Constrained Three-Dimensional Component Layout Using Simulated Annealing

S. Szykman
Engineering Design Laboratory,
Manufacturing Systems
Integration Division,
National Institute of Standards
and Technology,
Gaithersburg, MD 20899

J. Cagan
Department of Mechanical Engineering,
Carnegie Mellon University,
Pittsburgh, PA 15213

This research introduces a computational algorithm that uses simulated annealing to optimize three-dimensional component layouts. General component layout problems are characterized by three objectives: achieving high packing density, fitting components into a given container and satisfying spatial constraints on components. This paper focuses on the extension of a simulated annealing packing algorithm to a general layout algorithm through the implementation of a language of spatial constraints that are characteristic of layout problems. These constraints allow the designer to specify desired component proximities or to restrict translation or rotation of components based on a global origin or set of coordinate axes, or relative to other component locations or orientations. The layout of components from a cordless power drill illustrates the algorithm.

1 Introduction

This paper introduces a computational approach to constrained three dimensional component layout problems. Component layout optimization is a common problem in many engineering applications such as layout of automobile engine compartments and design of mechanical or electromechanical assemblies. Component layout tasks are characterized by three problem-independent objectives:

- achieving high packing density (due to trends in product miniaturization),
- fitting components into a specified container,
- satisfying spatial constraints on component placements.

In Szykman and Cagan (1995), we presented a simulated annealing approach to three-dimensional *packing* problems. These problems are a subset of general component layout problems that incorporate the first two objectives listed above—achieving high packing density and fitting components into a container if one is given. Packing algorithms alone, however, do not fully address the needs of layout designers. In this paper we build upon the packing algorithm to include various types of spatial constraints on component placement. The constraints on components can be applied globally (i.e., constraints based on a global origin or coordinate axes) or relative to other component locations and orientations. With the addition of this spatial constraint language, the extended algorithm can be used to solve more general problems that are representative of mechanical engineering layout tasks.

To achieve a more general algorithm, a language to enable representation of such constraints must be defined and a means of satisfying the potentially numerous constraints during the optimization must be developed. In practice, specifying and satisfying spatial constraints makes this technology feasible for many realistic design tasks. With this approach the tool demonstrates the potential to impact design practice; current approaches to layout design in industry often emphasize physical prototyping and manual placement, a costly and time-consuming process where satisfying coupled spatial relations can be difficult.

The following section discusses other approaches to component layout problems and in section 3 we describe the simulated annealing algorithm. Section 4 describes the spatial constraint language and illustrates how simulated annealing is used to satisfy constraints on components. The constrained layout of a power drill is then presented for illustration in section 5.

Contributed by the Design Automation Committee for publication in the Journal of Mechanical Design. Manuscript received Oct. 1994; revised Sept. 1996. Associate Technical Editor: D. A. Hoeltzel.

2 Related Work

Several research efforts have focused on layout problems that involve locating components in space while satisfying spatial relationships among them. These spatial relationships consist of a noninterference condition and may also include additional constraints on component placement.

A majority of the research in this area has been done in the field of operations research (OR), focusing primarily on the packing problem. The various OR approaches to packing problems are summarized in survey papers by Coffman et al. (1984), Dyckhoff (1990) and Dowsland and Dowsland (1992). Packing problem formulations typically do not allow representation of various types of spatial constraints on component placements. For a layout tool to be useful in general engineering applications, representation of these types of constraints is essential.

A variety of optimization techniques have been applied to three-dimensional component layout problems by Sandgren and Dworak (1988), Udy et al. (1988), Fujita et al. (1991), Kim and Gossard (1991) and Landon and Balling (1994). These approaches have a number of disadvantages associated with the nonlinear nature of layout problems: they either require feasible starting points, linearization of nonlinear equations (which can lead to poor convergence) or require the use of gradients which may be difficult to calculate. In addition, these techniques generally converge to local rather than global optima.

Two non-gradient based search techniques are hierarchical generate-and-test and constraint-directed search (e.g., Flemming et al., 1992). With both of these approaches, search is performed through a hierarchical tree-like structure where nodes correspond to states in the design space. Using hierarchical generate-and-test, the expansion of nodes is done using a branch-and-bound search. Constraint-directed search performs search in a similar tree-like space but uses a different kind of search control. Design specifications and constraints are reduced to sets of disjunctive constraints. The constraint-directed search is accomplished by instantiating one disjunct from one of the disjunctive constraints to form the root node of the tree; the tree is then pruned by removing any incompatible or trivially satisfied constraints.

One advantage of such schemes is that for small problems they are able to enumerate all possible feasible solutions, and can therefore guarantee global optimality within their adjacency restrictions. However, the drawback is that the state space grows exponentially, making this exhaustive search impractical for larger problems. Although both techniques use constraints to prune search so that many infeasible states are not generated, for problems that are not highly constrained, the number of feasible states can still suffer from a combinatorial explosion and can produce an "unmanageably large" set of feasible solutions (Flemming et al., 1992).

Simulated annealing (Kirkpatrick et al., 1983) does not share the drawbacks described above. In particular, simulated annealing can escape from local optima and is a zero-order algorithm that does not require the use of gradients. In addition, simulated annealing does not enumerate all states during search. By not doing an exhaustive search, simulated annealing sacrifices the ability to guarantee global optimality in exchange for the ability to search through a large set of solutions in a reasonable amount of time. Although global optimality is not guaranteed, in practice simulated annealing generally returns near-optimal solutions and does, at times, find global optima. Simulated annealing has successfully been applied to two dimensional layout problems in the domain of electrical engineering[1] (e.g., Sechen and Sangiovanni-Vincentelli, 1985; Cohn et al., 1991), where this technology is well established beyond the academic research community and is used extensively for circuit layout in industry.

In Szykman and Cagan (1995), we introduced a simulated annealing algorithm for three dimensional component packing that borrows concepts developed for two dimensional VLSI layout optimization. The packing algorithm is validated by solving a set of benchmark problems with known global optima and consistently converges to good designs for a problem with an unknown global optimum. This paper presents an extension of the packing algorithm to general engineering layout problems through the definition and implementation of a constraint language that enables a designer to impose spatial constraints on component placements.

3 The Simulated Annealing Algorithm

Simulated annealing (Kirkpatrick et al., 1983) is a stochastic optimization technique that has been used to solve continuous, ordered discrete and multi-modal optimization problems. In a typical simulated annealing algorithm, a feasible state is chosen as a starting state and evaluated (i.e., the value of the objective function at that point is calculated). A step is taken to a new design state which is then evaluated. If this step leads to an improvement in the objective function, the new design is accepted and becomes the current design state.

If the step leads to an inferior state, the step may still be accepted with some probability. This probability is a function of a decreasing parameter called *temperature,* based on an analogy with the annealing of metals, given by:

$$P_{\text{accept}} = e^{-(\Delta C/T)}, \qquad (1)$$

where ΔC is the change in objective function due to the move and T is the current temperature. The temperature starts out high and decreases with time. Initially, steps taken through the state space (and therefore the objective function space) are almost random, resulting in a broad exploration of the objective function space. As the probability of accepting inferior steps decreases, those steps tend to get rejected, allowing the algorithm to converge to an optimum once promising areas of the objective function space have been found.

To allow designers to incorporate a variety of design objectives, the formulation of this work utilizes a generic objective function, F, consisting of a weighted sum of the form:

$$F = W_1C_1f_1 + W_2C_2f_2 + \ldots + W_nC_nf_n, \qquad (2)$$

where f_i is the value of the ith objective, C_i is the ith normalizing coefficient, and W_i is the weight for the ith term. To avoid scaling problems between objective terms that differ greatly in order of magnitude, each of the terms, f_i, is multiplied by a coefficient, C_i. The value of C_i is one divided by the maximum value of f_i seen at the current temperature, which keeps values of the $C_i f_i$ terms between zero and one; the dynamic penalty coefficient is normalized at each temperature due to the large variance in magnitudes of the various objectives in the objective function across temperatures during the annealing run. Once the objective function terms are scaled using the coefficients, the weights, W_i, allow the user to specify their relative importance.

For a simple component packing task, the objective is to minimize the inverse of the volumetric packing density (maximizing the density). The first term of the objective function is given by:

$$f_1 = \frac{V_{bb}}{\sum_{i=1}^{n} V_i}, \qquad (3)$$

where V_{bb} is the volume of the bounding box of the design (a box that completely encloses the design), n is the number of components, and V_i is the volume of the ith component.

As a layout is perturbed, components are permitted to overlap (Jepsen and Gelatt, 1983). This overlap is penalized in the objective function using a second term, f_2, given by:

$$f_2 = \sum_{i=2}^{n} \left(\sum_{j=1}^{i-1} O_{ij}\right), \qquad (4)$$

where O_{ij} is the overlap between the ith and the jth component. Equation (4) corresponds to the sum of the terms that lie below the diagonal in an "overlap matrix". O_{ij} is defined as:

$$O_{ij} = \frac{V_i^*}{V_{avg}}, \qquad (5)$$

where V_i^* is the volume of overlap between the two components and V_{avg} is the average volume of the ith and jth components. The volume V_i^* may be either exact or a slight approximation to the true volume of overlap due to the use of bounding boxes to speed up the penalty calculation. This approximation can be made without loss of generality because as long as any overlap is penalized, the optimization will tend to eliminate overlap even if the penalty is not exactly equal to the overlap volume. Furthermore, since feasible final designs are free of overlap, the approximation does not affect their evaluation. A third objective function term (f_3), similar to f_2, packs components into a specified container by penalizing components that protrude from the container.

The simulated annealing algorithm described above uses an adaptive annealing schedule taken from Huang et al. (1986), and a method for efficient move selection introduced by Hustin and Sangiovanni-Vincentelli (1987) to solve three-dimensional packing problems. These techniques use statistical information about past performance of the algorithm to control the temperature and move selection, thereby improving the performance of the algorithm. The adaptive annealing schedule, the move selection strategy, and the move set (consisting of moves that can translate a component, rotate a component, or swap the locations of two components) are presented in detail in Szykman and Cagan (1995).

[1] Because VLSI circuits are generally modeled using multiple two-dimensional, or planar circuits, the need to extend this technology to three dimensions has not arisen in the domain of circuit layout. In contrast, the third dimension usually cannot be neglected for component layout problems in mechanical engineering applications.

The packing problems solved using the objectives described above can be thought of as unconstrained layout problems. This paper builds on the earlier packing algorithm and develops a more general layout algorithm by allowing various types of spatial constraints to be specified on component placements. The satisfaction of constraints is accomplished by adding additional terms to the objective function that penalize constraint violations. As the optimization minimizes the objective function, constraint violations are gradually eliminated. Two violation penalty terms involving spatial constraints are introduced: f_4 generates a penalty for violated inequality location constraints while f_5 generates a penalty for proximity constraints; equality constraints are immediately satisfied and propagated through the analysis.

The addition of the constraint satisfaction terms to the objective function is nontrivial due to the large number of constraints and their significant effect on the behavior of the objective function. Successful inclusion of these terms is crucial for the application of this technology in practice. These objectives and the approach towards the constraint satisfaction problem are described in greater detail below.

4 Representing and Satisfying Spatial Constraints

This section presents an approach to representing spatial constraints and a means of incorporating those constraints into the objective function. As previously discussed, extending the packing problem to a general layout problem has been quite limited in the literature due to the significant increase in complexity resulting from the addition of constraints to the problem. Our approach is to allow constraints to be violated during the optimization; infeasible designs are considered valid states and can be accepted by the simulated annealing algorithm. Due to penalties for infeasibilities, over time the optimization drives the layout into the feasible range, thereby satisfying spatial constraints.

To implement this approach, constraints must be described in a way that an appropriate level of penalty can be calculated. We initially describe a set of classes of constraints that are used alone, or in combination, to describe commonly used spatial relations. The constraint classes are defined so that the amount of constraint violation can be quickly calculated. A penalty is added to the objective function so that during the optimization, a feasible solution (assuming one exists) is found by driving penalties, and therefore violations, to zero. The calculations are formulated to take advantage of problems where the components are restricted to 90 deg rotations about the three coordinate axes, as is this implementation of the layout algorithm.

4.1 Types of Constraints. This section defines a constraint language that allows the designer to impose several types of spatial constraints that are characteristic of layout problems. The spatial constraints are divided into three categories: proximity constraints which specify general nearness and farness of components, location constraints which specify position of components, and orientation constraints which specify rotation of components. The constraint language also permits the user to specify both conjunctions and disjunctions of constraints.

Each of the constraints described below also contains a pointer to a constraint type which is not shown in Eqs. (6)–(17). The constraint type allows the constraints to be correctly parsed when they are initially entered as input to the algorithm. This constraint language contains various types of spatial constraints that are representative of those used in engineering layout problems.

4.1.1 Proximity Constraints. Proximity constraints specify general spatial relationships between components. The two types of proximity constraints are *near* and *far* constraints. Both types of proximity constraints take the form:

$$\langle\text{NAME-1}\rangle\langle\text{NAME-2}\rangle\langle\text{DISTANCE}\rangle, \qquad (6)$$

where NAME-1 and NAME-2 are the names of the components being constrained and DISTANCE is the separation distance at which the constraint becomes active. For example, given a proximity constraint:

$$\text{C1 C2 10.0}, \qquad (7)$$

if this equation represents a far constraint, the constraint indicates that the distance between components C1 and C2 should be 10.0 or more. As a near constraint, Eq. (7) specifies that components C1 and C2 should be separated by a distance of 10.0 or less. By applying the constraint twice, as both a far constraint and a near constraint, one can specify that the distance between the components should be as close as possible to 10.0. Unlike equality constraints, however, proximity constraints need not be satisfied exactly.

4.1.2 Location Constraints. Location constraints allow the user to restrict component translational placements using more specific relationships than allowed by proximity constraints, which specify separation distances but not coordinate directions. These constraints may be global constraints, restricting translation of components based on a global origin, or relative constraints which associate the location of one component with that of another.

A global location constraint (GLC) restricts the location of a component relative to a global origin. GLCs may be equality constraints or inequality constraints and take the form:

$$\langle\text{NAME}\rangle\langle\text{COORD.}\rangle\langle\text{LOCATION}\rangle\langle\text{SIGN}\rangle\langle\text{VALUE}\rangle, \qquad (8)$$

where ⟨NAME⟩ is the component name, ⟨COORD.⟩ is one of {X, Y, Z} indicating the coordinate being constrained, ⟨SIGN⟩ is one of {<=, =, >=}, and ⟨VALUE⟩ is the coordinate value for the constraint. The ⟨LOCATION⟩ is one of {MIN, C, MAX} indicating where on the component the constraint is being applied. To illustrate: assuming the Z coordinate is aligned with the vertical direction, the MIN, C and MAX locations refer to the coordinates at the bottom side, the center and the top side of the component, respectively.

The following equation gives an example of an inequality GLC that constrains the Z MIN coordinate (i.e., the bottom) of component C1 to be greater than or equal to one:

$$\text{C1 Z MIN} >= 1.0. \qquad (9)$$

Relative location constraints (RLCs) may also be equality or inequality constraints. Rather than locating components with respect to a global origin, RLCs locate components relative to one another. RLCs are of the form:

$$\langle\text{NAME-1}\rangle\langle\text{COORD.}\rangle\langle\text{LOCATION-1}\rangle\langle\text{SIGN}\rangle$$
$$\langle\text{NAME-2}\rangle\langle\text{COORD.}\rangle\langle\text{LOCATION-2}\rangle\langle\text{OFFSET}\rangle, \qquad (10)$$

where ⟨OFFSET⟩ is an optional value that specifies the distance by which the coordinate must be satisfied. The constraint:

$$\text{C1 Z MIN} >= \text{C2 Z MAX 1.0} \qquad (11)$$

indicates that the bottom (the Z MIN coordinate) of C1 must be higher than the top (the Z MAX coordinate) of C2 by a distance of at least 1.0. By using equality RLCs with no ⟨OFFSET⟩, various centerings, alignments and adjacencies between components can be achieved.

4.1.3 Orientation Constraints. Orientation constraints fix rotation of components and, like location constraints, may be global or relative. There are two kinds of global orientation constraints (GOCs). The first type of GOC is used to fix the orientation of a component, and takes the form:

$$\langle \text{NAME} \rangle = \langle \text{ORIENTATION} \rangle, \qquad (12)$$

where ⟨ORIENTATION⟩ is the desired component orientation. An example of this type of GOC is:

$$\text{C1} = \text{Z\_X\_Y}, \qquad (13)$$

where the orientation of Z_X_Y corresponds to an orientation where the component (local) X, Y and Z axes are aligned with the global Z, X and Y axes, respectively.

During layout optimization, components are rotated by the simulated annealing algorithm. The second type of GOC restricts allowable component orientations by fixing an axis of rotation for the component (in contrast to the previous GOC which constrains a component to a single orientation). This GOC takes the form:

$$\langle \text{NAME} \rangle = \langle \text{AXIS} \rangle, \qquad (14)$$

where ⟨AXIS⟩, one of {X, Y, Z}, is the allowable axis of rotation. The constraint shown in the following equation indicates that component C1 can only rotate about the Y axis:

$$\text{C1} = \text{Y}. \qquad (15)$$

Finally, a relative orientation constraint (ROC) restricts two components to have the same orientation. ROCs are of the form:

$$\langle \text{NAME-1} \rangle = \langle \text{NAME-2} \rangle, \qquad (16)$$

where ⟨NAME-1⟩ and ⟨NAME-2⟩ are the names of the constrained components. The ROC given by:

$$\text{C1} = \text{C2} \qquad (17)$$

constrains components C1 and C2 to have the same orientation.

4.2 Constraint Satisfaction: Propagation vs. Penalizing Violations. Let us define a *valid* design as one in which all the spatial constraints on components are satisfied. It is expected that the layout algorithm produce valid final designs. Two approaches to constraint satisfaction can be taken. One approach is to generate only valid designs. This can be achieved by propagating constraints after each iteration in the simulated annealing algorithm. If a component has a fixed coordinate, it is prevented from moving in a direction which changes that coordinate; when a component is translated or rotated, components constrained to it are moved accordingly.

An alternative approach is to permit constraint violations and penalize them in the objective function. As the algorithm runs, the penalties in the objective function cause constraint violations to be eliminated. The layout algorithm described in this paper uses a hybrid approach to constraint satisfaction that combines both of these methods, propagating equality constraints and penalizing violations of inequality constraints.

A component that has an equality constraint on one of its coordinates is only able to satisfy that constraint at a single value (i.e., its feasible region is a point) for that coordinate. Since simulated annealing is a stochastic algorithm, and because of a fixed minimum translation distance within the algorithm (Szykman and Cagan, 1995), it is possible that this one feasible coordinate value will never be reached. Therefore, to ensure valid final designs, initial locations for all components are selected by the algorithm so that the equality constraints are satisfied. Each time a step is taken, moves are propagated so that satisfaction of equality constraints is enforced at every iteration.

Actively propagating constraints has two disadvantages. First, constraint propagation after each move results in increased computational overhead. Second, simulated annealing algorithms perform better at layout optimization when components are allowed to enter infeasible regions of the design space.[2] Since inequality constraints do not share the problem of single feasible points that equality constraints have, rather than propagating the inequality constraints, they are satisfied by permitting constraint violations and penalizing them in the objective function. The penalty for each violated inequality location constraint is given by:

$$f_4 = DV * C\_Dim, \qquad (18)$$

where DV is the violation distance (i.e., the distance by which the component must be moved to satisfy the constraint) and C_Dim, the component dimension in the constrained coordinate direction, is either the X, Y or Z dimensions of the component.

As a result of the DV term in Eq. (18) the penalty decreases as the violation distance decreases to zero. This term is multiplied by C_Dim to make the penalty higher for larger components so that they are more likely to be moved; as the algorithm progresses it is easier to move smaller components than big ones. Thus, the heaviest violation penalties are caused by the largest components that violate constraints, as well as the ones that are located farthest from feasible positions.

Near and far proximity constraints are treated like ideal extension and compression springs with a free length of DISTANCE [see Eq. (6)]. A near constraint is not violated if the actual distance between components is less than DISTANCE. Similarly, far constraints are not violated if the components are more than DISTANCE apart. The violation of a proximity constraint is penalized in proportion to the "energy" of the "spring". This penalty is given by:

$$f_5 = DV_P^2, \qquad (19)$$

where DV_P, the proximity violation distance, is the difference between DISTANCE and the actual separation distance.

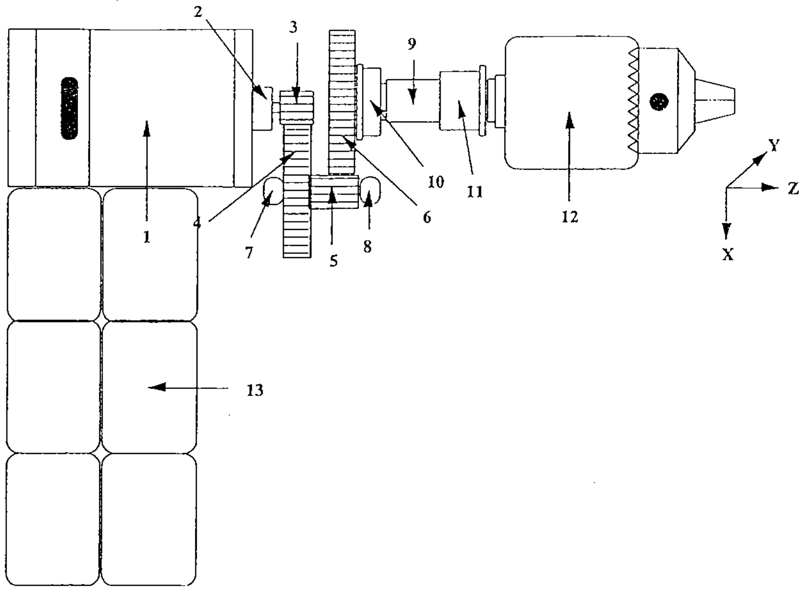
Summing the penalties given by Eqs. (18) and (19) over all the proximity and inequality location constraints give the constraint violation penalty terms f_4 and f_5, which are added to the objective function described in section 3. The use of this hybrid method for satisfying spatial constraints is illustrated in an example in section 5.

4.3 Consistency of Constraints. In this implementation, before the simulated annealing algorithm begins, the set of constraints applied to a problem is sent to a preprocessor to be analyzed for errors. There are a number of different errors that can arise within a constraint set consisting of the kinds of constraints discussed above. The constraint preprocessor checks for errors in three categories: redundant constraints, overcon-

Table 1 Examples of constraint sets with errors

Set	Constraints	Error Type
1	C1 X C = 0.0 C1 X C = 0.0	Redundant
2	C1 X C >= 0.0 C1 X C >= 1.0	Redundant
3	C1 X C >= 0.0 C1 X C <= 1.0	None
4	C1 X C = 0.0 C1 X C = 1.0	Overconstrained Component
5	C1 X C <= 0.0 C1 X C >= 1.0	Overconstrained Component
6	C1 X C = 0.0 C2 X C = 1.0 C1 X C = C2 X C	Unresolvable relative constraint

[2] This is why simulated annealing layout algorithms generally allow and penalize component overlap rather than forbidding it (Jepsen and Gelatt, 1983).



1.	Motor
2.	Motor bearing
3.	Gear 1
4.	Gear 2
5.	Gear 3
6.	Gear 4
7.	Bushing 1
8.	Bushing 2
9.	Shaft
10.	Journal bearing 1
11.	Journal bearing 2
12.	Chuck assemby
13.	Battery

Fig. 1 Modified Black & Decker® cordless power drill

strained components and unresolvable relative constraints. Each of these types of errors can lead to computational difficulties and inefficiencies. These categories are described below and are illustrated using simple examples in Table 1.

Redundant constraints cause additional computation time and occur when the feasible region resulting from two (or more) constraints are identical (i.e., repeated constraints) or if one feasible region is a subset of the other. Table 1 gives three examples of redundant constraints, where "C1" is a component name. (Recall that the letter "C" for ⟨LOCATION⟩ indicates that the constraint is applied at the center of the component.) Constraint set 1 is an example of a repeated constraint, and set 2 is an example of constraints where the feasible region of one constraint is completely entailed by the other. Note that set 3 is a valid constraint set because although the feasible regions overlap, one is not a subset of the other.

If constraints on a component result in disjoint feasible regions, the component is overconstrained and the constraints cannot be satisfied. Constraint sets 4 and 5 in Table 1 illustrate overconstrained components.

Unresolvable relative constraints occur when one or more relative constraints cannot be satisfied. Although this problem is similar to overconstrained components, the error is not due to a single component but to the relative location or orientation between components. In Table 1, the first two constraints in set 6 constrain the locations of components C1 and C2. Neither component alone is overconstrained, but the third constraint is a RLC that can not be satisfied. Attempting to satisfy these constraints using constraint propagation can lead to looping behavior in the propagation steps. This error can also occur along chains of multiple relative constraints.

In the current implementation, overconstrained components and redundant constraints are identified by comparing feasible regions for a component due to various constraints; if feasible regions for two constraints are disjoint, the component is overconstrained, and if one feasible region is a subset of another, one of the constraints is redundant. The ability to use this technique is related to the types of constraints and component geometries allowed in the current implementation; for other implementations of simulated annealing layout algorithms, a different approach to checking for overconstrained and redundant components may be necessary. Unresolvable relative constraints are identified by checking for the looping behavior during constraint propagation described previously.

Although the sets shown in the table use constraints on component locations as examples, the constraint preprocessor checks orientation constraints for the same kinds of errors. The advantage of this restricted implementation of orthogonal placement of components is that these types of errors in constraint sets can easily be identified.

Table 2 Constraints fixing motor location

Constraint	Type
MOTOR C X = 0.0	Equality GLC
MOTOR C Y = 0.0	Equality GLC
MOTOR C Z = 0.0	Equality GLC
MOTOR = X_Y_Z	Equality GOC

5 Example: Constrained Layout of the Cordless Power Drill

In this section, we apply the constrained layout algorithm to an actual engineering system. Figure 1 shows the components that comprise the power transmission of a Black & Decker® CD 1000 cordless power drill, at approximately .75× scale, modified by adding six batteries for further complexity. To allow rapid interference testing, the current implementation of the simulated annealing algorithm represents all objects as blocks or cylinders. Extensions by Kolli et al. (1996) based on this work have resulted in simulated annealing algorithms that allow representation of and interference testing for more general component geometries.

We represent the power drill using eighteen cylinders as shown in Fig. 1.[3] For this example, rather than the solution being driven by the constraints, the constraints are determined by the solution which is known *a priori*. Applying the simulated annealing algorithm to a problem with a known solution permits us to verify that it is indeed able to perform a layout task while satisfying a set of constraints on the components.

The spatial constraints in this example are representative of the types of constraints that are common in layout problems.

[3] Two small shafts—one linking the motor to gear 1, and the second on which the two bushings and gears 2 and 3 are mounted—are not included in the representation.

Table 3 Constraints for meshing gears 1 and 2

Constraint	Type
GEAR1 C Z = GEAR2 C Z	Equality RLC
GEAR1 C Y = GEAR2 C Y	Equality RLC
GEAR1 MAX X = GEAR2 MIN X	Equality RLC

First, using global constraints, the location of the motor is fixed at the origin as a reference, and its orientation is fixed. These constraints are shown in Table 2. Table 3 shows the set of constraints used to force adjacency between gears 1 and 2 so that they mesh. A similar set of constraints is used to ensure meshing between gears 3 and 4. The constraints that are used to give gear 1 and the motor a common axis of rotation are shown in Table 4. Analogous constraints are applied to give the motor bearing the same axis of rotation, to specify a second axis of rotation for the bushings and gears 2 and 3, and a third axis of rotation for gear 4, the bearings, the shaft and the chuck assembly.

Next, a group of RLCs is used to impose an ordering on components as shown (see Table 5): the motor bearing is placed directly to the right of the motor; gear 1 is located to the right of the motor bearing; gear 3 must be to the right of gear 2; bushings 1 and 2 are located directly to the left of gear 2 and directly to the right of gear 3, respectively; the chuck assembly is located to the right of journal bearing 2, which is positioned to the right of the shaft; journal bearing 1, which is placed to the left of the shaft, is in turn located to the right of gear 4.

Finally, the six batteries are constrained in order to fit into a handle which is located beneath the motor. The location and orientation of BATTERY1 is fixed beneath the motor using equality location and orientation constraints. A set of inequality relative location constraints is used to indicate that batteries 2, 4 and 6 should be somewhere to the left of batteries 1, 3 and 5, respectively, and that batteries 5 and 6 should be somewhere below batteries 3 and 4, which should in turn be below batteries 1 and 2. Because inequality constraints are used, the constraints can be satisfied by a configuration in which the batteries are far apart. To minimize the size of the handle, a set of proximity constraints is imposed to indicate desired proximities between the batteries. In all, over 90 constraints are applied to the various components.

Figure 2 shows the result of applying the simulated annealing layout algorithm to the constrained power drill components. The algorithm runs on a DEC Alpha 3000 workstation at a rate of over 1000 iterations per second. To illustrate the layout process using simulated annealing, a series of six layouts is shown beginning with the initial configuration in Fig. 2(a) (0 iterations) and progressing at 20,000 iteration intervals to the final design shown in Fig. 2(f) (approximately 100,000 iterations); the difference in apparent component size in the figures is due to differences in distances between components.

The initial configuration is generated by randomly selecting all coordinate values that are not constrained by equality constraints. The coordinate values that are restricted by equality constraints are selected by the algorithm so that those constraints are satisfied, as described in section 4.2. The initial design having satisfied equality constraints is shown in Fig. 2(a). Each of the 18 components has four variables (X, Y and Z coordinates and an orientation) for a total of 72 variables; because of equality constraints, the algorithm restricts some of the variables to single values.

Table 4 Constraints for common axis of rotation for motor and gear 1

Constraint	Type
MOTOR C X = GEAR1 C X	Equality RLC
MOTOR C Y = GEAR1 C Y	Equality RLC
GEAR1 = MOTOR	Equality ROC

Table 5 Constraints specifying spatial ordering

Constraint	Type
MTR_BRNG MIN Z = MOTOR MAX Z	Equality RLC
GEAR1 MIN Z >= MTR_BRNG MAX Z	Inequality RLC
GEAR3 MIN Z >= GEAR2 MAX Z	Inequality RLC
GEAR2 MIN Z = BUSHING1 MAX Z	Equality RLC
BUSHING2 MIN Z = GEAR3 MAX Z	Equality RLC
CHUCK MIN Z >= JRN_BRNG2 MAX Z	Inequality RLC
JRN_BRNG2 MIN Z >= SHAFT MAX Z	Inequality RLC
SHAFT MIN Z >= JRN_BRNG1 MAX Z	Inequality RLC
JRN_BRNG1 MIN Z >= GEAR4 MAX Z	Inequality RLC

As the optimization progresses, the effect of the various terms in the objective function can be observed. In Fig. 2(b) (20,000 iterations) the proximity constraints have brought the batteries close together and the inequality location constraints have affected the sequence of the power transmission components. By 60,000 iterations (Fig. 2(d)) the packing density objective has brought the power transmission components much closer together, and they are correctly ordered. At 100,000 iterations (Fig. 2(f)), component overlap has been eliminated, and all constraints are satisfied.

The algorithm is able to satisfy the set of spatial constraints while simultaneously optimizing the packing density of the components subject to those constraints. As a consequence of a finite minimum translation distance, perfect packings will rarely be achieved with this algorithm (Szykman and Cagan, 1995). This is illustrated in the final design by the minute gap that is visible between the shaft and the bearing to its left. For comparison, the size of that gap is less than 1 percent of the length of the motor and is smaller than the minimum translation distance in the move set. It should be noted that the minimum translation distance can be made arbitrarily small if gap sizes are found to be unacceptable for a particular problem.

To better assess the consistency of the algorithm, the problem was run 30 times. Solutions of equivalent quality to the one illustrated above were found 90 percent of the time. The remaining solutions converged to local optima where either some penalties on constraints remained or additional compactness was feasible; again the global optimum is not guaranteed but near-optimal solutions are consistently generated. On average the algorithm took approximately 130,000 iterations[4] (slightly over two minutes) to find the solutions with a standard deviation of about 36,400 iterations. The consistency of the performance of the algorithm indicates that the difficult problem of spatial constraint satisfaction for product layout can be solved using simulated annealing. The simulated annealing algorithm has also been applied to industry problems including the layout of electronic switching systems (Szykman, 1995). In contrast to the power drill problem where constraints specified the final layout configuration, with these industrial applications constraints only partially constrained the layout, permitting a wide variety of generated layouts.

The large number of iterations required is characteristic of the simulated annealing algorithm. Initially the algorithm performs a broad exploration of the space of possible solutions, followed by a "simmer" stage where it focuses in on the solution region, and finally it makes relatively minor perturbations to the solution to further optimize the configuration. Note that the number of iterations is not arbitrarily selected by the user

[4] Because of stricter convergence criteria, the algorithm ran on average about 265,000 iterations despite having found a solution in half that time. The performance can therefore be improved by adjusting the convergence criteria.

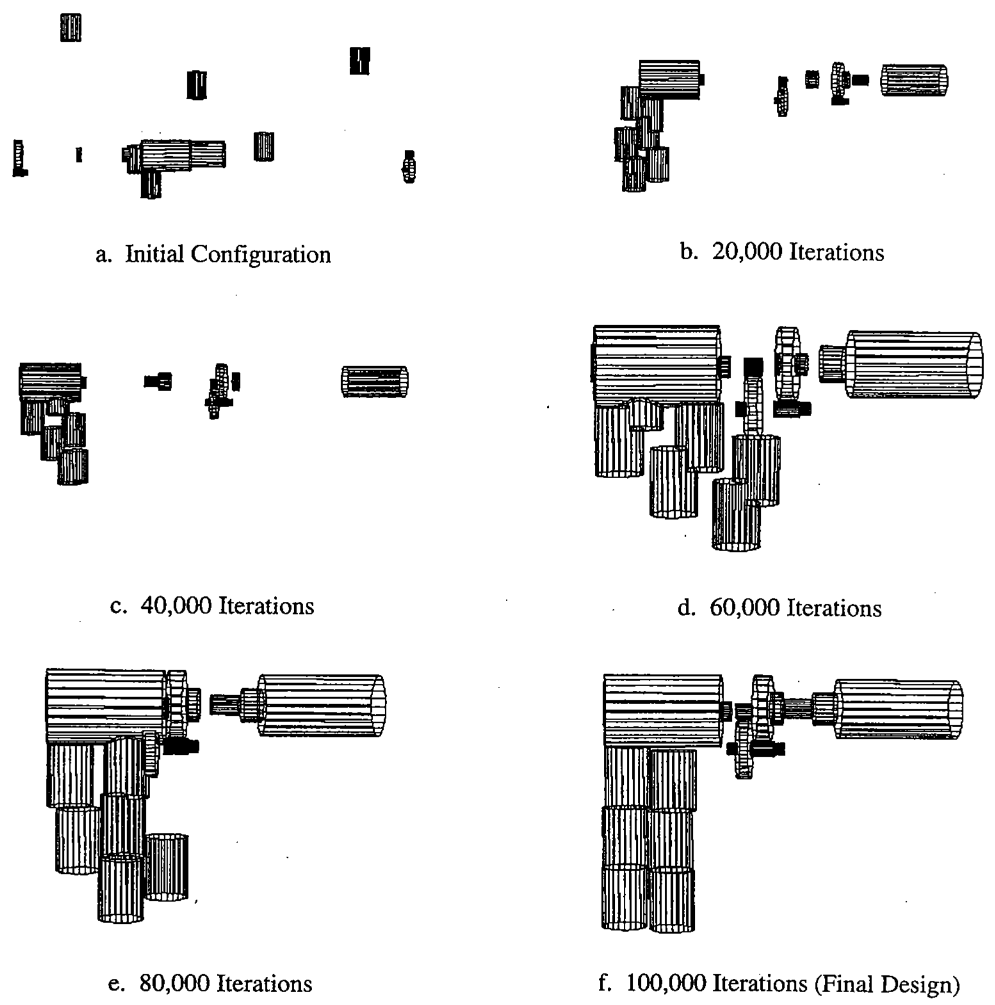


Fig. 2 Layout of power drill with batteries at 20,000 iteration intervals

but is determined by the adaptive annealing schedule mentioned in section 3. Although 130,000 iterations seems like significant computation, the algorithm performs that many iterations in under two and a half minutes. This is a relatively minor amount of time in comparison with the cost and effort required to physically model and manually place components, as is often done in industrial practice.

6 Concluding Remarks

This paper presents a computational algorithm that uses simulated annealing to solve three-dimensional component layout problems as characterized by three objectives: achieving high packing density, fitting components into a given container, and satisfying spatial constraints on components. This research builds on previous work on component packing to achieve a general layout algorithm by incorporating a method to satisfy spatial constraints. A language of those constraints on components that are characteristic of layout problems defines that set of operable constraints. With these constraints the designer specifies desired component proximities or restricts translation or rotation of components based on a global origin or set of coordinate axes, or relative to other component locations or orientations.

The layout algorithm presented in this paper is formulated independently of any domain information, and is applicable to general component layout problems such as electromechanical consumer products, heavy mechanical machinery, and transportation devices. This approach has been applied to a variety of other problems including layout of components in a chemical plant, layout of electronic switching systems (Szykman, 1995) as well as layout of a heat pump (Cagan et al., 1996).

Typically, design performance is a function of design objectives other than the generic objectives described in this paper. These objectives, as well as nonspatial constraints which are generally problem-dependent, can in theory be included as additional terms in the objective function once they have been represented. Because tradeoffs between design objectives may not be easily quantified, refining the objective function is a nontrivial, problem-specific task. Further, if the evaluation of such objectives is time-intensive, the runtime of the algorithm could become unacceptable. In its current form the algorithm is used to generate a variety of layouts of which the designer further modifies to satisfy additional performance constraints. Current research is examining ways to rapidly evaluate objectives which are traditionally time-intensive, such as with thermal analysis.

The current implementation has several limitations that affect the space of possible designs that can be generated. First, the shapes of components and containers are limited to rectangular blocks and cylinders; second, component rotations are limited to multiples of 90 deg; third, components are oriented along orthogonal axes. This restricted formulation allows rapid geometric interference testing and simplifies constraint checks. These limitations are independent of the simulated annealing algorithm; the current implementation can be augmented to eliminate them at the expense of additional computational resources. Current research is developing approaches to include continuous rotations and rapid evaluation of intersections of nonlinear surfaces (Kolli et al., 1996).

Acknowledgments

The authors would like to thank Rick Clark, Pratip Dastidar and Paul Weisser for their discussions regarding this work, as well as the National Science Foundation (under grant # DDM-9258090) and United Technologies (supplying matching funds) for supporting this work.

References

Cagan, J., Clark, R., Dastidar, P., Szykman, S., and Weisser, P., 1996, "HVAC CAD Layout Tools: A Case Study of University/Industry Collaboration," *Proceedings of the 1996 ASME Design Engineering Technical Conference and Computers in Engineering Conference: Design Theory and Methodology Conference, 96-DETC/DTM-1505,* Irvine, CA, August 19–22.

Coffman, E. G., Jr., Garey, M. R., and Johnson, D. S., 1984, "Approximation Algorithms for Bin-Packing—An Updated Survey," *Algorithm Design for Computer System Design,* G. Ausiello, M. Lucertini and P. Serafini, eds., Springer-Verlag, New York, pp. 49–106.

Cohn, J. M., Garrod, D. J., Rutenbar, R. A., and Carley, L. R., 1991, "KOAN/ANGRAM II: New Tools for Device-Level Analog Placement and Routing," *IEEE Journal of Solid-State Circuits,* Vol. 23, pp. 330–342.

Dowsland, K. A., and Dowsland, W. B., 1992, "Packing Problems," *European Journal of Operational Research,* Vol. 56, pp. 2–14.

Dyckhoff, H., 1990, "A Typology of Cutting and Packing Problems," *European Journal of Operational Research,* Vol. 44, pp. 145–159.

Flemming, U., Baykan, C. A., Coyne, R. F., and Fox, M. S., 1992, "Hierarchical Generate-and-Test vs. Constraint-Directed Search," *Artificial Intelligence in Design '92,* J. S. Gero, ed., Kluwer Academic Publishers, Boston, pp. 817–838.

Fujita, K., Akagi, S., and Hase, H., 1991, "Hybrid Approach to Plant Layout Design Using Constraint-Directed Search and an Optimization Technique," *Advances in Design Automation 1991: Proceedings of the 17th ASME Design Automation Conference,* Vol. 1, Miami, FL, September 22–25, pp. 131–138.

Huang, M. D., Romeo, F., and Sangiovanni-Vincentelli, A., 1986, "An Efficient General Cooling Schedule for Simulated Annealing," *ICCAD-86: IEEE International Conference on Computer-Aided Design—Digest of Technical Papers,* Santa Clara, CA, November 11–13, pp. 381–384.

Hustin, S., and Sangiovanni-Vincentelli, A., 1987, "TIM, a New Standard Cell Placement Program Based on the Simulated Annealing Algorithm," IEEE Physical Design Workshop on Placement and Floorplanning, Hilton Head, SC, April.

Jepsen, D. W., and Gelatt, C. D., Jr., 1983, "Macro Placement by Monte Carlo Annealing," *Proceedings of the IEEE International Conference on Computer Design,* November, pp. 495–498.

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

Kirkpatrick, S., Gelatt, C. D., Jr., and Vecchi, M. P., 1983, "Optimization by Simulated Annealing," *Science,* Vol. 220, No. 4598, pp. 671–679.

Kolli, A., Cagan, J., and Rutenbar, R. A., 1996, "Packing of Generic, Three Dimensional Components Based on Multi-Resolution Modeling," *Proceedings of the 1996 ASME Design Engineering Technical Conference and Computers in Engineering Conference: Design Automation Conference, 96-DETC/DAC-1479,* Irvine, CA, August 19–22.

Landon, M. D., and Balling, R. J., 1994, "Optimal Packaging of Complex Parametric Solids According to Mass Property Criteria," ASME Journal of Mechanical Design, Vol. 116, pp. 375–381.

Sandgren, E., and Dworak, T., 1988, "Part Layout Optimization Using a Quadtree Representation," *Advances in Design Automation 1988: Proceedings of the 14th ASME Design Automation Conference,* Kissimmee, FL, September 25–28, pp. 211–219.

Sechen, C., and Sangiovanni-Vincentelli, A., 1985, "The TimberWolf Placement and Routing Package," *IEEE Journal of Solid-State Circuits,* Vol. 20, No. 2, pp. 510–522.

Szykman, S., 1995, "Optimal Product Layout Using Simulated Annealing," Ph.D. dissertation, Carnegie Mellon University, Pittsburgh, PA.

Szykman, S., and Cagan, J., 1995, "A Simulated Annealing-Based Approach to Three-Dimensional Component Packing," ASME Journal of Mechanical Design, Vol. 117, No. 2(A), pp. 308–314.