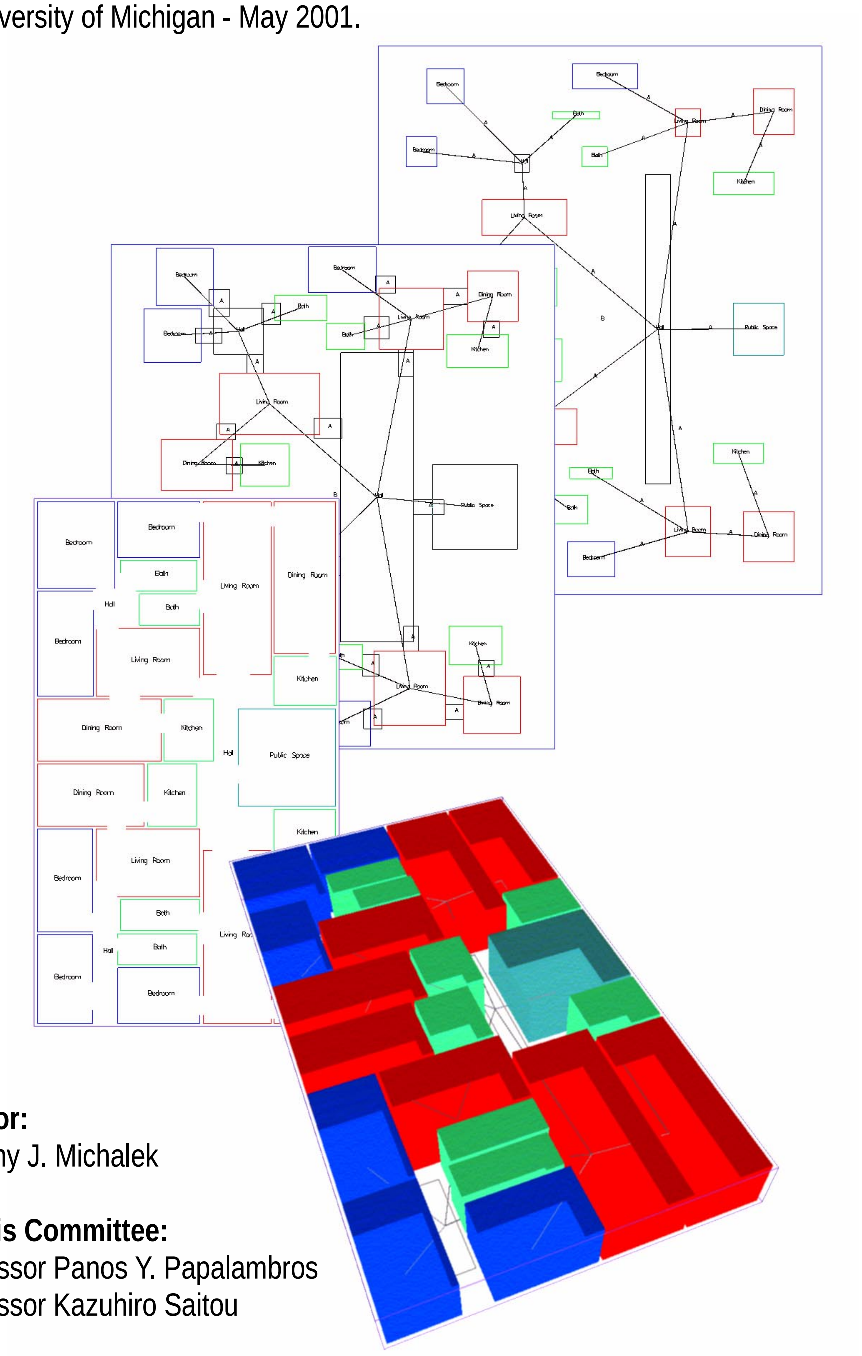
Interactive Layout Design Optimization

An interactive optimization tool for architectural floorplan layout design.

Submitted in partial fulfillment of the requirements for the degree of
Master of Science in Mechanical Engineering
at the University of Michigan - May 2001.



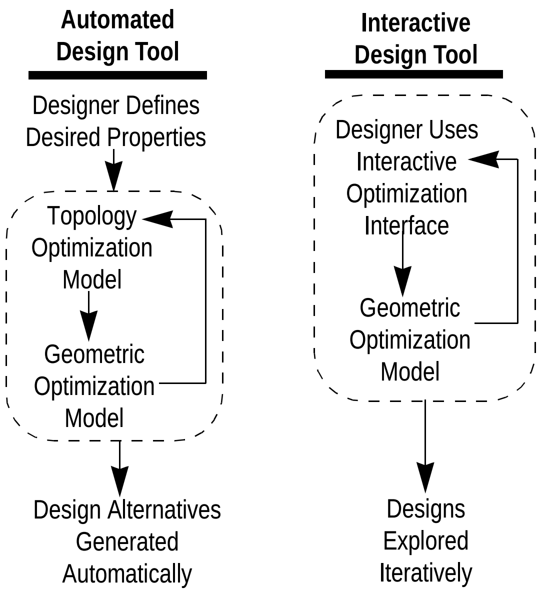
Author:
Jeremy J. Michalek

Thesis Committee:
Professor Panos Y. Papalambros
Professor Kazuhiro Saitou

ABSTRACT

Many areas of design involve both quantifiable and subjective goals, preferences, and constraints. Subjective aspects of design are typically ignored in optimization models because they are difficult to model with mathematics; however, they are extremely important in areas such as product design and architectural design. The objectives of this thesis are (1) to formulate quantifiable aspects of architectural floorplan layout design using computational optimization algorithms, (2) to provide a method for integrating mathematical optimization with human decision making, and (3) to develop the use of optimization techniques as a tool to aid early conceptual design. Two design tools have been developed: an *automated tool* and an *interactive tool*.

The *automated tool* uses a decomposition strategy to separate topological decisions (discrete design decisions) from purely geometric decisions (sizing and placement). The designer specifies desired design characteristics of the building, and the program automatically generates a population of feasible, goal-directed design alternatives.


Automated Design Tool
Designer Defines Desired Properties
Topology Optimization Model
Geometric Optimization Model
Design Alternatives Generated Automatically
Interactive Design Tool
Designer Uses Interactive Optimization Interface
Geometric Optimization Model
Designs Explored Iteratively

The *interactive tool* uses an object oriented representation with an interface that allows the designer to interact with the building layout optimization problem. Using the interactive tool, the designer can refine the problem definition on-the-fly and quickly explore solution alternatives and trade-offs while receiving both visual and computational feedback. By interacting with the optimization process, the designer can guide global search and take unmodeled preferences into account. This interactive approach is a novel use of optimization methods as an exploratory sketching tool for the early conceptual design phase.

Acknowledgements

This thesis work was completed with significant contributions from Ruchi Choudhary *(University of Michigan School of Architecture)*, and John Whitehead, Panayiotis Georgeopolous, and Adam Cooper of the Optimal Design Laboratory *(University of Michigan Mechanical Engineering Department)*. Special thanks to my thesis advisor, Panos Papalambros, and committee member Kazu Saitou who contributed much support and knowledge, and the entire Optimal Design Laboratory (including Brian Jensen) who encouraged, supported, and provided useful feedback.

TABLE OF CONTENTS

List of Figures

List of Tables

CHAPTER 1

OVERVIEW

Computational design tools for spatial layout planning present perhaps the most comprehensive challenges in the area of architectural design computation. Spatial design tools are the common ground where design representation, generation, evaluation and decision-making are required to be addressed simultaneously for the goal of realizing meaningful design exploration tools. Additionally, a multitude of ill-posed design intentions, non-explicit goals, and the non-deterministic nature of the design process itself add to the problem complexity. Given the difficulties, the problem of architectural layout design continues to challenge researchers from all areas of design computation.

Reported attempts to automate the process of layout design started over 35 years ago [1]. Researchers have used several problem representations and solution techniques to describe and solve the problem (details of previous research models are discussed in Chapter 2). This thesis presents an alternative automated layout method that generates goal-directed design alternatives given a set of design objectives and constraints.

Architectural design involves a mix of quantifiable and subjective goals, preferences and constraints. Aesthetic preferences and other subjective aspects of designs are typically ignored in automated models because these aspects are difficult to model with mathematics. Designers generally explore subjective aspects during the conceptual design phase by sketching and comparing design alternatives. Very few CAD packages address the needs of designers during this initial conceptual exploratory phase of design. Chapter 2 discusses this further. This thesis presents a novel interactive design tool that uses optimization to help the designer quickly generate and compare designs using visual and computational feedback to understand design trade-offs.

This chapter gives an overview of the work presented in the thesis. Two optimization models (a *geometry model* and a *topology model*) are introduced, and two design tools (an *automated tool* and an *interactive tool*) that use the optimization models are discussed.

1. 1 Optimization Models

Two separate optimization models have been developed to model different parts of the building layout design problem. The geometric optimization model defines position and size variables for each room, and the design variables have continuous domains. The topology optimization model defines design decision variables including room connectivity and rough position, and the design variables have discrete domains.

1. 1. 1 Geometric Optimization Model

The geometric optimization model is used in both the automated design tool and the interactive design tool. The model is discussed in detail in Chapter 3. The geometric floorplan layout problem is posed as a process of searching for the best location and size of a group of interrelated rectangular units. This representation assumes that most architectural floorplans can be described as combinations of rectangular shapes (including L-shapes, etc.). The problem is formulated mathematically with a set of design variables representing layout dimensions. Design objectives and constraints are formulated as functions of the design variables. Several optimization algorithms are used to solve for optimum layout geometry. The layout geometry optimization determines room position and size (topology is fixed during optimization). Figure 1 shows a sample layout geometry.

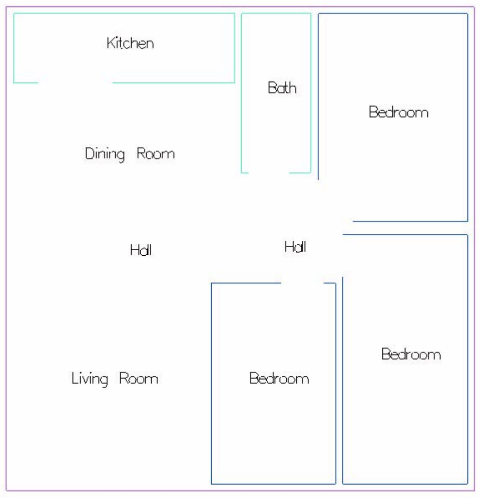


Figure 1. Sample layout geometry

1. 1. 2 Topology Optimization Model

The topology optimization model is used together with the geometry model in the automated design tool. The model is discussed in detail in Chapter 4. The topology layout problem is posed as a process of searching for the set of room connectivities and rough locations that yields the best geometry. The topology defines room connections (by doorway or accessway), rooms that lie on external walls, and the rough placement of rooms inside the building bounds. The problem is formulated with a set of discrete design variables (representing topology decisions) and design constraints. An evolutionary algorithm is used to search for high quality designs that meet all specifications (design 'quality' refers to the objective function value of the resulting geometric layout). Figure 2 shows an example topology. In this figure nodes represent room positions, and lines represent connections between rooms.

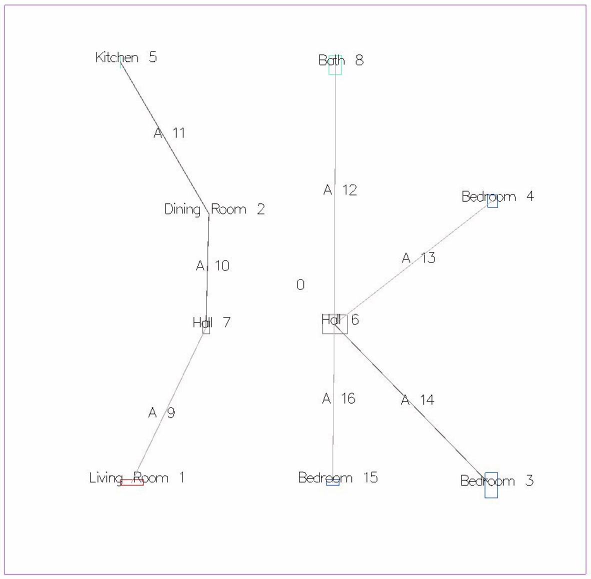


Figure 2. Sample layout topology

Theoretically, topologies could be evaluated based on topological objectives, such as openness, proximity, directionality, or symmetry; however, even though these aspects are often thought of as topological, they are difficult to evaluate without geometry. However, any design objective can be evaluated in a geometric layout because the geometry defines the layout completely (the geometric layout treats connectivity and materials as fixed values). This is why each topology is evaluated based on the best geometry that it can produce.

1. 2 Design Tools

Two design tools have been developed to assist a designer in exploring solutions to building floorplan layout problems: an *automated design tool*, and an *interactive design tool*.

1. 2. 1 Automated Design Tool

The mechanics of the automated design tool are discussed in detail in Chapter 3 and Chapter 4. By combining the topology and geometry models, the automated design tool can be used to generate high-quality design alternatives for consideration. In this process, the designer specifies topological and geometric constraints and objectives. For example, the designer might specify that the living room and bedrooms must be adjacent to external walls for natural lighting, there must be valid paths from every room to the entryway, and the kitchen must be connected to the dining room. The topology optimization algorithm will search for topology alternatives that satisfy these constraints (Chapter 4). Each valid topology is then translated into a local optimal geometric layout using the geometric optimization algorithm (Chapter 3). This process is shown in Figure 3. The topology algorithm searches for the topology that results in the best geometry, and the algorithm generates a population of feasible layout design alternatives. This automated design generator helps the design process because it formalizes design objectives and constraints, it can assist the search for solution alternatives, and it has the potential to generate novel designs that are not biased by the same set of assumptions and strategies that human designers use.

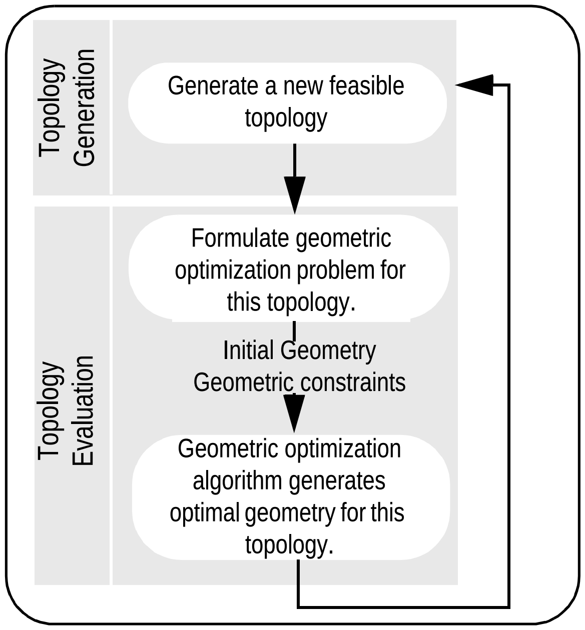

Topology Generation
Generate a new feasible topology
Topology Evaluation
Formulate geometric optimization problem for this topology.
Initial Geometry
Geometric constraints
Geometric optimization algorithm generates optimal geometry for this topology.

Figure 3. Automated building layout optimization method

Topology algorithm searches for the topology that results in the best geometry

1. 2. 2 Interactive Design Tool

Automated design generation tools are sometimes not adequate for problems such as architectural design for several reasons. Architectural design involves many subjective decisions about aesthetics or other preferences that are difficult to model or quantify mathematically. For design qualities that can be well defined, it is often difficult to foresee all issues that may affect the optimization model before observing some results. The usefulness of such a tool can be greatly improved if the designer is given the opportunity to refine the problem definition during the optimization process while receiving feedback.

In addition to modeling issues, automated tools face computational difficulties. Currently, the only methods that can compute or verify the global optimum of a function involve systematic exhaustive search with some kind of tree-pruning (branch and bound / constraint satisfaction programming (CSP) forward checking). These methods suffer from combinatorial explosion in large, highly constrained problems such as building layout. Other stochastic methods explore a smaller subset of the design space, and they can often find good solutions quickly; however, these algorithms do not guarantee solution quality in finite time because of their stochastic nature. Strict global optimality is less important in ill-defined problems like architectural design, which generally have some design preferences that are not well represented in the model. Instead, it is important that the designer is able to explore and compare high quality alternatives. Allowing the designer to use experience and intuition to guide the search can improve search time relative to well-defined objectives and constraints as well as take into account unmodeled objectives and constraints. In addition, the designer can assist gradient algorithms that may have computational difficulties for non-smooth objective and constraint functions by guiding the design away from first-order discontinuities.

A building geometry optimization design tool was created to allow the designer to interact with the problem in a number of ways, including interactively defining the problem, guiding the search for a solution, and exploring design alternatives (see Figure 4). This tool offers a new, powerful approach to using optimization in the design process. Instead of using optimization in the final design stages to fine-tune a solution to a well-defined problem, optimization is used to help the designer define the problem and to explore solution alternatives and trade-offs interactively, while receiving both visual and computational feedback. If the designer changes the problem formulation, the new formulation is automatically updated, and search in the new design space begins with the last design found before the change. The interactive design tool is discussed in Chapter 5.

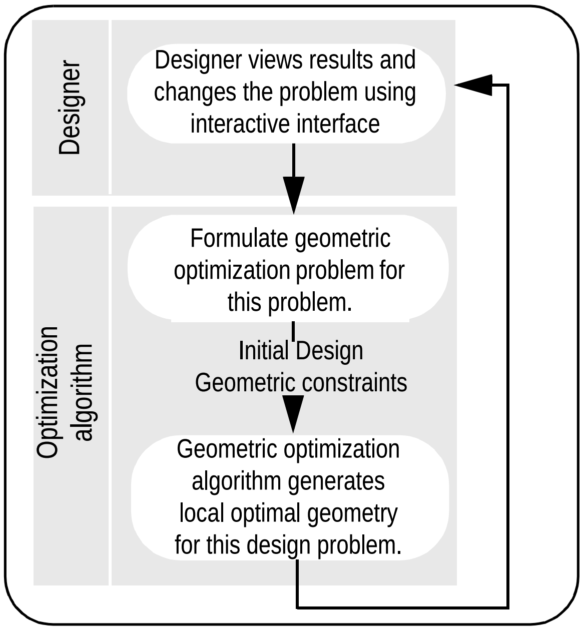

Designer
Designer views results and changes the problem using interactive interface
Optimization algorithm
Formulate geometric optimization problem for this problem.
Initial Design
Geometric constraints
Geometric optimization algorithm generates local optimal geometry for this design problem.

Figure 4. Interactive building layout optimization method

Designer uses feedback from the algorithm to refine or change the problem definition.

BACKGROUN

In this chapter, the automated layout design tool presented in the thesis is contrasted with other research attempts to automate layout design. The new tool offers a novel approach, and advantages are discussed. The interactive design tool is contrasted with other interactive optimization methods. It is compared to other CAD systems as a tool for design exploration in the early, conceptual phase of design, and it is presented as a novel way to use optimization in this early stage.

2. 1 Automated Spatial Configuration

Spatial configuration optimization is concerned with finding feasible locations and dimensions for a set of interrelated objects that meet all design requirements and maximize design quality in terms of design preferences. Spatial configuration is relevant to all physical design problems, and so it is a very important area of research. Research work on automation of spatial configuration includes component packing [16]-[18], route path planning [23], process and facilities layout, VLSI design [21][22], and architectural layout [1]-[13]. Architectural layout is particularly interesting because in addition to common engineering objectives such as cost and performance, architectural design is especially concerned with aesthetic and usability qualities of a layout, which are generally more difficult to describe formally. Also, the components in a building layout (rooms or walls) often do not have pre-defined dimensions, so every component of the layout is resizable.

Reported attempts to automate the process of layout design started over 35 years ago (Levin 1964 [1]). Researchers have used several problem representations and solution search techniques to describe and solve the problem. Table 1 outlines some of the major contributions to this field.

Table 1: Outline of Research in Automated Building Layout

Authors	System	Representation	Solution Strategy
Liggett Mitchell[2]		Fixed Grid Space Allocation	Constructive placement followed by iterative improvement
Sharpe Marksjo[3]	TOPAZ	Fixed Grid Space Allocation	Simulated annealing
Jo Jagielski Gero[4][5]	EDGE	Fixed Grid Space Allocation using an ordered schema	Evolutionary algorithms
Baykan Fox[6]	WRITE	Topology: search through combinations of disjunctive constraints Geometry: cartesian coordinates of each room edge. Constraints are defined by the specific topology	Topology: CSP enumeration techniques (backtracking) Geometry: CSP enumeration techniques (backtracking)
Schwarz Berry Saviv[7]	ABD	Topology: decision variables define specific topological relationships that define geometric constraints Geometry: Two constraint graphs define wall locations in x-dir and y-dir. Constraints are functions of decision variables	Topology: Enumerative or heuristic CSP search techniques Geometry: CSP enumeration techniques
Medjeoub Yannou[8]	ARCHi PLAN	Topology: specific room adjacency and room proximity define geometric constraints Geometry: Room coordinates, length and width	CSP solution enumeration techniques
Michalek Choudhary Papalambros		Topology: room connectivity defines geometric constraints, and rough placement is used only as a starting point for geometric optimization Geometry: room coordinates, length and width	Topology: Evolutionary algorithms Geometry: SQP

2. 1. 1 Fixed Grid Space Allocation

One approach to spatial allocation is to define the available space as a set of grid squares and use an algorithm to allocate each square to a particular room or activity [2]-[5] (see Figure 5). This problem is inherently discrete, non-linear, and multi-modal. Because of the combinatorial complexity, it cannot be solved exhaustively for reasonably-sized layout problems. Several heuristic strategies have been developed to find solutions without searching the design space exhaustively.

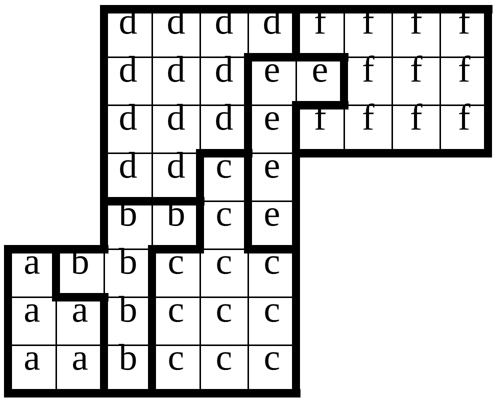


Figure 5. Sample fixed grid allocation layout

Liggett and Mitchell [2] use a *constructive placement* strategy, where space is allocated for rooms one at a time based on the best probable design move at each step. Using this strategy, squares will be completely allocated for the first room before the other rooms are considered. One by one, the rooms are allocated enough squares on the grid to meet size requirements. The placement of each room is guided by a probability model that tries to minimize cost in terms of cost of travel between spaces. Once a design is complete by placing all of the rooms, the algorithm uses an *iterative improvement* strategy to find a local optimum. In this phase, the space allocation is altered slightly to improve the design objectives. This heuristic approach is useful in many situations; however, it does not guarantee global quality or even feasibility.

Sharpe and Marksjo [3] use a metropolis (simulated annealing) algorithm to explore the space of potential allocations globally and stochastically. More recently, Gero [4][5] has used an ordered schema to describe the allocation of space and an evolutionary algorithm (genetic algorithm) to search for solutions. These stochastic solution strategies have more potential to search the design space globally. Gero was able to produce improved solutions to layout problems posed by Liggett and Mitchell as well as other layout problems.

The fixed grid allocation approach is a successful approach for allocating a pre-defined space into rooms or activities. This approach can be used for applications such as to redistribute activities in an office building during a reorganization, or to distribute activities in a newly purchased building. The approach may also be successful in generating new buildings with variable boundaries; however, this has not yet been reported.

2. 1. 2 Decomposition of Topology and Geometry

Another approach to representing the building layout design space is to decompose the problem into two parts: topology and geometry. Topology refers to logical relationships between layout components. Geometry refers to the position and size of each component in the layout. Topological decisions define constraints for the geometric design space. For example, a topological decision that *room i* is adjacent to the north wall of *room j* restricts the geometric coordinates of *room i* relative to *room j*.

Baykan and Fox [6] developed a system using Constraint Satisfaction Programming (CSP) techniques[1] to enumerate solutions to a graph representation of the layout design space. The topology space is described by defining which rooms are adjacent. Adjacency is represented as sets of disjunctive constraints, for example,

room i ***adj-to*** *room j*

would be represented as

(room i ***adj-to-north-of*** *room j) OR*

(room i ***adj-to-south-of*** *room j) OR*

(room i ***adj-to-east-of*** *room j) OR*

(room i ***adj-to-west-of*** *room j).*

1. To learn more about CSP formulation and solution techniques, see reference [37].

The space of topologies is searched using CSP backtracking to enumerate all feasible combinations of topological constraints.

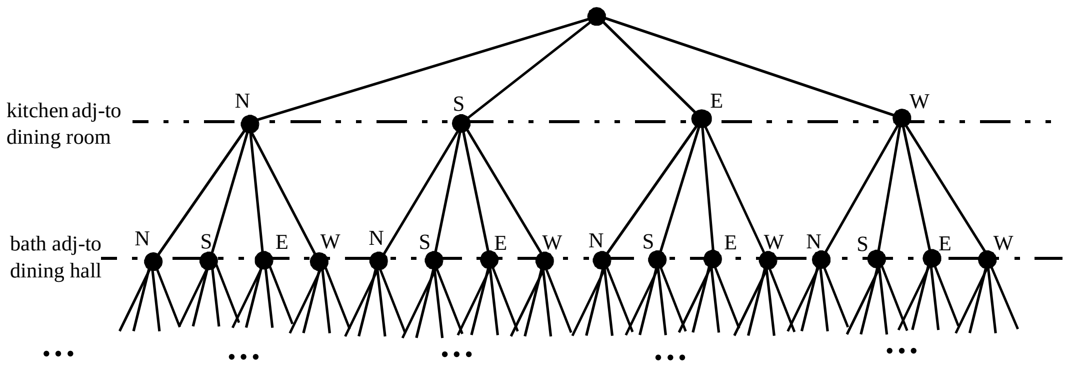


Figure 6. Sample search tree of topology combinations

Each topology combination is searched for feasible combinations of room edge coordinates [(x_1, y_1), (x_2, y_2)] that satisfy the constraints using backtracking. This approach is complete (will always find the solution if one exists); however, search time and space are intractable for large problems (a studio apartment layout is used as a tractable example).

Medjeoub and Yannou [8] have a similar representation, but first they enumerate all topologies that produce at least one feasible geometry. The designer is then able to review the topological possibilities and select those which s/he wants to explore geometrically. This approach can handle moderately-sized problems (up to twenty spaces including stairs and halls).

Schwarz, Berry, and Saviv [7] describe the same basic combinatorial topology search with a vector of decision variables, and geometric solutions are searched for each topology. Geometry is represented using two constraint graphs that describe wall positions in the x-direction and in the y-direction. Wall positions are constrained based on the topological decisions (decision variable values). They have shown success for small problems (up to nine rooms).

2. 1. 3 What is still needed

Existing solutions to the automated building layout problem are varied. Each representation and solution method has its own set of biases and assumptions. Many research attempts have yielded impressive results; however, there is still a need for improvements and additions.

1. Successful generation of global quality solutions has been achieved for medium-sized problems; however, there is still a need for a strategy that can handle larger problems computationally.
2. In all of the present decomposed solutions, the designer must specify which rooms are adjacent. In real design situations, specific adjacencies may be compromised, as long as there exists an acceptable pathway from one room to another. For example, in a house it may be important that there is a valid pathway from the bedroom to the foyer that does not pass through bathrooms or closets; however, the exact adjacency path may be flexible. There is a need for a system that can handle these path requirements without adjacency requirements.
3. Computational evaluation speed is a major drawback. It would be useful to take advantage of the speed of gradient-based algorithms on the geometric aspects of the layout, because they have a continuous variable nature. Gradient-based algorithms can efficiently handle large sets of constraints and objectives, especially those that can be represented linearly.
4. There is a need to generate layouts that can be easily manipulated and altered by a designer. It is rare that a designer would simply choose a computer generated layout without altering it, because many subjective architectural considerations have not been formalized into the mathematical model used by the computer. The designer should be able to quickly make changes to generated layouts.

The tool presented in this work addresses all of these concerns. The interactive design tool can work with much larger geometric layouts than reported in the literature, and the topology layout optimization can handle problems as large as those reported in the literature [8]. Generic path constraints have been implemented, the computational efficiency of gradient algorithms has been used where appropriate, and the design tool allows easy manipulation by designers.

2. 2 Interactive Optimization

2. 2. 1 Interactive Multi-Objective Optimization

Most work on interactive optimization focuses on multi-objective optimization. This is because preferences are expressed a posteriori; after some idea of the trade-off is gained from comparison of results. Optimization algorithms are generally designed to maximize or minimize a single objective function. Multi-objective interactive techniques use some variation of the Interactive Weighted Tchebycheff (IWT) approach [50]. In this approach the multi-objective problem is converted into a single objective problem by minimizing the weighted sum of distances to an ideal point. Figure 7 shows an example of this situation. Here, f_1 and f_2 are two competing objective functions. An optimization algorithm will generate different points on the Pareto curve depending on how the objectives are weighted. After obtaining a solution, the designer alters the weights (the relative importance) of each objective to tune trade-off preferences and move along the Pareto set. This is often an insufficient approach because there is poor understanding of the relationship between the location of the desired Pareto points and the corresponding weights.

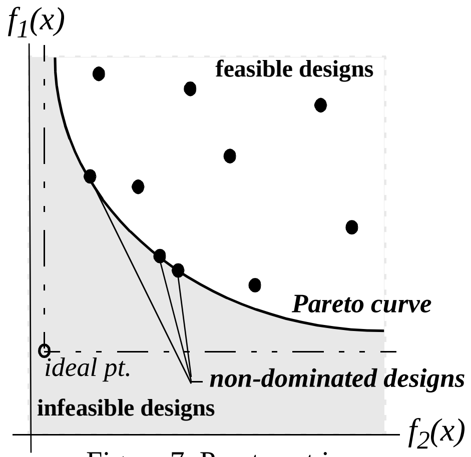


Figure 7. Pareto set in a multi-objective optimization problem

Several alternative methods involve the designer choosing an 'aspiration point': a design close to the Pareto set ([48], [47]). The algorithm then searches for a point on the Pareto set close to the aspiration point (see Figure 8). This is very important, because it allows the designer to interact with values that s/he can relate to physically, instead of an arbitrary weighting scheme. Other techniques have been developed, including an epsilon-

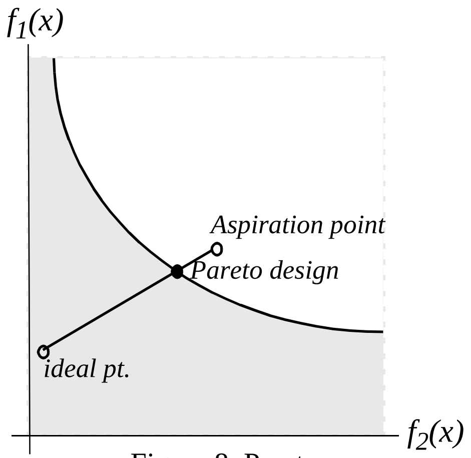


Figure 8. Pareto optimization using an aspiration point

inequality constraint method where new constraints are added to the problem based on the designer’s feedback (Azarm 1998 [50]).

In the thesis, the simple Interactive Weighted Tchebycheff approach with linear weights is used because the objective functions are generally not competing. More advanced interactive approaches for exploring the Pareto set could be added in the future to help with competing objectives. More importantly, the approach in this thesis expands on a concept known well in the multi-objective optimization literature -- that the designer must generally see some physical results in order to understand design trade-offs. In interactive multi-objective optimization, when the designer has seen results, s/he can use that information to alter the objective function definition by changing individual objective weights. This idea is expanded in the thesis, allowing the designer to see physical results of the entire design in real-time during optimization and to use this information interactively to change aspects of the problem statement or redirect search.

2. 2. 2 Interactive Design Space Exploration

In addition to weighted multi-objective optimization, some researchers have extended the interaction to a process where the designer can change the optimization model during the optimization process. *OptdesX*, a commercial optimization tool [52][53], allows the designer to monitor changes in the design variables during optimization. The designer can pause an optimization run to make changes in the design variables and move in the design space. For example, if the designer notices a variable, objective, or constraint value changing to an unacceptable value, s/he may realize immediately that the model has errors. Alternatively, if the designer notices that some of the design variables are ‘stuck’ in an undesirable region, the designer can stop the optimization run and ‘nudge’ the design by manually changing the value of the design variable. The change could result in moving search into a better local minimum. *OptdesX* offers a general platform for watching and interacting with the optimization process; however, visualization is limited to numerical information, and the designer cannot change the design problem itself without reprogramming and recompiling.

Tidd, Rinderle, and Witkin [55] developed a system for a designer to interact with design variables and resultant design behaviors. In this method, each design variable and behavior is represented in graphical form. By ‘pulling’ on the graphs as if they were physical objects, the designer influences the design objectives and changes the shape of the design space. The optimization algorithm reacts to this change and alters design variables accordingly to find a new local optimum. In this way, the designer can interact with the design problem in an intuitive way to understand design trade-offs from an entirely new perspective.

2. 2. 3 Interactive Building Layout Optimization

Arvin and House [54] created a physics-based system for architectural layout. In this system, rooms are connected with simulated springs and dampers, and a dynamic simulation is run to push and pull the rooms into position. The designer influences the design by adding springs or changing spring constants. It is unclear how cumbersome this process would be for an architect; however, it does give the designer a new way to explore the problem -- one where spaces (such as rooms) are the working elements instead of the lines used in sketching.

Liggett and Mitchell [45] used the probability model in their automated design tool to create an interactive design tool for building layout. Their fixed grid space allocation representation uses *constructive placement* to allocate rooms into the building space one at a time (see Section 2. 1. 1). Liggett and Mitchell developed an interactive design tool to provide graphic feedback to the designer on the probable ‘goodness’ of design moves at each step. The designer can then use this feedback, along with intuition and other considerations that are not be represented in the model, to guide the space allocation selection.

Because of the difficulty in modeling many architectural design preferences, many researchers feel that

> *“...attempts to use fully automated computer algorithms to solve the layout problem should be reexamined with a view of incorporating man’s visual capability into the procedures.” (Scriabin and Vergin [60])*

2. 2. 4 What is still needed

Integrating the computational power of optimization algorithms with the guiding power of human judgement is an extremely rich area for exploration in the design optimization field. Many design disciplines are concerned with aesthetic and usability issues that are difficult to quantify mathematically, but are easy to evaluate visually by a human. Also, many optimization algorithms get trapped in local minima or get diverted because of non-linearities. Building layout design in particular has both of these qualities, and a tool is needed that allows the designer to guide the search algorithms into areas of interest and away from computational traps. A human designer with a visual interpretation of movement in the design space has the potential to recognize computational traps and guide the search into preferred areas of the design space. The designer can also guide search into areas that are interesting for unmodeled subjective reasons.

This thesis work addresses these concerns with an interactive design tool that supports the designer by computationally optimizing aspects that can be modeled mathematically, while giving the designer control to make decisions influenced by subjective human judgement and intuition.

2. 3 Exploratory Design CAD Tools

Most CAD systems are very good at producing precise, accurate renderings of well defined designs. Many CAD systems also include detailed analysis packages for obtaining simulation feedback on the performance of a design. However, very few CAD packages address the needs of a designer during the initial conceptual exploratory phase of design when the problem is ill-defined, and the solution has not yet been decided.

> *The conceptual phase of design (or ideation phase) is the initial phase where the internal ideas of the designer are externalized or explored interactively and represented tentatively in some form using any medium... These are usually early sketches, rough mock-up models and concept renderings. Normally during the conceptual phase, designers quickly represent as many as possible different solutions in a short time. These are evaluated visually before exploring further possibilities. There are two important characteristics during this phase. First, the quick and intuitive representation of concepts. Secondly, the generation (in a very short time) of many different solutions and variations. Sketching is the most widely used imagery aid in evolving new ideas during the conceptual phase. (Stuyver and Hennessey [57])*

2. 3. 1 Ill-Defined Design Problems

The vast majority of design problems are ill-defined. In an ill-defined problem the initial constraints on the problem are not fully formulated. Resolving ill-defined problems is a process of searching for and refining a set of design constraints [61]. Few CAD tools exist to help a designer to refine design constraints.

Tidd, Rinderle and Witkin [55] developed a system for a designer to interact with design variables and resultant design behaviors (see Section 2. 2. 2). Arvin and House [54] created a similar physics-based system for architectural layout (see Section 2. 2. 3). These tools provide a glimpse of the potential of tools that help the designer to explore constraint options interactively during conceptualization.

2. 3. 2 Rapid Generation of Design Alternatives

Some commercial systems such as Working Model 2D allow designers to quickly model simple mechanisms to see how they will function. The designer can also receive feedback on forces and accelerations. Other drawing packages such as Alias, FormZ, and 3D Studio allow simple 3D shapes to be created and manipulated easily to explore basic forms.

Most architectural CAD tools are either drafting or analysis machines -- neither focusing on the rapid generation of design alteratives. Kharrufa, Saffo, Aldabbagh and Mahmood [46] developed an architectural CAD tool allowing the designer to interact with intuitive objects such as spaces instead of drawing lines. They suggested that

> *CAD could offer significant help... by presenting the architect with information such as expected cost and spatial allocation, which can improve the decision making. Furthermore, it could be used to simulate part of the building's expected performance in areas such as thermal loads and lighting (Kharrufa, Saffo, Aldabbagh and Mahmood [46]).*

Many architectural analysis tools exist for performing these kinds of simulations; however

> *The biggest stumbling block that prevents the computer being used to produce this extra information during the preliminary design stage is that the data concerning the building must first be input. This is a lengthy process that might defeat the aim of improving efficiency. (Kharrufa, Saffo, Aldabbagh and Mahmood [46])*

2. 3. 3 What is still needed

An interactive tool for early design stage conceptualization is needed that can

- help the designer to interactively refine the ill-defined design problem,
- provide an intuitive representation to interact with,
- provide a simple interface that enables rapid design and exploration of alternatives, and
- provide computational feedback on the performance of designs.

These needs are qualitative in nature. Nevertheless, they are important qualities for an exploratory design tool. The design tool presented in this thesis addresses all of these issues for architectural layout design. The interactive design tool provides assistance that allows the designer to refine the problem definition during exploration, and the use of optimization allows the designer to quickly generate high-quality layouts and receive both visual and computational feedback.

CHAPTER 3

OPTIMIZATION OF GEOMETRY

3. 1 Problem Formulation

The geometric optimization problem is posed as a process of finding the best location and size of a group of interrelated rectangular units. By formulating the geometric layout problem mathematically with a set of variables, objectives, and constraints, optimization algorithms can be used to solve for optimum layout designs. The geometric optimization problem has been formulated so that all objectives and constraints are continuous functions of the design variables, and all design variables have continuous domains. To the author’s knowledge, this is the only work in architectural layout design that uses this kind of formulation intended for gradient-based optimization.

3. 1. 1 Units

The layout problem is posed as a search for the best placement and size of a group of interrelated *Units* into a two dimensional cartesian space.

> ***Definition 1.*** *A* ***Unit*** *is a rectangular, orthogonal space defined to perform a specific architectural function.*

Examples of architectural functions include living spaces, storage spaces, facilities, and accessibility spaces.

There are several ways to represent a Unit mathematically. Figure 9 shows some alternative variable representations that were used as the research evolved. Each representation affects the shape of the design space and the gradient calculations used by the optimization algorithms, introducing bias into the solution strategy.

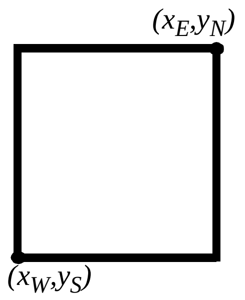


Figure 9.
(a)

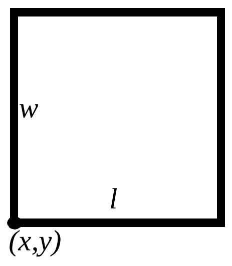


Figure 9.
(b)

An algorithm using model 9(a) can move each wall of a Unit independently of the others with a single variable; however, moving the entire Unit without altering its size requires changing at least two variables simultaneously. An algorithm using this representation may have trouble satisfying area constraints when a Unit needs to be moved.

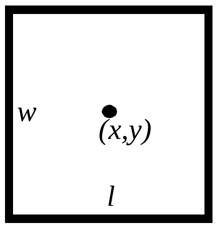


Figure 9.
(c)

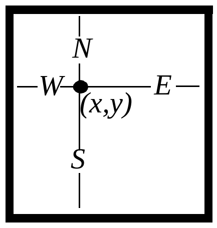


Figure 9.
(d)

Figure 9. Alternative variable representations of a Unit

Model 9(b) is unbalanced. The north and east walls can move independently, but the south and west walls cannot move directly without affecting the north or east walls.

An algorithm using model 9(c) can move a Unit without affecting its size; however, it cannot move any wall independently without changing at least two variables simultaneously. This representation may have trouble violating adjacency constraints when it needs to move walls.

Model 9(d) represents a Unit as a point in space *(x,y)*, and the perpendicular distance from that point to each of the four walls: *{N, S, E, and W}*. This model has more variables; however, it allows an optimization algorithm to change the position of a Unit independently without affecting its size (by changing x or y), and it can change any of the four wall positions independently (by changing *N, S, E,* or *W)*. Furthermore, *(x,y)* does not need to be the center of the Unit, and *{N, S, E, W}* need not be restricted to positive numbers. Although this model increases the problem dimensionality, it offers a lot of flexibility to make the best design moves at each step of optimization. To the author's knowledge, this is the only implementation of such a representation which has improved behavior of gradient algorithms in this problem.

Model 9(d) was adopted for most of the examples presented here (except where noted); however, the object-oriented implementation includes all of these representations, and it is easy to switch between them or add new representations.

3. 1. 2 Rooms

Definition 2. *A* ***Room*** *is a Unit that is used as a living space.*

A living space is any space that is considered to be used for sustained living activity. Typically this definition would not include pathways or hallways that are used for access, but may include closets or other Units. It is up to the designer to decide which Units should be Rooms. The differentiation between living space vs. non-living space is important only in optimization objectives to maximize the amount of space used for living and minimize all other space (see Section 3. 2. 15).

Figure 10 shows four different types of units which are discussed below. In Figure 10, “Living Room”, “Bedroom”, and “Bathroom” are Rooms.

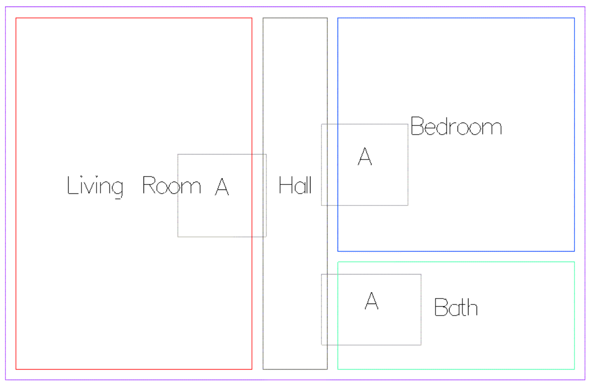


Figure 10. An example layout showing four different types of Units

3. 1. 3 Boundaries

> ***Definition 3.*** *A* **Boundary** *is a Unit that has other Units constrained inside itself and is not considered living space.*

Boundaries are used to group Units together. By default, the outer walls of the building are defined by a Boundary Unit, and all other Units are forced inside of this Boundary Unit.

In Figure 10, the large purple colored rectangle that defines the outer walls of the building is a Boundary.

3. 1. 4 Hallways and Accessways

> ***Definition 4.*** *A* **Hallway** *is a Unit with no physical walls that is not considered living space.*

> ***Definition 5.*** *An* **Accessway** *is a Hallway that is constrained to geometrically intersect two Units.*

Generally, a Hallway is used like a Room, but it does not function as living space. It functions only to provide a path between other rooms. Accessways are generally restricted to be small, and they are forced to intersect two other Units. They function to keep the two Units adjacent and connected, and to ensure that there is room for a door or opening.

In Figure 10, "Hall" is a Hallway. The Units labeled "A" are Accessways. Notice that each Accessway overlaps two other Units, ensuring access between those Units.

3. 1. 5 Windows

Units that are along external walls may have windows for natural lighting. Windows are generally added to Rooms and Hallways. By default, adding a window to one of the walls of a Room also adds an explicit constraint to force the Room against the respective wall of the building Boundary. It is then assumed that any room with a window is against an external wall for natural lighting calculations. Window height can be fixed for each Unit, and window width is a variable for each direction that has a window: ω_{N_i}, ω_{S_i}, ω_{E_i}, and ω_{W_i} represent the width of the north, south, east and west windows, respectively.

3. 1. 6 Constraints

The model formulation includes a toolbox of constraints that can be used to maintain relationships between Units. Design constraints have been developed to provide for the following relations:

Force Inside: *Used to force some Units inside of others, such as forcing all Rooms to be inside the building Boundary.*

Prohibit Intersection: *Used to prevent Rooms from intersecting and occupying the same space.*

Force Minimum Intersection: *Used to force Accessways to sufficiently overlap Rooms to guarantee space for a doorway or opening.*

Force To Edge: *Used to force Rooms against the edge of an external Boundary Unit to ensure feasibility of external doors for access or windows for natural lighting or emergency exit.*

Bound Size: *Used to provide bounds on acceptable length, width, and area of each Unit.*

Bound Ratio: *Used to bound Unit length-to-width ratios within an acceptable range.*

Feasible Window: *Used to ensure a window is small enough to fit on its wall.*

Bound Build Cost: *Used to provide bounds on acceptable estimated cost of building the structure.*

Bound Lighting: *Used to provide a bound on minimum acceptable natural lighting for each Room.*

Mathematical models of these constraints are explained in Section 3. 2.

3. 1. 7 Objectives

Several design objectives have been developed for measuring the quality of each design based on designer preference.

Minimize Heating Cost: *Minimize the estimated annual cost to heat the building.*

Minimize Cooling Cost*: Minimize the estimated annual cost to cool the building.*

Minimize Lighting Cost*: Minimize the estimated annual cost to illuminate the building.*

Minimize Wasted Space*: Minimize the amount of space contained in the building Boundary that is not occupied by living space.*

Minimize Accessway Size*: Minimize the size of Accessways to bring connected Units as close together as possible.*

Minimize Hallway Size*: Minimize the size of Hallways.*

The designer can choose a single optimization objective or may select several objectives and weight them in terms of importance. Mathematical models of each of these objective functions are explained in Section 3. 2.

3. 1. 8 Notes on an Earlier Model Formulation

The mathematical model used in this thesis work evolved as the project progressed. The final refined model has been described above, but some of the optimization tests used other earlier models to represent the problem. For completeness, earlier models will be described below and referenced when they were used.

In modeling Units, the model in Figure 9(b) was used for some of the early optimization runs. This was a reasonable model because initially it was assumed that the south and west walls of the building are external walls, and the north and east walls are not.

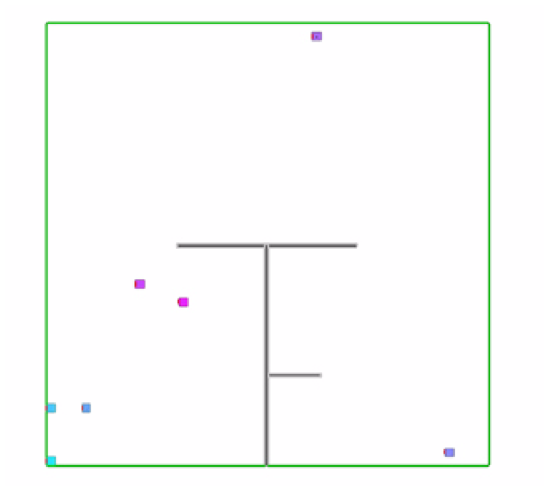


Figure 11. Example showing parameterized hall structure in early problem formulation.

The grey colored rectangles represent the hall structure inside the green building bounds. The colored squares represent Rooms inside the space.

In some of the earlier models, halls and accessways were not modeled as Units. Instead, hallways were represented by a parameterized hallway structure (see Figure 11). In this model, instead of Accessways, each Room has a door, represented by its center point. The door point is constrained to be inside the room and away from corners. Each door point is also constrained to connect with the hall structure. These constraints would force each door to an edge between its Room and the hall structure, guaranteeing connection and ensuring access to other Rooms and to an exit. The current model using Hallway Units and Accessway Units is more general, and it behaves better during optimization, so the early model will not be discussed further except to be referenced when used.

In addition, a mixed-discrete formulation was developed specifically for the hybrid SA/SQP solution method (Section 3. 4. 2). In this formulation, two discrete decision variables are added to each room. The first discrete variable, ϑ_{door}, determines which side of the Room the door is on {north, south, east, west}, and a continuous variable, *d*, determines where the door is located along that wall. The second discrete variable, ϑ_{ext}, determines if the Room will be forced against an external wall. In this model, it is assumed that the south and west walls are the only external walls, so the set of legal values for this variable is

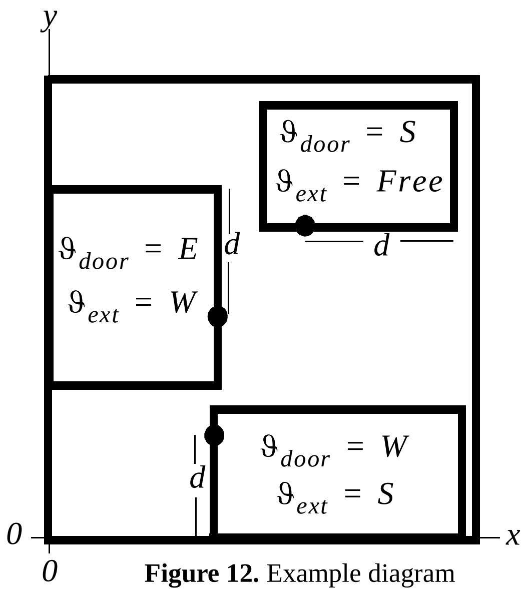


Figure 12. Example diagram showing the mixed-discrete early formulation

{south, west, free}. If $\vartheta_{ext} = south$, then the Room's y-coordinate variable is fixed. If $\vartheta_{ext} = west$, then the Room's *x*-coordinate variable is fixed. If $\vartheta_{ext} = free$, then the Room's *x* and *y*-coordinates are free variables. Figure 12 shows a graphical representation of how this formulation works, and Section 3. 4. 2 explains further how this formulation was used.

3. 2 Mathematical Optimization Formulation

The design optimization problem is formulated as

$$\begin{aligned} &\text{minimize } f(\boldsymbol{x}) \\ &\text{subject to } \boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{0}, \boldsymbol{g}(\boldsymbol{x}) \le \boldsymbol{0} \\ &\boldsymbol{x} \in \Re^n \end{aligned} \tag{1}$$

where ***x*** is the vector of alterable design variables, *n* is the number of variables, ***h(x)*** is a vector of equality constraints, and ***g(x)*** is a vector of inequality constraints written in negative null form[2]. The continuous problem formulation allows us to take advantage of powerful gradient-based optimization algorithms designed to search continuous design spaces. Gradient algorithms use gradient information to make function approximations and calculate best search directions from the approximations. These algorithms can have unstable behavior if the functions are not smooth (do not have continuous first derivatives), so it is important to formulate the problem so that objective and constraint functions are as smooth as possible.

3. 2. 1 Design Variables

The optimization variables, ***x*** in Eq. (1), follow the variables defined in Figure 9d. Variables for each Unit include a reference point location *(x, y)*, distances to each wall *(N, S, E, W)*, and any windows added to each Unit (ω).

$$\boldsymbol{x} = \bigcup_{i=1}^{n} \{x_i, y_i, N_i, S_i, E_i, W_i, \omega_{N_i}, \omega_{S_i}, \omega_{E_i}, \omega_{W_i}\} \tag{2}$$

$$x_i, y_i, N_i, S_i, E_i, W_i \in \Re$$

$$\omega_{N_i}, \omega_{S_i}, \omega_{E_i}, \omega_{W_i} \in \Re_+$$

2. See Papalambros and Wilde [28] for details on design optimization conventions

The window variables drop out when the window is not physically present for a specific Unit and direction.

3. 2. 2 Resultant Variables

In order to simplify calculations and notation, several resultant intermediate variables will be used to describe geometry that results from the design variables (see Figure 13). This is useful in an object-oriented implementation because the design variable model can be changed without rewriting every constraint (only resultant relation calculations need to be rewritten). The following resultant variables are calculated from the design variables as follows.

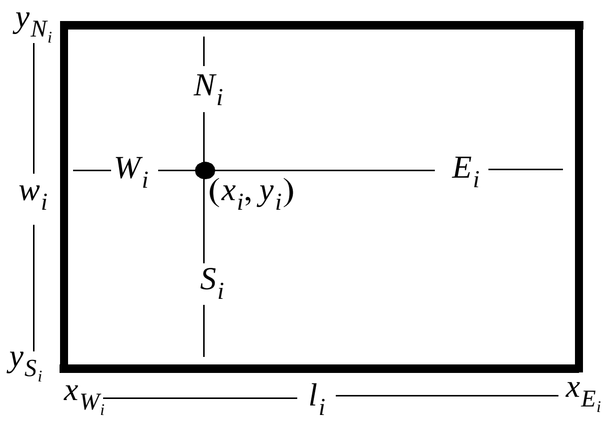


Figure 13. Diagram of design variables and resultant variables for a single Unit

$y_{N_i} = y_i + N_i$	Unit north wall location	(3)
$y_{S_i} = y_i - S_i$	Unit south wall location	(4)
$x_{E_i} = x_i + E_i$	Unit east wall location	(5)
$x_{W_i} = x_i - W_i$	Unit west wall location	(6)
$l_i = W_i + E_i$	Unit length	(7)
$w_i = N_i + S_i$	Unit width	(8)

Note that relations (3)-(8) are linear, so linear functions of these resultant variables are also linear functions of the original variables. Introducing this notation simplifies the model and makes it easier to understand.

3. 2. 3 Force Inside Constraint Group

The Force Inside Constraints are generally used to force Units into the main building Boundary or other grouping Boundaries. In order to force Unit *i* inside Unit *j*, the following constraints must all be satisfied:

$y_{N_i} \le y_{N_j}$ Unit *i* inside north wall of Unit *j*, (9)

$y_{S_j} \le y_{S_i}$ Unit *i* inside south wall of Unit *j*, (10)

$x_{E_i} \le x_{E_j}$ Unit *i* inside east wall of Unit *j*, (11)

$x_{W_j} \le x_{W_i}$ Unit *i* inside west wall of Unit *j*. (12)

3. 2. 4 Prohibit Intersection Constraint Group

Each Prohibit Intersection Constraint functions to prevent two Units from occupying the same space. By default, one Prohibit Intersection Constraint is added for every combination of Rooms, Hallways, and Accessways, except where two Units are forced to intersect, or where one Unit is forced inside of another.

In order to prevent Unit *i* from intersecting Unit *j*, Unit *i* must be entirely to the north, south, east, or west of Unit *j*. At least one of the following constraints must be satisfied

$y_{S_i} \ge y_{N_j}$ Unit *i* is north of Unit *j*, (13)

$y_{S_j} \ge y_{N_i}$ Unit *i* is south of Unit *j*, (14)

$x_{W_i} \ge x_{E_j}$ Unit *i* is east of Unit *j*, (15)

$x_{W_j} \ge x_{E_i}$ Unit *i* is west of Unit *j*. (16)

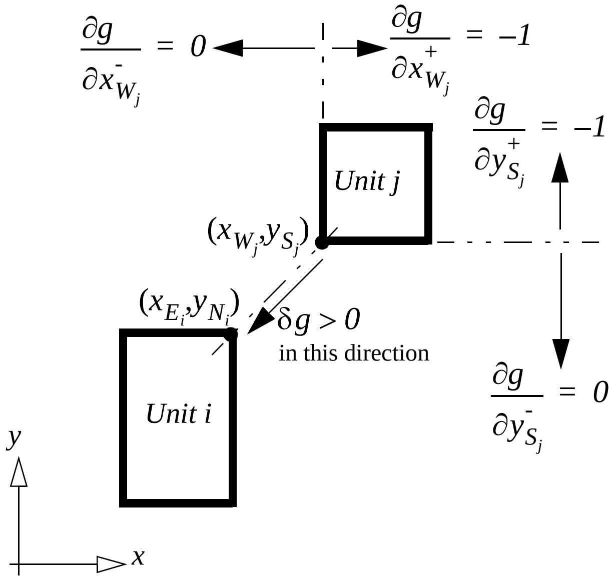
This logical disjunctive constraint set

$$(x_{W_i} \geq x_{E_j}) \text{ OR } (x_{W_j} \geq x_{E_i}) \text{ OR } (y_{S_i} \geq y_{N_j}) \text{ OR } (y_{S_j} \geq y_{N_i}) \qquad (17)$$

can be represented in negative null form using a min function

$$\min(x_{E_j} - x_{W_i},\ x_{E_i} - x_{W_j},\ y_{N_j} - y_{S_i},\ y_{N_i} - y_{S_j}) \leq 0 \qquad (18)$$

This nonlinear, non-smooth formulation is undesirable for gradient-based calculations; however, the nature of the constraint makes it unavoidable. With this formulation, the constraint function acts as a smooth linear function except when the close corners of two Units are nearly diagonal (see Figure 14). Several other mathematical representations were explored, but this representation seems to have the best behavior.



$\frac{\partial g}{\partial x^-_{W_j}} = 0$
$\frac{\partial g}{\partial x^+_{W_j}} = -1$
$\frac{\partial g}{\partial y^+_{S_j}} = -1$
Unit j
(x_{W_j}, y_{S_j})
(x_{E_i}, y_{N_i})
$\delta g > 0$
in this direction
$\frac{\partial g}{\partial y^-_{S_j}} = 0$
Unit i
y
x

Figure 14. An example nonlinearity of a disjunctive logic constraint represented as a min() function. In the figure, g represents the value of the prohibit intersection constraint: the left side of Eq. (18)

3. 2. 5 Force Minimum Intersection Constraint Group

Units are generally forced to intersect in order to ensure access (as Accessways do), or to make a more complex geometric shape by combining rectangular Units. Forcing intersection is the opposite of prohibiting intersection, so forcing intersection can be written as the conjunction of the following constraints

$$y_{S_i} \leq y_{N_j} \qquad \text{Unit } i \text{ overlaps north wall of Unit } j, \qquad (19)$$

$$y_{S_j} \leq y_{N_i} \qquad \text{Unit } i \text{ overlaps south wall of Unit } j, \qquad (20)$$

$$x_{W_i} \leq x_{E_j} \qquad \text{Unit } i \text{ overlaps east wall of Unit } j, \qquad (21)$$

$$x_{W_j} \leq x_{E_i} \qquad \text{Unit } i \text{ overlaps west wall of Unit } j. \qquad (22)$$

Although these constraints ensure intersection of the two Units, they permit intersection at a point. In designing architectural spaces, we are generally interested in intersection that provides enough room for a doorway or opening, so we must add an additional constraint. There will be enough room for a doorway or opening if the overlap in one of the cartesian directions is at least as large as the opening. Therefore, in addition to intersection, at least one of the following conditions must be satisfied

$${}_{N_j} - y_{S_i} \geq \max(d_i,\ d_j) \qquad \text{Unit } i \text{ overlaps north wall of Unit } j, \qquad (23)$$

$${}_{N_i} - y_{S_j} \geq \max(d_i,\ d_j) \qquad \text{Unit } i \text{ overlaps south wall of Unit } j, \qquad (24)$$

$${}_{E_j} - x_{W_i} \geq \max(d_i,\ d_j) \qquad \text{Unit } i \text{ overlaps east wall of Unit } j, \qquad (25)$$

$${}_{E_i} - x_{W_j} \geq \max(d_i,\ d_j) \qquad \text{Unit } i \text{ overlaps west wall of Unit } j. \qquad (26)$$

where d_i is the minimum size for a door or opening in Unit *i*. This disjunctive set of constraints can be represented in negative null form using a min function similar to Eq. (18).

$$\begin{aligned}\min\{ &\max(d_i,\ d_j) - x_{E_j} + x_{W_i},\ \ \max(d_i,\ d_j) - x_{E_i} + x_{W_j}, \\ &\max(d_i,\ d_j) - y_{N_j} + y_{S_i},\ \ \max(d_i,\ d_j) - y_{N_i} + y_{S_j} \} 0\end{aligned} \qquad (27)$$

Although this constraint function is nonlinear and non-smooth in part of the design space, it is linear in most of the design space (similar to Figure 14).

The complete Force Minimum Intersection Constraint Group is represented as a set of constraints that force intersection (Eq. (19)-Eq. (22)) and another constraint to ensure that the overlap is large enough for access (Eq. (27)).

3. 2. 6 Force To Edge Constraint Group

The Force To Edge Constraints are used to force a Unit to the edge of a Boundary because of a window or external door. It is assumed that the first Unit *i* has already been forced inside Unit *j* by another constraint. In order to force a Unit to a particular wall, one of the following constraints can be added:

$y_{N_i} = y_{N_j}$	Unit *i* against north wall of Unit *j*,	(28)
$y_{S_i} = y_{S_j}$	Unit *i* against south wall of Unit *j*,	(29)
$x_{E_i} = x_{E_j}$	Unit *i* against east wall of Unit *j*,	(30)
$x_{W_i} = x_{W_j}$	Unit *i* against west wall of Unit *j*.	(31)

If connection to an edge is important, but the specific edge is not important, (for instance, a building may require an external door, but it is not important which one), then the following constraint can be added to represent a disjunction of Eq. (28)-Eq. (31)

$$\min\left\{(x_{E_i} - x_{E_j})^2,\ (x_{W_i} - x_{W_j})^2,\ (y_{S_i} - y_{S_j})^2,\ (y_{N_i} - y_{N_j})^2\right\} = 0 \qquad (32)$$

This representation is non-smooth at Unit corners; however, it is quadratic in most of the design space (similar to Figure 14).

3. 2. 7 Bound Size Constraint Group

Three kinds of constraints are provided to bound the size of a Unit: minimum area, minimum length/width, and maximum length/width. Length is not distinguished from width. It is assumed that a maximum area constraint would not be used to bound the area. Instead, Unit area is only reduced to improve objective functions, such as cost objectives. Minimum area, A_{min}, minimum length/width, l_{min}, and maximum length/width, l_{max}, can be set for each Unit. Default values are applied to each Unit based on common room dimensions because results may not be physically meaningful if a lower bound on length/width or area is not provided.

$$A_{min_i} - l_i w_i \le 0 \qquad \text{minimum area,} \tag{33}$$

$$l_{min_i} - l_i \le 0 \qquad \text{minimum length,} \tag{34}$$

$$l_{min_i} - w_i \le 0 \qquad \text{minimum width,} \tag{35}$$

$$l_i - l_{max_i} \le 0 \qquad \text{maximum length,} \tag{36}$$

$$w_i - l_{max_i} \le 0 \qquad \text{maximum width.} \tag{37}$$

3. 2. 8 Minimum Ratio Constraint Group

Unit length-to-width ratio can be bounded to maintain a desired aesthetic scheme or prevent long, narrow Rooms that may not be usable. The Minimum Ratio constraint group consists of two constraints.

$$R_{min_i} l_i - w_i \le 0 \qquad \text{minimum width to length ratio,} \tag{38}$$

$$R_{min_i} w_i - l_i \le 0 \qquad \text{minimum length to width ratio.} \tag{39}$$

3. 2. 9 Build Cost Constraint

The build cost constraint is used to keep the construction cost below some value, Γ_{budget}. For simplicity, build cost is measured only in terms of material cost. Material costs for walls κ_{wall} and for windows κ_{ω} are specified as dollars per square foot of material, and other costs are ignored. The build cost constraint is calculated as

$$\kappa_{wall}(A_N + A_S + A_E + A_W) + \kappa_{\omega}(A_{\omega_N} + A_{\omega_S} + A_{\omega_E} + A_{\omega_W}) \le \Gamma_{budget} \tag{40}$$

where A_N, A_S, A_E, A_W are the areas of the external walls in each compass direction and $A_{\omega_N}, A_{\omega_S}, A_{\omega_E}, A_{\omega_W}$ are the areas of windows facing each compass direction. These quantities are computed in Eq. (49)-Eq. (56).

3. 2. 10 Feasible Window Constraint Group

In addition to the simple bound restricting window size to be positive, the window width cannot be larger than the wall it is on. Each window added to a Unit is given one of the following feasible window constraints (as appropriate).

$$\omega_{N_i} \le l_i \qquad \text{north window size,} \qquad (41)$$

$$\omega_{S_i} \le l_i \qquad \text{south window size,} \qquad (42)$$

$$\omega_{E_i} \le w_i \qquad \text{east window size,} \qquad (43)$$

$$\omega_{W_i} \le w_i \qquad \text{west window size.} \qquad (44)$$

3. 2. 11 Bound Lighting Constraint Group

A simple estimation of the amount of daylight entering a Unit with windows is calculated using environmental and material information. The following procedure is used.

Determine available daylight at the window exterior. IESNA [63] provides three standard skies for use in the evaluation of daylight designs. Approximate available daylight can be determined from these based on altitude and azimuth angles.

E_{vkd_m} = vertical sky illuminance (direct)

E_{vks_m} = vertical sky illuminance (sky)

for month m.

Find coefficient of utilization. The coefficient of utilization, C_U, is a function of the room geometry and window size, and it determines the fraction of the available daylight that enters the room. C_U can be found in pre-tabulated data [62] based on room depth, window width, and window height.

Determine net transmittance of windows. The net transmittance for a window facing direction j is calculated as

$$\mu_j = \frac{0.9\mu_G A_{\omega_j}}{A_j} \tag{45}$$

where j takes on each of the directions {N, S, E, W}, μ_G is the transmittance of the window (material property), A_{ω_j} is the area of the glass in direction j, and A_j is the area of the wall in direction j.

Determine Daylight at the Room Center. The horizontal illuminance at the center of room i is calculated as

$$E_i = \left(\sum_j \sum_m A_{\omega_j}(E_{vkd_{m_i}} + E_{vks_{m_i}})\mu_j C_U 10^2 \right) \tag{46}$$

for room i, where j takes on each direction {N, S, E, W}, and m spans the 12 months of the year. The illuminance is then converted into watts, Θ_i:

$$\Theta_i = \frac{E_i 10^{-3}}{A_i \beta_{eff}} \tag{47}$$

where A_i is the area of room i, and β_{eff} is the efficacy of the light source (assumed to be 80).

The required natural lighting per square foot, θ_{req_i}, is defined for each Unit by the designer (default 1 Watt/sq.ft.). Assuming uniform light distribution, total required natural lighting can be calculated as $A_i\theta_{req_i}$. The minimum percentage of required lighting that is provided by natural light, φ_{min_i}, can be specified by the designer.

The final constraint is written as:

$$\frac{\Theta_i}{A_i \theta_{req_i}} \ge \varphi_{min_i}. \tag{48}$$

3. 2. 12 Minimize Heating Cost Objective

The annual energy cost to heat the building is calculated as a function of the building Boundary Unit shape, volume, surface area, and material as well as environmental conditions.

Simplified calculations (ASHRAE [62]) are used to calculate an approximation. The procedure for calculating heating loads is as follows:

1. Calculate the net area of windows on each external wall. It is assumed here that windows on all Units are constrained against external walls

$$A_{\omega_N} = \sum_{i \in \mathrm{UNW}} \omega_{N_i} h_{\omega_i} \qquad \text{area of north windows,} \qquad (49)$$

$$A_{\omega_S} = \sum_{i \in \mathrm{USW}} \omega_{S_i} h_{\omega_i} \qquad \text{area of south windows,} \qquad (50)$$

$$A_{\omega_E} = \sum_{i \in \mathrm{UEW}} \omega_{E_i} h_{\omega_i} \qquad \text{area of east windows,} \qquad (51)$$

$$\omega_W = \sum_{i \in \mathrm{UWW}} \omega_{W_i} h_{\omega_i} \qquad \text{area of west windows.} \qquad (52)$$

(UNW, USW, UEW, and UWW refer to Units with north, south, east, or west windows respectively).

2. Calculate the net area of each external wall.

$$A_N = l_1 h_1 - A_{\omega_N} \qquad \text{area of north wall,} \qquad (53)$$

$$A_S = l_1 h_1 - A_{\omega_S} \qquad \text{area of south wall,} \qquad (54)$$

$$A_E = w_1 h_1 - A_{\omega_E} \qquad \text{area of east wall,} \qquad (55)$$

$$A_W = w_1 h_1 - A_{\omega_W} \qquad \text{area of west wall.} \qquad (56)$$

where 1 indicates Unit 1, which is assumed to be the building Boundary Unit.

3. Calculate heat loss. The heat loss calculation assumes that all heat is lost from the external walls and windows (no heat is lost through the roof). This model could be changed depending on what type of building is being modeled. The coefficient of transmittance for the wall, U_{wall}, and window, U_{ω}, are tabulated based on the materials used. The annual heat loss is calculated as

$$Q_{heat} = \sum_{i} \Delta T_i((A_N + A_S + A_E + A_W)U_{\text{wall}} + (A_{\omega_N} + A_{\omega_S} + A_{\omega_E} + A_{\omega_W})U_{\omega}) \tag{57}$$

where i is the set of months where heat is used, and ΔT_i is the average internal/external temperature difference for month i.

4. Calculate cost to maintain temperature. Gas heat is assumed, and the cost of gas per cubic foot, κ_{gas} and efficiency of the heater in Watts per cubic foot of gas, η_{heater}, can be specified. The heating cost objective function is formulated as

$$\text{minimize } \Gamma_{heat} = \frac{\kappa_{gas} Q_{heat}}{\eta_{heater}} \tag{58}$$

3. 2. 13 Minimize Cooling Cost Objective

The procedure for calculating cooling loads is more complicated than heating loads because heat due to solar gain must be taken into account.

1. Calculate the net area of windows on each external wall. Use Eq. (49) - Eq. (52).

2. Calculate the net area of each external wall. Use Eq. (53) - Eq. (56).

3. Calculate solar heat gain through the windows. Several parameters are important in calculating solar heat gain. Depending on the orientation of the windows (N, S, E, or W), the Solar Heat Gain Factor, β_{shgf}, can be found in tables for a given location [62]. The shading coefficient, β_{sc}, is a property of the glass [62], and the time-lag factor, β_{tlf}, is a tabulated function of glass type and window orientation [62]. The annual solar heat gain, Q_{solar}, is calculated as

$$Q_{solar} = \beta_{sc}\left(\sum_i (A_{\omega_N}\beta_{shgf_N}\beta_{tlf_N} + A_{\omega_S}\beta_{shgf_S}\beta_{tlf_S} + A_{\omega_E}\beta_{shgf_E}\beta_{tlf_E} + A_{\omega_W}\beta_{shgf_W}\beta_{tlf_W})\right) \tag{59}$$

where i is the set of months where air conditioning is used.

4. Calculate conductive heat gain through the building exterior. The orientation of each exterior wall and windows is accounted for in the factor. The cooling load due to conduction is calculated as

$$Q_{cond} = \sum_i \Delta T_i (U_\omega (A_{\omega_N}\beta_{tlf_N} + A_{\omega_S}\beta_{tlf_S} + A_{\omega_E}\beta_{tlf_E} + A_{\omega_W}\beta_{tlf_W}) + U_{wall}(A_N\beta_{tlf_N} + A_S\beta_{tlf_S} + A_E\beta_{tlf_E} + A_W\beta_{tlf_W})) \tag{60}$$

where i is the set of months where air conditioning is used.

5. Calculate the cost to maintain temperature. Electric cooling is assumed, and the rate of electricity, κ_{elec}, and efficiency of the air conditioning unit, η_{ac}, can be specified. The cooling cost objective function is formulated as

$$\text{minimize } \Gamma_{cool} = \frac{\kappa_{elec}(Q_{solar} + Q_{cond})}{\eta_{ac}}. \tag{61}$$

3. 2. 14 Minimize Lighting Cost Objective

This objective minimizes the cost spent on lighting the building by encouraging natural lighting. The amount of natural lighting in room i, Θ_i, is calculated as in Section 3. 2. 11, Eq. (47). The minimum daylight requirement per square foot, β_{light}, is set by the designer based on usage intention. The total required cost if all of this light is provided by electric lighting can be calculated as:

$$\Gamma_{elec} = \left(\sum_i \beta_{light_i} A_i\right) \beta_H 10^{-3}, \tag{62}$$

where i is the set of Units, and β_H is the number of hours of use per month.

The total cost to minimize is then the maximum possible electricity cost minus the cost savings from natural lighting (see Section 3. 2. 11):

$$\text{minimize } \Gamma_{light} = \Gamma_{elec} - \left(\sum_i \Theta_i\right) \beta_A 10^{-3}, \tag{63}$$

where i is the set of Units, and β_A is the number of hours of available light per month.

3. 2. 15 Minimize Wasted Space Objective

Wasted space refers to space that is not living space. This could be space used for hallways or un-allocated space inside the building Boundary. Wasted space is calculated as the area of the building Boundary minus the total area used as living space. The objective is formulated as

$$\text{minimize } \left(l_1 w_1 - \sum_{i \in \text{Rooms}} l_i w_i \right), \tag{64}$$

where 1 indicates Unit 1, which is assumed to be the building Boundary Unit.

3. 2. 16 Minimize Accessway Size Objective

This objective brings connected Units together. Accessways may be constrained to be small (Section 3. 2. 7) to keep Units together. Alternatively, the Minimize Accessway Objective can be used to bring Units together if possible, but allow them to be separated if necessary, providing that there is an Accessway between them. This method allows Accessways to function similarly to Hallways depending on the design situation. The objective is formulated as

$$\text{minimize } \left(\sum_{i \in \text{Accessways}} l_i w_i \right). \tag{65}$$

3. 2. 17 Minimize Hallway Size Objective

This objective is used to provide extra living space where possible. Generally, the Minimize Wasted Space Objective (Section 3. 2. 15) will naturally minimize Hallways in order to increase the area of each Room. However, there are some situations where the objective function is flat with respect to Hallway variables. An example is shown in Figure 15.

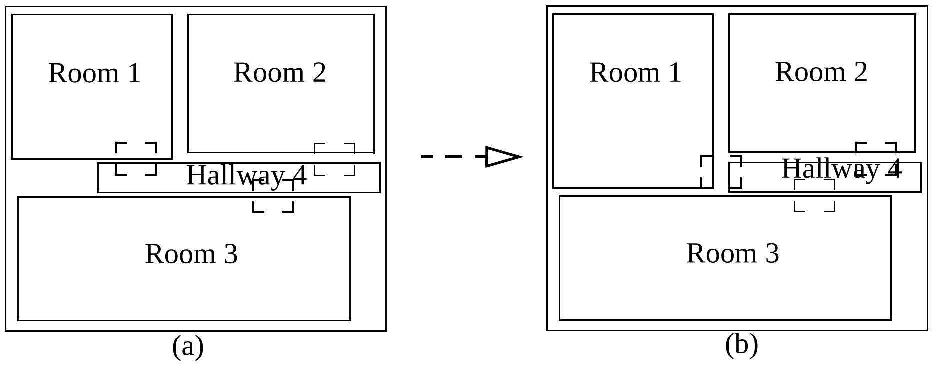


Figure 15. Example showing effects of the Minimize Hallway Objective

In Figure 15a, $\frac{\partial f}{\partial W_4} = 0$, where W_4 is the west wall position variable for Unit 4, and *f* is the value of the Minimize Wasted Space objective function. It appears to the algorithm as if there is no reason to change W_4; however, if W_4 is decreased enough, as in Figure 15b, then Room 1 is able to move in and take up the space. Using the Minimize Hallway objective will tend to provide more space for living space where possible. This is one example where designer interaction with the optimization problem during optimization would be helpful. The objective is formulated as

$$\text{minimize} \left(\sum_{i \in Hallways} l_i w_i \right) . \tag{66}$$

3. 2. 18 Multi-Objective Optimization

Multiple objectives can be selected and combined into a single objective function using a weighted sum of the individual objective functions.

$$(\boldsymbol{x}) \ = \ \sum_{j\,=\,1}^{N} w_j \ f_j(\boldsymbol{x}), \tag{67}$$

where $f_j(\boldsymbol{x})$ is the jth objective function, w_j is the weighing (relative importance) of the jth objective function, and N is the total number of objective functions. Appropriate weights may be difficult to set for objective functions measured in different units. After obtaining results, weights can be adjusted to compensate and to guide the design to desired results. The objectives in the toolbox do not compete in most of the design space, except for cost objectives, which are all measured in dollars. This makes multi-objective optimization much easier. In practice weights only need to be adjusted to keep the function values in the same order of magnitude to avoid computational problems.

3. 3 Local Optimization Methods

Several gradient-based optimization algorithms were used to solve the geometric layout for local optima.

3. 3. 1 Sequential Quadratic Programming

Sequential Quadratic Programming (SQP) is a method for solving continuous optimization problems as Eq. (1). A typical SQP algorithm is shown in Figure 16.[3]

1	Generate QP sub-problem: • 2nd order approximation of the Lagrangian • 1st order approximation of constraints Use an active set strategy to handle inequality constraints.	$f(\underline{x}) \approx L(\underline{x}_k) + \nabla L(\underline{x}_k)\delta x + \frac{1}{2}\underline{\delta x}^T \nabla^2 L(\underline{x}_k)\underline{\delta x}$ $h(\underline{x}) \approx h(\underline{x}_k) \quad \nabla h(\underline{x}_k)\underline{\delta x} = \underline{0}$
2	Solve the QP approximation to find search direction s_k. KKT conditions are linear, so the system of equations can be solved explicitly, or a more sophisticated method can be used.	$\underline{h} = \underline{0}, \underline{g} \le \underline{0}$ $\nabla f + \underline{\lambda}^T \nabla \underline{h} + \underline{\mu}^T \nabla \underline{g} = \underline{0}$ $\underline{\mu}^T \underline{g} = \underline{0}$ $\underline{\lambda} \ne \underline{0}, \underline{\mu} \ge \underline{0}$
3	Line search a penalty function of the objective in the direction of s_k (find a good value for α)	$\underline{x}_{n+1} = \underline{x}_n - \alpha(s_k)$
4	Iterate to convergence	

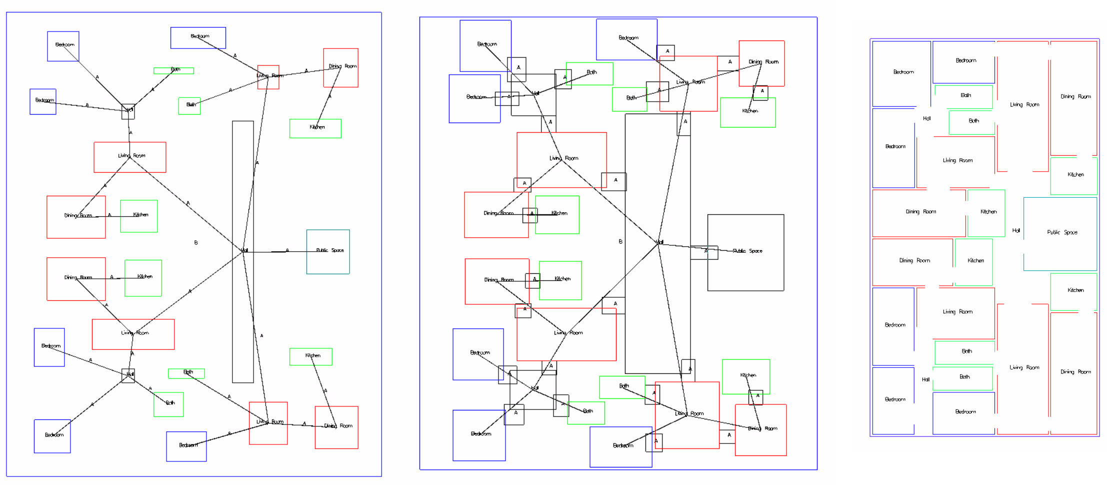
Figure 16. Description of the Sequential Quadratic Programming method for optimization.

OptdesX [53], an optimization package, was used to implement SQP on the early problem formulation described in Section 3. 1. 8. SQP successfully generated local optima if the initial design was feasible or near feasible. Infeasible starting points often led the algorithm to become trapped in infeasible space. The algorithm worked well; however, the problem was reformulated and re-implemented using CFSQP so that an object-oriented C++ code could be written and a GUI could be developed to interact graphically with the problem.

CFSQP, a C implementation of Feasible Sequential Quadratic Programming [31], was used to solve the building geometric layout problem presented in this thesis. FSQP is similar to SQP except that once a feasible design is found, search directions are altered to maintain feasibility at every iteration. If the initial design is infeasible, a penalty function strategy is used to find a feasible design. In addition, CFSQP uses a Quasi-Newton Method to approximate the Hessian of the Lagrangian $\nabla^2 L(\boldsymbol{x})$. Instead of using finite difference methods to approximate the Hessian at every design, the Quasi-Newton Method begins by assuming an arbitrary Hessian (usually the identity matrix). Then, at each iteration, the Hessian approximation is updated with new design space information. The Hessian updates are constructed so that the Hessian is guaranteed to be positive definite at every iteration. This is important for convergence properties. CFSQP also handles linear constraints separately so that they are solved more efficiently.

3. Refer to Papalambros and Wilde [28] for more details on SQP, Lagrangian, KKT conditions, line search, active set strategies, Quasi-Newton Methods, finite difference, and penalty functions.

The CFSQP algorithm has been successful at generating locally optimal geometric designs for the layout problem. A sample optimization of a particular layout problem is shown in Figure 17



(a) (b) (c)

Figure 17. Progression of the CFSQP algorithm optimizing a sample apartment complex building to minimize annual cost and wasted space; (a) shows the initial layout sketch provided by the designer (accessways shown as lines between Units); (b) is an intermediate feasible iteration (accessways shown also as rectangles); and (c) shows the completed design (accessways shown as wall openings for clarity).

CFSQP is very fast for moderately sized problems, and it is relatively stable; however, sometimes the algorithm becomes stuck and is unable to find the next move or determine KKT optimality. This may be partly due to non-smooth constraints (Eq. (18), Eq. (27), Eq. (32)). SQP is only provably convergent for smooth convex problems, and it may have unpredictable behavior if gradient functions are discontinuous. Still, in practice the algorithm almost always converges quickly, and convergence problems can usually be avoided by perturbing the design slightly to move it away from non-smooth areas of the design space. An auto-perturb feature has been implemented to automatically avoid this problem. The auto-perturb feature alters each design variable by a small random number if the algorithm gets stuck. This usually moves search away from the first-order discontinuity and produces a reliable solution.

3. 3. 2 Generalized Reduced Gradient

In addition to SQP, the OptdesX package was used to implement the Generalized Reduced Gradient (GRG) algorithm on the original layout formulation (see Section 3. 1. 8). A typical GRG algorithm is shown in Figure 18[4].

1	Use an active set strategy to determine which of the inequality constraints to assume active. Treat active inequality constraints as equality constraints and ignore inactive constraints.	$\underline{g}(\underline{x}) \Rightarrow \underline{h}_{assumed}(\underline{x})$
2	Split variables into decision variables and state variables. Number of state variables = number of independent constraints.	$f(\underline{x}) \Rightarrow f(\underline{d},\underline{s})$
3	Linearize $\underline{h}$, set $\partial\underline{h} = \underline{0}$ and solve for $\partial\underline{s}$ as a function of $\partial\underline{d}$	$\partial\underline{h} = \frac{\partial\underline{h}}{\partial\underline{s}}\partial\underline{s} + \frac{\partial\underline{h}}{\partial\underline{d}}\partial\underline{d} = \underline{0} \therefore \partial\underline{s} \approx -\left(\frac{\partial\underline{h}}{\partial\underline{s}}\right)_n^{-1}\left(\frac{\partial\underline{h}}{\partial\underline{d}}\right)_n \partial\underline{d}$
4	Write z($\underline{d}$) function as a projection of the f($\underline{x}$) function onto the $\underline{d}$ space (unconstrained reduced space)	$\partial z = \frac{\partial f}{\partial\underline{d}}\partial\underline{d} + \frac{\partial f}{\partial\underline{s}}\partial\underline{s}$ $\therefore \frac{\partial z}{\partial\underline{d}} \approx \left(\frac{\partial f}{\partial\underline{d}}\right)_n - \left(\frac{\partial f}{\partial\underline{s}}\right)_n\left(\frac{\partial\underline{h}}{\partial\underline{s}}\right)_n^{-1}\left(\frac{\partial\underline{h}}{\partial\underline{d}}\right)_n = \underline{0}$
5	Solve the unconstrained z($\underline{d}$) problem using Steepest Descent to find $\underline{d}$	$\underline{d}_{n+1} = \underline{d}_n - \alpha\left(\frac{\partial z}{\partial\underline{d}}\right)_n$
6	Iterate using Newton's Method to find $\underline{s}$ to satisfy constraints for $\underline{d}$ found in 4 [$\underline{h}(\underline{d},\underline{s}) = \underline{0}$]. If $\underline{s}$ cannot be found (too far for linear model to be valid), use a smaller α.	$\underline{s}_{m+1} = \underline{s}_m - \left(\frac{\partial\underline{h}}{\partial\underline{s}}\right)_m^{-1}\underline{h}(\underline{d},\underline{s}_m)$
7	Iterate to convergence (goto 2)	

Figure 18. Description of the Generalized Reduced Gradient method for optimization.

GRG was successful at generating local optima for the layout problem, and it is unclear if either GRG or SQP is a superior solution method for this formulation. However, GRG seemed to have more difficulty with discontinuous gradient functions, so SQP was pursued for the remainder of the problems.

3. 3. 3 Limitations of Local Search

These gradient-based search algorithms find locally optimal designs. This means that the design is better than any neighboring design; however, the solution is highly dependent on the starting point, and there is no guarantee that the design is of global quality. The design space of this problem contains many local optima, some of which have poor global

4. For more information on GRG, active set strategy, reduced space, steepest descent, and Newton's Method see Papalambros and Wilde [28].

quality. Also, if the starting point is highly infeasible, then the algorithms often cannot find feasible designs.

3. 4 Global Optimization Methods

Global optimization methods have been developed to overcome the limitations of local search and to find solutions of global quality. Several global search strategies were used to generate geometric layouts.

3. 4. 1 Simulated Annealing

Simulated Annealing (SA - also known as the metropolis algorithm) is a stochastic algorithm inspired by the way that molecules in metals minimize energy state during the annealing process. A typical SA algorithm is shown in Figure 19.

1	Choose initial design and initial temperature	$T = T_{init}, x_k = x_{init}$
2	Examine a random design in the neighborhood of the current design.	$\underline{x}_\Delta = \underline{x}_k + \underline{\delta x}$
3	If the new design has a better objective function value, accept it as the next design. If the new design is not better, accept it anyway with probability proportional to temperature.	if $(f(\underline{x}_\Delta) > f(\underline{x}_k))$ $\underline{x}_{k+1} = \underline{x}_\Delta$ else if $(random(0,1) < p(T))$ $\underline{x}_{k+1} = \underline{x}_\Delta$ else $\underline{x}_{k+1} = \underline{x}_k$
4	Decrease temperature according to some cooling schedule	$T = cool(T)$
5	Iterate until some terminating conditions	

Figure 19. Description of the Simulated Annealing algorithm.

Initially, the algorithm explores randomly, accepting almost every new point. As the temperature cools, the algorithm rejects more of the uphill moves. The idea is that by sometimes allowing the algorithm to accept an uphill move, there is a chance of working its way out of local minima, but as the algorithm progresses, it begins to accept only downhill moves, switching from global to local focus. SA is a stochastic algorithm, and it is not guaranteed to converge to a local or global minimum in finite time; however, it is often successful at finding solutions of global quality.

SA was implemented using several variations of the early formulation layout problem (see Section 3. 1. 8). Because SA cannot handle constraints explicitly, the constraints were modeled as penalty terms in the objective function. Due to the high dimensionality and highly constrained space of the problem, SA was unable to find a feasible design, even when run for several days on a trivial problem. Providing SA with a feasible starting point does not solve this problem because SA accepts many random design moves in the beginning of the algorithm, moving far from the initial design. If the cooling schedule and initial temperature are adjusted to prevent this, then the global search quality of SA is lost.

These results suggest that a stochastic algorithm is not a practical solution to a problem formulation with high dimensionality and a highly constrained space, although alternate formulations of the layout problem may be more suited to this kind of search (for example [3]).

3. 4. 2 Hybrid SA/SQP Search Method

A hybrid SA/SQP search strategy was developed to take advantage of the global qualities of SA and the efficiency of SQP in order to generate local optima of global quality. The method is outlined in Figure 20.

1	Choose initial design and initial temperature	$T = T_{init}, x_k = x_{init}$
2	Examine a random design in the neighborhood of the current design. Calculate the quality of this new design using step 3.	$\underline{x}_\Delta = \underline{x}_k + \underline{\delta x}$
3	Run SQP using the new design as a starting point Set the quality of the SA-generated design equal to the quality of the local optimum produced from the starting point.	$\underline{x}^*_{SQP} = SQP(\underline{x}_\Delta)$ $f(\underline{x}_\Delta) = f(\underline{x}^*_{SQP})$
4	If the new design has a better objective function value, accept it as the next design. If the new design is not better, accept it anyway with probability proportional to temperature.	$\text{if } (f(\underline{x}_\Delta) > f(\underline{x}_k)):\ \underline{x}_{k+1} = \underline{x}_\Delta$ $\text{else if } (random(0,1) < p(T)):\ \underline{x}_{k+1} = \underline{x}_\Delta$ $\text{else}:\ \underline{x}_{k+1} = \underline{x}_k$
5	Decrease temperature according to some cooling schedule	$T = cool(T)$
6	Iterate until some terminating conditions	

Figure 20. Description of the
SA/SQP hybrid algorithm

In this method, SA is used to search for a good starting point, and SQP is used to find the local minimum near each starting point. In this way SA can search the space more globally with large moves while SQP worries about the details. A sample objective function is shown in Figure 21. In this example, SQP can find six different local optima depending on where the starting point is chosen. Each point that SA selects is evaluated by locally optimizing it, so SA observes any point in the vicinity of a local optimum to have the objective value of that local optimum. In a sense, the objective function is being screened for SA. Notice in the example that the function SA observes has only two local optima instead of six. Also, an algorithm searching the resultant function can make larger design moves without as much danger of overstepping important features. The discontinuity of the resultant objective function is acceptable because SA does not require continuous functions, as long as they are defined over the entire domain. Also, the penalty function formulation that CFSQP uses to find feasible space is sometimes successful at finding a feasible optimum even when SA chooses an infeasible starting point (see Section 3. 3. 1).

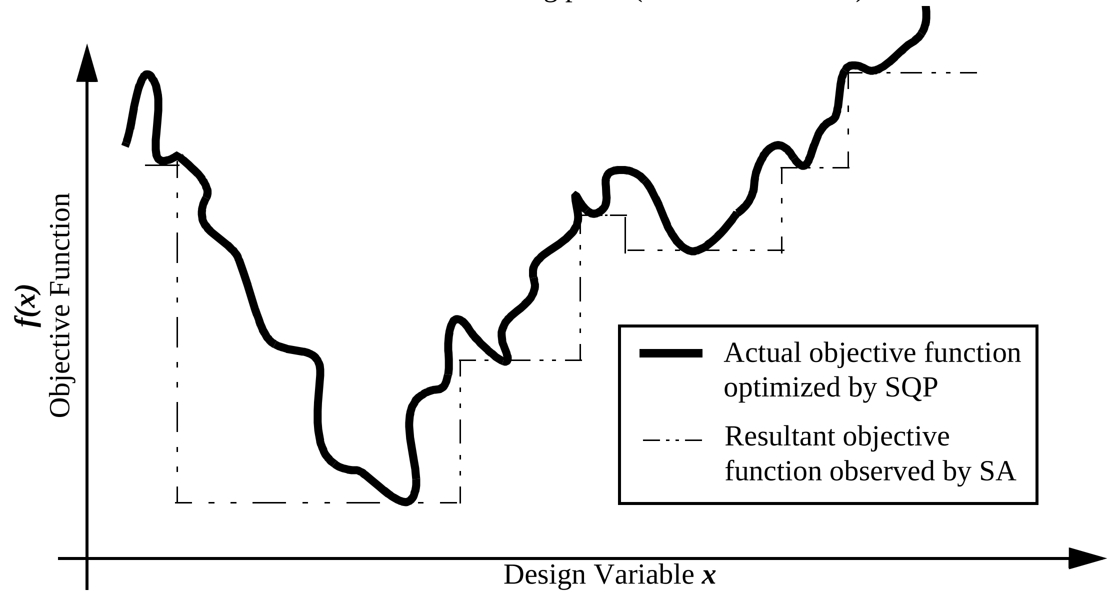

f(x)
Objective Function
Design Variable x
Actual objective function optimized by SQP
Resultant objective function observed by SA

Figure 21. Hybrid SA/SQP sample function with multiple local minima.

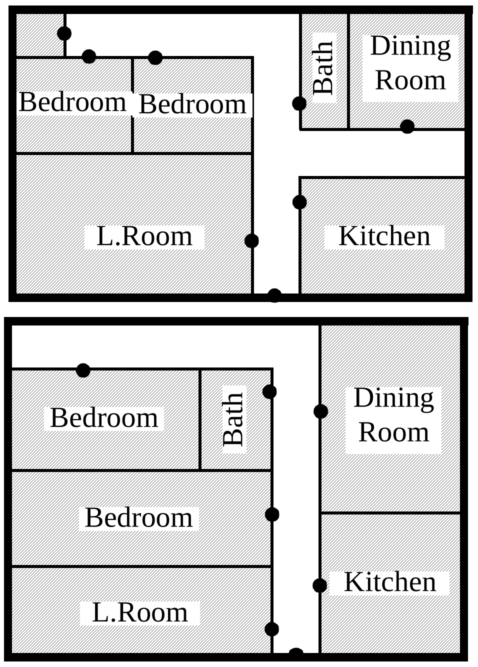


Figure 22. Sample results generated by the SA/SQP hybrid algorithm

The hybrid SA/SQP method was implemented on the early problem formulation for building layout (see Section 3. 1. 8). In this problem, the building external shape was specified within a tolerance. Local optima of reasonable global quality were obtained for up to seven room apartment layouts (70 variables, 269 constraints). Figure 22 shows two resultant layouts generated by this method. It is important to understand that these designs were generated automatically with no feasible initial starting point. This is a substantial improvement. Using SA alone, we were unable to produce even a feasible design. SQP is quick at generating solutions; however, the designer must define where Rooms should be placed relative to one another. In this problem, the early formulation specifies that all Rooms must be connected to a hallway structure, but it does not specify arrangement. The algorithm is able to automatically generate a quality feasible arrangement and optimize that geometry locally.

3. 4. 3 Evolutionary Algorithms

Evolutionary algorithms define a class of algorithms inspired by the natural evolution of organisms. Evolutionary algorithms include several sub-classes: genetic algorithms (GA), genetic programming (GP), evolutionary programming (EP), and evolutionary strategies (ES). In design optimization, evolutionary algorithms are used to evolve a population of designs over a number of generations. A typical evolutionary algorithm is described in Figure 23, although there are many variations.[5]

1	Generate an initial population of m designs (usually random)	$P_k = P_{init} = \{x_1, x_2, .. x_m\}$
2	Select two individuals from the population using a selection scheme. Usually higher quality designs are more likely to be selected.	$x_i = select(P_k)$ $x_j = select(P_k \setminus x_i)$
3	Call a crossover function to combine traits of the two designs into two new designs.	$(x_a, x_b) = crossover(x_i, x_j)$
4	Call a mutator function to randomly perturb each design with some probability.	$x_a = mutate(x_a)$ $x_b = mutate(x_b)$
5	Goto 2 and repeat until m new designs have been generated.	
5	Insert the new designs back into the population	$P_k = P_k \bigcup \{x_a, x_b\}$
6	Select m individuals from the population to discard using a selection scheme. Usually the lowest quality designs are discarded.	$P_k = P_k \setminus selectKill(P_k)$ $P_k = P_k \setminus selectKill(P_k)$
7	Goto 2 and iterate until some terminating conditions	$P_{k+1} = P_k$

Figure 23. Description of a typical
Evolutionary Algorithm

The selection function is written so that high quality designs are more likely to be chosen. The crossover operator combines traits from two designs to produce a new design with the assumption that high quality designs have high quality components. The crossover operator generates new combinations of these high quality components. In Genetic Algorithms, for example, the design variables are translated into binary strings called chromosomes, and crossover involves cutting the chromosomes at a random place along the string and swapping ends. The mutation function introduces some random perturbation to the system, and it is usually applied with low probability to introduce extra variance.

5. Refer to Bentley [42] for more information on evolutionary algorithms.

Genocop [40][41], a GA tool written in C, was used to implement a genetic algorithm to solve the layout problem for designs of global quality. Due to the high dimensionality and highly constrained nature of the problem formulation, the GA was only able to generate feasible designs for the most trivial problems. It is possible that finding the right set of GA parameters or using different heuristics could improve the ability to find feasible and quality designs; however, the complexity of the problem grows quickly with problem size, and it is unlikely that a GA would be practical for sizable problems. Besides, one should take advantage of the efficiency and reliability of gradient algorithms when the formulation has so many simple constraint functions.

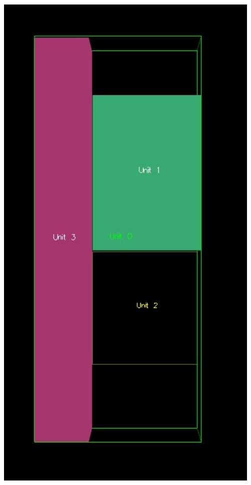


Figure 24. Sample geometry generated by Genocop

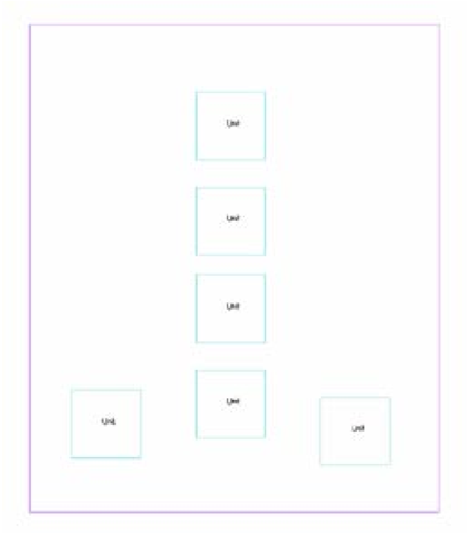
3. 4. 4 The Maximum Distance Distribution Method

One way to explore for solutions of global quality is to use a variation of an optimization technique referred to as the Maximum Distance Distribution Method (MDDM [29],[30]). This method was developed for discrete problems, but it also works for continuous problems. The concept is to use a local optimization algorithm to find a local minimum $\boldsymbol{x}^*$ using the formulation in Eq. (1). Once the local minimum is found, a new optimization problem is formulated to maximize the distance from $\boldsymbol{x}^*$ subject to an extra constraint that the new point must have an objective value at least as good as $f(\boldsymbol{x}^*)$.

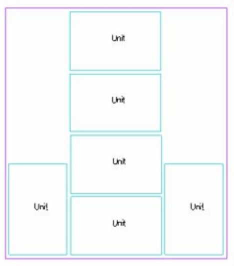
$$\begin{aligned} &\text{maximize } (\boldsymbol{x} - \boldsymbol{x}^*)^2 \\ &\text{subject to } \boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{0},\ \boldsymbol{g}(\boldsymbol{x}) \le \boldsymbol{0},\ f(\boldsymbol{x}) - f(\boldsymbol{x}^*) \le \boldsymbol{0} \\ &\boldsymbol{x} \in \Re^n \end{aligned} \tag{68}$$

If optimizing Eq. (68) yields a solution, $\boldsymbol{x}^\dagger$, in a new area of the design space, then optimizing Eq. (1) again with $\boldsymbol{x}^\dagger$ as a starting point will tend to yield a better local minimum. This process can be repeated by iteratively solving Eq. (1) and Eq. (68) to obtain better solutions. MDDM is not guaranteed to converge to the global optimum; however, in practice there are many situations where this method is successful at improving the quality of the local optimum returned. An example is provided in Figure 25. The method is especially useful if $f(\boldsymbol{x}^*)$ is flat in some feasible direction at $\boldsymbol{x}^*$.

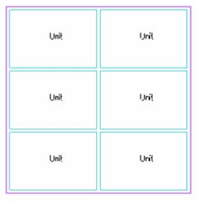
An auto-improve feature has been implemented to set up and run MDDM iteratively after finding a local minimum, and it has been successful at improving designs in some situations. Combining the auto-improve feature with the auto-perturb feature (see Section 3. 3. 1) one can create interesting results that move far from the initial starting point and sometimes obtain global solutions (see Figure 25). If the MDDM formulation cannot find a point satisfying Eq. (68) using local search, then the design is perturbed. If the perturbation moves the search area enough, the algorithm will explore other areas of the design space, and it may find another point. If not, the design will be perturbed again. Eventually, an improved design may be found. If an improved design has not been found by some limit number of perturbations, then the algorithm reverts to the last best known feasible design and terminates. This method is heuristic; however, in some situations it improves the design significantly before terminating.



(a)



(b)



(c)

Figure 25. Demonstration of the MDDM method for finding improved local optima.

An initial design (a) was optimized using CFSQP. The result is a local optimum (b) (the design cannot be improved by small changes in the design variables). The MDDM method was used to generate design (c), an improved local optimum for this problem.

3. 4. 5 Strategic Exploration

Another design exploration program was written to produce design alternatives by searching the space using a strategy of random design changes. To use this program, the designer sketches an initial design using the interactive design tool. The program then makes design moves of three types:

1.Swap the positions of two Units.
2. Perturb the position of a Unit.
3. Reduce the size of a Unit.

After each design move, the program attempts to re-optimize using the geometric optimization algorithm. The algorithm first attempts to find a feasible design using penalty methods. If it is unable to find a feasible design, the program makes another design move at random. When a feasible design is found, it is saved, and the program continues by making more random design moves. This strategy was used to generate designs for a simple three-bedroom apartment layout. The program generated 200 design alternatives overnight. A sample of generated designs is shown in Figure 26. Although this strategy is not rigorous, it is a useful tool for generating a spread of design alternatives that can be explored further with the interactive design tool (Chapter 5).

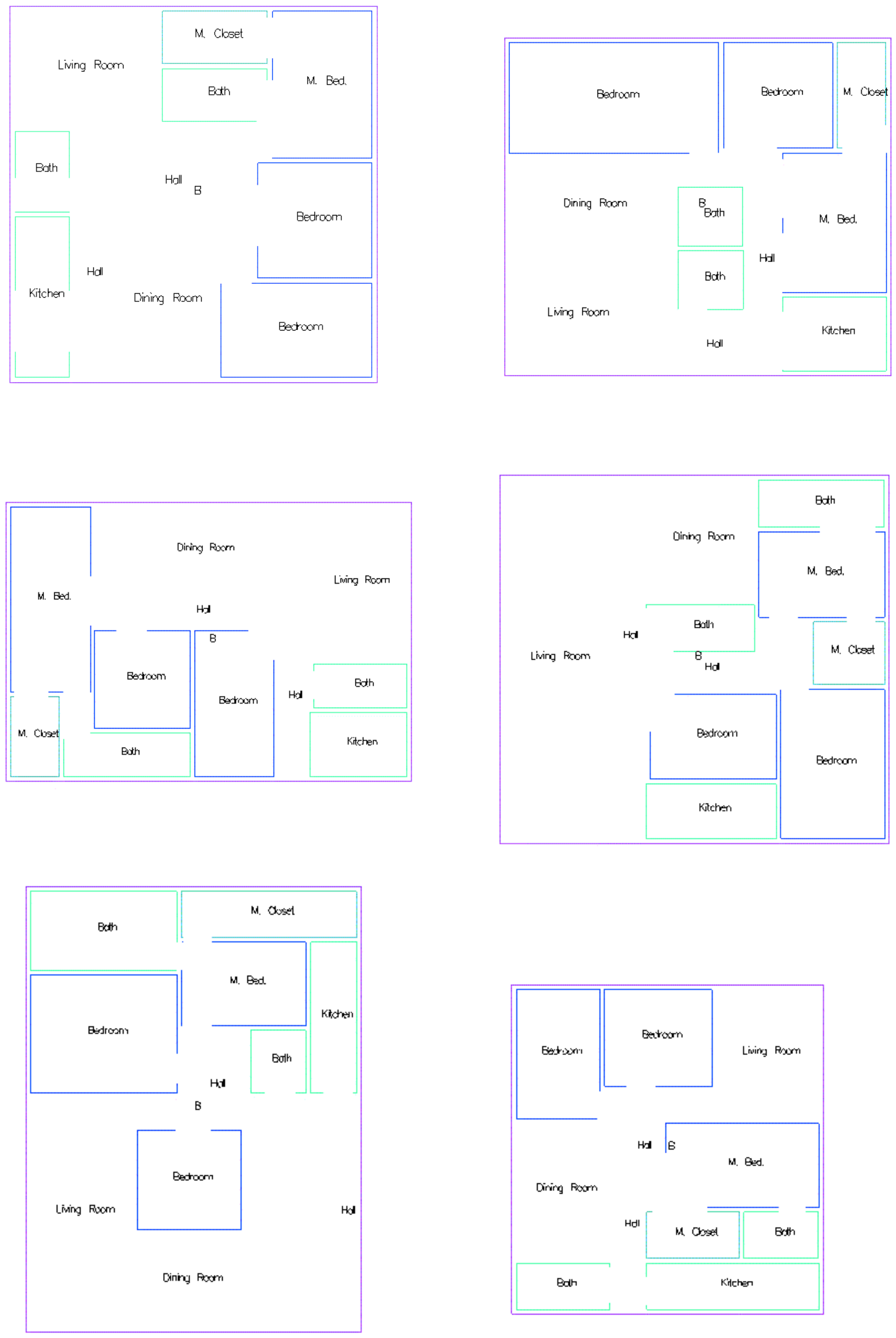
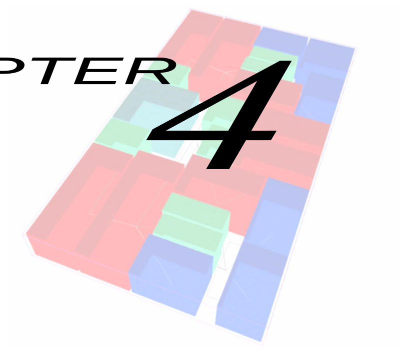


Figure 26. Sample designs generated by the strategic exploration algorithm.

3. 5 Summary

Given an initial geometry, geometric constraints, and a design objective, the geometric optimization algorithm is successful at generating a local optimal geometry by finding the best size and position for each Unit. The algorithm assumes that the layout geometry can be described by combinations of orthogonal rectangles. Additional methods have been implemented (Section 3. 4) to include some global exploration with the local search. These global methods are successful to some extent; however, it is better to use local search for situations when the designer wishes to explore specific areas of interest and guide search. In geometric optimization, the size and position of Units can vary, but the Units themselves and Unit interactions (connectivity, etc.) are treated as constant constraints. The designer can explore different Unit topologies by interactively trying various alternatives (Chapter 5), or the designer can use the topology optimization algorithm (Chapter 4) to automatically explore configuration alternatives by passing each valid alternative to the geometric optimizer.

CHAPTER 4



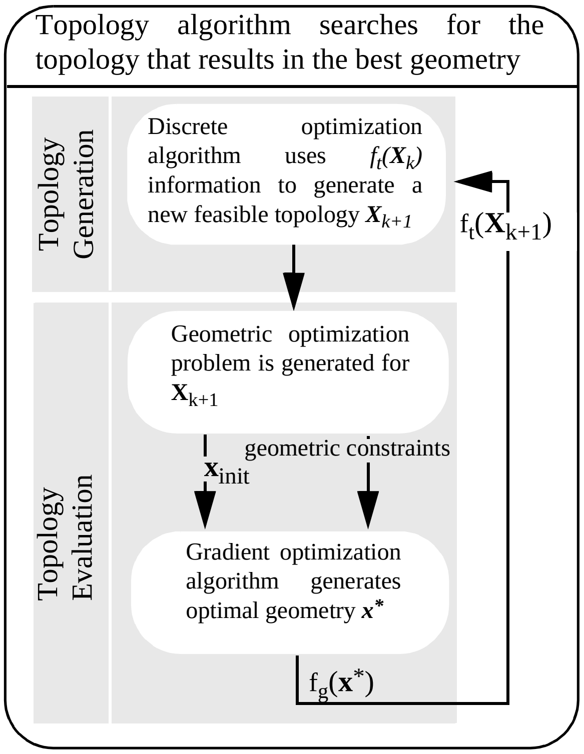
OPTIMIZATION OF TOPOLOGY

4. 1 Problem Formulation

The topology optimization problem is presented as a process of finding the best set of relationships between rooms in a space. In this formulation, relationships include connectivity, and initial rough location. Connectivity defines which rooms are directly connected by a doorway or open pathway. Rough location defines rough arrangement of rooms. Other models ([7],[8]) have used decision variables to define topological spatial relationships (i.e.: adj-to-north-of, adj-to-south-of, etc...). However, the use of rough room position to describe spatial relationships does not enforce these relationships during geometric optimization, so the geometric optimization algorithm has more freedom to manipulate the geometry.

Design objectives must be defined in order to evaluate a given topology. Topologies could be evaluated based on topological qualities, such as openness, proximity, directionality, or symmetry; however, even though these aspects are often thought of as topological, they are difficult to evaluate without rough geometry. It is best to evaluate objectives using a geometric layout, therefore we evaluate each topology based on the best geometry that can be generated from it. Using this method, layouts can be optimized for any objective that can be formulated in terms of geometry or topology.

Figure 27 shows the topology optimization process. A discrete optimization algorithm uses information from previous topologies to generate new topologies. Each new feasible topology, $\boldsymbol{X}$, is translated into a geometric optimization problem (Section 3. 2). A locally optimal geometry, $\boldsymbol{x}^*$, is found (Section 3. 3), and the quality of that geometry, $f_g(\boldsymbol{x}^*)$, defines the quality of the topology that generated it, $f_t(\boldsymbol{X})$. The discrete optimization algorithm searches for the topology that generates the best geometry.



Topology algorithm searches for the topology that results in the best geometry
Topology Generation
Discrete optimization algorithm uses $f_t(\boldsymbol{X}_k)$ information to generate a new feasible topology $\boldsymbol{X}_{k+1}$
$f_t(\mathbf{X}_{k+1})$
Topology Evaluation
Geometric optimization problem is generated for $\mathbf{X}_{k+1}$
$\mathbf{x}_{init}$
geometric constraints
Gradient optimization algorithm generates optimal geometry $\boldsymbol{x}^*$
$f_g(\mathbf{x}^*)$

Figure 27. Building topology optimization method

4. 2 Mathematical Model

4. 2. 1 Variables

The variables for the topology optimization problem are the initial grid position of each room, and the connectivity between each room and every other room/external wall.

$$x_i, y_i \in Z_+ \tag{69}$$
$$\phi_{ij} \in \{0, 1\}$$
$$\forall i \in \{rooms\}, \forall j \in (\{rooms > i\} \cup \{extwalls\})$$

where (x_i, y_i) represents integer cartesian coordinates of room *i*, and ϕ_{ij} represents the existence of a connection between room *i* and room *j* (or external wall *j*). Figure 28 shows a visual representation of the design variables.

It is important to note that topological decisions about relative position of rooms (i.e.; room *i* is-north-of room *j*) are represented here using absolute positions of rooms. It is necessary to use absolute position in this representation so that the topology can be

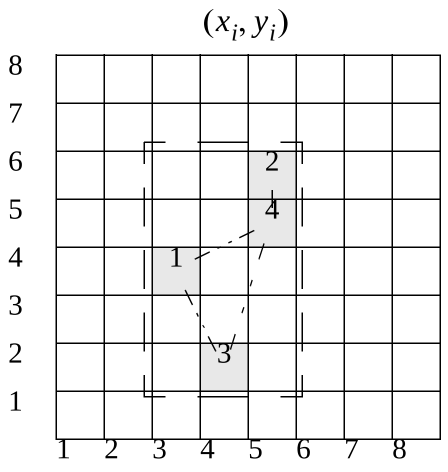


Figure 28a. Room position grid showing (x,y) for each room. Phantom lines show room connections. The dashed line shows the implied boundary.

ϕ_{ij}

	N	S	E	W	1	2	3	4
N					*0*	*1*	*0*	*0*
S					*0*	*0*	*1*	*0*
E					*0*	*1*	*0*	*1*
W					*1*	*0*	*0*	*0*
1						*0*	*1*	*1*
2							*0*	*1*
3								*1*
4								

Figure 28b. Room connectivity, ϕ_{ij}. A "1" represents a connection between *i* and *j*. Connections to compass directions (N, S, E, W) represent windows or doors on external bounds of the building.

Figure 28. A 4-room example showing design variables in the topology formulation

translated into initial geometry for geometric optimization. Several other methods of representing topological decisions ([7],[8]) do not use absolute positions; however, it is necessary in this strategy because the geometric optimization algorithm requires a starting design with geometric information. For example, if the starting design defined all Units to be positioned at (0,0), the geometric optimization algorithm would be unable to use gradient information to search for feasible geometries because of the nature of the constraints at that point (see Section 7. 1. 9). The use of absolute positions has several consequences:

The mapping from topology to geometry is not injective (one-to-one). It is possible for more than one topology to generate the same geometry. This means that computation time can be wasted searching similar topologies.

The mapping from topology to geometry is not surjective (onto). Because each room is represented as a grid point, each topology could be interpreted geometrically in several ways (see Figure 29). It is not clear, however, that every geometric alternative can be generated using the topology definition in this thesis.

The space of topology combinations is exponential.

For a grid size of mXm squares and n rooms, there are m^{2n} possible room position combinations. This is a large space of possibilities. Compare this to the 4^n possible connection combinations generated when each room relationship can take on the 4 allele values: {adj-to-north, adj-to-south, adj-to-east, adj-to-west}.

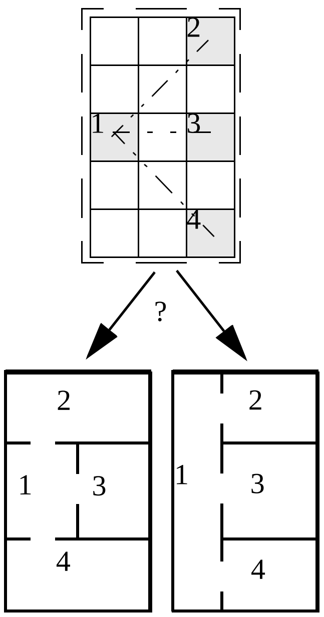


Figure 29. Alternative geometric interpretations of a topology.

Because of these limitations, this representation is not well suited to small problems where all solutions need to be enumerated (see Section 7. 1. 9 for thoughts on improving this shortcoming). It is not clear that the representation can enumerate all possible topology alternatives; however, this method is powerful for larger problems where heuristic search is necessary. This is because in practice heuristic search algorithms can often find reasonable quality designs quickly, while enumeration algorithms must systematically explore designs one by one, which can often take too long to be practical.

4. 2. 2 Overlap Constraints

This constraint ensures that no two rooms occupy the same space.

$$|x_i - x_j| + |y_i - y_j| \geq 1 \quad \forall i \neq j \tag{70}$$

4. 2. 3 Connectivity Constraints

Connectivity constraints are defined by the designer for each problem. The constraints describe how a certain room is required to be connected to an outside wall, to another room, or how certain rooms are required to *not* be connected. For example,

$\phi_{ij} = 1$ room *i* required to connect to room *j*, (71)

$\phi_{ij} = 0$ room *i* required not to connect to room *j*,(72)

$\phi_{iN} + \phi_{iS} + \phi_{iE} + \phi_{iW} \geq 1$ room *i* required to connect to at least one external wall. (73)

4. 2. 4 Path Constraints

Path constraints are defined by the designer for each problem. A path may be required between all combinations of rooms, or a path may be required between certain rooms. For example, a path could be required from the bedroom to the kitchen without passing through a bathroom or closet. These constraints involve room connectivity, and they are generated for each specific constraint with an algorithm (see Appendix B). An example is shown in Figure 30. In this 4-room example, at least one path from room 1 to room 4 must be connected excluding paths passing through room 3:

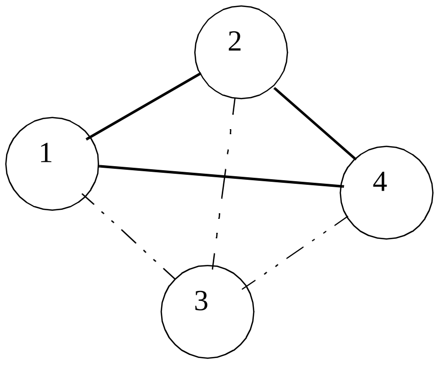


Figure 30. Example connectivity graph showing alternative paths

$$\phi_{14} + (\phi_{12}\phi_{24}) \geq 1 \tag{74}$$

This constraint will be satisfied if rooms 1 and 4 are connected ($\phi_{14} = 1$), or if both room 1 is connected to room 2 ($\phi_{12} = 1$) *and* room 2 is connected to room 4 ($\phi_{24} = 1$).

4. 2. 5 Planarity Constraints

The room connectivity and position must be such that the geometry can be realized with a two-dimensional (planar) floorplan. One way to ensure planar feasibility is to draw lines between connected nodes on the position grid and ensure that no two lines cross. These lines will be allowed to share endpoints as long as they do not share any interior point. This constraint is difficult to represent with a closed form mathematical function; however, the planarity check function can be found in Appendix C.

In Figure 31, the planarity check would interpret the following: 2-3 does *not* intersect 3-4 because they share only an end node (node 3). 1-4 intersects 2-3 because they cross (non-planar), and 3-4 intersects 2-5 because they cross at node 4 (which is an interior point of line 2-5).

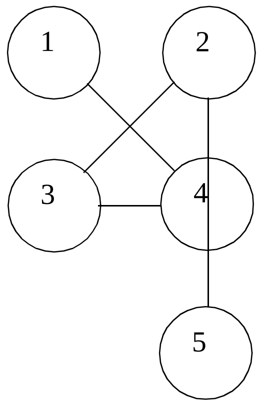


Figure 31. Example connectivity graph showing alternative paths

4. 2. 6 Envelope Constraints

Units that are forced to be connected to an external wall must lie on the external envelope of Units on that wall. The four constraints below are added for each unit *i*.

$$\phi_{iN}(\max(y_1, y_2, \ldots y_n) - y_i) = 0 \tag{75}$$

$$\phi_{iS}(y_i - \min(y_1, y_2, \ldots y_n)) = 0 \tag{76}$$

$$\phi_{iE}(\max(x, x_2, \ldots x_n) - y_i) = 0 \tag{77}$$

$$\phi_{iW}(x_i - \min(x_1, x_2, \ldots x_n)) = 0 \tag{78}$$

4. 2. 7 Objective

The objective of the topology optimization problem is to minimize the objective value of the resultant local optimal geometry formed by the topology.

$$\text{minimize}(\ f_g(\ \text{SQP}(X)\ )) \tag{79}$$

where f_g is the objective value of the geometry, and SQP is a function that returns the local optimum geometry for the topology $\boldsymbol{X}$. Notice that **x** and **y** determine starting locations for rooms in the geometry formulation while ϕ defines constraints for the geometry as well as windows and accessways (see Figure 32). The optimization objective can be anything defined by the geometry. Typically, we have optimized for the topology that produces the most cost efficient layout.

Only feasible topologies are passed to the geometric optimizer (SQP). If the topology violates any constraints, then the design is evaluated using penalty functions.

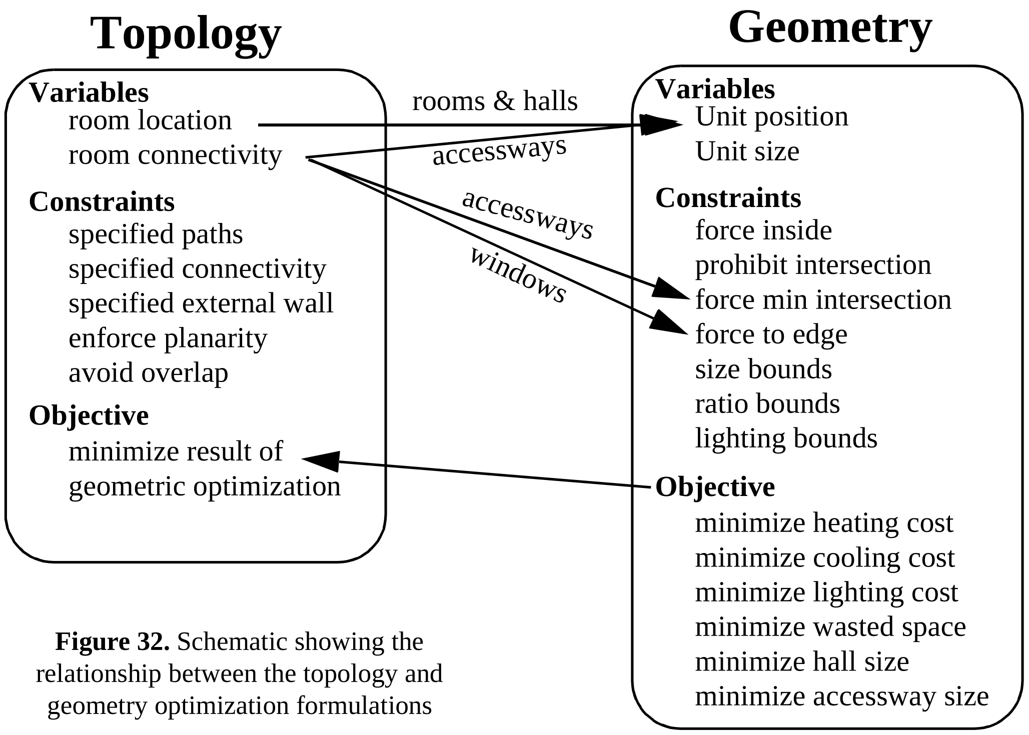


Figure 32. Schematic showing the relationship between the topology and geometry optimization formulations

4. 2. 8 Penalty functions

Many non-linear discrete search algorithms do not have an explicit way to handle constraints. Constraints can be handled either by limiting the operators to produce only feasible moves or by using penalty functions to penalize the objective function for infeasible designs. In this formulation, penalty functions are used, and infeasible designs are not passed to the geometric optimizer.

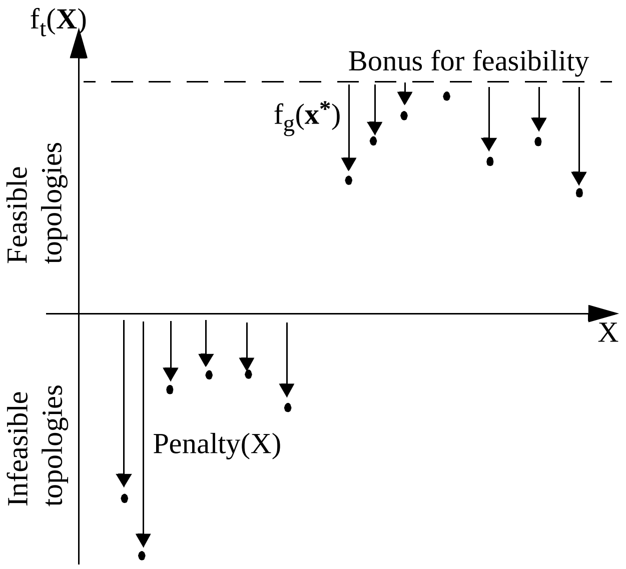


Figure 33. Formulation of the topology objective function

Figure 33 shows how this penalty function works. Here we are trying to maximize the topology objective function $f_t(\boldsymbol{X})$. The following procedure is used to evaluate a topology:

If the design is infeasible, f_t returns a negative value that is penalized for each constraint violated, and for the extent of violation.

If the design is feasible, **X** is passed to the geometric optimization algorithm. Assuming the geometric algorithm finds a feasible geometry ($\mathbf{x}^*$), f_t returns a bonus value (B) minus the objective function value of the geometric optimum $f_g(\mathbf{x}^*)$. The bonus value is set so that it is larger than any objective value $f_g(\mathbf{x})$.

Using this method, all infeasible topologies return negative function values, all feasible topologies return positive objective function values, and feasible topologies that result in better geometries (*lower* $f_g(\boldsymbol{x})$) have a better objective function value *(higher* $f_t(\boldsymbol{X})$).

4. 3 Global Optimization Methods

The discrete topology design space is multi-modal, highly constrained, and highly infeasible, so it must be searched with a global scope. The space of topologies could be searched exhaustively with a CSP enumeration algorithm [37] or branch and bound; however, combinatorial explosion will cripple the algorithm for problems of significant size. Furthermore, enumeration is unnecessary in a problem where many of the implicit design goals (such as aesthetic intent) are not generally defined mathematically, but instead must be judged. It is not meaningful to produce a strict global optimum; instead, it is more useful to produce an array of quality design alternatives to explore. For this reason, evolutionary algorithms (Section 3. 4. 3) were selected. Evolutionary algorithms search heuristically, and they can be stopped at any point during the optimization process to return a population of best designs found. This heuristic search, combined with penalty functions, can often find quality feasible designs to large problems that are intractable for systematic search methods.

4. 3. 1 Constraint Satisfaction Algorithms

Constraint satisfaction algorithms include a number of techniques for improving search speed and reducing complexity including backtracking, node-consistency checking, arc-consistency checking, path-consistency checking, forward checking, look-ahead, and look-back schemes. Also, heuristic search strategies and value ordering techniques can reduce search time.[6] These techniques make a CSP representation a good alternative for some classes of problems. Specifically, these techniques have been developed for binary constraint satisfaction problems (where each constraint involves only one or two variables). Theoretically this is useful because any constraint can be translated into a group of binary constraints by introducing new variables For example, Figure 34 shows how a constraint involving three variables ($x + y = z$) can be translated into a four-variable problem with only binary and unary constraints. In this translation, a new encapsulated variable, w, is introduced. The domain of w is the set of vector combinations of the individual variables. Theoretically, this binarization conversion method allows any constraint problems to be solved using binary CSP methods; however, in practice binarization is generally not worth doing because the new variables add significant complexity to the search space.

6. For more information on CSP solution strategies, see [37].

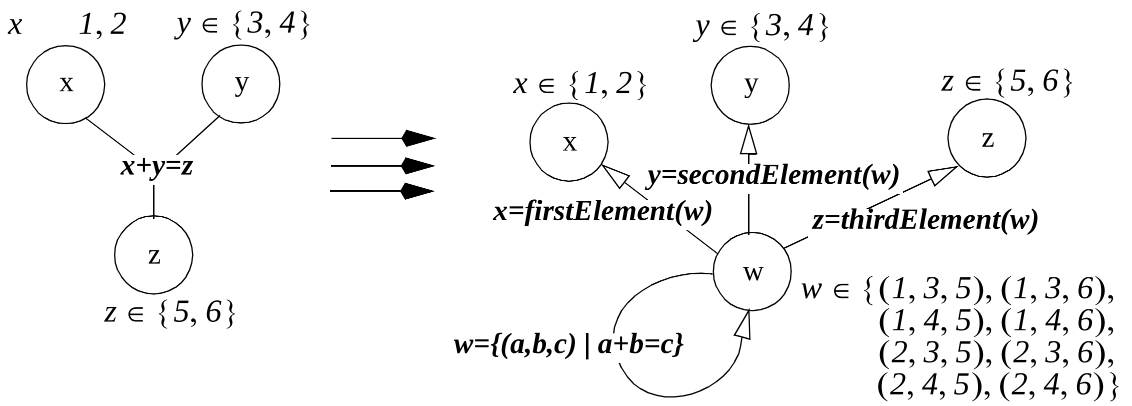


Figure 34. Conversion of a 3-variable constraint into 2-variable and 1-variable constraints by introducing an encapsulated variable

CSP solution techniques were not explored for this problem representation because of the large number of multi-variable constraints and constraints that were difficult to express explicitly. For example, the path constraints and planarity constraints would be difficult to represent as a set of binary constraints. It is important to note that other problem formulations may be more suited to this CSP representation ([6],[7],[8]).

4. 3. 2 Evolutionary Algorithms

An evolutionary algorithm for topology layout was implemented using GAlib, an evolutionary algorithms optimization package [37]. A SteadyStateGA was selected (described in Figure 23). Selector, crossover, and mutation functions are defined below, and an example to demonstrate the process is shown in Figure 35.

Selector: A Roulette Wheel selector was used to select high quality designs with greater probability than low quality designs. (Design A and B are selected from the population P_k.)

Sexual Crossover: When sexual crossover is used, two parents are selected from the population, and two new children are produced using mixed room connectivity from both parent. (One child is shown as Design D.)

Asexual Crossover: When asexual crossover is used, one parent is selected from the population, and one new child is produced by swapping connectivity values between rooms or by swapping room positions. (Shown as Design C.)

Mutation: After crossover, new designs are mutated slightly. Room locations (x,y) are incremented or connectivities are flipped with low probability. (Shown as Design E and F.)

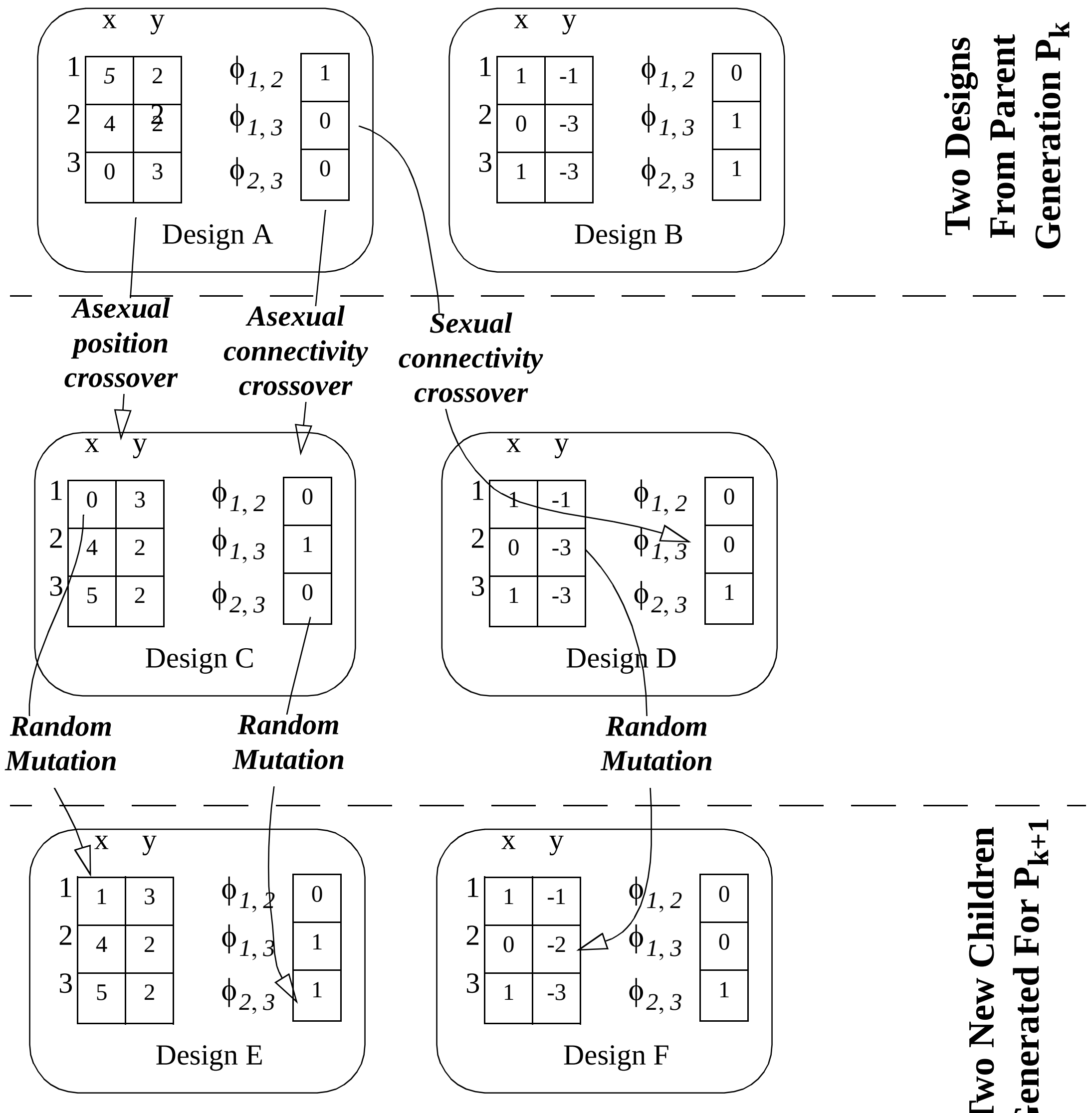


Figure 35. Process of generating a new design population using crossover and mutation operators

The evolutionary algorithm implementation is able to generate quality feasible designs for medium-sized example problems. An example problem is explored in Chapter 6.

4. 4 Summary

Given constraints about building topology, the topology algorithm is able to search for feasible topologies using stochastic techniques. Because of the stochastic nature, the algorithm cannot guarantee a solution in finite time; however, in practice the algorithm has been successful at generating feasible topologies for medium-sized problems (see Chapter 6). The topology algorithm evaluates each feasible topology based on the best geometry that can be generated from it using the geometry algorithm, assuming that all design objectives can be evaluated based on the geometric outcome. In this process, room connectivities are translated as constraints in the geometric algorithm, and room positions are translated into starting points for the geometric design variables. Using this approach, the topology algorithm searches for the topology that will result in the best geometry.

CHAPTER 5

INTERACTIVE DESIGN OPTIMIZATION

A building geometry optimization design tool was created to allow the designer to interact with the design problem in a number of ways including interactively defining the problem, guiding search for a solution, and exploring design alternatives. This tool offers a new powerful approach to using optimization in the design process. Instead of using optimization in the final stages to fine tune a solution to a well-defined problem, the optimization tool is used to help to refine the problem itself and to interactively explore solution alternatives and trade-offs while receiving both visual and computational feedback.

The geometric optimization approach uses CFSQP (Section 3. 3. 1), which guarantees that once a feasible design has been found, every following iterate will be feasible. This means that each iteration of the algorithm yields a feasible design alternative, and the progression of the algorithm moves toward improved design alternatives. Allowing a designer to see this progression of designs visually and intuitively introduces opportunities for making design and modeling decisions based on the progression of the algorithm. The object-oriented implementation allows for these changes to be made during optimization. If the problem formulation is changed, the new formulation is automatically updated, and search in the new design space begins with the last design found before the change. In this case CFSQP will search a new design space with changed shape and possibly changed dimensionality; however, the user sees an uninterrupted progression of designs.

This chapter presents a description of the interactive design tool capabilities and describes advantages of the tool. A case study using the tool is presented in Chapter 6.

5. 1 Interactive Problem Definition

The interactive design tool allows the designer to add, delete, and modify objectives, constraints, and Units during optimization to refine the problem definition. The designer can set up the initial problem and start optimization. At each iteration, the current design is displayed. The designer can watch how the design is changing and use that information to change the problem definition at any point during the optimization. This is useful because design is an iterative process for the designer as well as for the algorithm. When the designer has visual feedback, s/he can realize new preferences or forgotten constraints, and s/he can explore how changes in the problem definition affect the design solutions.

Refining the problem during optimization is accomplished using an object-oriented representation of the building. Each time an optimization is performed, the program translates the object-oriented representation into a set of mathematical design variables, objectives and constraints. This automatic translation allows a new mathematical design space to be formulated automatically with a different dimensionality or different objective and constraint functions. Underneath, the problem is actually being regenerated, and a completely new mathematical optimization is started in the new space at a point analogous to where the designer left off in the old space. However, the mathematical reformulation is hidden from the designer, so it appears as though the optimization is progressing naturally with the new design change. This allows a new way to think about modeling changes during optimization because the designer sees modeling changes as simple design moves that can be easily added and experimented with during conceptualization.

5. 1. 1 Multi-Objective Optimization

If the designer chooses more than one objective function to optimize, the individual objectives are combined into a single objective function using a weighted sum (Eq. (67)). The weights, or relative importance, of each individual objective can be changed to reflect preference in competing objectives. Defining appropriate weights for a set of objectives is nontrivial; however, using the interactive tool, the designer can change objective weights as s/he observes how the design is progressing during optimization. Defining appropriate weight values is easier once the designer can see how the designs react to a particular set of weights. Furthermore, a good set of weights in one area of the design space may be poor in another area, so it is important to have the flexibility to change them during optimization. It is common practice to revise objective weights *after* finding an optimum; however, this method allows the designer to interject *during* the process if s/he sees the design migrating toward an undesirable area of the design space. In the future, other methods of describing objective preference, such as using aspiration points ([47],[48]), could be implemented to make the process more intuitive for the designer.

5. 1. 2 Addition, Deletion, and Modification of Objectives

As the designer receives feedback of the optimization progression, s/he may want to change the definition of the design objective.

Adding an Objective: A designer watching the design progression will be able to notice many types of layout deficiencies visually, and s/he can add a new objective to enforce a preference away from the deficiency. For example, if layout solutions are lacking on use of space for living areas, the designer can add a Minimize Wasted Space objective.

Deleting an Objective: After seeing the optimization progression, the designer may decide that some objectives are unimportant, or the designer may wish to remove an objective to simplify the problem or observe how the design progression reacts without the objective. If all objectives are removed, then the algorithm terminates when it finds a feasible design.

Modifying an Objective: The designer can modify weights (relative importance) of each objective to specify preference (discussed in Section 5. 1. 1).

5. 1. 3 Addition, Deletion, and Modification of Constraints

As the designer receives feedback of the optimization progression, s/he may want to change the definition of the design constraints.

Adding a Constraint: If the design progresses into an undesirable area of the design space, the designer can dynamically add new constraints to prevent search in that area.

Deleting a Constraint: After seeing the optimization progression, the designer may decide to remove certain constraints in order to achieve a better solution. Often decisions about which constraints should be ignored cannot be made until a designer has seen some physical designs.

Modifying a Constraint: Some constraints can be relaxed by modifying a numerical bound, such as a minimum area constraint. Once the designer has seen some feasible design alternatives, s/he may choose to relax certain numerical bounds in order to achieve a better solution.

5. 1. 4 Addition, Deletion, and Modification of Units

As the designer receives feedback of the optimization progression, s/he may want to change the layout elements themselves.

Adding a Unit: Extra units may be added to change the problem (i.e., add an extra bedroom or closet) or to enforce connectivity (i.e., add an extra accessway).

Deleting a Unit: Units may be deleted if they become extraneous in a particular layout. Rooms may be deleted (explore a two bedroom instead of a three bedroom apartment) or forced connections may be relaxed (i.e.: remove accessway (connectivity constraint) and allow two Rooms to separate).

Modifying a Unit: Units can be stretched or moved during optimization to force search into a different area of the design space. Modification can be used to guide search into an area of interest (discussed in Section 5. 2).

5. 1. 5 Change of Variable Formulation

The geometric layout design problem can be formulated with several alternative variable representations (see Section 3. 1. 1) each having its own representation used by CFSQP (defining which terms are optimization variables and which terms are resultant calculations). The object-oriented representation, however, can calculate constraints and objectives regardless of which design parameters are used as optimization variables. It would be easy to switch between representations during optimization if one representation produced better optimization behavior in certain areas of the design space.

5. 2 Interactive Optimization

With the ability to modify design variables during search, the designer can guide the the optimization process. Because the design variables are geometric in nature, the designer can interact with the variables in an intuitive way. If the designer sees the design moving into an undesirable area of the design space, s/he can intervene and force search into a new area of the space by manipulating Units. This method uses the designers experience and intuition to guide global search along with the efficiency and accuracy of gradient algorithms to direct local search.

In addition, the designer can also help the optimization algorithm to avoid computational traps. Gradient based algorithms assume that all functions of the design variables are continuous and have continuous derivatives. If the problem representation violates these assumptions, the algorithm may have unpredictable behavior. In particular, the geometric building layout formulation presented in Chapter 3 has non-smooth gradient constraint functions in some areas of the design space. If the algorithm has computational difficulties near one of these areas, usually it will appear to be stuck, and the designer can nudge the Units slightly away from the non-smooth area to resume normal optimization. Typically enlarging the building boundary slightly is enough to resolve the computational difficulty. This ability is important for problems that have some irregularities.

5. 3 Interactive Design Exploration

5. 3. 1 Interactive Sketching

The interactive layout optimization tool can function as an interactive sketchpad for exploring design alternatives. As a typical procedure, the designer would

Define Rooms & Halls: *Define which Rooms will be included in the building (kitchen, bedroom, etc...) and what are acceptable sizes for each Room (length, sq. ft.).*

Move Rooms Into Rough Location: *Rough dimensions also can be set by stretching.*

Define Connections: *Add Accessways to define which Rooms will be connected.*

Choose an Objective: *Choose an objective to optimize for*

Add Additional Constraints: *Add any special constraints besides those added by default*

Optimize: *The optimization algorithm will compact the geometry into a locally optimum layout.*

Examine Results: *Check results visually and check estimated performance values calculated by the objective function.*

Iterate: *Use the information to refine the problem definition or guide search into a new area.*

The problem setup (steps 1-5) can be completed in less than one minutes for a typical two-bedroom apartment. Optimization for the same problem usually terminates within a few seconds. At this point, making changes to the design, such as relocating a room, is just a matter of dragging rooms into alternative positions and re-optimizing. This process is extremely fast, and many design alternatives can be easily examined both visually and computationally. The speed and simplicity offers a lot of potential as an exploratory tool. Examining alternative configurations is faster than sketching, and possibly more intuitive because the designer manipulates objects (Rooms, Halls,...) instead of lines.

5. 3. 2 Design Feedback

In addition to receiving quick visual feedback about various configurations, the design tool also provides computational feedback about design performance with respect to the design objectives. The designer can immediately see quantitative data about design alternatives, including

Performance Cost: *An estimate of the annual heating, cooling, and lighting cost.*

Build Cost: *An estimate of the cost of materials (glass and walls) to build.*

Lifetime Cost: *An estimate of the net financial cost over the lifetime of the building including trade-off between build cost and annual performance cost, taking into account annual interest rate.*

Natural Lighting: *An estimate of the lighting levels in each room for a given environment.*

Living Space: *An estimate of the building area that is used for living space in comparison to area used for passageways or wasted space.*

In the current implementation, all of these estimates are rough estimates, which is appropriate for a conceptual exploratory tool; however, more accurate models could be used if necessary. One drawback to using gradient-based optimization techniques is that these functions must all be smooth, clean functions in order to behave well during optimization; however, complex non-smooth simulation functions can be smoothed using surrogate modeling techniques to fit into this model [43]. The use of rough models to provide computational feedback to the designer during conceptualization offers the potential for consideration of important computational objectives early in the design process as well as the opportunity to explore how design changes affect building performance.

CHAPTER 6

DEMONSTRATION STUDY

This chapter reports on studies that illustrate the use of the automated design tool and the interactive design tool.

6. 1 Automated Design Tool

A large scale problem was implemented to test the scalability of the automated building generation algorithm. This example problem involves a small apartment complex with three separate apartments. Rooms and specifications are shown as below.

Table 2: Room Specifications for Demonstration Problem

Apt	Room	Min Area (sq.ft.)	Min length & width (ft.)	Max length & width (ft.)
-	Public Entry	9	3	100
1	Living Room	160	12	40
1	Dining Room	100	10	30
1	Kitchen	100	8	40
1	Bedroom	120	10	40
1	Bathroom	30	5	20
2	Living Room	160	12	40
2	Dining Room	100	10	30
2	Kitchen	100	8	40
2	Bedroom 1	120	10	40
2	Bedroom 2	120	10	40
2	Bathroom	30	5	20
3	Living Room	160	12	40
3	Dining Room	100	10	30
3	Kitchen	100	8	40
3	Bedroom 1	120	10	40
3	Bedroom 2	120	10	40
3	Bathroom	30	5	20

Topological constraints are defined as in Section 4. 2. Constraints that are specific to this problem are listed below.

Table 3: Topology Specifications for Demonstration Problem

Constraint Type	Section	Constraint
Overlap	Section 4. 2. 2	No two Units can occupy the same space
Connectivity	Section 4. 2. 3	PublicEntry must connect to the LivingRoom of each apartment
Connectivity	Section 4. 2. 3	PublicEntry must connect to an external wall
Connectivity	Section 4. 2. 3	All bedrooms must connect to an external wall
Path	Section 4. 2. 4	In each apartment, there must be a path from the Kitchen to the LivingRoom that may pass through the DiningRoom
Path	Section 4. 2. 4	In each apartment, there must be a path from the Bathroom to the LivingRoom that may pass through the DiningRoom and Kitchen
Path	Section 4. 2. 4	In each apartment, there must be a path from the DiningRoom to the LivingRoom that may pass through the Kitchen
Path	Section 4. 2. 4	In each apartment, there must be a path from each Bedroom to the LivingRoom that may pass through the DiningRoom
Accessways	Section 4. 2. 5	Accessway lines connecting Units cannot intersect
Envelope	Section 4. 2. 6	Units that are connected to an external wall must lie on the boundary envelope of rooms

This problem was run for 20,000 generations (100 designs each generation) to search for global solutions (see Chapter 4 for details). Feasible designs take much longer to evaluate than infeasible designs (because they are passed to the geometric optimization algorithm), so a second termination criterion was added to terminate after 50 feasible designs were found. This criterion was intended to make search time more consistent.

Two sample solutions shown in Figure 36 were generated on separate runs using the automated design tool.

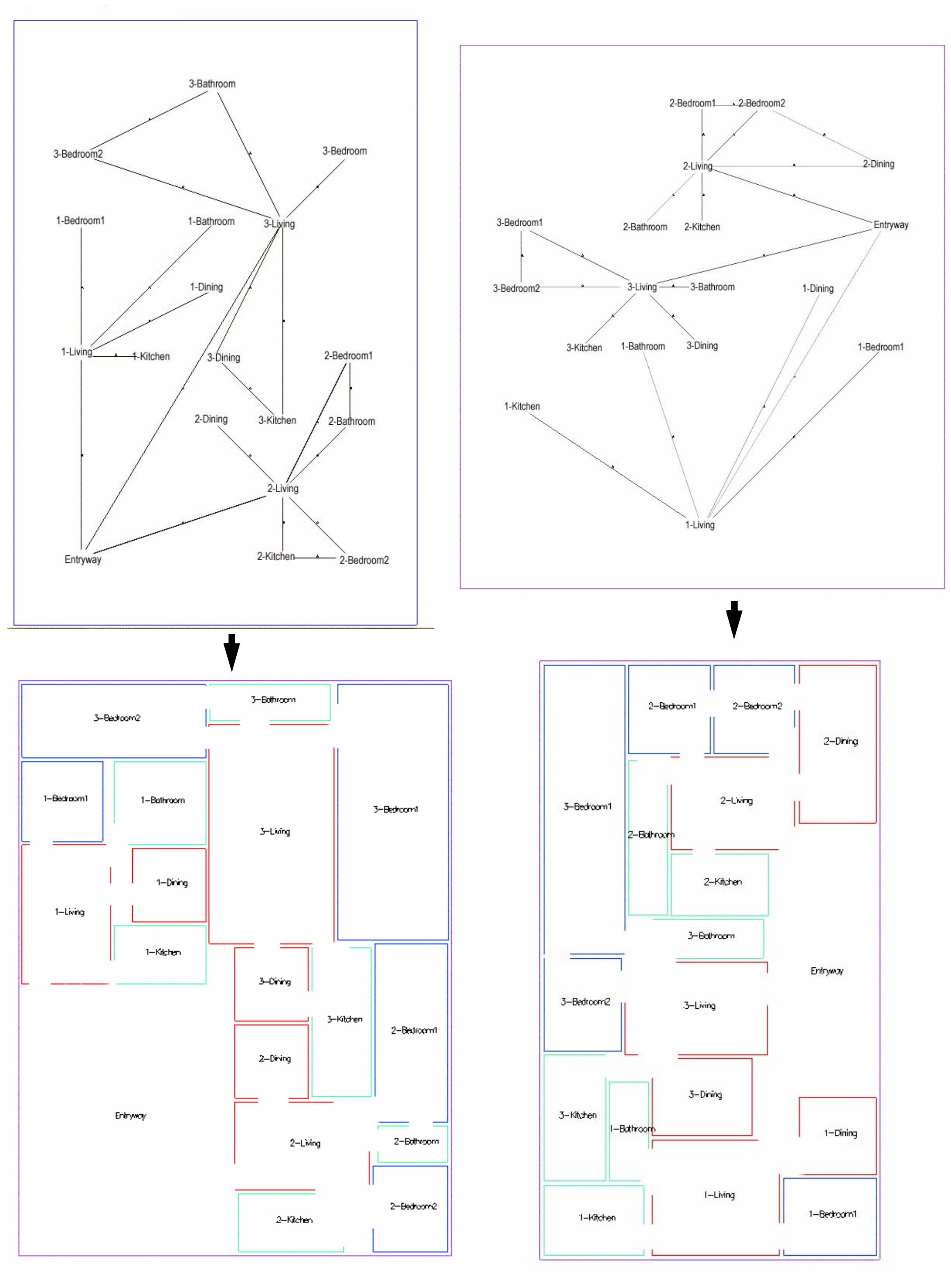
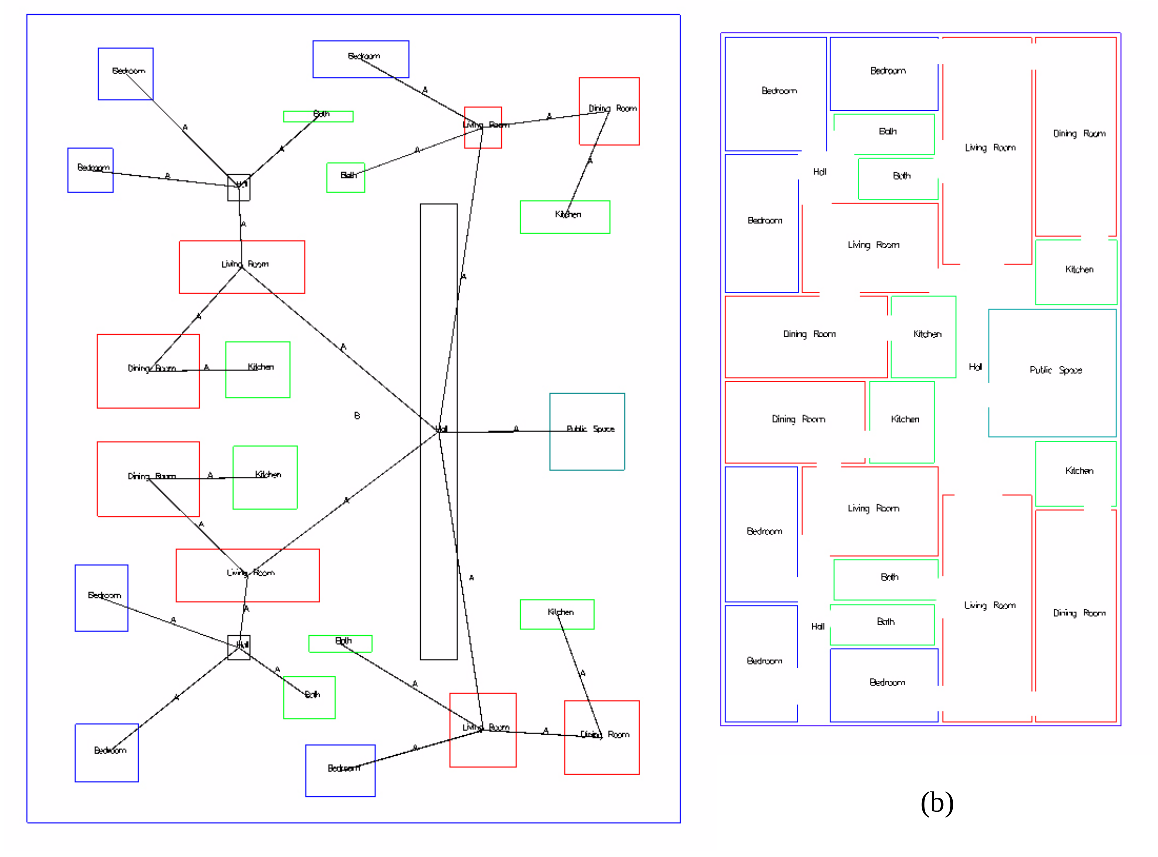


Figure 36. Sample designs generated by the automated design tool

The algorithm was able to generate local optimal solutions to this fairly complex problem. However, global search is quite limited due to combinatorial complexity. Once a feasible topology is found, it will have a much higher probability of being selected as a parent design by the evolutionary algorithm because it will have a much higher fitness value than infeasible designs. Thus, new designs tend to be very similar to the first feasible design found, and other designs are usually discarded. The result is that the algorithm tends to fixate on the first feasible solution it finds, exploring mostly variations of that solution. The algorithm can be run several times to produce design alternatives, but generally when it is run once, the final population converges to variations of one main design theme. This is a serious limitation for global search, and the algorithm is more useful as a feasible-design-finder than as a true optimizer. For smaller problems, the evolutionary algorithm is still able to search a significant range of the design space to find global quality solutions. The evolutionary algorithm may be better suited to examples like this than enumeration algorithms. For this problem, the evolutionary algorithm can consistently find solutions in under 20,000 generations (2×10^6 design evaluations). By comparison, even if the room position grid was reduced to 20 X 20 squares an enumeration algorithm would have $2^{57} \times 20^{(2 \cdot 18)} = 10^{64}$ possible combinations to search. This is far too many combinations to evaluate, and it is unclear how many need to be evaluated to find a feasible solution.

The geometric optimization problem does not have the same combinatorial nature that the topology problem has, and it is able to handle much larger problems. An example shown in Figure 17, and repeated below in Figure 37, contains 23 rooms, three hallways, one boundary, and 25 accessways for a total of 52 units. This geometric optimization problem contains 312 variables and 1578 constraints.



(a)

(b)

Figure 37. Sample apartment complex geometric optimization.
(a) Initial layout sketch provided by the designer.
(b) Optimized design.

6. 2 Interactive Building Exploration

Typically, optimization algorithms are used to fine tune design parameters to improve a pre-existing parameterized design concept. The interactive optimization software presented in this thesis is intended to use optimization as a tool to aid in design conceptualization and exploration. A sample design problem is shown here for laying out a simple apartment. The example will show how the optimization tool can be used to help the designer to:

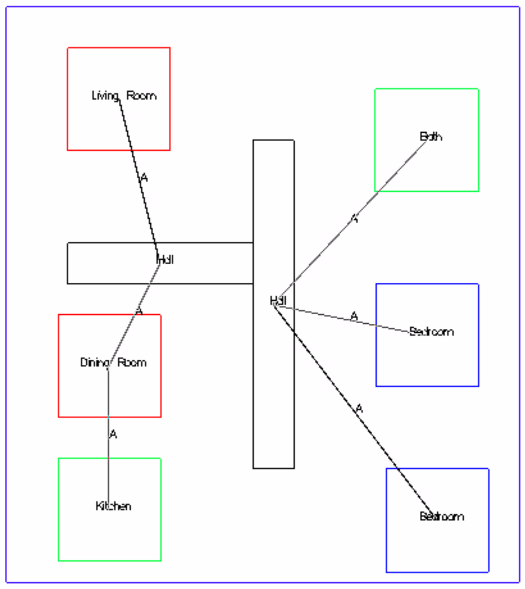
-- Quickly sketch design concepts

-- Receive visual and computational feedback on design concepts

-- Formalize the design problem objectives and constraints

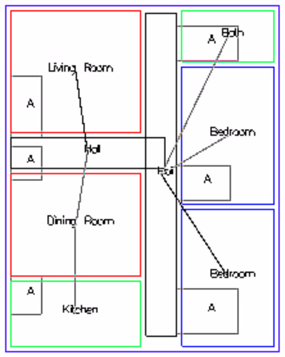
-- Use visual and computational feedback to refine the problem definition

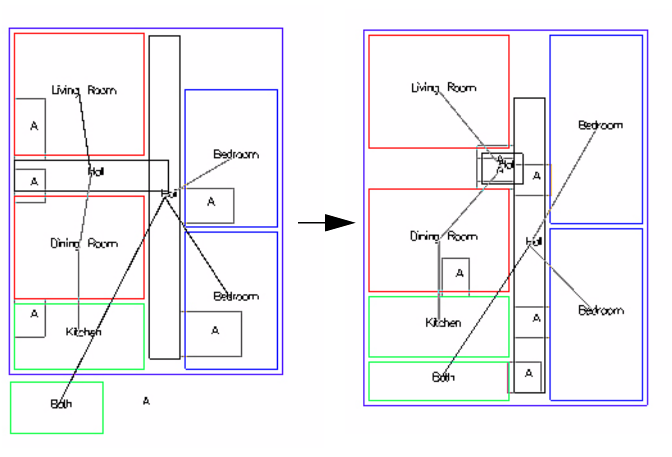
-- Quickly explore design alternatives and design trade-offs



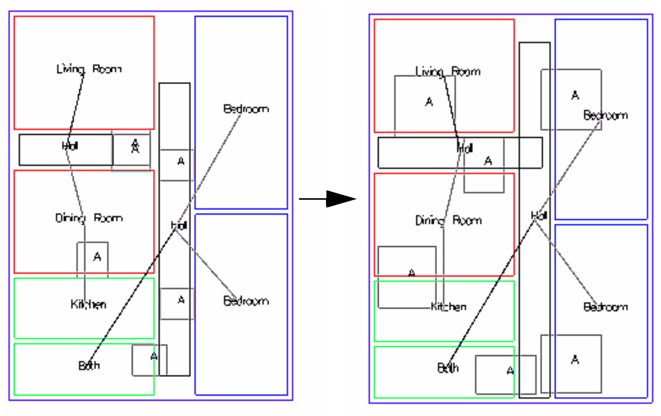
First, the initial layout concept is sketched out. This design will consist of two bedrooms, a full bath, kitchen, dining room and living area with a hallway structure consisting of two main paths. Connections are defined as shown in the picture by adding accessways between Units. Each Unit is initialized with its own default constraints for minimum area, length, and width (Section 3. 2. 7). Default constraints are automatically added to prohibit intersection between all Room-Room and Room-Hallway combinations (Section 3. 2. 4). Default constraints are added to force all Units inside the building bounds (Section 3. 2. 3). Each Accessway is added by the designer to guarantee connected access between two Units by doorway or opening (accessways represented here as lines drawn between Units). Default constraints are added that force intersection between the Accessway and each of the connected Units such that the intersection overlap is large enough for a door or opening (Section 3. 2. 5). Using the defaults, this entire process takes about one minute.

Next, design objectives are specified. This design is optimized to minimize annual heating cost (Section 3. 2. 12) and minimize wasted space (Section 3. 2. 15). The optimization process takes a few seconds, and the solution is shown here (accessway connectivity is represented as lines between Units, and actual accessway position is shown by the rectangular Units marked “A”).

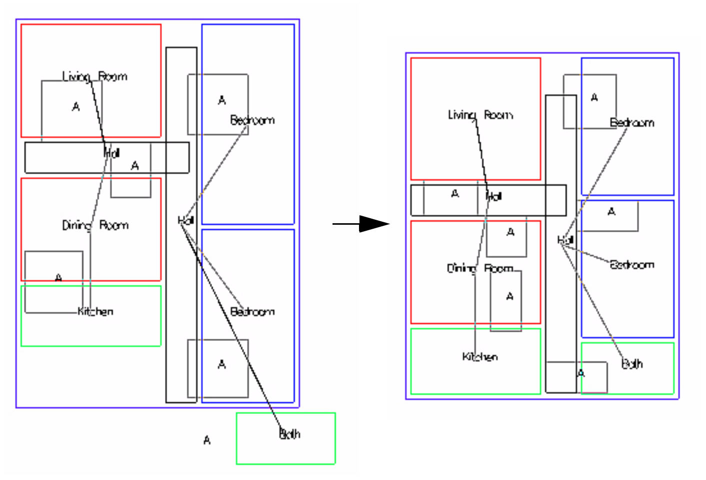




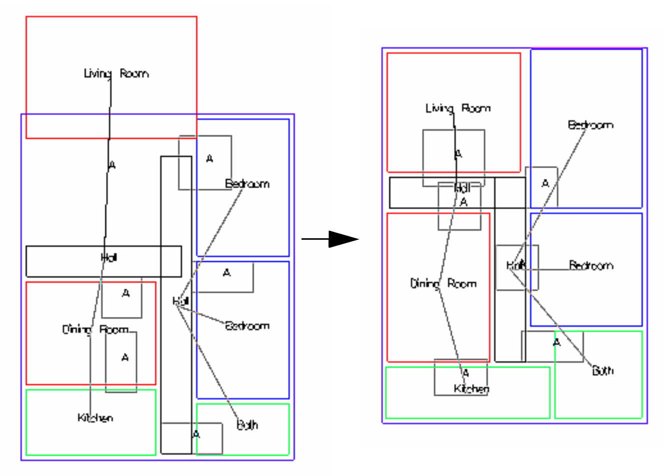
Suppose that after viewing the results, the designer decides that it will be more economical if the bath and kitchen are close together so that piping can be clustered. The bath is dragged down below the kitchen, and the design is re-optimized. The optimization algorithm repacks the Units. Results show that the designer forgot an important constraint. The Hallway between the living room and dining room was intended to be an entryway, but the designer failed to specify that the Hallway must connect to an external wall. The designer now adds a new constraint to force the hall against the west wall (Section 3. 2. 6) and optimizes the design again.



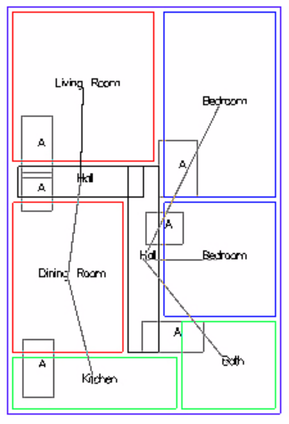
Results show that another constraint has been forgotten. The two main Hallway Units should be connected. Another constraint is added to force a sufficient intersection between the two Hallways (Section 3. 2. 5). The new formulation is re-optimized. Results now match the designer’s intentions. It is much easier for designers to be sure that they have included design intentions in a model if they see results. This tool allows the designer to quickly see results and adjust the problem definition if necessary. The layout is saved as a design alternative, and the designer continues to explore other alternatives.



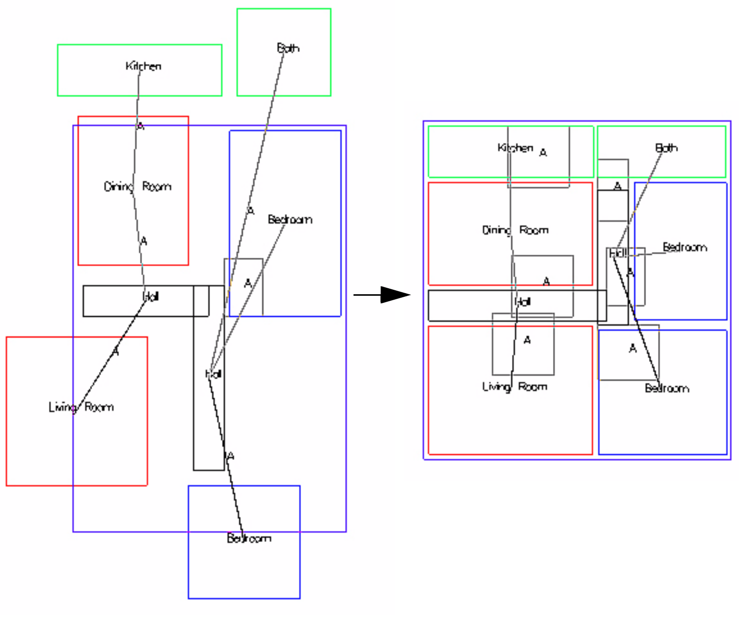
The designer considers moving the bath to the east side of the apartment to separate the public space from the private space better, and also to allow more room for the private rooms while shrinking the bedrooms. Another design alternative is produced. The ability to make design changes and quickly examine results, consequences, and trade-offs is very useful. The effects of design changes can be seen quickly - even faster than sketching with pencil and paper.



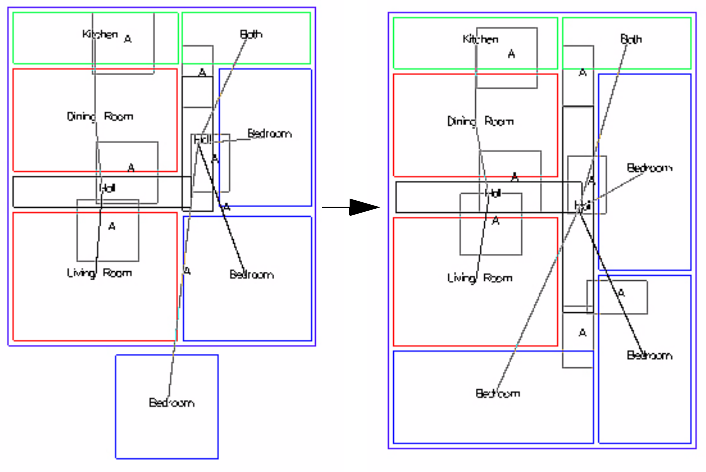
The designer notices that the doorway (accessway) to the north bedroom could be moved south, and the living room could take up the space currently occupied by a hallway. The living room is moved north of the hallway and enlarged so that it uses the space. The apartment is re-optimized.



The designer examines the results and decides that the default minimum area for the living room (120 sq. ft.) was too small. The minimum area is increased to 150 sq. ft. and the design is re-optimized. This ability to change constraint parameters allows designers to perform parametric studies intuitively and examine trade-offs. Many constraints, such as minimum allowable room area constraints, are flexible because these constraints can be relaxed if doing so provides a significant gain somewhere else. A designer using this tool can often see visually where relaxing constraints can improve the design, and it is quick and easy to explore constraint relaxation. In this case, the constraint is increased to provide more space. The algorithm enlarges the living room to meet the new constraints. A new design alternative has been created. The designer saves this design and moves on to explore.

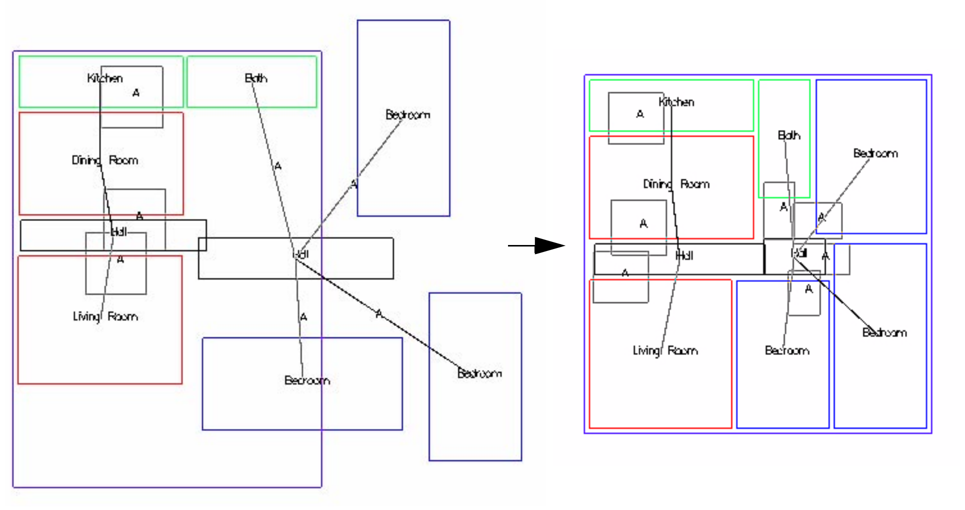


The designer wants to reduce the size of the apartment. The last design measured 27 X 40 (1080 sq. ft.). A new design arrangement is considered by moving the kitchen and bath to the north end, and moving the living room to the south. The design is re-optimized. The resulting design measures 30 X 33 (990 sq. ft.) - a reduction of 90 sq. ft. The design alternative is saved.



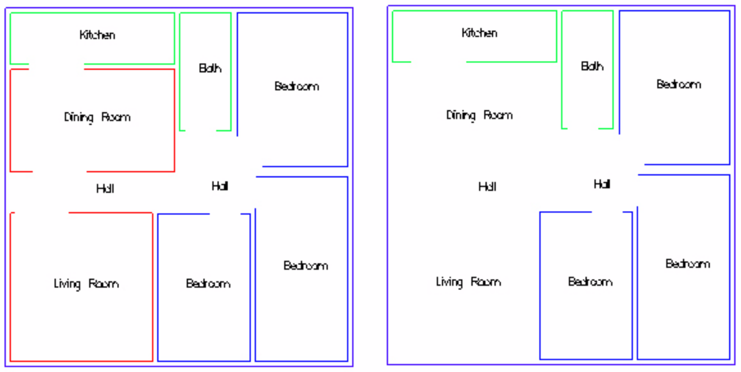
The designer decides to consider adding a third bedroom. The designer places the third bedroom and adds an Accessway. The new design is optimized. The new three-bedroom layout can now be compared to the two bedroom layout. Adding an extra bedroom increases the apartment size from 30 X 33 (990 sq. ft.) to 30 X 42.5 (1275 sq. ft.), and the heating cost will rise an estimated 15%. Providing the designer with computational feedback as well as visual feedback can aid in decision making in the early conceptual stages of design. The new three-bedroom layout is saved.

After examining the results, the designer decides that the private area should be more distinctly separate from the public area. The three bedrooms are moved to the west side of the apartment, and the new design is optimized.

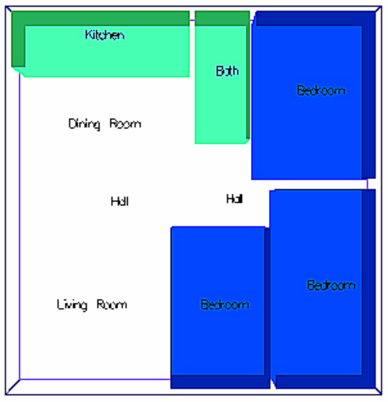


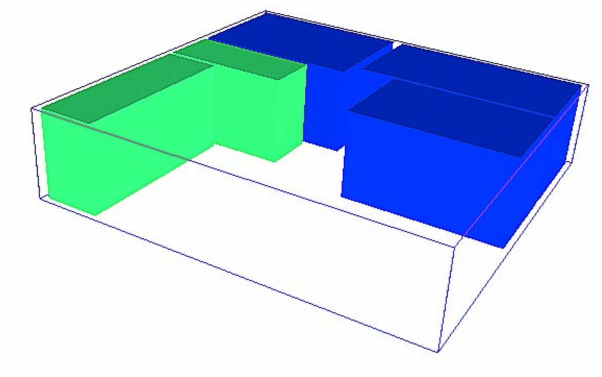
The result is accepted and saved as an alternative three bedroom arrangement. The design can be viewed in several different ways to get a better feeling for the layout.

Accessways can be viewed as openings or doorways, and the living room and dining room can be viewed without physical walls.

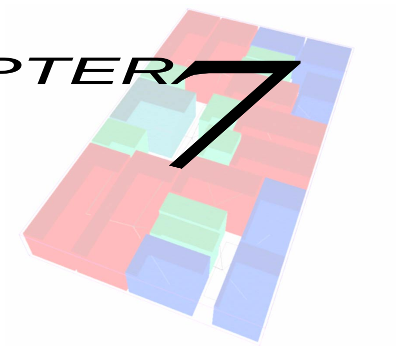


The layout can be viewed in three dimensions to give a different feel for the space allocation. Three dimensional views can be used to help visualize the look and feel of the interior space.





This example has shown how the interactive optimization design tool can be used to quickly generate and compare design alternatives visually and computationally. The designer uses the optimization tool during conceptualization to help refine the problem goals, understand design trade-offs, and explore design options. The entire process of generating, visualizing, examining computational feedback, and using that information to explore new designs in the above example was completed in about ten minutes. This tool offers speed for design concept sketching with the power of computational feedback and optimization results. The tool helps the designer to understand design trade-offs and to define design objectives and constraints more formally as s/he views real results.



FUTURE DIRECTION

This chapter discusses possible future directions and ideas for improvement of the automated tool and the interactive tool.

7. 1 Automated Design Optimization Improvements

7. 1. 1 Improve Design Toolbox

New constraints and objectives can be added to the representation to improve optimization behavior, better represent architectural criteria, and improve the quality of resultant layouts. Constraint sets can be added to allow designers to more easily deal with complex shaped rooms or building boundaries made of multiple Units. Current objective and constraint functions can be improved for better accuracy. Also, additional objectives and constraints can be added to model building codes, structural supports, routing of wiring, piping and ducts, human traffic patterns, or other aspects of architectural design that can be quantified.

7. 1. 2 Explore Shape Grammars

Shape grammars offer an alternative possible representation for exploring topology and geometry simultaneously. Shape grammars have a strong presence in the field of architecture, and the integration of a shape grammar with an optimization search algorithm could yield interesting results.

7. 1. 3 Material Selection

In the current implementation, materials are assumed. New variables can be added to the topology representation to represent material selection decisions for windows and walls. This is especially important in examining the trade-off between build cost and performance cost.

7. 1. 4 Variable Number of Hallways

The number of hallways in a layout is somewhat arbitrary in this representation because complex hallways structures are made up of some number of rectangular hallway units. In the topology layout, a fixed number of hallways is assumed, and hallways are removed from the geometric layout if they are not being used (less than 2 connections). An improved way to deal with a variable number of hallways could be developed to improve layout solutions.

7. 1. 5 Diversity

Solution alternatives generated in a single run of the algorithm tend to be very similar (although they differ greatly between runs). A diversity objective could be incorporated to increase diversity of the resultant design population.

7. 1. 6 Multiple Floors

An ability to add extra floors can be added with spaces shared between the floors (such as stairs or high ceilings).

7. 1. 7 Complex Shapes

A more generalized Unit component that can represent non-rectangular and non-orthogonal shapes would be necessary to generalize this tool to handle a larger class of problems. It is important, however, that the representation remain simple for two reasons: (1) The problem must be fairly simple in order to get fast, reliable results from the optimization algorithm, which is necessary for the interactive tool. (2) If the representation of the geometry is too complex, it will be difficult to interact with, and the interactive tool will loose some of its use as a sketching tool. There is a compromise between speed and the level of simplicity and approximation.

7. 1. 8 Alternative Global Search Methods

Alternative global search methods can be explored, such as CSP algorithms, that search the design space more rigorously and guarantee completeness. Alternate strategies may be more successful at generating layouts for large problems. The topology strategy presented in this paper is heuristic, and it is not guaranteed to find a solution in finite time. Also, the evolutionary algorithms used in this thesis can be explored further to compare different parameter values or new mutation and crossover functions.

7. 1. 9 Topology Definition

The topology can be defined in a new way so that topology decisions create different kinds of constraints in the geometry optimization problem. This could be especially powerful if the topology decisions are designed to map to **linear** constraints in the geometry optimization problem. Linear constraints can be handled efficiently by gradient algorithms. It may be possible to construct a topology model that produces only linear constraints for the geometric optimization algorithm. One idea would be to define topology similar to definitions by [7] and [8] (see Chapter 2). Topology decision variables could make decisions such as “Room1 *is-west-of* Room2”. This would produce a linear constraint in the geometry model instead of the nonlinear Prohibit Intersection Constraint that is usually used (see Section 3. 2. 4). Using this kind of topology, it is possible that the rough position topology variables could be completely eliminated. If all constraints were linear, an initial design where all Units are positioned at (0,0) would be manageable by the geometry algorithm. In its current form, the geometry algorithm cannot handle such a starting point because intersection constraints are nonlinear and non-smooth if two rooms share the same center position. The rough position variables are used to generate initial geometry that the geometry algorithm can manage, and it uses the rough position variables to search different configurations. If decisions such as “Room1 *is-west-of* Room2” were used instead to search different configurations, the mapping from topology to local optimal geometry might be both injective and surjective (meaning that each valid topology will create a different local optimal geometry, and that all possible local optimal geometry could be created by a specific topology). This would be a significant improvement over the current representation which is not injective or surjective (see Section 4. 2. 1).

7. 2 Interactive Design Exploration Improvements

7. 2. 1 Develop Interface

The interface can be improved to commercial application quality to improve intuitive interaction and increase speed and ease of use.

7. 2. 2 Interface with Design Constraints

Interface with design constraints is text based in the current representation. A system for creating, displaying, and interacting with constraints graphically can be developed to give the designer more of a physical feel for the design constraints and improve the process of interactively refining the problem definition.

7. 2. 3 Trust Regions

Trust regions (bounding boxes) can be used to limit the size of optimization moves during each iteration. This may be useful in interactive optimization if the designer can control the size of the bounding box, forcing the design moves to slow down when s/he wants more interaction, and allowing optimization to speed up when s/he wants more computational efficiency.

7. 2. 4 Generalization

The interactive tool can be generalized to work with exploration of other classes of product design.

CHAPTER 8

CONCLUSIONS

This chapter summarizes and discusses the automated design tool and the interactive design tool.

8. 1 Automated Building Layout Design

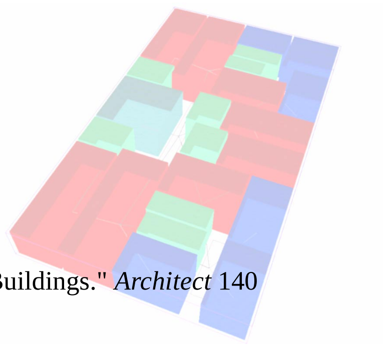
Two automated optimization algorithms have been used to automate the generation of design layouts: the geometry and topology algorithms. The geometric algorithm, built on rigorous gradient-based algorithms, is efficient and robust, and it has been successful at optimizing geometry for large problems. In its present state, it is most useful as an aid for design exploration, rather than design automation, because results are highly dependent on the starting point defined by the designer. Several tools have been implemented for searching the geometric space more globally, including a hybrid SA/SQP that uses SA to choose starting points for SQP local optimization and a strategic program that uses a set of design moves to explore the space and find feasible layouts. These tools have been successful at automatically finding alternative arrangements for rooms and exploring many local minima.

A second topology optimization algorithm was built on top of the geometric algorithm to search feasible topology alternatives and find the feasible topology that generates the best geometry. The topology problem defines room connectivity and rough placement, and it is highly nonlinear and discrete in nature. Only global methods, such as enumeration algorithms and heuristic algorithms, are useful for exploration of this kind of multi-modal design space. Heuristic evolutionary algorithms were selected for this thesis because of the combinatorial explosion that threatens enumeration algorithms. The results are interesting, but limited. Heuristic algorithms also suffer from combinatorial explosion, and solving a problem with around 20 rooms takes nearly a day of computation. Furthermore, because the algorithms are heuristic, they cannot guarantee convergence in finite time, and a designer cannot know how long he will have to wait to find a solution. One difficulty evaluating with topology combinatorial search algorithms for building layout is that results have not been reported for buildings with more than 20 rooms. Automation of small problem solutions is not as interesting from a design point of view because small problems often can be explored in less time using intuition and sketching -- especially when the subjective nature of aesthetic elements is considered. Automation of topology solutions would be much more useful for large scale, highly constrained problems. One example would be to layout a public space, such as an airport, and include additional considerations such as queuing, routing, and scheduling in the model. For small problems that are highly dependent on designer judgement as well as computational factors, an interactive approach that takes advantage of designer knowledge and intuition is recommended. One advantage to the approach presented here is that the final design generated by the algorithm can be used as a starting point for interactive design exploration.

8. 2 Interactive Layout Optimization

The interactive design optimization tool shows significant potential for computational optimization algorithms to be used in the early conceptual stages of the design process to help designers explore solutions and trade-offs. Design conceptualization is an extremely important part of the design process, and it is one that computer tools typically ignore because of the poorly understood nature of creativity and subjective judgement. The interactive tool assists the designer with design generation and evaluation, rather than attempting to automate these processes completely. This approach allows the designer to maintain control, to use quantitative and subjective judgements where appropriate, and so it supports creative exploration. Such conceptual support tools may finally begin to fill an important gap in the present array of computer design support capabilities.

REFERENCES



Automated Building Layout

1. Levin, P. H. "Use of Graphs to Decide the Optimum Layout of Buildings." *Architect* 140 (1964): 809-15.

2. Liggett, Robin S., and William J. Mitchell. "Optimal Space Planning in Practice." *Computer-Aided Design* 13, no. 5 (1981): 277-88.

3. Sharpe, R., B. S. Marksjo, J. R. Mitchell, and J. R. Crawford. "An Interactive Model for the Layout of Buildings." *Applied Mathematical Modeling* 9 (1985): 207-14.

4. Jo, Jun H., and John S. Gero. "Space Layout Planning Using an Evolutionary Approach." *Artificial Intelligence in Engineering*, no. 12 (1998): 149-62.

5. Jagielski, Romuald, and John S. Gero. "A Genetic Programming Approach to the Space Layout Planning Problem." *CAAD Futures* (1997): 875-84.

6. Baykan, Can A., and Fox Mark S. "Spatial Synthesis by Disjunctive Constraint Satisfaction." *Artificial Intelligence in Engineering Design*, no. 11 (1997): 245-62.

7. Schwarz, A., D. M. Berry, and E. Shaviv. "Representing and Solving the Automated Building Design Problem." *Computer-Aided Design* 26, no. 9 (1994): 689-98.

8. Medjdoub, B., and B. Yannou. "Separating Topology and Geometry in Space Planning." *Computer-Aided Design* 32 (1999): 39-61.

9. Arvin, Scott A., and Donald H. House. "Modeling Architectural Design Objectives in Physically Based Space Planning." *ACADIA* (1999): 212-25.

10. Gero, J. S., and V. Kazakov. "Learning and Reusing Information in Space Layout Problems Using Genetic Engineering." *Artificial Intelligence in Engineering* 11, no. 3 (1997): 329-34.

11. Rosenman, Mike. "Case-Based Evolutionary Design." *Ai Edam-Artificial Intelligence for Engineering Design Analysis and Manufacturing* 14, no. 1 (2000): 17-29.

12. Rosenman, M. A., and J. S. Gero. "Evolving Designs by Generating Useful Complex Gene Structures." *Evolutionary Design by Computers* (1999): 345-64.

13. Park, Kwang-Wook, and Donald E. Grierson. "Pareto-Optimal Conceptual Design of the Structural Layout of Buildings Using a Multicriteria Genetic Algorithm." *Computer-Aided Civil and Infrastructure Engineering* 14 (1999): 163-70.

14. Faucher, Didier, and Marie-Laure Nivet. "Playing With Design Intents: Integrating Physical and Urban Constraints in CAD." *Automation in Construction* 9 (2000): 93-105.

15. Peponis, J., and Craig Zimring. "Designing Friendly Hospital Layouts. The Contributions of Space-Syntax." *Journal of Healthcare Design* 8 (1996): 109-16.

Other Automated Layout

16. Yin, Su, and Jonathan Cagan. "An Extended Pattern Search Algorithm for Three-Dimensional Component Layout." *Transactions of the ASME* 122 (2000): 102-8.

17. Cagan, Jonathan, Drew Degentesh, and Su Yin. "A Simulated Annealing-Based Algorithm Using Hierarchical Models for General Three-Dimensional Component Layout." *Computer-Aided Design* 30, no. 10 (1998): 781-90.

18. Szykman, S., and J. Cagan. "Constrained Three-Dimensional Component Layout Using Simulated Annealing." *ASME Transactions* 119 (1997): 28-35.

19. Choo, Hyun Jeong, and Iris D. Tommelein. "Space Scheduling Using Flow Analysis." *Proc. Seventh Annual Conference of the International Group for Lean Construction*, July 1999

20. Rong, Bai. "Automated Generation of Fixture Configuration Design." *Journal of Manufacturing Science and Engineering* 119 (1997): 208-19.

21. Koide, Tetsushi, and Shin'ichi Wakabayashi. "A Timing-Driven Floorplanning Algorithm With the Elmore Delay Model for Building Block Layout." *Integration, the VLSI Journal*, no. 27 (1999): 57-76.

22. Wang, Yinglin, and Huizhong Wu. "Method of Constraint Graphs Used in Spatial Layout." *Ruan Jian Xue Bao Journal of Software* 9, no. 3 (1998): 200-205.

23. Ito, Teruaki. "A Genetic Algorithm Approach to Piping Route Path Planning." *Journal of Intelligent Manufacturing*, no. 10 (1999): 103-14.

24. Davidson, R., and D. Harel. "Drawing Graphs Nicely Using Simulated Annealing." *ACM Transactions on Graphics* 15, no. 4 (1996): 301-31.

25. Fadel, Georges M., Avijit Sinha, Todd McKee. “Packing Optimisation Using a Rubberband Anology” *Proceedings of DETC’01 2001 ASME Design Engineering Technical Conferences* DETC2001/DAC-21051 September 2001.

26. Kim, J.J., and D.C. Gossard “Reasoning on the Location of Components for Assembly Packaging” ASME Journal of Mechanical Design, Vol. 113, No. 4 1991 pp 402-407.

27. Chapman, C.D., K. Saitou, and M.J. Jakiela. “Genetic Algorithms as an Approach to Configuration and Topology Design” Journal of Mechanical Design, Transactions of the ASME, v 116 n 4 Dec 1994, pp 1005-1012.

Optimization References

28. Papalambros, Panos Y., and Douglass J. Wilde. Principles of Optimal Design - Modeling and Computation - Second Edition. Cambridge, England: *Cambridge University Press*, 2000.

29. Kott, G. J., and G. Gabriele. "A Tunnel Based Method for Mixed Discrete Constrained Nonlinear Optimization." *ASME Design Automation Conference* 1998.

30. Kott, G. J. "A Method for Mixed Variable Constrained Nonlinear Optimization Based on a New Function Bounding Technique." *PhD Thesis*, Rensselaer Polytechnic Institute, 1998.

31. Zhou, J. L., and A. L. Tits. "An SQP Algorithm for Finely Discretized Continuous Minimax Problems and Other Minimax Problems With Many Objective Functions." *SIAM Journal of Optimization* (1995).

32. Lawrence, C. T., J. L. Zhou, and A. Tits. “User’s Guide for CFSQP Version 2.3: A C Code for Solving (Large Scale) Constrained Nonlinear (Minimax) Optimization Problem, Generating Iterates Satisfying All Inequality Constraints” Institute for Systems Research, University of Maryland, Technical Report TR-94-16r1, 1995.

33. Zhou, J. L., and A. Tits. "Nonmonotone Line Search for Minimax Problems." *Journal of Optimization Theory and Applications* 76, no. 3 (1993): 455-76.

34. Panier, E., and A. Tits. "On Combining Feasibility, Descent and Superlinear Convergence in Inequaltiy Constrained Optimization." *Mathematical Programming* 59 (1993): 261-76.

35. Mayne, D. Q., and E. Polak. "Feasible Directions Algorithms for Optimization Problems With Equality and Inequality Constraints." Mathematical Programming 11 (1976): 67-80.

36. Bonnans, J. F., E. Panier, A. Tits, and J. Zhou. "Avoiding the Maratos Effect by Means of a Nonmonotone Line Search: II. Inequality Problems - Feasible Iterates." SIAM Journal on Numerical Analysis 29, no. 4 (1992): 1187-202.

37. Bartak, Roman. "Constraint Programming: In Pursuit of the Holy Grail." http://kti.ms.mff.cuni.cz/~bartak/constraints 1999.

38. Koziel, Slawomir, and Zbigniew Michalewicz. "Evolutioniary Algorithms, Homomorphous Mappings, and Constrained Parameter Optimization.".

39. Wall, Matthew. "Galib: A C++ Library of Genetic Algorithm Components - Version 2.4 - Document Revision B"."1996.

40. Koziel, S. and Michalewicz, Z., "Evolutionary Algorithms, Homomorphous Mappings, and Constrained Parameter Optimization" *Evolutionary Computation*, Vol.7, No.1, pp.19-44, 1999.

41. Koziel, S. and Michalewicz, Z., "A Decoder-based Evolutionary Algorithm for Constrained Parameter Optimization Problems, Proceedings of the 5th Parallel Problem Solving from Nature" Springer-Verlag, Lecture Notes in Lomputer Science, Vol.1498, Amsterdam, September 27--30, 1998, pp.231--240.

42. Bentley, Peter J. Evolutionary Design by Computers. San Francisco, CA: Morgan Kaufmann Publishers, Inc., 1999.

43. Guuvaerts, P. Geostatistics for Natural Resources Evaluation. Oxford University Press, New York, NY 1997

Interactive Optimization

44. Schwarz, A., D. M. Berry, and E. Shaviv. "On the Use of the Automated Building Design System." *Computer-Aided Design* 26, no. 10 (1994): 747-61.

45. Liggett, Robin S., and William J. Mitchell. "Interactive Graphic Floor Plan Layout Method." *Computer-Aided Design* 13, no. 5 (1981): 289-98.

46. Kharrufa, S., A. Saffo, H. Aldabbagh, and W. Mahmood. "Developing CAD Techniques for Preliminary Architectural Design." *Computer-Aided Design* 20, no. 10 (1988): 581-88.

47. Tappeta, Ravindra, and John E. Renaud. "Interactive Multiobjective Optimization Design Strategy for Decision Based Design."*Proceedings of the 1999 ASME International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*.

48. Diaz, Alejandro. "Interactive Solution to Multiobjective Optimization Problems." *International Journal for Numerical Meghods in Engineering* 24 (1987): 1865-77.

49. Miettinen, Kaisa, and Marko M. Makela. "Interactive Multiobjective Optimization System WWW-NIMBUS on the Internet." *Computers & Operations Research*, no. 27

(2000): 709-23.

50. Palli, N., S. Azarm, P. McCluskey, and R. Sundararajan. "An Interactive Multistage Epsilon-Inequality Constraint Method for Multiple Objectives Decision Making." *Journal of Mechanical Design* 120 (1998): 678-86.

51. Wieghardt, K., D. Hartmann, and K. R. Leimbach. "Interactive Shape Optimization of Continuum Structures." *Engineering Structures* 19, no. 4 (1997): 325-31.

52. Balling, Rick, Alan Parkinson, and Joseph Free. "Interactive Optimization in Engineering Design."Computing in Civil Engineering, *Proceedings of the 3rd Conference* 1984.

53. Parkinson, A. R., R. J Balling, and J. C. Free. “Optdes. BYU: A Software System for Optimal Engineering Design” Computers in Engineering 1984, Advanced Automation: 1984 and Beyond, *Proceedings of the 1984 International Computers in Engineering Conference and Exhibit*.

Exploratory Design Tools

54. Arvin, Scott A., and Donald H. House. "Modeling Architectural Design Objectives in Physically Based Space Planning." *ACADIA* (1999): 212-25.

55. Tidd, William F., James R. Rinderle, and A. Witkin. "Design Refinement Via Interactive Manipulations of Design Parameters and Behaviors." *Design Theory and Methodology - DTM '92* American Society of Mechanical Engineers New York, NY, USA: ASME Design Engineering Division, 1992.

56. Parmee, C., and C. R. Bonham. "Towards the Support of Innovative Conceptual Design Through Interactive Designer/Evolutionary Computing Strategies." *Ai Edam-Artificial Intelligence for Engineering Design Analysis and Manufacturing*, no. 14 (2000): 3-16.

57. Stuyver, Ralph, and Jim Hennessey. "A Support Tool for the Conceptual Phase of Design." http://www.io.tudelft.nl/research/IDEATE/papers/stuy_hci/stuy_hci.htm 1997.

58. Hennessey, Jim. "The IDEATE Project: Exploring Computer Enhancements for Conceptualizing." http://www.io.tudelft.nl/research/IDEATE/papers/henn_els/henn_els.htm 1997.

59. Duffy, A. H. B., A. Persidis, and K. J. MacCallum. "NODES: A Numerical and Object Based Modelling System for Conceptual Engineering Design." *Knowledge-Based Systems* 9 (1996): 183-206.

60. Scriabin, M., Vergin, R “Comparison of Computer Algorithms and Visual Based Methods for Plant Layout” *Management Science* v 22 n 2 Oct 1975 p 172-181

61. Simon, H.A, “The Structure of Ill-structured Problem” *Artificial Intelligence* 4, 1973 p 181-201

Architectural Design References

62. ASHRAE “Fundamentals, American Society of Refrigeration, Heating and Air-conditioning Engineers, Atlanta, GA 1997

63. IESNA “IESNA Lighting Education: Intermediate Level, Illumination Engineers Society of North America. 1998

Appendix A: Mathematical Nomenclature

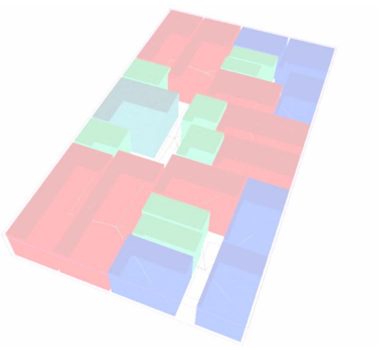


Table 4: Mathematical Nomenclature

Variable	Type	Description
n	Parameter	Number of rooms
$\boldsymbol{x}$	Vector	Vector of design variables for the geometry optimization problem
$f_{\boldsymbol{g}}(\boldsymbol{x})$	Function	Objective function of the geometric design variables
$\boldsymbol{w_j}$	Parameter	Weight of objective function j in a multi-objective formulation
$\boldsymbol{g}(\boldsymbol{x})$	Function	Vector of inequality constraint functions of the geometry design variables
$\boldsymbol{h}(\boldsymbol{x})$	Function	Vector of equality constraint functions of the geometry design variables
$\boldsymbol{x}^*$	Vector	Local optimal geometric design solution
$\boldsymbol{x}^\dagger$	Vector	Local optimal geometric design solution to the MDDM formulation
$x_{\boldsymbol{i}}$	Variable	The reference point position of Unit i in the Cartesian x-direction
$y_{\boldsymbol{i}}$	Variable	The reference point position of Unit i in the Cartesian y-direction
$N_{\boldsymbol{i}}$	Variable	Perpendicular from Unit i reference point to north wall of Unit i
$S_{\boldsymbol{i}}$	Variable	Perpendicular from Unit i reference point to south wall of Unit i
$E_{\boldsymbol{i}}$	Variable	Perpendicular from Unit i reference point to east wall of Unit i
$W_{\boldsymbol{i}}$	Variable	Perpendicular from Unit i reference point to west wall of Unit i
ω_{N_i}	Variable	Length of north window of Unit i
ω_{S_i}	Variable	Length of south window of Unit i
ω_{E_i}	Variable	Length of east window of Unit i
ω_{W_i}	Variable	Length of west window of Unit i
ϑ_{door_i}	Variable	Discrete variable that determines which side of Unit i a door is on (alt. form.)
ϑ_{ext_i}	Variable	Discrete variable that determines which external wall Unit i is against (alt. form.)
$\boldsymbol{y}_{\boldsymbol{N}_i}$	Result	Cartesian y-coordinate of the north wall of Unit i
$\boldsymbol{y}_{\boldsymbol{S}_i}$	Result	Cartesian y-coordinate of the south wall of Unit i
$\boldsymbol{x}_{\boldsymbol{E}_i}$	Result	Cartesian x-coordinate of the east wall of Unit i
$\boldsymbol{x}_{\boldsymbol{W}_i}$	Result	Cartesian x-coordinate of the west wall of Unit i

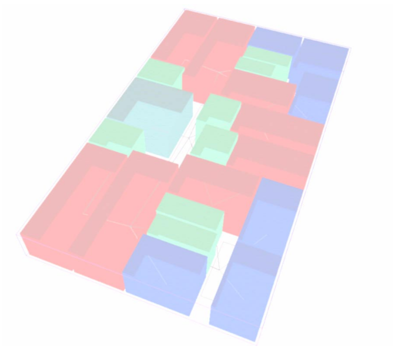
Table 4: Mathematical Nomenclature

Variable	Type	Description
l_i	Result	Length of Unit i (x-direction)
w_i	Result	Width of Unit i (y-direction)
A_i	Result	Area of Unit i
h_i	Parameter	Height of Unit i
A_N	Result	Area of the north wall of the building
A_S	Result	Area of the south wall of the building
A_E	Result	Area of the east wall of the building
A_W	Result	Area of the west wall of the building
A_{ω_N}	Result	Total area of window glass on the north wall of the building
A_{ω_S}	Result	Total area of window glass on the south wall of the building
A_{ω_E}	Result	Total area of window glass on the east wall of the building
A_{ω_W}	Result	Total area of window glass on the west wall of the building
d_i	Parameter	Minimum width of doorways or openings in Unit i
A_{min}	Parameter	Minimum allowable area of Unit i
l_{min}	Parameter	Minimum allowable length/width of Unit i
l_{max}	Parameter	Maximum allowable length/width of Unit i
R_{min_i}	Parameter	Minimum allowable length-to-width and width-to-length ratio for Unit i
ΔT_i	Parameter	Average internal/external temperature difference during month i
U_{wall}	Parameter	U-value of wall material (quality of material)
U_{ω}	Parameter	U-value of window material (quality of material)
β_{shgf}	Parameter	Solar heat gain factor (function of geographical location)
β_{sc}	Parameter	Shading Coefficient (property of glass)
β_{tlf}	Parameter	Time lag factor (property of glass and orientation)
η_{heater}	Parameter	Estimated efficiency of gas heater
η_{ac}	Parameter	Estimated efficiency of air conditioning unit
Q_{heat}	Result	Estimated annual heat loss through building walls during heated months
Q_{solar}	Result	Estimated solar heat gain per year during air conditioned months
Q_{cond}	Result	Est. conductive heat gain through the building exterior during air cond. months
Γ_{heat}	Result	Estimated annual cost to heat the building
Γ_{cool}	Result	Estimated annual cost to cool the building

Table 4: Mathematical Nomenclature

Variable	Type	Description
$\Gamma_{\boldsymbol{budget}}$	Parameter	Maximum allowable material cost to construct the building
$\kappa_{\boldsymbol{wall}}$	Parameter	Estimated cost per unit area of wall
κ_{ω}	Parameter	Estimated cost per unit area of window glass
$\kappa_{\boldsymbol{elec}}$	Parameter	Estimated cost of electricity
$\kappa_{\boldsymbol{gas}}$	Parameter	Average cost of gas per cubic foot
$\Theta_{\boldsymbol{i}}$	Result	Average illuminance from natural lighting in Unit i
$\theta_{\boldsymbol{req_i}}$	Parameter	Minimum required average illuminance per square foot in Unit i
$\varphi_{\boldsymbol{min_i}}$	Parameter	Minimum allowable percentage of required lighting from natural source for Unit i
$\boldsymbol{X}$	Vector	Vector of design variables for the topology optimization problem
$f_{\boldsymbol{t}}(\boldsymbol{X})$	Function	Objective function of the topology design variables
$\phi_{\boldsymbol{ij}}$	Variable	Connectivity between Unit i and Unit j
Penalty($\boldsymbol{X}$)	Function	Function to penalize a topology, $\boldsymbol{X}$, for being infeasible
B	Parameter	Bonus given to feasible topologies in the topology optimization
$\boldsymbol{P_k}$	Vector	The k^{th} population of designs in an evolutionary algorithm progression

Appendix B: Topology Path Constraint Calculations

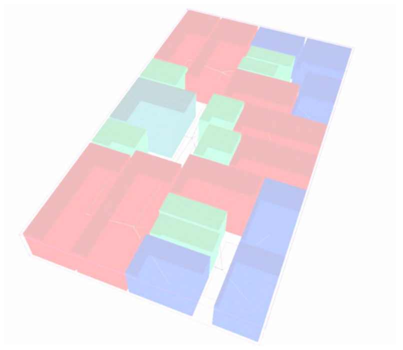


This is C++ code that checks if a connected path exists from a start Unit to a goal Unit while passing through only legal units (as defined by the designer in the problem definition). The first checkForPath function recursively calls the second function checking for longer paths each call.

```
int BuildingGenome::checkForPath(int start, int goal, vector<int> legalUnits) {
     // Check for a path of length i through any legalUnits between start and goal
     // for each possible value of i.  Returns 1 if any legal path is found
     if (getConnectivity(start, goal) == 1) return 1;
     for (int i=1; i < legalUnits.size(); i++) {
               if (checkForPath(start, goal, legalUnits, i)) {
                         return 1;
               }
     }

     // No path has been found:
     return 0;
}

int BuildingGenome::checkForPath(int start, int goal, vector<int> legalUnits, int pathLength) {
     // This is a recursive formula used to find a connectivity path between
     // start and goal of length "pathLength".  It returns 1 if there is a path
     // through legalUnits from start to goal of length pathLength.  0 otherwise.

     // Path lengths must be >= 1
     if (pathLength < 1) return 0;

     // A pathLength of 1 means a direct connection (connectivity(start, goal) = 1)
     if (pathLength == 1) return getConnectivity(start, goal);

     // To check for a path of length pathLength > 1, check for a direct
     // connection between the start and some room i, and a path
     // of length (pathLength-1) between room i and the goal.
```

```
    // Try this for all rooms i
    for (int i=0; i < legalUnits.size(); i++) {
            if (i != start && i != goal) {
                    if (getConnectivity(start, i) == 1) {
                            // Construct remaining units to avoid loops
                            vector<int> remainingUnits;
                            for (int j=0; j < legalUnits.size(); j++) {
                            if (legalUnits[j] != i) remainingUnits.push_back(legalUnits[j]);
                            }
                            if (checkForPath(i, goal, remainingUnits, pathLength - 1)) {
                            return 1;
                            }
                    }
            }
    }

    // No path has been found:
    return 0;
}
```

Appendix C: Topology Planarity Constraints



This is a C++ function to check if a given topology is ‘planar’. In the check, connections between units are represented as lines. This function checks to see if any of the lines intersect. There are special rules about intersecting at endpoints included in the comments. The main goal is to avoid passing designs to the geometric optimizer only if there is no possible geometry that can satisfy the input.

```
int checkLineIntersection(double Ax1, double Ay1, double Ax2, double Ay2,
                          double Bx1, double By1, double Bx2, double By2) {
        double Adx = (Ax2-Ax1);
        double Ady = (Ay2-Ay1);
        double Bdx = (Bx2-Bx1);
        double Bdy = (By2-By1);

        double Amaxx = (Ax2>Ax1 ? Ax2 : Ax1);
        double Amaxy = (Ay2>Ay1 ? Ay2 : Ay1);
        double Aminx = (Ax2<Ax1 ? Ax2 : Ax1);
        double Aminy = (Ay2<Ay1 ? Ay2 : Ay1);

        double Bmaxx = (Bx2>Bx1 ? Bx2 : Bx1);
        double Bmaxy = (By2>By1 ? By2 : By1);
        double Bminx = (Bx2<Bx1 ? Bx2 : Bx1);
        double Bminy = (By2<By1 ? By2 : By1);

        // First reject far away line segments using bounding box
        if (Aminx > Bmaxx) return 0;
        if (Aminy > Bmaxy) return 0;
        if (Bminx > Amaxx) return 0;
        if (Bminy > Amaxy) return 0;

        // First deal with points
        // ---------------------
```

```
if (Adx == 0 && Ady == 0) {
          // A is a point
          if (Bdx == 0 && Bdy == 0) {
                    // B is a point
                    if (Ax1 == Bx1 && Ay1 == By1) {
                              // The two points are the same
                              return 1;
                    }
                    else // The two points are different
                              return 0;
          }
          else {
                    // B is a line
                    if (Bdx == 0) {
                              // B is a vertical line
                              if (Ax1 == Bx1 && Ay1 >= Bminy && Ay1 <= Bmaxy) {
                              // Point A is on vertical line B
                              return 1;
                              }
                              else // Point A is not on vertical line B
                              return 0;
                    }
                    else if (Ay1 - By1 == (Bdy/Bdx)*(Ax1 - Bx1)) {
                              // Point A is on line B
                              return 1;
                    }
                    else // Point A is not on line B
                              return 0;
          }
}
if (Bdx == 0 && Bdy == 0) {
          // B is a point, A is a line
          if (Adx == 0) {
                    // A is a vertical line
                    if (Bx1 == Ax1 && By1 >= Aminy && By1 <= Amaxy) {
                              // Point B is on vertical line A
                              return 1;
                    }
                    else // Point B is not on vertical line A
                              return 0;
          }
          if (By1 - Ay1 == (Ady/Adx)*(Bx1 - Ax1)) {
                    // Point B is on line A
                    return 1;
          }
          else // Point B is not on line A
                    return 0;
}

// Next deal with the infinite slope case:
// ---------------------------------------
if (Adx == 0) {
          // Line A is vertical
          if (Bdx == 0) {
                    // Line B is also vertical
                    if (Ax1 == Bx1) {
                              // Both lines have same x: colinear
                              if (Amaxy > Bminy && Aminy < Bmaxy) {
                              // Line segments overlap
                              return 1;
                              }
                              else // Line segments are separated in y-dir
                              return 0;
                    }
                    else // Vertical lines separated in x-dir
                              return 0;
          }
          else {
                    // Line B is not vertical:  Find intersection with x=Ax1=Ax2
                    double y = By1 + (Ax1 - Bx1)*(Bdy/Bdx);
                    if (y >= Aminy && y <= Amaxy &&
                              ((y > Bminy && y < Bmaxy) || (y==Bminy && y==Bmaxy)) &&
                              Ax1 > Bminx && Ax1 < Bmaxx) {
                              // Line B crosses line A within segment A
                              // and inside of segment B
                              return 1;
                    }
                    else if (y > Aminy && y < Amaxy &&
                              y >= Bminy && y <= Bmaxy &&
                              Ax1 >= Bminx && Ax1 <= Bmaxx) {
                              // Line B crosses line A within segment B
                              // and inside of segment A
```

```
                        return 1;
                }
                else return 0;
        }
}
else if (Bdx == 0) {
        // Line B is vertical, Line A is not.
        // Find intersection of line A with x=Bx1=Bx2
        double y = Ay1 + (Bx1 - Ax1)*(Ady/Adx);
        if (y >= Bminy && y <= Bmaxy &&
                ((y > Aminy && y < Amaxy) || (y == Aminy && y==Amaxy)) &&
                Bx1 > Aminx && Bx1 < Amaxx) {
                // Line A crosses line B within segment B
                // and inside of segment A
                return 1;
        }
        else if(y > Bminy && y < Bmaxy &&
                        y >= Aminy && y <= Amaxy &&
                        Bx1 >= Aminx && Bx1 <= Amaxx) {
                // Line A crosses line B within segment B
                // and inside of segment A
                return 1;
        }
        else return 0;
}


// NOW DEAL WITH FINITE SLOPE CASES
// -------------------------------
// calculate y-intercepts:
double A_b = Ay1 - (Ady/Adx)*Ax1;
double B_b = By1 - (Bdy/Bdx)*Bx1;
if ((Ady/Adx) == (Bdy/Bdx)) {
        // Lines are parallel
        if (A_b == B_b) {
                // Lines are colinear
                if (Aminx < Bmaxx && Amaxx > Bminx) {
                        // Line segments overlap
                        return 1;
                }
                else // Line segments do not overlap
```

```
                                        return 0;
                        }
                        else // Lines are non-colinear
                                return 0;
                }
                else {
                        // Slopes are not equal, find intersection pt
                        double xx = (B_b - A_b)/((Ady/Adx)-(Bdy/Bdx));
                        double yy = (Ady/Adx)*xx + A_b;
                        if (xx >= Aminx && xx <= Amaxx &&
                                yy >= Aminy && yy <= Amaxy &&
                                xx > Bminx && xx < Bmaxx &&
                                ((yy > Bminy && yy < Bmaxy) || (yy==Bmaxy && yy==Bminy))) {
                                // Intersection point is within segment A
                                // and inside of segment B
                                return 1;
                        }
                        else if(xx > Aminx && xx < Amaxx &&
                                        ((yy > Aminy && yy < Amaxy) || (yy == Amaxy && yy==Aminy)) &&
                                        xx >= Bminx && xx <= Bmaxx &&
                                        yy >= Bminy && yy <= Bmaxy) {
                                // Intersection point is within segment B
                                // and inside of segment A
                                return 1;
                        }
                        else { // Line intersection is not inside of either segment
                                // NOTE: The two line segments may share endpoints
                                return 0;
                        }
                }
}
```