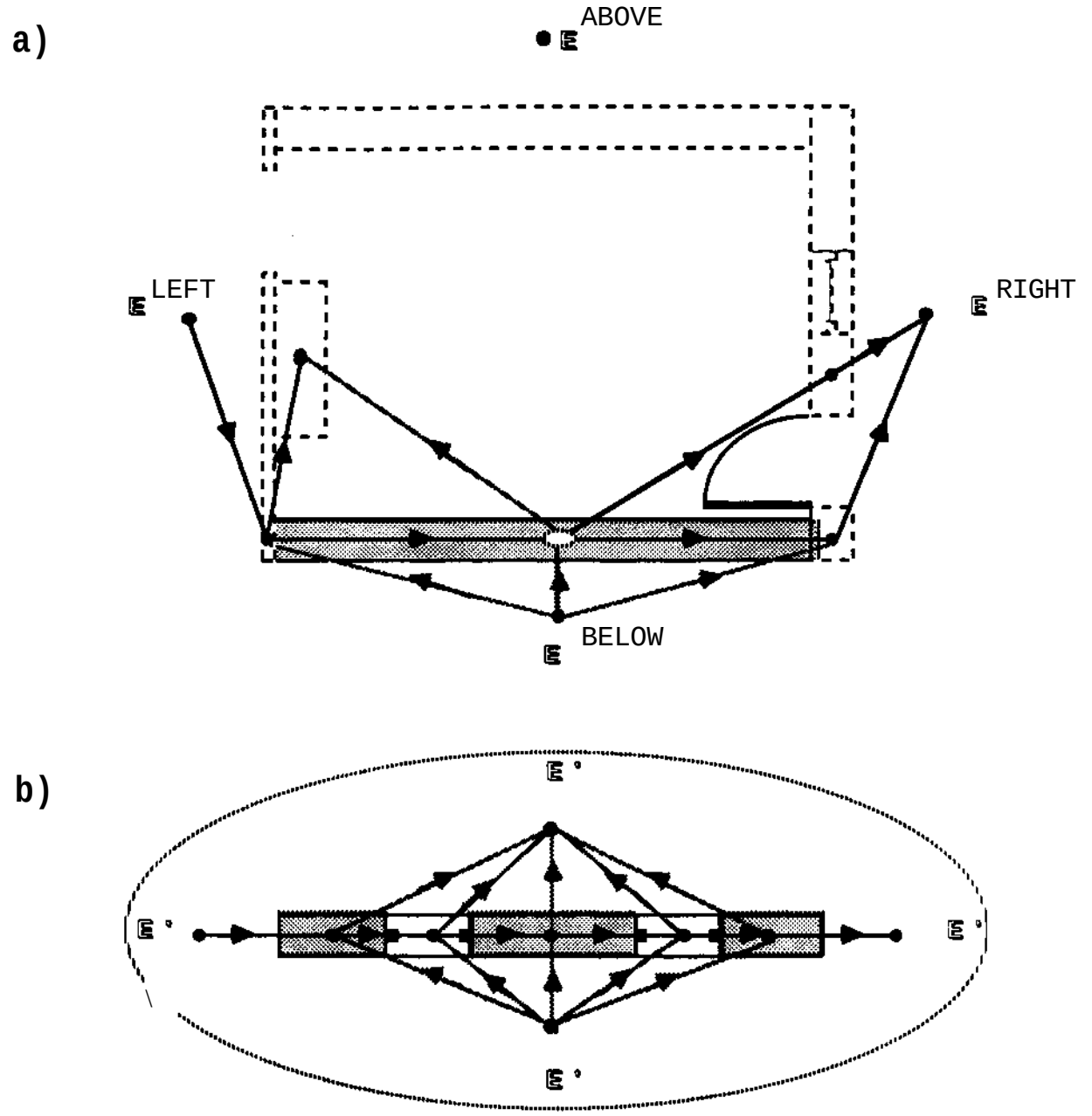


Figure 4-6: Example of a GOB used in context, and its internal arrangement

What if a GOB is not a rectangle?

Kundu recognizes and describes the same issue with reference to his own particular representation, but his meaning is easily understood in the general sense: [Kundu 88]

> The subregions in a r*-plan are a specialized form of clusters, namely, that have a rectangular boundary. A general theory that includes non-rectangular subiegions as an abstraction unit (cluster) seems a difficult task... More generally, one may allow merging two subregions if they share a portion of their boundary. In any case, some control must be exercised in such merging operation (sic) or else the subspaces of the abstraction will soon cease to have any meaning.

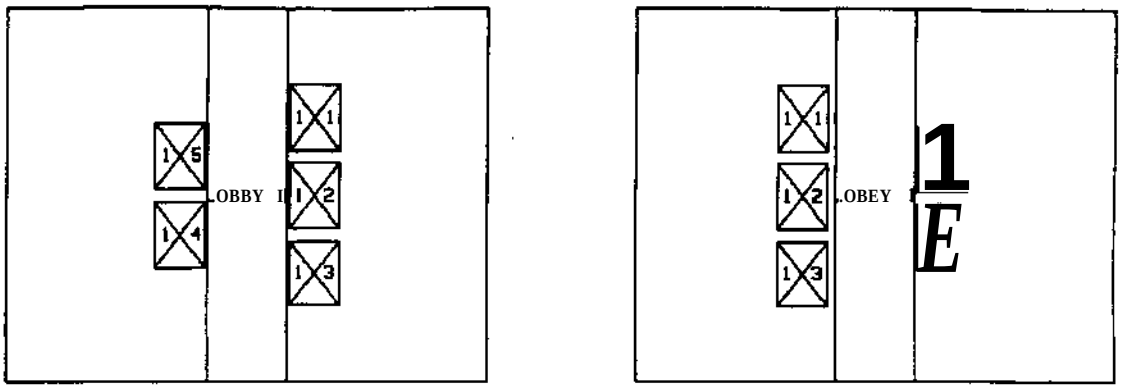


Figure 4-7: First phase solutions to a building-core layout problem using LOOS.

a) b)

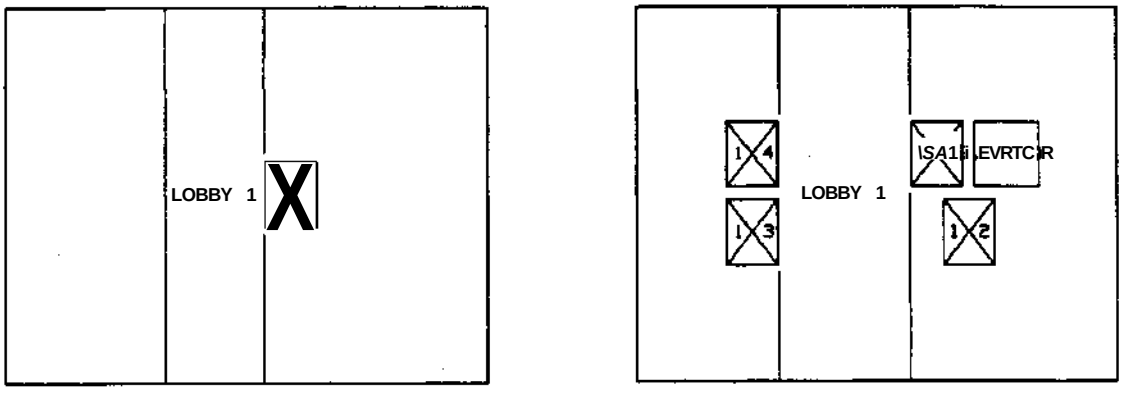


Figure 4-8: Developing solutions with all primitive objects

Starting the core layout problem with a high level GOB evokes a number of issues which may have to be dealt with generally. The issue of the possible non-rectangularity and needed 'overlap' or 'merging' of GOBs with one another seems unavoidable, so I suggest some alternative ways of handling this:

- *Only allow GOBs which are true rectangles.* This will cover a certain percentage of cases and still buy great leverage, as exemplified by the kitchen example shown in 4-3. However it is too restrictive of the application of the power of abstraction which can be represented by GOBs in space planning.
- ***Handle non-rectangularity by allowing for overlap with dimensional constraints stored in a GOB.*** This could possibly be generalized to the idea of GOBs having 'features' (comer cut-outs, slots, etc.)
- ***Handle non-rectangularity by decomposing and modeling a GOB by more components if required(not*** necessarily primitive objects but subGOBs). This method seems to be more in keeping with the capability of LOOS for modeling loosely packed arrangements of rectangles and would be the preferred alternative at the outset, until more experience and knowledge is gained. However, this strategy involves other issues beyond dealing with rectangularity which are continued in the outer listing.
- ***Other alternatives***

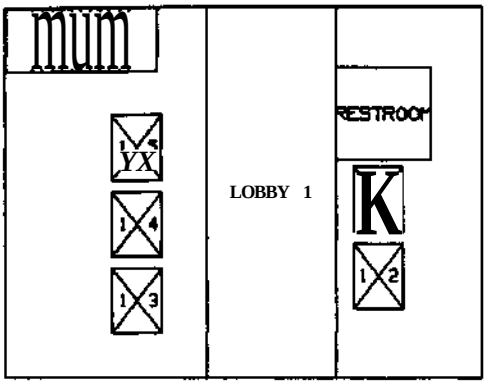


Figure 4-9: The "restroom" would overlap the bounding box of a GOB representing the "lobby and elevators" as a single aggregation.

43.23 Additional possibilities for GOBs

The above discussions brought out those issues about the behavior GOBs that are required to fulfill the minimum requirements of abstract planning objects in the layout process. Below are listed some additional possibilities involving the behavior of GOBs which may be explored in the research if time allows and they evolve as fruitful directions. A warning is in order This section describes possibilities that should be viewed as more speculative and subject to drastic change or cancellation upon experimentation.

Dynamic creation of an abstraction: when a group becomes a GOB

In the example presented in Figure 4-6, the 'wall-segment' abstraction was hand built and read in as part of the input file for the problem. Obviously, the burden of abstracting that wall should not be placed on the user of the system, but should be automated in the preprocessor. To facilitate this, certain types of objects must have the property that if they are recognizable as an aggregation, they will be automatically aggregated to reduce the complexity of subsequent generation.[68] This seems relatively straightforward if the aggregation forms a densely packed rectangle, such as the 'wall-segment'. However, automatic abstraction of loosely-packed, non-rectangular clusters into a single aggregate unit opens up a number of research issues. In any case, even with the simpler rectangular restriction, the generalization of the concept of automatic abstraction suggests that during the course of the generation cycle (not restricted to just the pre-processing phase, as in this example) certain types of objects may be synthesized into higher level GOBs in a bottom up fashion.

This will cause the system to alter the graph on that particular branch of the search tree by reducing a sequence of vertices whose objects share an important and consistent semantic relationship into a single vertex attributed by a GOB. As discussed in reference to Figures 4-5 and 4-6, the GOB will have as part of its semantic content, a representation for the subgraph consisting of the formerly articulated components. (This relationship may be recursive as GOBs may contain other GOBs as "primitives" from their point of view.) The value of such a capability would be the dynamic reduction of the complexity of

[6g] **This may be related to the potential for using clustering algorithms in the formation of groups or compositions of objects, as suggested in 4.2.3.2.**

the graph representing a developing candidate solution. One caveat is that this will increase the complexity of bookkeeping required for keeping track of the level of development of different branches of the solution space. In LOOS, the level of development or number of objects inserted was simply equated with the depth in the search space or the number of nodes in the graph. As always, the value of a strategy has to be weighed carefully against any resultant increased complexity in the overall system and processing cost of the mechanisms which realize it.

Synthesizing from an abstraction: when a GOB becomes a graph

As discussed, the overriding strategy in employing GOBs is to fully design the layout at the level of abstraction in which they represent individual (abstracted) nodes, apart from their own development as layout subproblems at a lower level of abstraction. However, it is possible to envisage other behavior and control regimes which might prove necessary and efficient in certain domains or for specific types of abstracted aggregations of objects. If so, then during the course of the generation cycle, certain GOBs may be instantiated into their detailed components (It is a question whether this would be done only temporarily to answer a query, or permanently). The subgraph representing the articulated components for a GOB may be developed (the system will push down to a lower level generation cycle and develop alternatives).

Or if completed alternatives exist, under certain conditions, the GOB vertex on the higher level may be "unpacked" and the subgraph, representing the best alternative layout for the GOB, will be intersticed into the higher level graph, thus adding all of the detailed components into the higher level partial solution. Under certain circumstances it may even prove to be efficient to do this for all the alternatives for the GOB, evaluate each at the higher level (by running through the tester), and pick the one to be retained. At this point the unpacked GOB could be repacked into a single vertex, or it could effectively have disappeared by dissolving into the articulation of its components into the higher level graph.

Mixing levels of abstraction in the context of layout design

Mixing of abstraction levels in the layout process could happen anytime as a result of the synthesizing or decomposing of GOBs. Also, the process may start with a mixture of objects from different levels of abstraction so long as none of more primitive objects is contained in any of the composite objects being placed in the same generation cycle. For example, the list of design objects for placement in a core may contain: restrooms, lobby, utility closet, and elevator banks or alternatively the elevators may be placed individually. As another example the list of design objects for placement in a kitchen with eating area may be: woik area, eating area; or alternatively - sink, refrigerator, stove, work counter, eating area; or another alternative - woik area, table, chair 1, chair 2, chair 3, etc.; or finally, both eating area units and work area units may be articulated as primitive objects as they are in the current LOOS system.

4.4 Environment and Implementation

Extending LOOS to create ABLOOS does not imply building on top of the current implementation, but it does imply accepting the core elements of the architecture. In this regard the generator, which required a considerable effort to implement, test and debug will be adopted essentially as a black box. The overall concept of a separate knowledge based tester will be accepted but its structure may change substantially based on the experience gained with previous versions and to accommodate the use of abstractions in the design objects (the data). Pre and post-processors will probably continue to be dictated by the requirements of the problem domains, with certain limited aspects remaining as a stable domain independent base. The addition of abstractions may be significant in regard to these components, but only in the sense of initializing the problem-solving with higher level pieces (abstractions) in the pre-

processor, or detailing such pieces in the post-processor. The controller will continue to work in terms of a basic BNB strategy, but may require substantially more elaboration in terms of levels and strategies to accommodate and make productive use of layout designing using hierarchical abstractions.

LOOS was coded in a multi-paradigm environment consisting of CommonLoops (a PCL version of CLOS [Keene 89]) on top of Common Lisp running on a workstation with graphics capabilities. This choice of environment was important to the implementation of the LOOS architecture.[69] The generator is algorithmic and creates highly structured graphs for which the built-in capabilities of Common Lisp for handling generalized sequences were extremely useful in saving programming effort. Also, incremental development of the knowledge-based evaluator was greatly assisted by the special properties - multiple inheritance and multi-methods - of the object-oriented extension CommonLoops. Multiple inheritance permits easy implementation of the merging of concepts to define new concepts(the multiple inheritance idea is shown abstractly in Figure 4-6 as a way of specifying a GOB.) Multi-methods permit the specialization of methods on the basis of class membership of more than one of the method arguments. In many object-oriented environments only the class of the first argument is significant Multi-methods simplify the creation of protocols for operations involving more than one object. A graphics capability is essential to the knowledge-acquisition loop because domain experts must be able to view and criticize developing layouts in a medium that is comprehensible and comfortable to them.

The implementation environment preferred for the development of ABLOOS is the Common Lisp/Common Loops combination on a workstation running Unix and XI1. This environment will provide a powerful and portable platform for ABLOOS, including whatever graphics are developed to support the research. It will also make readily available the core components of the LOOS approach to others who wish to use it in connection with other research efforts. I choose the same multi-paradigm programming environment, first, because I wish to have both LOOS and ABLOOS running in the same environment in order to bootstrap the development of ABLOOS by woildng on problems that LOOS currently runs (inefficiently); second, because I believe that it to be a superior environment(to rule base environments, for instance) for developing certain knowledge-based design systems My experience in building versions of the LOOS system in varying environments leads me to concur with the observations of some other researchers in this regard:

- The PRIDE paper handling system was designed by representing all of the required knowledge - goals, methods, constraints, adviseContext, advice, and calculations - as *objects*. Mittal, et al, state that they found there were advantages of object representations with messages(methods) over forcing all knowledge into the form of rules [Mittal 86c].
- Similarly, Tong suggests that in the mapping of the system design from the function level into the program level, multi-paradigm programming environments (LOOPS, KEE) are helpful, if not essential. They permit implementation decisions of best fit, e.g., "Which aspects of my problem-solving system should I implement in which paradigms?" [Tong 87].
- Frayman [Frayman 87]comments on the importance of environments providing a wide variety of representational primitives and the possibility of using intrinsic properties where possible in facilitating knowledge base maintenance and knowledge acquisition "...the benefits associated with hierarchical organizations, in particular, factoring information and representing commonparts only once in the system, providing indexing of knowledge for processing, and using the inheritance for providing defaults ... that can be used for newly defined classes of components....Functional hierarchies also facilitate making partial choices

[69]For a fuller discussion of the reasons for this implementation choice see [Hemming 88a].

...thereby eliminating unnecessary search."

4.5 Schedule of research

4.5.0.1 Sequence of tasks to be completed

The research will be accomplished successfully through incremental development of the new system which suggests the following sequence of tasks:

1. **Porting LOOS:** The core of the LOOS system is the generator, and the BNB controller. The first task is to get the core system running in the proposed environment - Common Lisp, PCL, and XI1. Also, it is probably worth translating one or more of the knowledge-bases of LOOS (residential kitchens and/or office building cores) into an updated PCL in the new environment This would allow the new system to be tested on some of the simple problems that LOOS already runs. Running the same problems would also allow a direct comparison of the efficiency of hierarchical system vs. the original in the number of candidates generated, and the solution set produced.

2. **Non-recursive use of GOBs:** Build a pilot version of ABLOOS, dealing with the level of problem discussed in the first example presented. This will get the basic structure and functionality of a GOB in place, and coordinate its working in conjunction with the overall system components, the generator, control strategy and a suitable tester to include test knowledge for the GOB(s) involved, e.g., the abstracted wall-segment in the example.

3. **Recursive use of GOBs:** Extend the functionality of GOBs to handle more difficult issues and behavior, especially control issues revolving around how to handle non-rectangular GOBs, and when to decompose and place GOBs into context.

4. **Test problems:** Complete the pilot version(l) by solving a class of new problems that LOOS cannot handle (problems and domains, other than the simple test domains of LOOS, are yet to be defined.)

5. **Analysis:** Appraise the results of version(l), and re-design and specify the changes needed for a final version(2) of ABLOOS (version(2) will not be implemented as part of this research, unless time permits.) The specification for the final version will integrate the experience gained in the research with the original intention to most clearly express the final results.

6. **Writing:** Report the results of the research in the dissertation.

4.5.0.2 Schedule of completion of tasks

A schedule for completing the proposed research is shown in Figure 4-10.

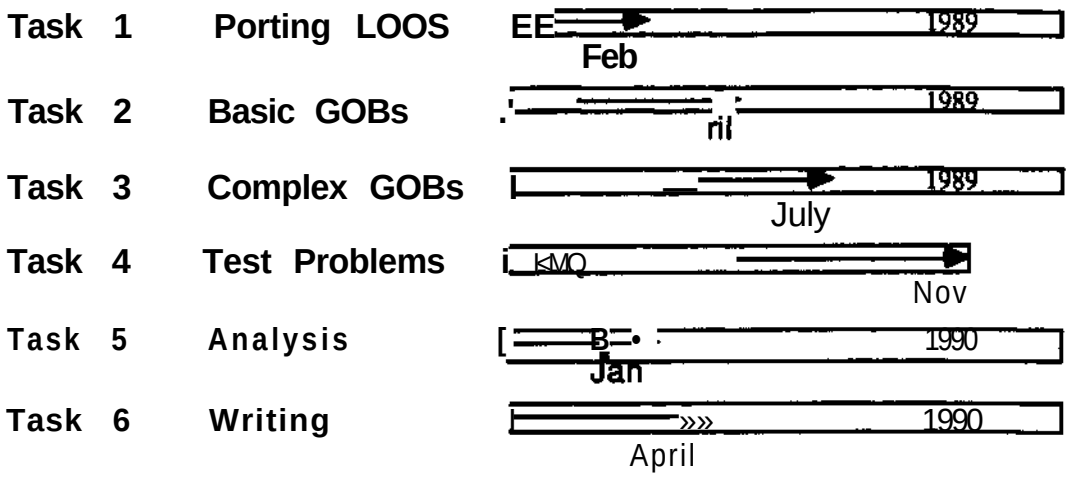


Figure 4-10: Schedule to accomplish research tasks defined in plan

5. Significance and Contribution

. The significance of the proposed thesis is that it is aimed at understanding a major issue of intelligence in a difficult and complex domain. That issue is controlling complexity while systematically exploring design alternatives in an automated design system. The design system exists, illustrating the feasibility of a particular paradigm. It presents a unique opportunity to further test the potential of combining planning techniques with a design problem solver. The artificial intelligence concepts utilized to solve the class of design problems are: problem representation and definition; problem decomposition into modular subproblems; constraint management for certain and efficient pruning; generate and test mechanisms; interaction between local and global constraints; search strategies.

The major contributions of this thesis are as follows:

1. Space planning cuts across disciplines as do the general synthesis issues that the thesis addresses. The issues raised deal with the representation and abstraction of geometric objects in dealing with the pervasive problem of geometry in design. These issues are of general interdisciplinary interest, not limited to just 2-D layout. Therefore, the research will result in an advancement of the state-of-the art knowledge about design systems, especially with respect to planning capabilities and abstraction mechanisms.

2. Of general interest, the research will produce the prototype of a design system that uses multiple abstraction capabilities to generate alternatives in a demanding domain. The formulation of the proposed research in conjunction with an approach to automated layout will allow the precise study of problem decomposition in an interesting and challenging context

3. Of specific interest, the research will extend in a substantial way an effective space planning approach that is applicable to layout problems in a variety of domains. The research will contribute to the efficiency, generality, flexibility and adaptabilily of the approach. It will produce the prototype of a space planning system with capabilities that are far superior to those that are commercially available.

I. A brief review of AI planning research

LI The formal roots of planning

At the heart of Artificial Intelligence research is the physical symbol system hypothesis of Newell and Simon [Newell 76]. This hypothesis, which seems not to be a *priori* provable and hence may only be empirically tested, nevertheless provides the basis for the belief that it is possible to build programs that can perform intelligent tasks now performed by people [Rich 83]. To enable machines to perform as generally intelligent agents it has been necessary to model and formalize the process by which they may attempt to solve problems. Making use of such a model enables the conversion of informal problem descriptions into systematic representations and procedures which the machine can use. A description of a very important formal problem-solving model known as a *state-space search* is:[70]

1. Define the state space which is the space that contains all possible potential solution configurations of the objects relevent to the problem. This space may be implicitly defined and need not(and in many cases could not!) be explicitly enumerated.
2. Specify the initial state(s), or the situation(s) from which the problem solver may start
3. Specify the goal state(s), or states which can be recognized as characterizing solutions to the problem.
4. Specify the rules that define the actions or operators available to move around, or go from one state to another, within the problem space.

Given that the above are specified, the problem-solving process may be formalized as finding a sequence of operations which will transform a given initial state into a desired goal state; some means must be available to compare intermediate states with the desired attributes of the goal state in order to identify when one has reached the goal or solution.

If enough knowledge is available regarding the problem, the problem solver may know just what to do and may apply direct methods in moving through the problem space from the initial state to the goal state, thereby describing the solution path. However since direct methods are rarely known for solving even marginally hard problems, generally some method of systematically *searching* through the problem space must be employed. That is why *search* is at the core of many intelligent processes, and of Artificial Intelligence. It may be defined as: using the set of rules that define allowable actions(number 4 above), *in combination with an appropriate control strategy,* to move through the problem space until a path from the initial state to the goal state is found. In some cases we are interested in the goal state itself as the solution, and in other cases the path(constituting a plan) to get there. The requirements of the control strategy are that it perturb the system(cause motion, or change from state to state), that it be systematic(capable of exploring every state and no state more than once), and that it be efficient.

Blind search

Two well known systematic control strategies which can perform a blind, exhaustive search of a problem-space are *breadth-first search(BFS)* and *depth-first search(DFS)* of which DFS is usually preferable

[70]Search is, of course, not the only way to represent and think about problem solving tasks. Alternative metaphors for viewing problem-solving tasks are* as logical problems, using formal reasoning methods; as constraint satisfaction processes. In most cases the same problem-solving algorithm can be viewed at various time in terms of each of these metaphors. [SEMON83 A]. See also [Laird 84,p 10]. As we shall see, the more advanced stragegic planning systems at various times seem to combine aspects of all of these metaphors.

because it uses less resources. If these *blind methods* are used alone with no knowledge to reduce the search effort the search is called a *brute force search.* A severe drawback to using brute-force search is that, for many problems, the number of states in the search space grows exponentially with the number of steps we take into it(the number of sequential operators applied), and thus these methods are *not nearly efficient enough to be practical.* Even a sophisticated model of blind search, such as depth-first iterative deepening has a performance which is exponential in the depth of the search [Korf 87]. Even when coupled with pruning techniques such as *branch and bound,* which still guarantees an optimal or shortest solution path, these search methods are doomed by a combinatorial explosion of the problem space.

Heuristic search

To conduct search with a control strategy that has any practical hope of successfully traversing the problem space, AI researchers came up with the concept of heuristic search. Heuristic search involves employing a little knowledge about the search space (in the form of a *heuristic evaluation function)* to improve the efficiency of the search, but usually at the expense of sacrificing the guaranteed completeness of the exploration of the problem space and the optimality of the solution found. Heuristics are *rules of thumb,* or approximate partial knowledge that might aid the discovery of the solution, but can not guarantee to do so. They involve "...the formulation of the problem to be solved as combinatorial search, with the heuristics cast in specific roles to guide the search, such as the selection of which step to take next, the evaluation of a new state in the space, the comparison of the present state to the posited goal-state, and so on[11] [Newell 82]. Therefore heuristics arc techniques or guides that offer *good,* but possibly not perfect solutions, *a good deal of the time,* but not always. They represent an important class of knowledge that arises in many tasks and it is usually the first knowledge about a task that becomes available to the problem solver, for instance, heuristic evaluation functions used to rate states in the space are a source of knowledge that has been most studied in the context of search problems. The heuristic search strategies generally constitute the *weak methods* because they make limited demands for knowledge of the task environment and can therefore be used in many domains; they are ubiquitous in AI, employed whenever problem-solving involves search [Laird 84]. Good general purpose heuristics exist which are (weakly) useful in many problem domains, and there are also useful domain specific heuristics: [Korf 85]

> A heuristic evaluation function is a function that is relatively cheap to compute form a given state, and that provides an estimate of the distance from that state to the goal. Most of the weak methods except for generate and test (which is a brute-force technique) rely on such a function, either explicitly or implicitly. For example, the evaluation function is the essence of simple heuristic search.... If we view the number of subgoals remaining to be satisfied as an evaluation function, then even means-ends analysis use an evaluation function.

However, general purpose heuristic search control strategies are categorized as weak methods since their effectiveness in overcoming the potential combinatorial explosion in the problem space is highly dependent on how the knowledge of the the problem domain may be exploited to reduce search. The set of weak methods also includes [Rich 83][71]:

---

[71]Note that Korf [Korf 85] includes only heuristic search, hill-climbing, generate-and-test, and means-ends-analysis in the set of weak operators. He considers problem reduction, or subgoaling, to be one of the much more powerful search reducing methods that planners introduced along with *abstraction* and *macro-operators.* Also a note of confusion [Cohen 82,p 30]: Newell and Simon initially made no distinction between blind search and heuristic search; thus they included Breadth-First Search and Depth-First Search in heuristic search, but excluded generate-and-test! The current understanding of the term stems from Nilsson [Nilsson 71] who classified heuristic search as containing domain specific information sufficient to restrict search, that is, going beyond that needed merely to formulate a class of problems as search problems. Whether or not the restriction is foolproof, the search is then called heruistic raher than blind. This would seem to adequately distinguish blind search methods - those that treat the space syntactically - from, heuristic methods which add domain knowledge; but it perhaps set the stage for some confusion about search in knowledge-based systems! So goes the developmental history of our knowledge.

- Heuristic search
- Generate-and-test
- Hill climbing
- Means-ends analysis
- Constraint satisfaction
- Problem reduction

These methods are a considerable improvement over blind search but even when problem characteristics are analyzed and one or more of these methods is carefully selected to best match, they are generally too weak to solve hard problems by themselves. Heuristic evaluation functions effectively reduce the branching factor of a search from that of brute force search allowing for somewhat larger problems to be solved. On the other hand, the complexity of heuristic search is still exponential, therefore more powerful knowledge sources are needed, subgoals, abstraction, etc which are the methods of *planning* [Korf 87]. The virtue of the heuristic methods- the fact that they require only a small amount of knowledge about a problem and hence are extremely general - allows them to be applied when not enough knowledge about a problem is available to employ a more powerful solution technique. Hence, their virtue is also their vice - or *weakness.*

Beyond heuristic search

Advanced problem-solving, including planning, involves the combining of one or more of these basic problem-solving strategies with one or more knowledge representation mechanisms together with a divide and conquer approach to problem decomposition to whatever extent possible [Rich 83]. The early planning systems, and most of the subsequent systems, have been based on a model using a combination of means-ends analysis(MEA) and problem reduction in a search space using a representation that captures both. The GPS system [Ernst 69] is credited with introducing the MEA method and Gelemter's Theorem prover [Gelemter 63] with introducing problem reduction into problem solvers.[72]

A drawback of the heuristic search techniques employing an evaluation function is that they assume the possible application of all operators at each state. MEA provides for the selection of the best operator to reduce the difference between the current state and the goal state. Consequently, operators are ordered in addition to states and the number of operators that has to be considered at each move is substantially reduced. In the problem reduction approach the problem solver works by reducing problems into sets of simpler subproblems repeatedly and recursively until primitive problems are reached whose solutions are immediate.

The generalized notation for a problem reduction search is an AND/OR graph, whereas for a state space search it is an OR graph. In an AND/OR graph the AND nodes represent subgoals, all of which must be achieved in order to achieve the parent goal; the OR nodes represent alternative decompositions of the parent goal into different sets of such required subgoals. Thus, the OR nodes represent splits in the path through alternative states in the problem space; potentially any of the states may be the solution, or may be on the path to the solution. In the planning paradigm reducing a task(the problem given) consists of

[72]The GPS system also used problem reduction, or goal reduction, as part of its problem space model in that it reduced goals to subgoals. Actually, MEA may be viewed as a subset of the goal reduction approach which implies a depth first search and assumes independent subgoals. MEA always picks the first subgoal on the list (of goals or subgoals at a level) to solve as a priority. Goal reduction as implemented in Gelemter's system is the more general model where the problem-solving process is driven by the backward chaining, *unordered* goal reduction process.

finding a collection of subtasks such that the problem task may be carried out by achieving the conjunction (AND) of all of the subtasks; the aggregation of these subtasks then represents the *plan.* Retrieval of relevant alternative sets of subtasks(OR choices in the decomposition of the problem task) is an important problem in itself and several mechanisms have been tried. Most of the planning systems to date have relied upon some form of "pattern directed invocation"[73] of operator choice - where the operator is composed of a set of subtasks. But no matter how the retrieval is accomplished the problem solver must be able to deal with the possibility of more than one choice of reduction being applicable. Given this situation one approach the problem solver can take is to explore all of the possibilities systematically by backtracking or use a more sophisticated control device. The choice of exploring all possibilities leads directly to the classic AND/OR graph-search approach [Nilsson 71] as cited in [MCDERMOTT78] which is why it became the basis for the early "successful" planning systems. As we shall see later more powerful and effective methods for choosing the next relevant operator are being tried, such as meta-planning, or applying the full power of the problem-solving system recursively to make that choice. That choice is, in turn, part of the larger problem of choice in AI problem solvers called the *control problem:* Which of its potential actions should an AI system perform at each point in the problem-solving process? Addressing that question has lead to the development of sophisticated architectures for the control of problem solvers, including the *blackboard model* which has been applied to the planning process [Hayes-Roth 85]; but more on that later!

Subproblem interactions

Heuristic search may be conducted in both a problem reduction representation or a state space representation or the mixed model. However, in any model involving search, *interdependent subproblems* create a new level of difficulty. A major difficulty of any system employing decomposition as a search-reducing, "divide and conquer" technique, is the potential *interaction* of not wholly independent subproblems. Two common examples involving interacting subproblems are: problems requiring consistent binding of variables; problems involving the expenditure of scarce resources. A variant of the scarce resource problem arises in robot planning tasks(such as those performed by STRIPS) and stimulated the research into methods to handle it: this is where the application of an operator **representing a robot action *solving one sub-problem may make inapplicable the operator needed to solve another subproblem.***

Planning

The brief overview above of the fonnalization of problem-solving and search in AI is important because it points out where and why the concept of *planning* entered the picture as a significant area of problem-solving methodology in AI in its own right. One of the most important motivations for planning in problem-solving is that of introducing tools or methods to assist the weak methods in reducing search as much as possible. The weak methods are dedicated to the *control* of exponential expansion, whereas the more intensively knowledge based methods or *strong methods* of which planning is a part are dedicated to its prevention [Newell 76].

---

[73] **All pattern directed invocation schemes may be described as some variant of the procedure: extract** and match features of the **problem task to an indexed set of subtask schemata which can be instantiated to yield a plan.** This practice of retrieving **techniques according to the form of a goal and trying each alternative** in **turn until one succeeds** is **attributed** to Hewitt [Hewitt **71] as cited in [Waldinger 77]. Such retrieval is usually a part of the MEA strategy and under that regime** the problem solver can **only be single-mindedly aware of one goal at a time. Hence, the assumption** of decomposability becomes functionally **entrenched and may lead to problems. Ideally, as Waldinger states in the above source, we would like to be able to apply whatever techniques are available to each conjunctive goal while somehow combining the results** into **a** single coherent plan that **achieves them all.**

The methodology of planning falls somewhere in the middle of a spectrum of problem-solving techniques bounded on one end by "knowing what to do directly" and on the other end by "blind search." A good early example of the use of planning to assist problem-solving is the DENDRAL system in which a planner is used to redefine the problem in terms that reduce the subsequent search effort of the problem solver. DENDRAL infers the structure of organic compounds using mass spectrogram and nuclear magnetic resonance data, and the overall system employs *aplan-generate-test* strategy. The planning part of the system consists of automatic inferencing of constraints on candidate structures derived from mass spectrometry and produces two lists: one of recommended structures and another of disallowed structures. The generator must then operate within the plan's constraints and is thus constrained to explore(or search for) only a fairly limited set of structures which include the mandatory substructures and exclude the forbidden substructures. In this case, as Rich points out, neither the planner nor the main problem solver is capable alone of solving the problem, but the combination is effective [Rich 83, p 74]:

> This combination of planning, using one problem-solving method (in this case, constraint satisfaction) with the use of the plan by another problem- solving method, generate-and-test, is an excellent example of the way techniques can be combined to overcome the limitations that each possesses individually.

Thus, certain of the weak methods, in particular -MEA, problem reduction, and constraint satisfaction - have been incorporated into planning systems which are stand-alone problem solvers(as in robot planning) or which assist other problem solvers(as in the DENDRAL system above). In this way, although they are too weak to solve hard problems by themselves, the weak methods continue to provide the framework into which domain-specific knowledge can be placed and so they continue to form the core of most AI systems [Rich 83, Sacerdoti 80]. Another interesting thing to note about the relationship between search and planning is their complimentarity: whereas search is inherently flexible and needs structure for efficiency, having a plan to solve a problem is inherently efficient but needs elements of search for flexibility.

Goal decomposition, interactions and planning strategies

Much of the early work in illuminating the nature of problem-solving and the processes(which can be automated) that accomplish it was done by Newell and Simon [Newell 72]. A very important idea developed by the above researchers is that in complicated problem domains, problems may be decomposed into ever simpler subproblems which are easier to find solutions for and may eventually be solved directly; then these partial solutions may be combined, in a simple additive manner, to yield a solution for the larger problem. The concept of *additivity* - the simple conjunction of the solutions to subproblems to solve the larger problem - was introduced into the planning paradigm through the work of Newell and Simon on GPS [Newell 72]. Unless it is possible to decompose problems in this manner the problem space will include the combinations of all components of the problem and will be too large to manage and explore in the time available. This complication of the problem space has not only to do with the exponential expansion of search, but also with how to represent states in the problem space so that a new state may be efficiently computed whenever moving from one problem state to the next This issue of how to determine what changes and what does not in the world model whenever there is a change of state is generally known as the *frame problem.* It is also of vital importance in problem-solving systems because of its potentially exponential cost as the number of potential states and the number of axioms describing the world increase. The restrictive assumptions made by the strategic planners with regard to representation and methodology help greatly to finesse the frame problem and to reason about interactions. Implementing this approach, the STRIPS system formalized the operators of the search space in the predicate calculus; operators were represented as rules with pre-conditions and post-conditions. See the *STRIPS assumption* in [Fikes 71] and current comments on the continued value of the STRIPS language for the form and content of actions in [Dean 88].

Of course, subproblems may not be entirely independent, but experience suggests that in many domains they are nearly decomposable; that is, we may proceed by assuming the independence of the subproblems and record and handle their interactions as we go along (more on this later!). The methods of planning were developed largely to handle problem decomposition, subproblem interaction and state representation with increasing flexiblility while retaining enough efficiency to be practical.

As a proximate assessment then, the history of the development of planning in AI can be identified with the use of methods to assist in efficently accomplishing problem-solving in certain domains by reducing search, efficiently representing the problem space and changes of state, and managing the interactions of subgoals. The formal properties of these methods are for the most part not yet proven, and there is strong evidence that the goal of most of these planning systems - *domain-independent conjunctive planning* - useful in realistic domains may not be possible [Chapman 85]. This is in part due to the fact that the effectiveness of the algorithms of the strategic planners is based on a representation language and scheme dependent on the assumptions of perfect information and prediction. However, some of the methods which have been developed may be generalizable and useful, particularly as techniques or subcomponents for domain specific problem solvers in more complex domains. They involve techniques for all of the following and more:

- problem representation and decomposition
- searching for solutions in a world model
- the handling of subproblem interaction
- error handling
- communication among processes or parts of a system
- recording of reasons for choices (affording dependent backtracking or an explanation facility)
- learning, both short-term,or within a problem-solving session,and long-term or across sessions.
- execution monitoring and re-planning

I.2 A brief chronology of the developments in planning

STRIPS, the beginnings of a paradigm

The earliest planners were conceived on the insight that often in search problems the mechanism for satisfying goals by top-down processing, or backward chaining, is more efficient than that of forward chaining, which may be considered to be a form of generate-and-test. Simple back-chaining was combined with the "difference reducing" stategy of MEA and a simplified action representation(operators with declarative preconditions and effects) to produce the STRIPS paradigm of planning [Fikes 71]. Backward chaining is a trivial process when there is only one goal to be achieved, but becomes more difficult when there are more, as satisfaction of one goal can produce a state which precludes the satisfaction of another. The STRIPS approach represents a general paradigm for backward chaining to satisfy multiple goals.

Interestingly, though many of the strategic planning systems have improved over the basic STRIPS approach, most still relie in part on variations of *backward chaining, MEA and a STRIPS-like operator/action representation.* Each of these methods helps in a clear way to reduce the search or representation burden of the problem solver. MEA, which is a special form of ordered goal reduction and hence also involves backward chaining, reduces search by partitioning the operator set so that operators in

a given subset are relevant to only a single subgoal. This partitioning reduces the branching factor of the search by allowing the problem solver to focus on a subset of the operators at any given time. Necessary assumptions of this method are that the problem-solving operators can be classified according to the kinds of difference they reduce and that the subgoals be nearly independent; that is, there is an ordering among the operators such that they can be applied in a sequence without interfering with one another. If the subgoals are completely *independent* the search effort is effectively reduced in complexity by dividing both the base -B- (branching factor of the search space) and the exponent -D-(depth to the goal in the search space) of the complexity function $O(B^0)$ by the number of subgoals [Korf 87]. However, a severe drawback to the simple use of MEA as the primary source of knowledge in a planner is that it still develops large search spaces and does not avoid dead-ends.

Abstraction based and hierarchical planning: ABSTRIPS, NOAH, MOLGEN, etc

The next idea employed, hierarchical abstraction, attempted to deal with the still combinatorial seach spaces of the the STRIPS paradigm by focusing the planner on the hardest parts of the problem first; see [Siklossy 73] and more well known, ABSTRIPS [Sacerdoti 74] which abstracted the goals into fixed levels of *criticality.* Actually, abstraction was first employed in a problem-solver/planner in GPS, which abstracted a more general representation of one aspect of its problem space, and in which the ordering of operators and differences in its difference table may also be viewed as in implicit hierarchy. Subsequently, NOAH [Sacerdoti 75] and many other systems following it abstract the problem-solving operators. They plan initially with generalized operators and later refine these, via *hierarchical decomposition,* to the primitive problem-solving operators given in the problem space. MOLGEN [Stefik 81a, Stefik 81b] goes one step further and abstracts both the operators and objects in its problem space. By representing interactions between operators explicitly as *constraints* MOLGEN introduced general constraint reasoning into planning whereby a planner is able to reason about ordering constraints for operations, conditions on objects, etc. Many interrelated ideas and issues are tangled in the terms *abstraction planning.hierarchical planning,* and *constraint-based reasoning;* what they have in common is a reliance on abstractioa These important ideas, particularly the power of abstraction in planning, are sorted out further below under the discussion of strategies. In all cases however, hierarchical planning involves representing the problem, and planning, in one or more abstraction spaces. In general terms, a plan is first generated in the highest, most abstract space and the details of this *skeletal* plan are fleshed out in the lower abstraction spaces. Thus hierarchical planning provides a means of ignoring details that obscure or complicate a solution, and *"By searching an abstracted representation of a space, the combinatorics can be reduced"* [Hayes-Roth 83, p 70]. The importance of hierarchical methods for taming large search problems has been recognized for many years. Minsky [Minsky 63] is one of the earliest and most often quoted sources who pointed out the enormous value of finding intermediate goals in the solution space (abstract partial solutions) which he called "planning islands":

> It will be worth a relatively enormous effort to find such 'islands' in the solution of complex problems...Thus practically any ability at all to 'plan,* or 'analyze,' a problem will be profitable, if the problem is difficult.

Using such hierarchical methods is thus seen as the "essence" of planning and can reduce search by what Minsky called a "fractional exponent".

Linear planning - issues, elaborations: HACKER, WARPLAN, INTERPLAN, et al

The abstraction levels approach of ABSTRIPS helped to solve some of the difficulty with the basic STRIPS approach, but other problems remained which lead to research on *conjunct ordering, subgoal interaction* and the *non-linear nature ofplans.* A very important property resulting from the assumption of independence is that an optimal solution for the problem as as whole can be obtained by simply concatenating together the optimal solutions of the individual subproblems, in any order! In the planning

literature this property is rather confusingly called the *linear assumption.* The name probably resulted from the fact that all of the early planners were limited to a one-dimensional representation for the developing plan as a linear sequence of operators. These planners attempted to solve a conjunction of goals, in the order given in the problem statement by simply appending the operators necessary to solve each subgoal encountered to the end of the sequence. Their representation implied a precedence to the order of solving the goals even if none existed. Furthermore, in most domains, subgoals are not perfectly independent and though their order of satisfaction is usually not specified in the problem statement, it may be critical to finding a solution. As mentioned above, *interactions* occur when solving one goal prior to another prevents the solution of the latter. Interacting subgoals can quickly compound the search effort because additional search results from premature commitment to an arbitrary ordering of interacting subgoals. *Backtracking* involves replanning from a *choice point* that failed and it can be very costly. What then was the value of assuming independence of the subgoals? In the Handbook of Artificial Intelligence [Cohen 82] it is characterized as an historically important heuristic: 'The orderings of problem-solving operators is the factorial of the number of operators, so it is obvious that all orderings cannot be examined in the hope of finding one that does not fail because of interacting operators." The linear assumption says that, assuming subgoal independence, one ordering is as good as any other, so choose one and then fix any interactions that emerge; it is used in cases where there is no *a priori* reason to order one operator ahead of another. Thus the linear assumption is an assumption with a heuristic basis, but often without a realistic basis, which is another reason why it is a confusing term and has frequently been referred to disparagingly!

In any case, within the linear plan-representation paradigm, researchers experimented with various representations and plan manipulation operators of increasing sophistication to accomplish appropriate subgoal ordering and achievement during planning [Warren 74, Tate 75, Rieger 77, Waldinger 77]; one system HACKER [Sussman 75] raised the assumption of complete independence of subgoals to a strategy where the planner first forms a complete ordered plan based on the assumption and then attempts to *debug* the plan when it fails by using a library of known classes of bugs resulting from dependencies among the subgoals. The linear assumption, which is definitively associated with Sussman [Sussman 75,p 59] - "..subgoals are independent and thus can be sequentially achieved in an arbitrary order" - was used in HACKER more in the interest of having the system *learn* how to handle simultaneous goals than with the facilitating of the planning itself. If a bug were encountered in the plan for which there was no answer in the "bug" library, then HACKER would attempt to synthesize a generalized subroutine for handling that bug; the newly learned "skill" could then be incorporated in the library for use in subsequent plans in future problem solving situations.

Non-linear and least-commitment planning: NOAH, MOLGEN, et al

An alternative assumption to the linear assumption was that it might be better to defer the ordering of operators than to order them arbitrarily. This idea has led to a whole line of *non-linear* planners, starting with NOAH, which, in addition to the hierarchical decomposition of operators mentioned above, contributed a special representation called a *procedural network* which allowed the constraints on operator sequencing to be expressed only as required, as a partial ordering. That is, NOAH orders operators only to eliminate interactions that arise and establishes partial orderings on sequencing to resolve those interactions by considering the preconditions of operators. The deferring of the ordering of operators until enough information is available to do so with certainty is an instance of the *least-commitment approach* to decision making. This approach contrasts with the linear assumption which advocates *committing* to any order of operators and then fixing it. The non-linear approach has been refined, extended and used in many subsequent strategic planning systems [Tate 77, Vere

83, Wilkins 84] culminating in a rigorous reconstruction of the methodology in [Chapman 85]. The least-commitment approach to ordering operators arises in a slightly different form in Stefik's MOLGEN [Stefik 81a]. As mentioned above MOLGEN represents conditions on both operators and objects, including operator ordering, explicitly as *constraints*. MOLGEN will not order operators until there are constraints available to guide it; furthermore it avoids commiting itself to using operators or objects without constraints because premature commitment may conflict with other parts of its plan. The non-linear and constraint reasoning approaches both work because they are able to infer constraints that hold between operators and make decisions only when they will not interfere with past or future decisions. The least-commitment method is a *constructive* approach because it avoids backtracking and having to undo bad decisions as much as possible. This is in constrast with the *destructive* approach inherent in the linear planners which make early commitment to the ordering of operators through assumption or guess and then have to restructure or fix the plan in response to problems that arise.

Meta-planning

In order to manage its constraint reasoning approach which necessarily involves the use of abstract representations ("..as it chooses specific operators or objects to replace abstract ones (by refinement) it introduces *constraints* into the plan." [Cohen 82]), MOLGEN extends the work on hierarchical planning to include a layered control structure. Planning at abstract levels is combined with reasoning about details using constraints. The *constraint formulation-propagation-satisfaction* cycle is a constructive process where abstract parts of the plan are usually refined only when there are the necessary constraints available. To manage this complex process a hierarchically organized control structure is introduced consisting of design, plan and strategy levels. This approach was an instance of giving planners meta-level control capability so they can reason about their reasoning (or plan about planning) as well as planning about solving the problem at hand. One of the important choices that a planner has to make is which of the possible alternative decompostions (OR choices in the AND/OR problem reduction approach) of a given task into its relevant subtasks is most appropriate (likely to succeed). In general meta-level reasoning explores more powerful and explicit methods for choosing what the problem-solver should do next than the previous exhaustive or pattern-directed approaches. Within the planning arena, meta-level reasoning has been called *meta-planning* and the methods and issues involved have been further explored by [Stefik 81b, Wilensky 81, Wilensky 83, Birkel 86] among others.

Opportunistic planning

Related to the ideas of meta-planning and constraint propagation (which is an explicit mechanism for moving information between subproblems and exploiting the synergy between decisions in different subproblems [Hayes-Roth 83,p 108]) is the idea of *opportunistic* planning. Opportunistic planning, as demonstrated in the OPM system [Hayes-Roth 79] is a theory of planning designed to model human planning capability and is based on the blackboard control architecture spawned by the Hearsay systems. This type of control structure is much more complex, flexible and dynamic than that of the other approaches. The term opportunistic refers to the asychronous manner in which planning decisions are made - only when there is a reason to do so. The ordering of operators that characterizes a plan is developed piecewise in localized clusters since it is driven partly by opportunities to include detailed, *bottom-up* problem-solving decisions in the developing plan. This is in contrast with *top-down* decomposition process characteristic of hierarchical planning, in which the detailed problem-solving actions are not decided until the last possible moment. Thus, in terms of representing, reasoning about, and handling the interactions inherent in planning/problem-solving, that is, in terms of decision making and commitment, the opportunistic model falls somewhere between the simple early commitment heuristic of the linear planners, and the more elaborate least-commitment approach of the later planners.

Skeletal planning, macros, case-based planning and learning

Another set of ideas important to problem-solving were integrated into planning research from early on. These are the intertwined ideas of *learning,* and the use of pre-formed plan segments, sometimes called *skeletal plans*[74]* or *macros* to provide planners with a mechanism to avoid having to repeatedly plan typical tasks from scratch. HACKER is an early example of an approach to planning that took learning into account and two important ideas surface from that work: first, planning should consist of using (re-using) known plans wherever possible (and debugging them in the current context as required); second, once a planner has constructed a useful plan, it should store the plan for later use. The most trivial and direct re-use of a plan segment might involve the retrieval of an exact sequence of primitive operations. However generally the attempt has been to retrieve an *abstracted* version of a plan that contains the basic steps and *instantiate* each of the plan steps within the environment of the particular problem by weaving together the experience implicit in the stored plan and a heavy dose of domain knowledge about the current context The idea of abstracted plans is found as early as the STRIPS planner which parameterized successful plans in order to generalize them [Fikes 72]. The generalized plans were stored in structured, indexed tables for retrieval and were called MACROPS (for macro-operators). Central to the utility of the skeletal planning approach is knowledge for competent plan selection and plan-step refinement. These tasks require an approach on how experiences are retrieved and exploited during planning. More recently this style of planning is re-emerging as a new paradigm called *Case-based planning(CBP)* which takes as its premise that the organization of experience is paramount in formulating new plans and re-using old ones [Dean 88]. Thus, storage and retrieval are vital, and most case-based approaches involve some theory of learning and memory organization. The concept of skeletal-planning or CBP has roots and direct precedent in Schank and Abelson's work with scripts; in their view, plans are generalized scripts that explain events related to, but not exactly like those the user has seen before [Schank 77].

Refined strategic planning and beyond

Many other related, but alternative views and heuristics for planning have been taken. One example is the utility of first attempting to disprove that goals can be reached before attempting to reach the goals [Siklossy 77, Kibler 81]. Further research has extended the hierarchical, non-linear, least-commitment paradigm to begin to deal with difficult issues of time and resource consumption. The focus and basic questions of planning have greatly expanded and become somewhat diffused as the gap has widened between the strategic and reactive branches of the discipline as mentioned above. Relaxing the restrictive assumptions of the traditional paradigm has become necessary as planning is extended into areas such as: hierarchical planning and distributed problem-solving; planning by multiple (cooperating) agents; recognition and repair of error conditions in execution; path planning, or planning and spatial reasoning for navigation of robots; etc.

---

[74] **The precompiled decompositions of operators in hierarchical planning have also been called *skeletons* but they differ in that they are generated as choices are made and the plan develops, rather being recalled from a store of plans.**

References

[Araya 87] Agustin A. Araya and Sanjay Mittal.
Compiling Design Plans from Descriptions of Artifacts and Problem Solving Heuristics.
In *International Joint Committee for Artificial Intelligence*, pages 552-558. IJCAI, Milan, Italy, August, 1987.

[Archea 86] Archea, J.
Puzzle-Making: What Architects Do When No One is Looking.
In Harfinann, A. C, et. al. (editors), *The Computability of Design, 1986 SUNY Buffalo Symposium on CAD*. SUNY Center for Tomorrow, Buffalo, NY, December, 1986.

[Baykan 87] Can A. Baykan.
Opportunistic Constraint Satisfaction in Space Planning.
Ph.D. Thesis Proposal, Carnegie-Mellon University, December, 1987.

[Bernstein 88] Mark Bernstein.
A self-starter who gave us the self-starter.
Smithsonian, July, 1988.

[Biricel 86] Biikel, Paul A.
Metaplanning: Controlling Planning in a Complex Domain.
March, 1986.
Dissertation Proposal, CS Dept. CMU.

[Bond 88] Alan H. Bond and Les Gasser (editors).
Readings in Distributed Artificial Intelligence.
Morgan-Kaufmann Publishers, Inc., San Mateo, Calif., 1988.

[Brown 85] David C. Brown and B. Chandrasekaran.
Knowledge and Control for Design Problem Solving.
Technical Report, Ohio State University, July, 1985.

[Brown 86] David C. Brown and B. Chandrasekaran.
Knowledge and Control for a Mechanical Design Expert System.
Computer :92-100, July, 1986.

[Buchanan 71] Bruce G. Buchanan, Edward A. Feigenbaum, and Joshua Lederberg.
A Heuristic Programming Study of Theory Formation in Science.
In *IJCAI 71*, pages 40-50. UCAI, 1971.

[Chakrabarti 86] P. P. Chakrabarti, S. Ghose, and S. C. DeSaricar.
Heuristic Search through Islands.
Artificial Intelligence 29:339-347,1986.

[Chandrasekaran 88]
B. Chandrasekaraa
Design: An Information Processing-Level Analysis.
Technical Report 88-BC-DESIGN, The Ohio State University, January, 1988.

[Chapman 85] David Chapman.
Planning For Conjunctive Goals.
Technical Report 83-85, Artificial Intelligence LaboratoryJtflT, November, 1985.

[Chamiak 86] Chamiak, Eugene and McDermott, Drew.
Introduction to Artificial Intelligence.
Addison-Wesley Co., Reading, MA, 1986.

[Cheeseman 88] Peter Cheeseman.
Uncertainty and Planning: A Summary, In 'DARPA Santa Cruz Workshop on Planning'.
AI Magazine : 124-127, Summer, 1988.

[Christiansen 85] Christiansen, Alan D.
The History of Planning Methodology: An Annotated Bibliography.
SIGART Newsletter (94), October, 1985.
planning, review, techniques,problem solving.

[Cohen 82] Feigenbaum and Barr (editors).
The Handbook of Artificial Intelligence.
William Kaufnann, Inc., One First Street, Los Altos, Calif, 1982.

[Coyne 85a] R. D. Coyne and J. S. Gero.
Design Knowledge and Sequential Plans.
Environment and Planning B: Planning and Design 12(4):401-418,1985.

[Coyne 85b] R. D. Coyne and J. S. Gero.
Design Knowledge and Context.
Environment and Planning B: Planning and Design 12(4):419-442,1985.

[Coyne 86] R. D. Coyne and J. S. Gero.
Semantics and the Organization of Knowledge in Design.
Design Computing l(l):68-89,1986.

[Darden 87] Lindley Darden.
Viewing the History of Science as Compiled Hindsight.
AI Magazine :33-41, Summer, 1987.

[Davis 80a] Randall Davis.
Meta-Rules: Reasoning about Control.
ai 15:179-222,1980.

[Davis 80b] Randall Davis.
Content Reference: Reasoning about Rules.
ai 15:223-239,1980.

[de Kleer 79] J. de Kleer, J. Doyle, G. L. Heele, and G. J. Sussman.
Explicit Control of Reasoning.
Artificial Intelligence: An MIT Perspective.
MIT Press, Cambridge, Massachusetts, 1979, pages 93-116.

[Dean 88] Thomas Dean.
Planning Paradigms, In 'DARPA Santa Cruz Workshop on Planning'.
AI Magazine :115-119, Summer, 1988.

[Eastman 73] Charles M. Eastman.
Automated Space Planning.
Artificial Intelligence 4:41-64,1973.

[Eastman 75] Charles M. Eastman (editor).
Spatial Synthesis in Computer-Aided Building Design.
John Wiley and Sons, New Yoik, New Yoik, 1975.

[Eastman 81] Charles M. Eastman.
Recent Developments in Representation in the Science of Design.
In *18th Design Automation Conference*, pages 13-21. IEEE, 1981.

[Eastman 85] Charles M. Eastman.
Abstractions: A Conceptual Approach for Structuring Interaction with Integrated CAD Systems.
Comput. & Graphics 9(2):97-105,1985.

[Ernst 69] Ernst.
GPS: A Case Study in Generality and Problem Solving.
Academic Press, New Yoik, NY, 1969.

[Fikes71] Fikes, R.E.; Nilsson, N.J.
STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving.
Artificial Intelligence 2:189-208,1971.

[Fikes 72] Fikes, Richard E.; Hart, Peter E.; Nilsson, Nils J.
Learning and Executing Generalized Robot Plans.
Artificial Intelligence 3:251-288, 1972.

[Fisher 86] Edward L. Fisher.
An AI-Based Methodology for Factory Design.
AI Magazine :72-85, Fall, 1986.

[Hemming 78] Hemming, U.
Wall Representations of Rectangular Dissections and their Use in Automated Space Allocation.
Environment and Planning B 5:215-232,1978.

[Hemming 79] Ulrich Hemming.
Representing an Infinite Set of Solutions through a Finite Set of Principal Options.
In *Proceedings of the 10th Annual Conference of the Environmental Design Research Association*, pages 190-197. Environmental Design Research Organization, Buffalo, NY, 1979.

[Hemming 80] Hemming, U.
Wall Representations of Rectangular Dissections: Additional Results.
Environment and Planning B 7:247-251,1980.

[Hemming 86a] U. Hemming.
On the Representation and Generation of Loosely Packed Arrangements of Rectangles.
Environment and Planning B: Planning and Design 13:189-205,1986.

[Flemming 86b] Hemming, U., Coyne, R., Glavin, T., and Rychener, M.
A Generative Expert System for the Design of Building Layouts.
Applications of Artificial Intelligence in Engineering Problems.
Springer, New Yoik, NY, 1986, pages 811-821.

[Hemming 86c] Hemming, U., Coyne, R., Glavin, T., and Rychener, M.
ROOS1 - Version One of a Generative Expert System for the Design of Building Layouts.
In *Proceedings of the International Joint Conference on CAD and Robotics in Architecture and Construction*, pages 157-166. Marseilles, France, 1986.

[Flemming 87] U. Flemming.
Rule-Based Systems in Computer-Aided Architectural Design.
In *Proceedings of the First International Symposium on Computer-Aided Design in Architecture and Civil Engineering.* Barcelona, Spain, April, 1987.

[Flemming 88a] U. Flemming, R.F. Coyne, T. Glavin, and M. Rychener.
A Generative Expert System for the Design of Building Layouts, Version 2.
Technical Report EDRC-48-08-88, Engineering Design Research Center, Carnegie-Mellon University, Pittsburgh, PA, August, 1988.

[Flemming 88b] Flemming, U., Coyne, R., Glavin, T., and Rychener, M.
A Generative Expert System for the Design of Building Layouts - Version 2.
Artificial Intelligence in Engineering: Design.
Computational Mechanics Publications, Southampton, 1988, pages 445-464.

[Flemming 88c] Flemming, U., Coyne, R., Glavin, T., and Rychener, M.
A Generative Expert System for the Design of Building Layouts (Final Report).
Technical Report (in preparation), Engineering Design Research Center, Carnegie-Mellon University, 1988.

[Fox 81] Mark Fox.
An Organizational View of Distributed Systems.
In *IEEE Transactions on Systems, Man, and Cybernetics SMC-11*, pages 70-80. IEEE, 1981.

[Fox 83] M. S. Fox.
Constraint-Directed Search: A Case Study of Job-Shop Scheduling.
Technical Report CMU-CS-83-161, CMU-RI-TR-83-22, Carnegie-Mellon University, December, 1983.

[Fox 86] Mark S. Fox.
Observations on the Role of Constraints in Problem Solving.
In *Proceedings of the Sixth Canadian Conference on Artificial Intelligence*, pages 172-187. Montreal, Quebec, Canada, May, 1986.

[Francis 74] R. L. Francis and J. A. White.
Facility Layout and Location: An Analytical Approach.
Prentice-Hall, Englewood Cliffs, N. J., 1974.

[Frayman 87] F. Frayman and S. Mittal.
COSSACK: A Constraints-Based Expert System for Configuration Tasks.
In *Proceedings of the Second International Conference on Applications of Artificial Intelligence to Engineering*, pages 143-165. Boston, MA, August, 1987.

[Funk 80] Funk, Wagnall, P. Barrett, C. Cohen, et. al. (editors).
Funk & Wagnalls Standard Dictionary.
Lippincott & Crowell, New York, NY, 1980.

[Galle 86] P. Galle.
Abstraction as a Tool of Automated Floor-Plan Design.
Environment and Planning B: Planning and Design 13:21-46, 1986.

[Gelernter 63] Gelernter.
Realization of a Geometry Theory Proving Machine.
Computers and Thought.
McGraw-Hill, New York, NY, 1963, pages 134-152.

[Genesereth 83] Michael R. Genesereth.
An Overview of Meta-Level Architecture.
In *Proceedings of the National Conference on Artificial Intelligence*, pages 119-124. AAAI, 1983.

[Gero 85a] J. S. Gero, A. D. Radford, R. Coyne, and V. T. Akiner.
Knowledge-Based Computer-Aided Architectural Design.
Knowledge Engineering in Computer-Aided Design.
Elsevier Science Publishers B. V., North-Holland, 1985, pages 57-87.

[Gero 85b] Gero, J.S.; Coyne, R.D.
Knowledge-Based Planning as a Design Paradigm.
Woiking Paper, Computer Applications Research Unit, August, 1985.

[Gero 88] John Gero, and Mary Lou Maher.
Chunking Structural Design Knowledge as Prototypes.
In. Center for Computational Mechanics, Third International Conference on AI in Engineering, August, 1988.

[Greeno 78] James G. Greeno.
Natures of Problem-Solving Abilities.
Handbook of Learning and Cognitive Processes.
Lawrence Erlbaum Associates, Hillsdale, NJ, 1978, pages 239-270, Chapter 6.

[Hall 71] Patrick A. V. Hall.
Branch-and-Bound and Beyond.
In *International Joint Conference on Artificial Intelligence*, pages 641-650. IJCAI, 1971.

[Havens 83] W. S. Havens and A. K. Mackworth.
Representing Knowledge of the Visual World.
IEEE Computer 16(10):90-96,1983.

[Hayes-Roth 79] Barbara Hayes-Roth, Frederick Hayes-Roth, Stan Rosenschein, and Stephanie Cammarata.
Modeling Planning as an Incremental, Opportunistic Process.
In *International Joint Conference on Artificial Intelligence*, pages 375-383. IJCAI, 1979.

[Hayes-Roth 83] Frederick Hayes-Roth, Donald A. Waterman, and Douglas B. Lenat (editors).
Teknowledge Series in Knowledge Engineering. Volume 1: *Building Expert Systems.*
Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 1983.

[Hayes-Roth 85] Hayes-Roth, Barbara.
A Blackboard Architecture for Control.
Artificial Intelligence 26:251-321,1985.
blackboard architecture, hearsay, cognitive model of planning.

[Henrion 78] Max Henrion.
Automatic Space-Planning: A Postmortem?
In Latombe (editor), *Artificial Intelligence and Pattern Recognition in Computer Aided Design*, pages 175-196. EFIP, North-Holland Publishing Company, 1978.

[Hewitt 71] Carl Hewitt.
Procedural Embedding of Knowledge in Planner.
In *International Joint Conference on Artificial Intelligence*, pages 167-182. IJCAI, 1971.

[Hobbs 85] Jerry R. Hobbs.
Granularity.
In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence,* **pages** 432-435. IJCAI, Los Angeles, Calif., 1985.

[Jacob 77] **F. Jacob.**
Evolution and Tinkering.
Science :116M167, June 10,1977.

[Keene 89] Sonya E. Keene.
Object-Oriented Programming in COMMON USP.
Addison-Wesley, Reading, MA, 1989.

[Khokhani 85] Kanti H. Khokhani, Angela Digiacomo, and Dennis G. Weinel.
Automatic Placement of Non-Uniform Rectangular Components.
VLSI Design :50-66, May, 1985.

[Kibler81] Dennis Kibler and Paul Morris.
Don't Be Stupid.
In *International Joint Conference on Artificial Intelligence,* pages 345-347. IJCAI, 1981.

[Koopmans 57] J. C. Koopmans and M. J. Beckmann.
Assignment Problems and the Location of Econometric Activities.
Econometrica 25:53-76, 1957.

[Korf77] R. E. Korf.
A Shape Independent Theory of Space Allocation.
Environment and Planning B 4:37-50,1977.

[Korf85] Richard E. Korf.
Macro-Operators: A Weak Method for Learning.
ai 26:35-77,1985.

[Korf87] Richard E. Korf.
Planning as Search: A Quantitative Approach.
ai 33:65-88,1987.

[Kundu 88] Sukhamay Kundu.
The Equivalence of the Subregion Representation and the Wall Representation for a Certain Class of Rectangular Dissections.
Communications of the ACM 31(6):752-763, June, 1988.

[Laird 84] John E. Laird.
Universal Subgoaling.
Thesis CMU-CS-84-129, Carnegie-Mellon University, May, 1984.

[Lawler 66] E. L. Lawler and D. E. Wood.
Branch-and-Bound Methods: A Survey.
Operations Research 14:699-719, 1966.

[Liggett 81] Robin S. Liggett and William J. Mitchell.
Optimal Space Planning in Practice.
Computer Aided Design 13(5):277-288, September, 1981.

[Linden 88] Theodore Linden.
Plan and Goal Representations, In 'DARPA Santa Cruz Workshop on Planning'.
AI Magazine :119-122, Summer, 1988.

[Mackworth 85] Alan K. Mackworth, Jan A. Mulder, and William S. Havens.
Hierarchical Arc Consistency: Exploiting Structured Domains in Constraint Satisfaction Problems.
Computer Intelligence 1:118-126,1985.

[Manheim 64] Marvin L. Manheim.
Hierarchical Structure: A Model of Design and Planning Processes.
M.I.T. Report No. 7, Massachusetts Institute of Technology, Cambridge, MA, 1964.

[Mcdennott 78] McDermott, Drew.
Planning and Acting.
Cognitive Science (2):71-109,1978.
theory of problem solving, non-complete planning.

[Michalski 83] Ryszard S. Michalski and Robert E. Stepp.
Learning From Observation: Conceptual Clustering.
Machine Learning: An Artificial Intelligence Approach.
Tioga Publishing Company, Palo Alto, CA, 1983, pages 331-364, Chapter 11.

[MiUer60] G.A. Miller, E. Galanter, and K.H. Pribram.
Plans and the Structure of Behavior.
Henry Holt and Company, New York, 1960.

[Minsky 63] Marvin Minsky.
Steps Toward Artificial Intelligence.
Computers and thought.
McGraw-Hill, New York, NY, 1963, pages 406-450.

[Minton 88] Steve Minton.
Learning Search Control Knowledge: An Explanation-Based Approach.
PhD Thesis, Carnegie-Mellon University, 1988.

[MitcheU 77] William J. Mitchell.
Computer-Aided Architectural Design.
Petrocelli/Charter Publishers, Inc., New York, New York, 1977.

[Mitchell 85] Tom M. Mitchell, Louis I. Steinberg, and Jeffrey S. Shulman.
A Knowledge-Based Approach to Design.
IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI-7(5):502-510, September, 1985.

[Mittal 86a] Sanjay Mittal and Mark Stefik.
Constraint Compaction: Managing Computational Resources for Efficient Search.
Technical Memo, Xerox Palo Alto Research Center, April, 1986.

[Mittal 86b] Sanjay Mittal and Agustin Araya.
A Knowledge-Based Framework for Design.
In *Fifth International Coherence on Artificial Intelligence,* pages 856-865. AAAI, Philadelphia, PA, August, 1986.

[Mittal 86c] Sanjay Mittal, Clive L. Dym, and Mahesh Morjaria.
PRIDE: An Expert System for the Design of Paper Handling Systems.
Computer :102-114, July, 1986.

[Mittal 87] Sanjay Mittal and Felix Frayman.
Making Partial Choices in Constraint Reasoning Problems.
In *AAAI '87 Proceedings - Volume* 2, pages 631-636. AAAI, July, 1987.

[Mostow 85] J. Mostow.
Toward Better Models of the Design Process.
AI Magazine 6(l):44-57,1985.

[Mostow 87] Jack Mostow and Mike Barley.
Automated Reuse of Design Plans.
In *International Conference on Engineering Design*, pages 632-647. Boston, MA, August, 1987.

[Newell 72] A. Newell and H. A. Simon.
Human Problem Solving,
Prentice-Hall, Englewood Cliffs, NJ, 1972.

[Newell 76] Newell, Allen and Simon, Herbert.
Computer Science as Empirical Inquiry: Symbols and Search.
acm, March, 1976.

[Newell 82] Allen Newell.
The Knowledge Level.
Artificial Intelligence 18:87-127,1982.

[Newell 83] Allen Newell.
Intellectual Issues in the History of Artificial Intelligence.
The Study of Information: Interdisciplinary Messages.
John Wiley and Sons, New York, New York, 1983.

[Nii 86] H. Penny Nii.
Blackboard Systems: The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures.
The AI Magazine :38-53, Summer, 1986.

[Nilsson71] N. J. Nilsson.
Problem Solving Methods in Artificial Intelligence.
McGraw-Hill, New York, NY, 1971.

[Orelup 87] Mark F. Orelup; John R. Dixon; and Melvin K. Simmons.
Dominic II: More Progress Towards Domain Independent Design by Iterative Redesiga
In *WAM*, pages 1-14. ASME, December, 1987.

[Oxman 87] Rivka Oxman and John S. Gero.
Using an Expert System for Design Diagnosis and Design Synthesis.
Expert Systems 4(l)A-l5$_t$ February, 1987.

[Pfefferkom75] Charles E. Pfefferkom.
The Design Problem Solver A System for Designing Equipment or Furniture Layouts.
Spatial Synthesis in Computer-Aided Building Design.
John Wiley and Sons, New York, New York, 1975, pages 98-146, Chapter 5.

[Polya 57] G. Polya.
How to Solve It.
Doubleday Publishers, Garden City, NY, 1957.

[Rich 83] Rich, Elaine.
Artificial Intelligence.
McGraw Hill, Inc., N.Y., N.Y., 1983.

[Rieger77] Chuck Rieger and Phil London.
Subgoal Protection and Unravelling During Plan Synthesis.
IJCAI :487-93, 1977.

[Rosenberg 86] Jerry M. Rosenberg.
Dictionary of Artificial Intelligence and Robotics.
John Wiley and Sons, New Yoik, New Yoik, 1986.

[Rosenbloom 86] Paul S. Rosenbloom, John E. Laird, Allen Newell, Andrew Golding, and Amy Unruh.
Current Research on Learning in Soar.
The Kluwer International Series in Engineering and Computer Science. Volume Knowledge Representation, Learning and Expert Systems.Afac/iMe *Learning: A Guide to Current Research.*
Kluwer Academic Publishers, Boston, MA, 1986, pages 281-290.

[Rychener 83] M. D. Rychener.
Expert Systems for Engineering Design: Problem Components, Techniques, and Prototypes.
Technical Report DRC-05-02-83, Carnegie-Mellon University, December, 1983.

[Sacerdoti 74] Sacerdoti, E.D.
Planning in a Hierarchy of Abstraction Spaces.
Artificial Intelligence 5(2): 115-135,1974.

[Sacerdoti 75] Sacerdoti, Earl D.
The Non-Linear Nature of Plans.
In *ICJAI-75,* pages 181-188. icjai, 1975.
planning, STRIPS, procedural net, partial ordering of plans.

[Sacerdoti 77] Eari D. Sacerdoti.
Artificial Intelligence Series: A Structure for Plans and Behavior.
Elsevier North-Holland, Inc., New York, 1977.

[Sacerdoti 80] Sacerdoti, Earl D.
Problem Solving Tactics.
AI Magazine 3(1):7-15, Winter, 1980.
Summary of planning tactics in problem solving.

[Sato 80] Keiichi Sato and Charles L. Owen.
A Prestructuring Model for System Arrangement Problems.
In *Proceedings of the 17th Design Automation Conference,* pages 226-236. IEEE, Minneapolis, MINN, June, 1980.

[Schank 77] R.C. Schank, and R.P. Abelson.
Scripts, plans, goals, and understanding.
Lawrence Erlbaum, Hillsdale, N.J., 1977.

[Sheil 83] B. SheiL
Power Tools for Programmers.
Datamation: 131-144,1983.

[Siklossy 73] L. Siklossy and J. Dreussi.
An Efficient Robot Planner which Generates its own Procedures.
In *IJCAI-73,* pages 423-30. UCAI, 1973.

[Siklossy 77] L. Siklossy and Clive Dowson.
The Role of Preprocessing in Problem Solving Systems.
In *IJCAI-5,* pages 465-471. UCAI, Cambridge, MA, 1977.

[Simon 75] Simon, H.
Style in Design.
Spatial Synthesis in Computer-Aided Building Design.
John Wiley and Sons, New York, N. Y., 1975.

[Simon 81] Herbert A. Simon.
The Sciences of the Artificial.
The MIT Press, Cambridge, MA, 1981.

[Stallman 77] R. Stallman and G. Sussman.
Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis.
Artificial Intelligence 9:135-196, 1977.

[Stefik 81a] Stefik, Mark.
Planning with Constraints (MOLGEN: Part 1).
Artificial Intelligence 16(2):111-140, 1981.

[Stefik 81b] Stefik, Mark.
Planning and Meta-Planning (Molgen: Part 2).
Artificial Intelligence 16(2):141-170, 1981.

[Sussman 75] Gerald J. Sussman.
A computer model of skill acquistion.
American Elsevier, New York, 1975.

[Swartout 88] William Swartout ed.; Thomas Dean; Theodore Linden; Richard Fikes; Peter Cheeseman; Drew McDermott.
DARPA Santa Cruz Workshop on Planning.
AI Magazine :115-130, Summer, 1988.

[Takewaki 85] Toshiaki Takewaki; Taizo Miyachi; Susumu Kunifuji; and Koichi Furukawa.
An Algebraic Manipulation System Using Meta-level Inference Based on Human Heuristics.
ICOT Technical Report TR-140, ICOT Research Center, Institute for New Generation Computer Technology, October, 1985.

[Tate 75] Austin Tate.
Interacting Goals and Their Use.
IJCAI :215-18, 1975.

[Tate 77] Austin Tate.
Generating Project Networks.
IJCAI :888-93, 1977.

[Tate 85] Austin Tate.
A Review of Knowledge-Based Planning Techniques.
In M. Merry (editor), *Expert Systems 85*, pages 89-111. British Computer Society at Warwick University, UK, Cambridge University Press, December, 1985.
Also published in Knowledge Engineer's Review, Vol. 1, No. 2, June 1985, published by the British Computer Society Specialist Group on Expert Systems. Also published as AIAI-TR-9, AIAI, Edinburgh University.

[Tenenberg 86] Josh Tenenberg.
Planning With Abstraction.
aaai , 1986.

[Tommelein 87a] I. D. Tommelein, M. V. Johnson, Jr., B. Hayes-Roth, and R. E. Levitt.
SightPlan: An Expert System for the Layout of Temporary Facilities on a Construction Site.
In *IFIP WG 5.2, Working Conference on Expert Systems in Computer-Aided Design.* EFDP, Sydney, Australia, February, 1987.

[Tommelein 87b] Iris D. Tommelein, Raymond E. Levitt, Barbara Hayes-Roth.
Using Expert Systems for the Layout of Temporary Facilities on Construction Sites.
Managing Construction Worldwide.
E. and R N. Spon, London, England, 1987, pages 566-577.

[Tong 87] Christopher Tong.
Toward an engineering science of knowledge-based design.
AIfor Engineering, Vol.2, No.3 , July, 1987.

[Vere 83] Vere.
Planning in Time: Windows and Durations for Activities and Goals.
In *Trans. PAMI,* pages 246-267. IEEE, 1983.

[Waldinger77] Waldinger, Richard.
Achieving Several Goals Simultaneously.
Machine Intelligence 8.
Ellis Horwood Limited, Chichester, England, 1977, pages 94-136, Chapter 6.

[Warren 74] David H.D. Warren.
WARPLAN: A System For Generating Plans.
Technical Report 76, University of Edinburgh, June, 1974.

[Weyhrauch 80] R. W. Weyhrauch.
Prolegomena to a Theory of Mechanized Formal Reasoning.
Artificial Intelligence 13:1-2,1980.

[Wilensky81] Wilensky, Robert.
Meta-Planning: Representing and Using Knowledge About Planning in Problem Solving and Natural Language Understanding.
Cognitve Science 5:197-233,1981.
metaplanning.

[Wilensky 83] Robert Wilensky.
Planning and Understanding.
Addison-Wesley, Reading, MA, 1983.

[Wilkins 84] David E. Wilkins.
Domain-independent Planning: Representation and Plan Generation.
Artificial Intelligence 22:269-301,1984.

[Woodbury 87] R. Woodbury.
The Knowledge Based Representation and Manipulation of Geometry.
PhD Thesis, Carnegie-Mellon University, December, 1987.

[Zozaya 88] Carlos Antonio Zozaya-Gorostiza.
Knowledge-Based Planning for Construction Projects.
PhD Thesis, Carnegie-Mellon University, April, 1988.