Eng. Opt., 2002, Vol. 34(5), pp. 461–484

Taylor & Francis
Taylor & Francis Group

ARCHITECTURAL LAYOUT DESIGN OPTIMIZATION

JEREMY J. MICHALEK[a,*], RUCHI CHOUDHARY[b] and PANOS Y. PAPALAMBROS[a]

[a]*Optimal Design Laboratory, Department of Mechanical Engineering, University of Michigan, Ann Arbor, Michigan 48109-2125, USA;* [b]*College of Architecture and Urban Planning, University of Michigan, Ann Arbor, Michigan 48109-2125, USA*

(Received 28 August 2001; In final form 26 February 2002)

This article presents an optimization model of the quantifiable aspects of architectural floorplan layout design, and a companion article presents a method for integrating mathematical optimization and subjective decision making during conceptual design. The model presented here offers a new approach to floorplan layout optimization that takes advantage of the efficiency of gradient-based algorithms, where appropriate, and uses evolutionary algorithms to make discrete decisions and do global search. Automated optimization results are comparable to other methods in this research area, and the new formulation makes it possible to integrate the power of human decision-making into the process.

Keywords: Optimization; Architectural design; Floorplan; Layout

1 INTRODUCTION

Spatial configuration 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, so it is an important area of inquiry. Research on automation of spatial configuration includes component packing [11–13], route path planning [18], process and facilities layout, VLSI design [16, 17], and architectural layout [3–10]. 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 [3]. Researchers have used several problem representations and solution search techniques to describe and solve the problem.

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 [4–7] (see Fig. 1). This problem is inherently discrete and multi-modal. Because of the

* Corresponding author. E-mail: michalek@umich.edu

ISSN 0305-215X print; ISSN 1029-0273 online
DOI: 10.1080/0305215021000033735

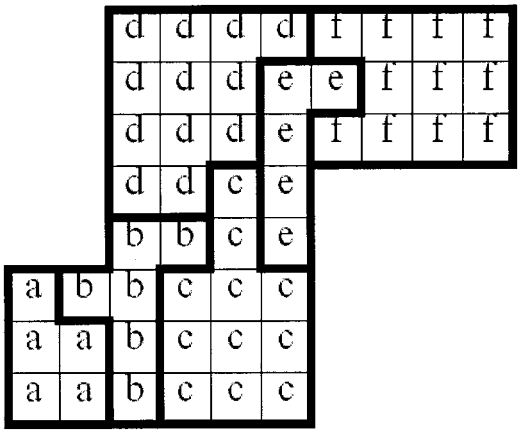

FIGURE 1 Sample fixed grid allocation layout.

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. Liggett and Mitchell [4] use a *constructive placement* strategy followed by an *iterative improvement* strategy. In this method, space is allocated for rooms one at a time based on the best probable design move at each step. Other researchers have used stochastic algorithms for search [5–7].

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*. Researchers have developed decision-tree-based combinatorial representations and used constraint satisfaction programming techniques to enumerate solutions without exhaustive search. Baykan and Fox [8] and Schwarz, Berry, and Saviv [9] developed variations of this model and have been able to enumerate solutions for a studio apartment and for a nine-room building respectively. Medjeoub and Yannou [10] developed a similar model, but they use a technique of first enumerating all topologies that can produce at least one feasible geometry. The designer is then able to review the feasible topological possibilities and select those which s/he wants to explore geometrically. This technique reduces computation dramatically, and they have shown success for up to twenty rooms.

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. It would be useful to take advantage of the speed of gradient-based algorithms on the geometric aspects of the layout, because they involve continuous variables.

This article develops a mathematical model for the geometric decisions in the layout problem that allows efficient solution with gradient-based and hybrid local–global methods. This model is then embedded into another model used for topology decisions that is solved with heuristic global methods. The geometric optimization process allows fast solution of large complex problems that also enables a true interactive design process described in a sequel article [2]. The topology optimization component has had limited success due to the combinatorial nature of the topology decisions. The interactive optimization tool can be downloaded from http://ode.engin.umich.edu.

2 OPTIMIZATION OF GEOMETRY

The geometric optimization problem is posed as a process of finding the best location and size of a group of interrelated rectangular units. A new decision model is formulated where all objectives and constraints are continuous functions, and all design variables have continuous domains.

A *Unit* is defined as a rectangular, orthogonal space allocated for a specific architectural function. Examples of architectural functions include living spaces, storage spaces, facilities, and accessibility spaces. For simplicity, this representation assumes that all *Units* can be represented as rectangles or combinations of orthogonal rectangles. This simple representation can model a large array of architectural layouts, and more complex shapes could be added to the model to expand this array. Figure 2 shows a *Unit* represented 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 than necessary to describe the shape; 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). Although this model increases the problem dimensionality, it offers a lot of flexibility to make the best design moves at each step of the optimization.

Units are grouped into several categories based on their function: *Rooms*, *Boundaries*, *Hallways*, and *Accessways*. *Rooms* are *Units* used for sustained living activity as determined by the designer. The differentiation between living space *vs.* non-living space is important only in optimization objectives that maximize the amount of space used for living relative to all other space. A *Boundary* is a *Unit* that has other *Units* constrained inside of it, and it is not considered living space. A *Hallway* is a *Unit* with no physical walls that is not a living space. *Hallways* function as pathways. An *Accessway* is a *Hallway* that is constrained to geometrically intersect two *Units*. *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 between the rooms.

In Figure 3, the external rectangle represents the building *Boundary*, the living room, bedroom, and bathroom are *Rooms*, the hall is a *Hallway*, and the three *Units* labeled "A" are *Accessways* that define space for a doorway between *Units*.

Units that are along external walls may also have windows for natural lighting. Window height can be fixed for each *Unit*, and window width is a variable. $\omega_N, \omega_S, \omega_E, \omega_W$ represent the width of the north, south, east and west windows, respectively.

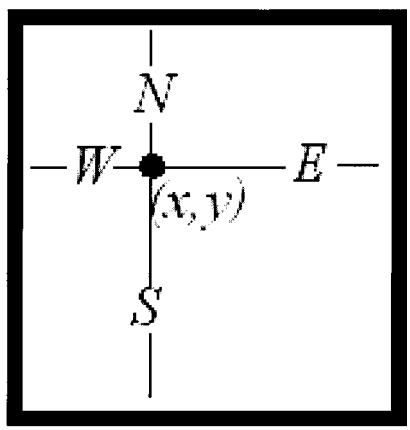

FIGURE 2 Representation of a Unit.

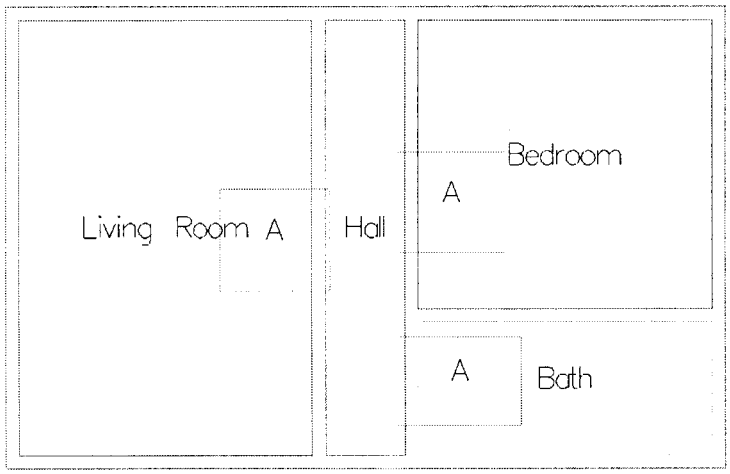

FIGURE 3 An example layout showing four different types of Units.

2.1 Mathematical Geometry Optimization Model

The design optimization problem is formulated as

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

where $\boldsymbol{x}$ is the vector of design variables, n is the number of variables, and $\boldsymbol{h}(\boldsymbol{x})$ and $\boldsymbol{g}(\boldsymbol{x})$ are vectors of equality and inequality constraints.

2.1.1 Design Variables

Variables for each Unit include a reference point location (x, y), distances to each wall (N, S, E, W), and the size of any windows added to each Unit:

$$\begin{gathered} \boldsymbol{x} = \bigcup_{i=1}^{n} \{x_i, y_i, N_i, S_i, E_i, W_i, \omega_{N_i}, \omega_{S_i}, \omega_{E_i}, \omega_{W_i}\} \\ x_i, y_i, N_i, S_i, E_i, W_i \in \Re; \quad \omega_{N_i}, \omega_{S_i}, \omega_{E_i}, \omega_{W_i} \in \Re_+ \end{gathered} \tag{2}$$

The window variables drop out when the window is not physically present for a specific Unit and direction. In order to simplify calculations and notation, several "intermediate" variables are used to describe geometry that results from the design variables. The following resultant variables are calculated from the design variables.

$$y_{N_i} = y_i + N_i \quad \text{Unit north wall location} \tag{3}$$

$$y_{S_i} = y_i - S_i \quad \text{Unit south wall location} \tag{4}$$

$$x_{E_i} = x_i + E_i \quad \text{Unit east wall location} \tag{5}$$

$$x_{W_i} = x_i - W_i \quad \text{Unit west wall location} \tag{6}$$

$$l_i = W_i + E_i \quad \text{Unit length} \tag{7}$$

$$w_i = N_i + S_i \quad \text{Unit width} \tag{8}$$

These relations are linear, so linear functions of these intermediate variables are also linear functions of the original variables.

2.1.2 Geometric Design Constraints

The following constraint groups form a toolbox of constraints that can be applied where appropriate for a particular layout problem. Using the software described in the companion paper [2], default constraints are automatically added to the model whenever the designer adds a room, window, etc. The designer can also add, delete, or modify constraints individually.

The *Force Inside Constraint Group* forces Units into the main building Boundary or other grouping Boundaries. In order to force Unit i inside Unit j, the following four constraints must all be satisfied:

$$y_{N_i} \le y_{N_j}, \quad y_{S_j} \le y_{S_i}, \quad x_{E_i} \le x_{E_j}, \quad \text{and} \quad x_{W_j} \le x_{W_i} \tag{9}$$

The *Prohibit Intersection Constraint* functions to prevent two Units from occupying the same space. By default, one *Prohibit Intersection Constraint* is added for each 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, at least one of the following constraints must be satisfied

$$(x_{W_i} \ge x_{E_j}) \ \text{OR} \ (x_{W_j} \ge x_{E_i}) \ \text{OR} \ (y_{S_i} \ge y_{N_j}) \ \text{OR} \ (y_{S_j} \ge y_{N_i}) \tag{10}$$

The logical disjunction can be represented in negative null form using a min function

$$\min(x_{E_j} - x_{W_i}, x_{E_i} - x_{W_j}, y_{N_j} - y_{S_i}, y_{N_i} - y_{S_j}) \le 0 \tag{11}$$

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.

The *Force Intersection Constraint Group* is used when Units are 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

$$y_{S_i} \le y_{N_j}, \quad y_{S_j} \le y_{N_i}, \quad x_{W_i} \le x_{E_j}, \quad \text{and} \ x_{W_j} \le x_{E_i} \tag{12}$$

Although these constraints ensure intersection of the two Units, they permit intersection at a point. Designers of architectural spaces are generally interested in intersection that provides enough room for a doorway or opening. To model this, an additional constraint is included to enforce overlap in one of the Cartesian directions that is at least as large as the doorway or opening. Therefore, in addition to intersection, at least one of the following conditions must be satisfied

$$y_{N_j} - y_{S_i} \ge \max(d_i, d_j) \quad \text{Unit } i \text{ overlaps north wall of Unit } j, \tag{13}$$

$$y_{N_i} - y_{S_j} \ge \max(d_i, d_j) \quad \text{Unit } i \text{ overlaps south wall of Unit } j, \tag{14}$$

$$x_{E_j} - x_{W_i} \ge \max(d_i, d_j) \quad \text{Unit } i \text{ overlaps east wall of Unit } j, \tag{15}$$

$$x_{E_i} - x_{W_j} \ge \max(d_i, d_j) \quad \text{Unit } i \text{ overlaps west wall of Unit } j. \tag{16}$$

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. (11).

$$\begin{aligned}\min\{&\max(d_i, d_j) - x_{E_j} + x_{W_i},\ \max(d_i, d_j) - x_{E_i} + x_{W_j},\\ &\max(d_i, d_j) - y_{N_j} + y_{S_i},\ \max(d_i, d_j) - y_{N_i} + y_{S_j}\} \le 0\end{aligned} \tag{17}$$

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 Eq. (11)). The complete *Force Minimum Intersection Constraint Group* is represented as a set of constraints that force intersection (Eq. (12)) and another constraint to ensure that the overlap is large enough for access (Eq. (17)).

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 as appropriate.

$$y_{N_i} = y_{N_j},\ \ y_{S_i} = y_{S_j},\ \ x_{E_i} = x_{E_j},\ \ \text{or} \quad x_{W_i} = x_{W_j} \tag{18}$$

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 direction the door faces), then the following constraint can be added to represent the disjunction in Eq. (18).

$$\min\left\{(x_{E_i} - x_{E_j})^2, (x_{W_i} - x_{W_j})^2, (y_{S_i} - y_{S_j})^2, (y_{N_i} - y_{N_j})^2\right\} = 0 \tag{19}$$

This representation is non-smooth at Unit corners; however, it is quadratic in most of the design space.

The *Bound Size Constraint Group* includes three kinds of constraints to bound the area of a Unit: minimum area, minimum length/width, and maximum length/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.

$$A_{\min_i} - l_i w_i \le 0 \qquad \text{minimum area} \tag{20}$$

$$l_{\min_i} - l_i \le 0 \quad \text{and} \quad l_{min_i} - w_i \le 0 \qquad \text{minimum length/width} \tag{21}$$

$$l_i - l_{\max_i} \le 0 \quad \text{and} \quad w_i - l_{\max_i} \le 0 \qquad \text{maximum length/width} \tag{22}$$

The *Minimum Ratio Constraint Group* can be used 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.

$$R_{\min_i} l_i - w_i \le 0 \quad \text{and} \quad R_{\min_i} w_i - l_i \le 0 \tag{23}$$

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

$$\kappa_{\text{wall}}(A_N + A_S + A_E + A_W) + \kappa_\omega(A_{\omega_N} + A_{\omega_S} + A_{\omega_E} + A_{\omega_W}) \le \Gamma_{\text{budget}} \tag{24}$$

where A_N,A_S,A_E,A_W are the areas of the external walls in each compass direction and A_{ω_N},A_{ω_S},A_{ω_E},A_{ω_W} are the areas of windows facing each compass direction. These quantities are computed in Eq. (30) and Eq. (31).

The *Feasible Window Constraints* ensure that the window width cannot be larger than the wall it is on. In addition to the simple bound restricting window size to be positive, this ensures feasible window size. Each window added to a Unit is given one of the following *Feasible Window Constraints* as appropriate.

$$\omega_{N_i} \leq l_i, \quad \omega_{S_i} \leq l_i, \quad \omega_{E_i} \leq w_i, \quad \text{or} \quad \omega_{W_i} \leq w_i \tag{25}$$

The *Bound Lighting Constraint* is used to ensure minimum natural lighting for specific rooms. 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: First, available daylight at the window exterior is determined. IESNA [31] 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.

$$E_{vkd_m} = \text{vertical sky illuminance (direct)}$$
$$E_{vks_m} = \text{vertical sky illuminance (sky)}$$

for month m. The coefficient of utilization, C_U, a function of the room geometry and window size, determines the fraction of the available daylight that enters the room. C_U can be found in pre-tabulated data [30] based on room depth, window width, and window height. The net transmittance for a window facing direction j is calculated as

$$\mu_j = \frac{0.9\mu_G A_{\omega_j}}{A_j} \tag{26}$$

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. Finally, the daylight at the room center is calculated. The horizontal illuminance at the center of room i is calculated as

$$E_i = \left(\sum_j \sum_m A_{\omega_j} (E_{vkd_{m_i}} + E_{vks_{m_i}}) \mu_j C_U 10^2 \right) \tag{27}$$

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:

$$\Theta_i = \frac{E_i 10^{-3}}{A_i \beta_{\text{eff}}} \tag{28}$$

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 $A_i \theta_{\text{req}_i}$. The minimum percentage of required lighting that is provided by natural light, $\varphi_{\min_i}$, can be specified by the designer. The final constraint is written as:

$$\frac{\Theta_i}{A_i \theta_{\text{req}_i}} \geq \varphi_{\min_i} \tag{29}$$

2.1.3 Geometric Design Objectives

Several objectives have been defined that can be used independently or together depending on the designer's goals.

The *Minimize Heating Cost Objective* estimates heating loss during cold months. 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 [30]) are used as an approximation. The procedure for calculating heating loads is as follows. It is assumed that windows on all Units are constrained against external walls, so the net area of windows on each external wall is:

$$A_{\omega_D} = \sum_{i \in \text{Unit}_D} \omega_{D_i} h_{\omega_i} \tag{30}$$

Here D takes on the four directions {north, south, east, west}, and Unit_D refers to Units that have windows in direction D. The net area of each external wall is

$$A_D = l_1 h_1 - A_{\omega_D} \tag{31}$$

Here D takes on the directions {north, south, east, west}, and 1 indicates Unit 1, which is assumed to be the building Boundary Unit. 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

$$Q_{\text{heat}} = \sum_i \Delta T_i((A_N + A_S + A_E + A_W)U_{\text{wall}} + (A_{\omega_N} + A_{\omega_S} + A_{\omega_E} + A_{\omega_W})U_\omega) \tag{32}$$

where i is the set of months where heat is used, and ΔT_i is the average internal/external temperature difference for month i. Finally, the cost to maintain temperature is calculated. 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 then

$$\text{minimize } \Gamma_{\text{heat}} = \frac{\kappa_{\text{gas}} Q_{\text{heat}}}{\eta_{\text{heater}}} \tag{33}$$

The *Minimize Cooling Cost Objective* estimates heat gain during hot months. The procedure for calculating cooling loads is more complicated than heating loads because heat due to solar gain must be taken into account. The procedure works as follows. First, the net area of windows on each external wall is calculated using Eq. (30), and the net area of each external wall is calculated using Eq. (31). Next, the solar heat gain through the windows is estimated. 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 [30]. The shading coefficient, β_{sc}, is a property of the glass [30], and the time-lag factor, β_{tlf}, is a tabulated function of glass type and window orientation [30]. The annual solar heat gain, Q_{solar}, is calculated as

$$Q_{\text{solar}} = \beta_{\text{sc}} \left(\sum_i (A_{\omega_N} \beta_{\text{shgf}_N} \beta_{\text{tlf}_N} + A_{\omega_S} \beta_{\text{shgf}_S} \beta_{\text{tlf}_S} + A_{\omega_E} \beta_{\text{shgf}_E} \beta_{\text{tlf}_E} + A_{\omega_W} \beta_{\text{shgf}_W} \beta_{\text{tlf}_W}) \right) \tag{34}$$

where i is the set of months where air conditioning is used. Next, the conductive heat gain through the building exterior is estimated. The orientation of each exterior wall and windows is accounted for in the factor. The cooling load due to conduction is calculated as

$$Q_{\text{cond}} = \sum_i \Delta T_i (U_\omega (A_{\omega_N} \beta_{\text{tlf}_N} + A_{\omega_S} \beta_{\text{tlf}_S} + A_{\omega_E} \beta_{\text{tlf}_E} + A_{\omega_W} \beta_{\text{tlf}_W}) + U_{\text{wall}} (A_N \beta_{\text{tlf}_N} + A_S \beta_{\text{tlf}_S} + A_E \beta_{\text{tlf}_E} + A_W \beta_{\text{tlf}_W})) \tag{35}$$

where i is the set of months where air conditioning is used. Finally, the cost to maintain room temperature is calculated. 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 then

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

The *Minimize Lighting Cost 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 Eq. (28). 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:

$$\Gamma_{\text{elec}} \left(\sum_i \beta_{\text{light}_i} A_i \right) \beta_H 10^{-3} \tag{36}$$

where i is the set of Units, and β_H is the number of hours of use per month. The total cost is then the maximum possible electricity cost minus the cost savings from natural lighting:

$$\text{minimize } \Gamma_{\text{light}} = \Gamma_{\text{elec}} - \left(\sum_i \Theta_i \right) \beta_A 10^{-3} \tag{37}$$

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

The *Minimize Wasted Space Objective* minimizes building 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

$$\text{minimize } \left(l_1 w_1 - \sum_{i \in \text{Rooms}} l_i w_i \right) \tag{38}$$

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

The *Minimize Accessways Objective* brings connected Units together. Accessways may be constrained to be small 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

$$\text{minimize } \left(\sum_{i \in \text{Accessways}} l_i w_i \right) \tag{39}$$

The *Minimize Hallway Objective* is used to provide extra living space where possible. The objective is formulated as

$$\text{minimize}\left(\sum_{i \in \text{Hallways}} l_i w_i\right) \tag{40}$$

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

$$f(\boldsymbol{x}) = \sum_{j=1}^{N} w_j f_j(\boldsymbol{x}) \tag{41}$$

where $f_j(\boldsymbol{x})$ is the jth objective function, w_j is the weighting (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 presented here 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.

2.2 Geometry Model Solution Methods

2.2.1 Local Optimization Method

CFSQP, a C implementation of Feasible Sequential Quadratic Programming [25], was used to solve the building geometric layout problem presented above. 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 also handles linear constraints separately so that they are solved more efficiently. A sample optimization of a particular layout problem is shown in Figure 4.

CFSQP is very fast for moderately sized problems using this formula, and it is relatively stable; however, sometimes the algorithm becomes stuck, partly due to non-smooth constraints (Eqs. (11), (17), (19)). 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.

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

2.2.2 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.

Both Simulated Annealing (SA) and Genetic Algorithms (GA) were implemented to search the geometry design space for global solutions. Because of the highly constrained

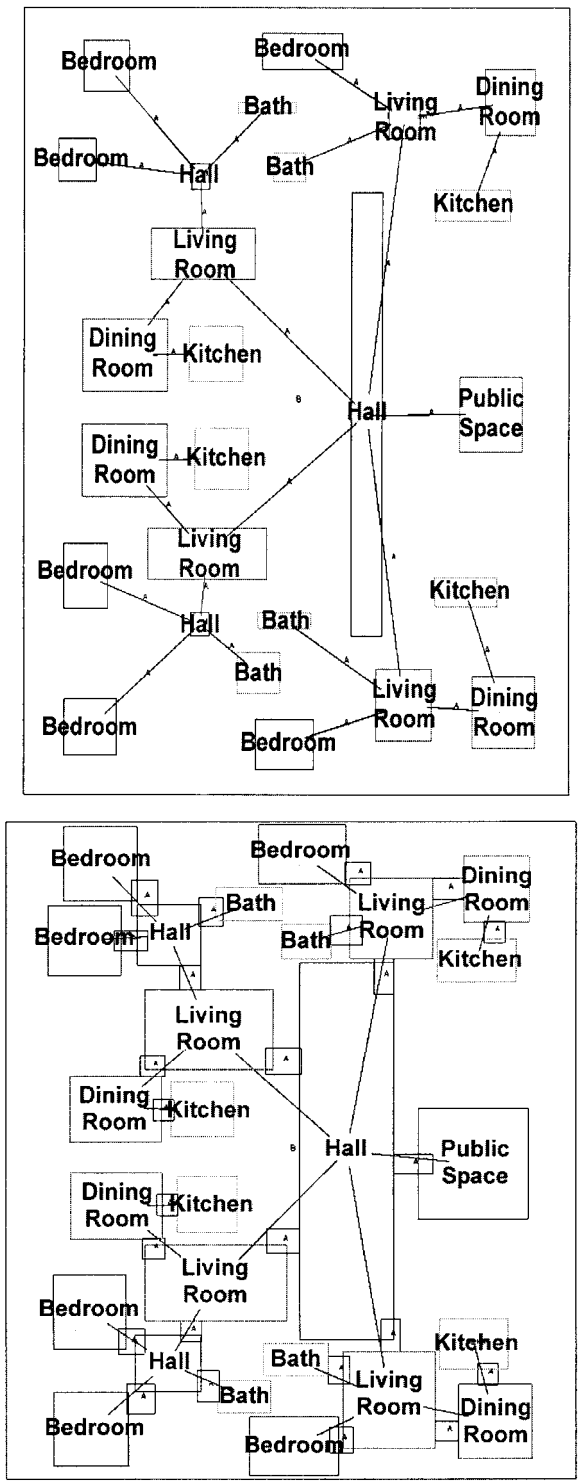

FIGURE 4 Progression of the CFSQP algorithm optimizing a sample apartment complex building to minimize annual cost and wasted space; (a) shows the initial layout sketch provided by the designer (accessways shown as lines between Units); (b) is an intermediate feasible iteration (accessways shown also as rectangles); and (c) shows the completed design (accessways shown as wall openings for clarity).

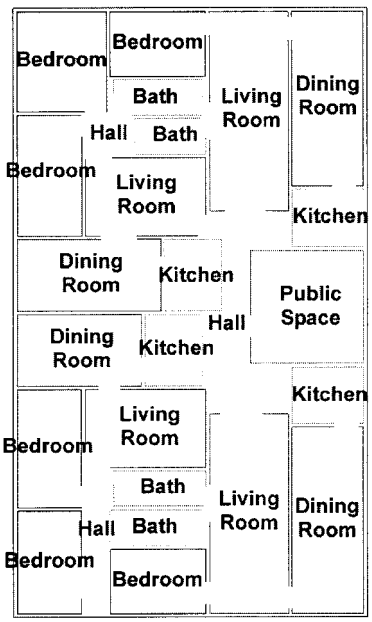

FIGURE 4 (*Continued*)

nature of the formulation, neither algorithm was successful at finding feasible designs for small problems. This does not mean that the algorithms cannot be successful at laying out architectural spaces; however, both algorithm are ill-suited to the formulation presented here.

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 works as shown in Figure 5.

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 6. 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

1	Choose initial design and initial temperature	$T = T_{init}, \underline{x}_k = \underline{x}_{init}$
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.	if $(f(\underline{x}_\Delta) > f(\underline{x}_k))$: $\underline{x}_{k+1} = \underline{x}_\Delta$ else if $(rand(0,1) < p(T))$: $\underline{x}_{k+1} = \underline{x}_\Delta$ 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 5 Description of the SA/SQP hybrid algorithm.

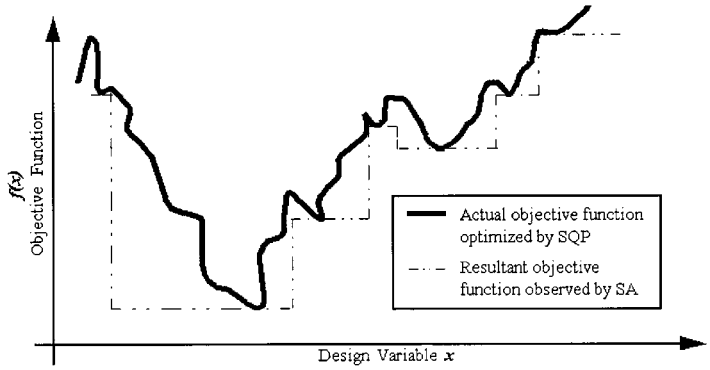

FIGURE 6 Hybrid SA/SQP sample function with multiple local minima.

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 hybrid SA/SQP method generated local optima of reasonable global quality for up to seven room apartment layouts (70 variables, 269 constraints – see Ref. [1] for resulting layouts). 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 arrangement is not specified by the designer. The algorithm is able to automatically generate a quality feasible arrangement and optimize that geometry locally.

Another way to search for solutions of global quality is to use a variation of an optimization technique referred to as the Maximum Distance Distribution Method (MDDM [23, 24]). 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 $f(\boldsymbol{x}^*)$.

$$\begin{array}{ll} \text{maximize} & (\boldsymbol{x}-\boldsymbol{x}^*)^2 \\ \text{subject to} & \boldsymbol{h}(\boldsymbol{x})=\boldsymbol{0}, \\ & \boldsymbol{g}(\boldsymbol{x}) \le \boldsymbol{0}, \\ & \boldsymbol{f}(\boldsymbol{x})-\boldsymbol{f}(\boldsymbol{x}^*) \le \boldsymbol{0} \\ & \boldsymbol{x} \in \Re^n \end{array} \tag{42}$$

If optimizing Eq. (42) 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 often yield a better local minimum. This process can be repeated by iteratively solving Eqs. (1) and (42) 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 7. The method is especially useful if $f(\boldsymbol{x}^*)$ is flat in some feasible direction at $\boldsymbol{x}^*$.

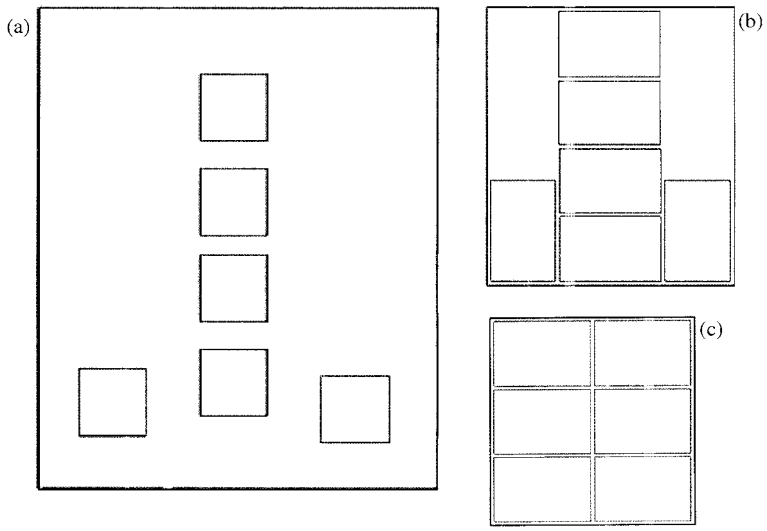

FIGURE 7 Demonstration of the MDDM method for finding improved local optima. An initial design (a) was optimized using CFSQP. The result is a local optimum (b) (the design cannot be improved by small changes in the design variables). The MDDM method was used to generate design (c), an improved local optimum for this example.

Another design exploration program was written to produce design alternatives by searching the space using a strategy of random design changes. The program makes design moves of three types: (1) swap the positions of two Units, (2) perturb the position of a Unit, and (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 one of the three design moves 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. 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 (see Ref. [2]).

3 OPTIMIZATION OF TOPOLOGY

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 ([9, 10]) 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.

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 *each topology is evaluated 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 8 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. A locally optimal geometry $\boldsymbol{x}^*$ is found using CFSQP, 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.

3.1 Mathematical Topology Optimization Model

3.1.1 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.

$$\begin{aligned} &x_i, y_i \in Z_+ \\ &\phi_{ij} \in \{0, 1\} \\ &\forall\, i \in \{\text{rooms}\}, \quad \forall\, j \in (\{\text{rooms} > i\} \cup \{\text{extwalls}\}) \end{aligned} \tag{43}$$

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 9

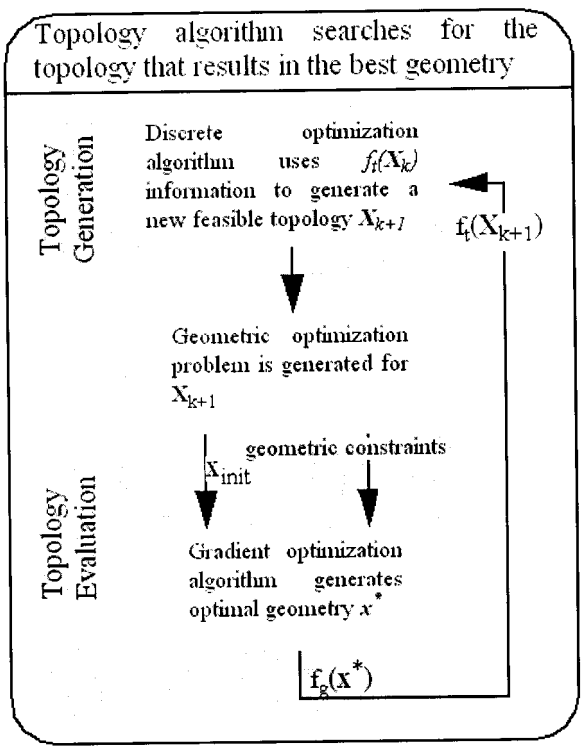

FIGURE 8 Building topology optimization method.

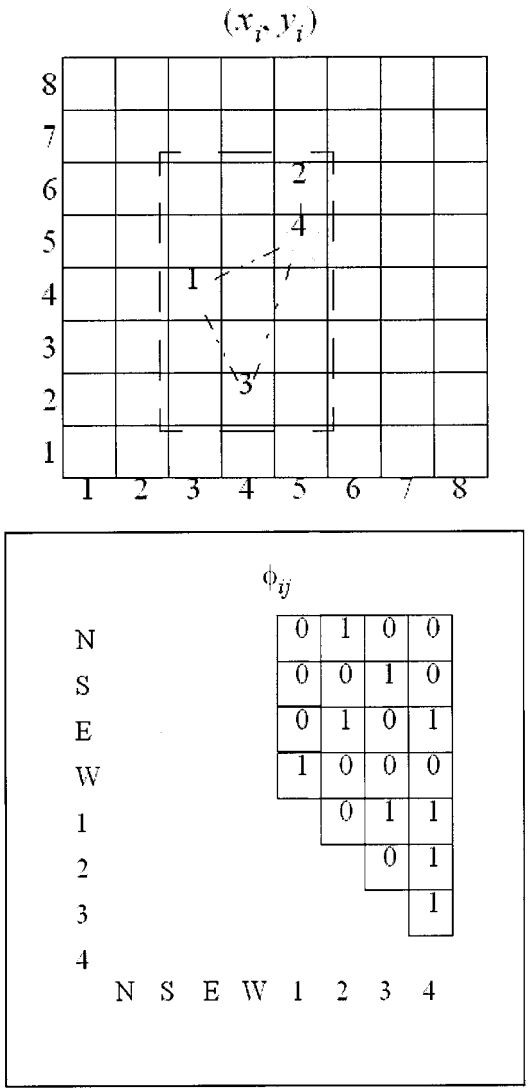

FIGURE 9 A 4-room example showing design variables in the topology formulation. (a) Room position grid showing (x, y) for each room. Phantom lines show room connections. The dashed line shows the implied boundary.

shows a visual representation of the design variables. It is important to note that topological decisions about relative positions of rooms (*i.e.*, *room i* is-north-of *room j*) are represented here using absolute positions of rooms. Several other methods of representing topological decisions ([9, 10]) do not use absolute positions; however, it is necessary in this strategy because the geometric optimization algorithm requires a starting design with geometric information. The use of absolute positions has several consequences: (1) 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. (2) 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. It is not clear, however, that every possible

geometric alternative can be generated using the topology definition in this paper. (3) The space of topology combinations is exponential.

Because of these limitations, this representation is not well suited to problems where all solutions need to be enumerated. 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.

3.1.2 Topology Design Constraints

The following constraints form a toolbox of constraints that can be applied where appropriate for a particlar topology layout problem.

The *Overlap Constraint* ensures that no two rooms occupy the same space.

$$|x_i - x_j| + |y_i - y_j| \geq 1 \quad \forall\, i \neq j \tag{44}$$

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,

$$\phi_{ij} = 1 \;\text{ room } i \text{ required to connect to room } j \tag{45}$$

$$\phi_{ij} = 0 \;\text{ room } i \text{ required not to connect to room } j \tag{46}$$

$$\phi_{iN} + \phi_{iS} + \phi_{iE} + \phi_{iW} \geq 1 \;\text{ room } i \text{ required to connect to at least one external wall} \tag{47}$$

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 Ref. [1]).

Planarity Constraints ensure 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. (see Ref. [1])

Envelope Constraints ensure that 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

$$\phi_{iN}(\max(y_1, y_2, \ldots, y_n) - y_i) = 0 \tag{48}$$

$$\phi_{iS}(y_i - \min(y_1, y_2, \ldots, y_n)) = 0 \tag{49}$$

$$\phi_{iE}(\max(x_1, x_2, \ldots, x_n) - y_i) = 0 \tag{50}$$

$$\phi_{iW}(x_i - \min(x_1, x_2, \ldots, x_n)) = 0 \tag{51}$$

3.1.3 Objective

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

$$\text{minimize}(f_g(\text{SQP}(\boldsymbol{X}))) \tag{52}$$

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 Fig. 10). The optimization objective can be anything defined by the geometry. Typically, this research has 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.

3.1.4 Penalty Functions

In this formulation, penalty functions are used to represent constraints, and infeasible designs are not passed to the geometric optimizer. Figure 11 shows how this penalty function works. Here the topology objective function $f_t(\boldsymbol{X})$ is maximized. 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, $\boldsymbol{X}$ 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 $f_g(\boldsymbol{x}^*)$. The bonus value is set so that it is larger than any objective value $f_g(\boldsymbol{x})$. Using this method, all infeasible topologies return negative function values, all feasible

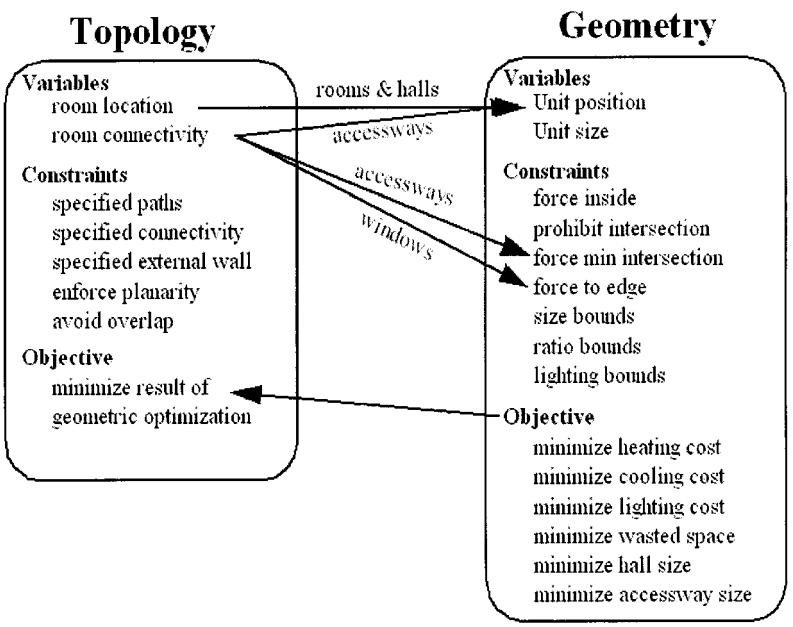

FIGURE 10 Schematic showing the relationship between the topology and geometry optimization algorithms.

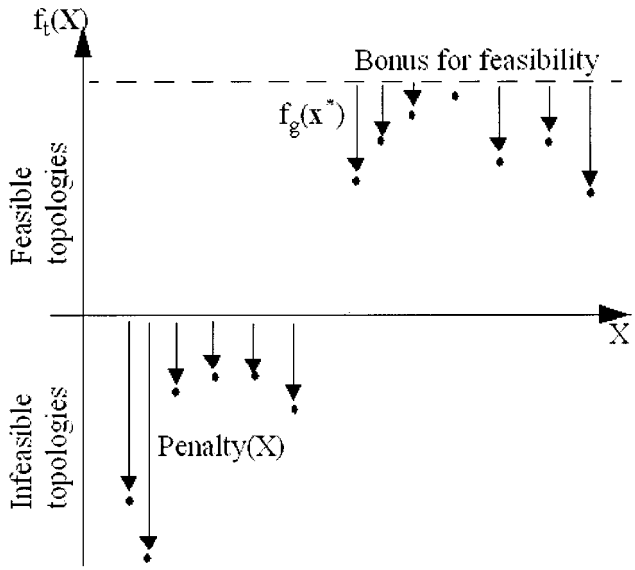

FIGURE 11 Formulation of the topology objective function.

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* $f_t(\boldsymbol{X})$).

3.2 Topology Model Solution Method

The discrete topology design space is combinatorial, multi-modal, and highly constrained, so it must be searched with a global scope. The space of topologies could be searched exhaustively with a constraint satisfaction programming enumeration algorithm [27] 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 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.

An evolutionary algorithm for topology layout was implemented using the GAlib optimization package [28]. A SteadyStateGA was selected. A Roulette Wheel selector was used to select high quality designs with greater probability than low quality designs. When sexual crossover is used (randomly), two parents are selected from the population, and two new children are produced using mixed room connectivity from both parents. When asexual crossover is used (randomly), one parent is selected from the population, and one new child is produced by swapping connectivity values between rooms or by swapping room positions. After crossover, new designs are mutated slightly. Room locations (x,y) are incremented or connectivities are flipped with low probability.

The evolutionary algorithm implementation is able to generate quality feasible designs for medium-sized problems.

TABLE I Room Specifications for Demonstration Problem.

Apt	*Room*	*Min. area (sq. ft.)*	*Min. length and width (ft.)*	*Max. length and 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

4 DEMONSTRATION EXAMPLE

A realistic problem was implemented to test the scalability of the automated building generation algorithm. The example involves a small apartment complex with three separate apartments. Rooms and specifications are shown in Table I. Constraints that are specific to this problem are listed in Table II. This problem was run for 20,000 generations (100 designs each generation) to search for global solutions. Feasible designs take much longer to evaluate than infeasible designs (because feasible designs 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 between runs.

The sample topology and resulting geometry solution shown in Figure 12 were generated using the automated design tool.

TABLE II Topology Specifications for Demonstration Problem.

Constraint type	*Constraint*
Overlap	No two Units can occupy the same space
Connectivity	Public Entry must connect to the Living Room of each apartment
Connectivity	Public Entry must connect to an external wall
Connectivity	All bedrooms must connect to an external wall
Path	In each apartment, there must be a path from the Kitchen to the Living Room that may pass through the Dining Room
Path	In each apartment, there must be a path from the Bathroom to the Living Room that may pass through the Dining Room and Kitchen
Path	In each apartment, there must be a path from the Dining Room to the Living Room that may pass through the Kitchen
Path	In each apartment, there must be a path from each Bedroom to the Living Room that may pass through the Dining Room
Accessways	Accessway lines connecting Units cannot intersect
Envelope	Units that are connected to an external wall must lie on the boundary envelope of rooms

The algorithm generates local optimal solutions, but the global search is quite limited due to combinatorial complexity. Once a feasible topology is found, it has a much higher probability of being selected as a parent design by the evolutionary algorithm because it has 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. For this problem, the evolutionary algorithm can consistently find solutions in under 20,000 generations (2×10^6 design evaluations).

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. The example shown in Figure 4, 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.

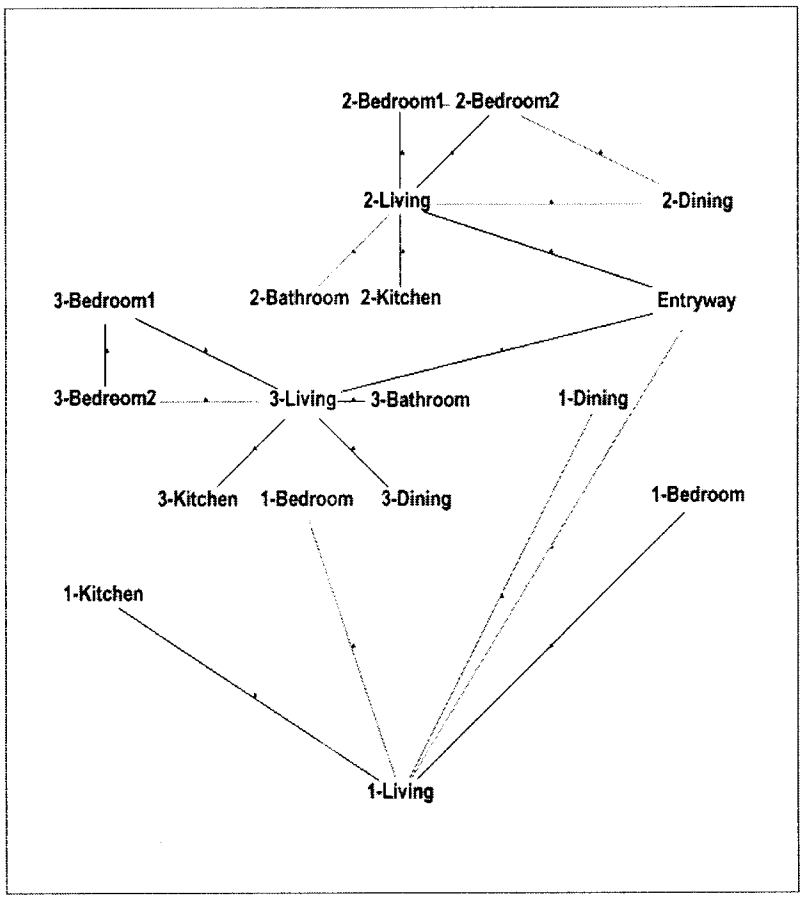

FIGURE 12 Sample design topology and final geometry generated by the automated design tool.

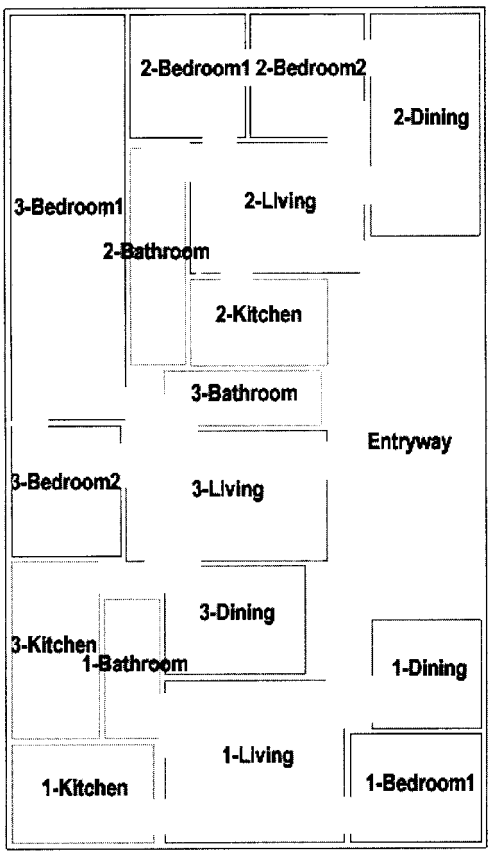

FIGURE 12 (*Continued*)

5 CONCLUSIONS

Two automated optimization algorithms have been used to automate the generation of design layouts: the geometry and topology algorithms. The geometry algorithm, built on rigorous gradient-based algorithms, is efficient and robust, and 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 algorithm and a strategic programming method. 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 results are interesting, but limited. 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 (see Ref. [2]).

Possible improvements include the addition of new shapes, objectives, and constraints to the design toolbox to address more complex geometry, material selection, building codes, structural elements, routing of wires, pipes and ducts, and more accurate models. Additionally, the topology search can be improved if the topology can be defined in a new

way so that topology decisions create linear constraints in the geometry optimization problem. Using a different definition of topology, such as in Refs. [3–5], it is possible to eliminate the rough position topology variables, and ensure that the mapping from topology to local optimal geometry is both injective and surjective (meaning that each valid topology will create a different locally optimal geometries, and that all possible local optimal geometry could be created by a specific topology).

Acknowledgements

Special thanks to Professor Kazuhiro Saitou, John Whitehead, Panayiotis Georgiopoulos, and Adam Cooper for their contributions to the topology optimization model and solution strategy.

References

[1] Michalek, J. (2001). Interactive layout design optimization. *MS thesis*, University of Michigan.
[2] Michalek, J. and Papalambros, P. Y. (2002). Interactive design optimization of architectural layouts. *Engineering Optimization* (this issue).
[3] Levin, P. H. (1964). Use of graphs to decide the optimum layout of buildings. *Architect*, **14**, 809–815.
[4] Liggett, R. S. and Mitchell, W. J. (1981). Optimal space planning in practice. *Computer-Aided Design*, **13**(5), 277–288.
[5] Sharpe, R., Marksjo, B. S., Mitchell, J. R. and Crawford, J. R. (1985). An interactive model for the layout of buildings. *Applied Mathematical Modeling*, **9**, 207–214.
[6] Jo, J. H. and Gero, J. S. (1998). Space layout planning using an evolutionary approach. *Artificial Intelligence in Engineering*, **12**, 149–162.
[7] Jagielski, R. and Gero, J. S. (1997). A genetic programming approach to the space layout planning problem. *CAAD Futures*, 875–884.
[8] Baykan, C. A. and Fox, M. S. (1997). Spatial synthesis by disjunctive constraint satisfaction. *Artificial Intelligence in Engineering Design*, **11**, 245–262.
[9] Schwarz, A., Berry, D. M. and Shaviv, E. (1994). Representing and solving the automated building design problem. *Computer-Aided Design*, **26**(9), 689–698.
[10] Medjdoub, B. and Yannou, B. (1999). Separating topology and geometry in space planning. *Computer-Aided Design*, **32**, 39–61.
[11] Yin, S. and Cagan, J. (2000). An extended pattern search algorithm for three-dimensional component layout. *Transactions of the ASME*, **122**, 102–108.
[12] Cagan, J., Degentesh, D. and Yin, S. (1998). A simulated annealing-based algorithm using hierarchical models for general three-dimensional component layout. *Computer-Aided Design*, **30**(10), 781–790.
[13] Szykman, S. and Cagan, J. (1997). Constrained three-dimensional component layout using simulated annealing. *ASME Transactions*, **119**, 28–35.
[14] Choo, H. J. and Tommelein, I. D. (1999). Space scheduling using flow analysis. *Proc. Seventh Annual Conference of the International Group for Lean Construction*, 299–312.
[15] Rong, B. (1997). Automated generation of fixture configuration design. *Journal of Manufacturing Science and Engineering*, **119**, 208–219.
[16] Koide, T. and Wakabayashi, S. (1999). A timing-driven floorplanning algorithm with the Elmore delay model for building block layout. *Integration, the VLSI Journal*, **27**, 57–76.
[17] Wang, Y. and Wu, H. (1998). Method of constraint graphs used in spatial layout. *Ruan Jian Xue Bao Journal of \Software*, **9**(3), 200–205.
[18] Ito, T. (1999). A genetic algorithm approach to piping route path planning. *Journal of Intelligent Manufacturing*, **10**, 103–114.
[19] Fadel, G. M., Sinha, A. and McKee, T. (2001). Packing optimisation using a rubberband analogy. *Proceedings of DETC'01 2001, ASME Design Engineering Technical Conferences*, DETC2001/DAC-21051.
[20] Kim, J. J. and Gossard, D. C. (1991). Reasoning on the location of components for assembly packaging. *Journal of Mechanical Design, Transactions of the ASME*, **113**(4), 402–407.
[21] Chapman, C. D., Saitou, K. and Jakiela, M. J. (1994). Genetic algorithms as an approach to configuration and topology Design. *Journal of Mechanical Design, Transactions of the ASME*, **116**(4), 1005–1012.
[22] Papalambros, P. Y. and Wilde, D. J. (2000). *Principles of Optimal Design–Modeling and Computation*, 2nd ed. Cambridge University Press, Cambridge, England.
[23] Kott, G. J. and Gabriele, G. (1998). A tunnel based method for mixed discrete constrained nonlinear optimization. *ASME Design Automation Conference*, DETC98/DAC-5592.
[24] Kott, G. J. (1998). A method for mixed variable constrained nonlinear optimization based on a new function bounding technique. *PhD thesis*, Rensselaer Polytechnic Institute.

[25] Zhou, J. L. and Tits, A. L. (1995). An SQP algorithm for finely discretized continuous minimax problems and other minimax problems with many objective functions. *SIAM Journal of Optimization*, **6**, 461–487.
[26] Lawrence, C. T., Zhou, J. L. and Tits, A. (1995). 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.*
[27] Bartak, R. (1999). Constraint programming: In pursuit of the Holy Grail. http://kti.ms.mff.cuni.cz/∼bartak/constraints.
[28] Wall, M. (1996). GAlib: A C++ library of genetic algorithm components-Version 2.4 -Document revision B. galib-request@mit.edu, http://lancet.mit.edu/ga/dist/galibdoc.pdf.
[29] Bentley, P. J. (1999). *Evolutionary Design by Computers.* Morgan Kaufmann Publishers, Inc., San Francisco, CA.
[30] ASHRAE (1997). *Fundamentals.* American Society of Refrigeration, Heating and Air-conditioning Engineers, Atlanta, GA.
[31] IESNA (1998). *IESNA Lighting Education: Intermediate Level.* Illumination Engineers Society of North America.