Shape annealing solution to the constrained geometric knapsack problem

Jonathan Cagan

The paper introduces a technique called *shape annealing* as a solution to an extension of the knapsack problem which is referred to as the *constrained geometric knapsack problem* that is used for the layout or packing of parts, components, and configurations. *Shape annealing*, a variation of the simulated annealing stochastic optimization technique, incorporates *shape grammars* to dictate permissible item orientations in packing problems. Stochastic mutation based on a potential function creates a packing that converges toward the optimum. Results are acceptable but generally not provably optimal, and the algorithm runs in polynomial time and space complexity. The constrained geometric knapsack problem includes geometric constraints and reduces in zero dimensions to the ordinary knapsack problem. The problem has a wide variety of applications in design and manufacturing.

Keywords: layout and packing, simulated annealing, knapsack problem

Layout and packing problems find application in a variety of industries. These problems can be modelled by knapsack problems and have received wide attention in the literature as discussed in Reference 1. The problem, which occurs in layout, cutting stock, scheduling and capital budgeting contexts, is to pack as many of a set of items into a knapsack as possible, bounded by some scalar quantity such as weight, to maximize the knapsack value; each packed item has a specified value itself. The knapsack problem has been shown to be NP-hard. In this paper we address a more complicated problem, called the *constrained geometric knapsack problem*, where the boundary of the geometric space of the knapsack must not be violated and, further, the items of the knapsack must not overlap and must orient themselves to each other in a prespecified way. Such a problem may arise in layout problems where surface interactions of components are pertinent.

We propose a solution to the constrained geometric knapsack problem called *shape annealing*, originally introduced in the architecture literature by Cagan and Mitchell[2] to evaluate and control the generation of shapes. Shape annealing combines the concepts of the stochastic optimization technique of *simulated annealing*[3] with *shape grammars*[4] which specify relations between geometric shapes. The resulting algorithm runs in polynomial time and space complexity and produces maximal solutions* that are acceptable although not provably optimal.

The next section discusses simulated annealing and shape grammars. The shape annealing algorithm is then introduced. The constrained geometric knapsack problem is formally presented and the shape annealing solution is applied to a simple problem with an exponentially large number of possible layout solutions.

BACKGROUND

Simulated annealing

Simulated annealing is a stochastic optimization technique which has been demonstrated to solve continuous (see, for example, References 5–8) or ordered discrete (see, for example, References 3 and 9) optimization problems of fixed structure. Kirkpatrick *et al.*[3] developed the simulating annealing algorithm based on the Monte Carlo technique by Metropolis *et al.*[10], referred to as the *Metropolis algorithm*. The idea is analogous to the annealing of metals. A cooling schedule is defined giving a temperature reduction over a certain number of iterations. Temperature (T) is a potential function with no physical meaning; the variable is called *temperature* to maintain the analogy with metal annealing. At high temperatures the selection of a solution point is quite random while at lower temperatures the solution is more stable; the metal annealing analogy is that at high temperatures the molecules are in a highly random state while at lower temperatures they reach a stable minimum energy state.

Department of Mechanical Engineering, Carnegie Mellon University, Pittsburgh, PA 15213, USA
Paper received: 2 November 1992. Revised: 10 August 1993

*A maximal solution is considered one in which a local change will not improve the solution.

0010-4485/94/10/0763–08

With simulated annealing, a feasible solution, s, is randomly selected and the 'energy' (i.e. the objective function) at that state, E, is evaluated. A different feasible state s′ is then selected and the solution objective is evaluated to E'. For problem minimization, if $E' < E$, then s′ becomes the new solution state. If $E' \geq E$, then there is a probability based on the temperature that the new state will still be accepted. Acceptance is determined by the probability calculation*:

$$\Pr\{s'\} = e^{-[(E'-E)/ZT]}$$

where Z, a function of temperature T, is a normalization factor. A random number, r, between 0 and 1 is generated and compared with $\Pr\{s'\}$. If $r < \Pr\{s'\}$ then the new state is accepted; otherwise the old state is retained. The temperature (potential function) is reduced and the process continues until convergence is reached or the temperature reaches zero. Generally, the size of the mutation space is also reduced and asymptotes to zero. Typically, the algorithm is run for several iterations at a given temperature until equilibrium is reached or until a certain number of iterations has occurred. The temperature is then reduced by a fixed amount and the algorithm is again run until convergence or an iteration limit is achieved. When there is no accepted new solution at a given temperature the minimum is assumed to have been found.

Under rigorous mathematical conditions, it can be proven that, if *equilibrium* (i.e. convergence) is reached at each temperature and if the temperature is reduced slowly enough, then the algorithm is guaranteed to converge on the global optimum[11]. In practice these conditions are typically not met; simulated annealing is then used to search only for a good solution and the precise globally optimal solution is often sacrificed.

Shape grammars

Shape grammars were introduced in the architecture literature by Stiny[4] as a formalism for shape generation. A simple set of grammatical rules are defined that transform one shape into a different shape. Only modifications specified by shape rules are permitted. What is most interesting is that, from very simple shape grammars, quite elegant architectural layouts have been generated. Examples include villas in the style of Palladio[12], Mughul gardens[13], prairie houses in the style of Frank Lloyd Wright[14], Greek meander patterns[15], and suburban Queen Anne houses[16].

Stiny[4] defines *shapes* as limited arrangements of straight lines in a Cartesian coordinate system with real axes and an associated Euclidean metric. Boolean operations of union and difference are defined on these shapes, as well as the transformation properties of translation, rotation, reflection, scale, and composition. Finally, distinguishing information about an individual shape can be associated through *labels*. This results in an algebra of shapes and a grammar formalism for algorithms from which languages of shapes are derived.

A shape grammar has four components: a finite set of geometric shapes (S), a finite set of label symbols (L) which specify additional characteristics about the shape, a finite set of shape rules (R) which can manipulate the shape, and an initial shape (I) which can be manipulated to form a new shape. Thus, from I, R transforms the set (I, L) into a new set (S, L), and, in general, R transforms one set of shapes, (S, L), into another, $(S, L)'$, as

$$(S, L) \xrightarrow{R} (S, L)'$$

An example shape grammar, which will be employed as an example in the fifth section, is shown in *Figure 1*, taken from Reference 17. This grammar can transform a half hexagon into three different shapes specified by Rules 1, 2 and 3; Rule 4 is given for completeness and indicates termination of the grammar. From any one of the new shapes, the rules can be applied to a half hexagon within the shape to create a different shape. Given this language, a countably infinite number of shapes can be generated. The dots are labels indicating from which side a rule can be applied; thus this grammar produces snake-like, serially generated shapes. Note that one could imagine other possible shape rules to add to this set; however, only shapes that can be derived from the given rule set are permissible.

A shape grammar such as the one illustrated in *Figure 1* can be used to generate a countably infinite design space. This space can be modelled as a decision tree where each branch models the application of a shape rule. In this example, each node representing a current design can branch into one of three directions based on each of the three primary rules. The search of such a vast space and control of any generated solutions becomes a complicated (NP-hard) problem. The next section presents the shape annealing algorithm as an approach to generating, searching, and controlling such vast spaces and illustrates its power for layout problems.

SHAPE ANNEALING

Algorithm

As presented in Reference 2, shape annealing utilizes the simulated annealing algorithm to determine whether a

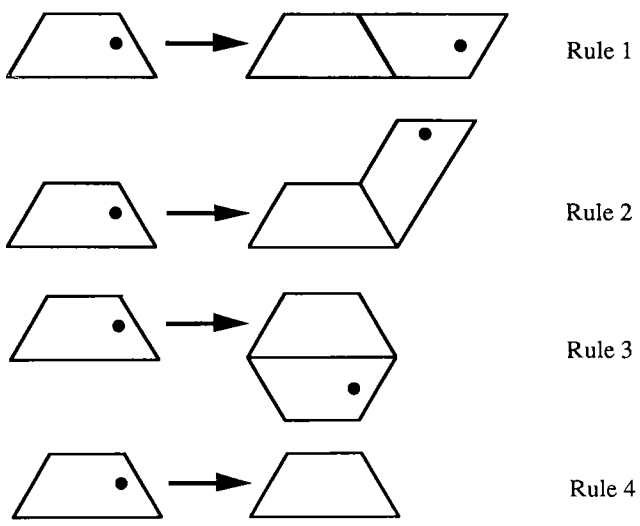


Figure 1 Example shape grammar [Taken from Reference 17.]

*Other probability calculations are possible; this one is used in the current algorithm.

randomly selected shape rule should be applied at a given configuration state (i.e. intermediate shape). Given a current configuration state, an eligible rule is selected and applied to that state. If the new design does not violate any constraints then it is sent to the Metropolis algorithm which compares the new state to the old state and, on the basis of the temperature function, determines whether to accept it or not. If the rule violates a constraint then the old state is maintained as the current state.

By only applying additive rules (i.e. those rules which add a new component onto the solution), the design may rapidly converge on an inferior solution because it can work itself into a state where no rule can be applied without a constraint violation. Shape annealing can backtrack out of these local solutions by reversing the previous rule. For every additive rule, shape annealing introduces a subtractive opposite which removes the component from the solution; if the subtractive rule (called *rule reversal*) is fired immediately following its companion additive rule then the effect of the additive rule is removed.

Shape annealing is a sort of Monte Carlo technique in that there is no continuum of solutions; however, the algorithm is different from the traditional Monte Carlo technique in that the parameters for randomness are controlled via the annealing schedule, and convergence is similarly controlled. In the original algorithm, temperature is reduced by a constant reduction factor at each iteration. Cagan and Reddy[18] demonstrate that the shape annealing algorithm is sensitive to the annealing schedule as well as the probability distribution of removal rules. They utilize a polynomial annealing function which gives a slower and smoother drop in temperature and doubles the probability that the most recent rule would be reversed. In that application, the packing factor increases by 15% from the original algorithm of Cagan and Mitchell. More sophisticated annealing schedules can be found in the VLSI literature[21] and can be adapted for improvement in algorithm efficiency.

The original shape annealing algorithm is given below. The maximum number of outer loop iterations, t, and the maximum number of inner loop iterations, m, need to be determined for the specific problem being solved on the basis of convergence criteria. The total number of loops will be at most $k=(t)(m)$.

```
Begin SHAPE_ANNEAL
  T=1.
  Define initial state;
  Evaluate state;
  While T>0 do
    success=0;
    For mutations=1 to m do
          /* at each temperature mutate m times
            until convergence or limit reached */
      Let temp_state=state;
      Generate rule;
      If (rule is applicable)
      Then begin
        apply rule to temp_state;
        if (verify constraints of temp_state)
        Then begin
          Evaluate temp_state;
          Test temp_state with Metropolis;
          If (accept)
          Then
            state=temp_state;
            success=success+1;
          End
        End
      End
      If (success>limit) then break;
    End
    If (success=0) then break;
                          /* no improvement ...
                              solution found */
    reduce T;
  End
End
```

Complexity

It is interesting to note that the size of the space of designs (the *state space*) is potentially infinite; the algorithm finds a way to generate a maximal packing from the exponentially large number of possible configurations with minimal storage requirements. At any stage the algorithm only stores the current configuration* of n items, O(n), and the shape grammar which is O(r) where r is the number of rules. Storage complexity is thus O($n+r$); for $n \gg r$, the storage complexity becomes O(n).

The worst-case time complexity for running the algorithm would have the algorithm run without convergence (k, and thus (t)(m), specified steps), where at each step one item is generated, O(1), and tested to guarantee no overlap, O($n-1$), and no violation of the b boundaries, O(b). An upper bound on this worst case considers the generation of the nth item at each iteration: O($k(n+b)$).

The best case would generate the completed solution in n iterations with one new item being generated and tested at each of the n iterations plus one temperature iteration (of m attempts to find a better solution) to test the solution:

$$O\left(\left(\sum_{i=1}^{n}(i+b)\right)+m\right)$$

The ratio of the worst case to the best case gives

$$O\left(\frac{k(n+b)}{\left(\sum_{i=1}^{n}(i+b)\right)+m}\right)$$

An upper bound on this ratio with $n(n+b) \gg m$ would be

$$O\left(\frac{k(n+b)}{n(n+b)}\right)=O\left(\frac{k}{n}\right)$$

Thus the ratio and all bounds are polynomial in n. For large k, the bounds and ratio are dominated by the number of iterations.

The next section introduces the constrained geometric knapsack problem as a model of factory and part layout problems. Shape annealing is proposed as a technique to generate good solutions to this NP-hard problem in polynomial time.

*O(n) means 'order n'.

CONSTRAINED GEOMETRIC KNAPSACK PROBLEM

Knapsack problem

Knapsack problems have received wide attention in the literature. Given decision variables (X_i) of available items, each with its own weight (W_i) and value (V_i), that can be instantiated as many times as necessary so that $N(X_i)$ represents the integer number of times each item is used, the knapsack problem is to maximize the value in the knapsack such that the weight of the n objects does not violate its capacity. Formally, the knapsack problem is defined as follows.

Maximize

$$\sum_{k=1}^{n} V_k N(X_k)$$

subject to

$$\sum_{k=1}^{n} W_k N(X_k) \leqslant W_{\max} \qquad N(X_k) \in Z^+$$

where Z^+ is the set of positive integers. The knapsack problem has been shown to be NP-hard[1]. For small n, the problem can be directly solved with integer programming techniques. For large n, however, bounding techniques and approximation techniques have been used to determine a solution. Martello and Toth[1] present a thorough discussion of the class of knapsack problems and both exact and approximate solution algorithms.

Geometric knapsack problem

The knapsack problem considers only the weight and value components of the items, which are important aspects in scheduling and resource allocation problems. A more difficult problem relates to *geometry*. Suppose that the available *space* in which the items can be placed is fixed, as are the dimensions of the knapsack. Also, suppose that the items cannot overlap (i.e. occupy the same space). This more complex problem is called the *geometric knapsack problem*. Let X_{k_p} indicate a *specific* item (p) in the class of items X_k. Also, let S_{total} be the space (location and volume) bounding the knapsack in $\mathbb{R}^3$, and $S(X_{k_p})$ be the space of the k_pth item in $\mathbb{R}^3$. The *geometric knapsack problem* is formulated as follows.

Maximize

$$\sum_{k=1}^{n} V_k N(X_k)$$

subject to

$$\sum_{k=1}^{n} W_k N(X_k) \leqslant W_{\max}$$

$$N(X_k) \in Z^+$$

$$S(X_{k_p}) \cap S(X_{j_r}) = \varnothing \qquad \forall\, j_r \neq k_p$$

$$S(X_{k_p}) \subseteq S_{\text{total}} \qquad \forall\, k_p$$

The last two constraints respectively state that any one component cannot occupy the same space as any other component and that each component must fall within the space of the knapsack.

This problem, in 3D, can be seen as a problem where objects are placed in a container, and the items settle into a tight packing. For 2D, we have a sort of 'cookie-cutter' problem where cookies of a certain type each have their own utility, the cookie dough is rolled flat and cookies are cut to get the most value from the given dough (the closest planar packing is desired). For 1D, the problem is like laying out pieces of wire on a given line. Finally, for 0D, the problem reduces to the knapsack problem because the bounded space becomes irrelevant. Thus the geometric problem is harder to solve than the traditional knapsack problem and remains NP-hard.

Constrained geometric knapsack problem

There is an even more specific problem than the geometric knapsack problem that considers geometric orientation. Suppose that, in addition to the *space* being fixed, there is some criteria on the items based on their *orientation*. In particular, the items must fit into the knapsack with a certain orientation to each other. This may be required because of magnetic properties of the objects, relative surface roughness, accessibility between the items, or even aesthetic reasons, among others. This more complex problem is called the *constrained geometric knapsack problem*.

Let O be a function relating the *orientation* of item j to that of item k via *orientation relation type* O_i. Let $L(X_q)$ be the *label* of X_q indicating the unique characteristics of that class of items. Also, let O_{ijq} be a description of allowable orientations O_i between the classes of items X_j and items X_q; O_{ijq} is a variable, one of whose discrete values may be selected as the desired orientation. There are a prescribed set of orientation relation types between each class of items, and a prescribed set of classes of items available. Finally, let O_{jq} be the orientation relation function between items in the classes of items X_j and items X_q which chooses one of the orientations O_{ijq}. The *constrained geometric knapsack problem* is formulated as follows.

Maximize

$$\sum_{k=1}^{n} V_k N(X_k)$$

subject to

$$\sum_{k=1}^{n} W_k N(X_k) \leqslant W_{\max}$$

$$N(X_k) \in Z^+$$

$$S(X_{k_p}) \cap S(X_{j_r}) = \varnothing \qquad \forall\, j_r \neq k_p$$

$$S(X_{k_p}) \subseteq S_{\text{total}} \qquad \forall\, k_p$$

$$O_{ijq} = O[L(X_j), L(X_q), O_i]$$

$$O_{jq} = O_{1jq} \vee O_{2jq} \vee \ldots \vee O_{ijq}$$

The last two constraints state that there are i predefined orientations between classes of items X_j and items X_q, and that the orientation between items in the two classes is specified by those orientations (the symbol $\vee$ indicates the disjunctive OR). In practice the constraints need to specify enough geometric information to model the relative orientations. The fifth section will give an example of these orientations as related to the example grammar illustrated in *Figure 1*.

Applications of the constrained geometric knapsack problem can be found in layout problems such as material pattern layout, sheet metal stamping, product packaging, and cutting stock where orientations between items are imposed (for example because of material grain direction). Other solutions to layout problems without geometric interactions between parts are discussed in References 1, 19 and 20. In this reformulated problem, if O_{ijq} is specified such that all possible orientations are included, then the problem reduces to the geometric knapsack problem; if all possible orientations are included and in addition S_{total} is not specified, then the problem reduces to the traditional knapsack problem. However, the heuristic techniques presented in the literature for the knapsack problem are ineffective for the extended problem because of the added geometric complexity.

We propose to represent knapsack items as shapes with the properties of a shape as described by Stiny[4]. The legal orientations of those shapes with respect to each other for the constrained geometric knapsack problem are represented by the shape grammar. Utilizing the shape annealing algorithm, a good solution to this problem that satisfies all geometric constraints can be generated in polynomial time with minimal storage capacity. The algorithm is generic in that any shape grammar and any set of boundaries can be used in the same implementation. Note that in the constrained geometric knapsack problem there is no specified order for how the items fit in the knapsack, but each fits within a given set of orientations without occupying the same space. With shape annealing, however, order in laying out the solution is specified, possibly restricting the solution, but guaranteeing that item orientation is valid.

Through this approach to layout design, a simple geometric representation is used to generate intricate constrained geometric knapsack configurations. This is important; shape annealing provides an environment in which an elementary shape grammar that describes a complete language of desired shape is specified from which the problem solution will then be generated with a simple algorithm. Only the local grammar information is required; the maximal solution will be found without the need for the designer to understand the global intricacies.

EXAMPLE

As an example application of the shape annealing solution to the constrained geometric knapsack problem, consider the 2D problem of packing as many half hexagons as possible into a predefined volume of fixed dimensions such that the items do not overlap. Further, consider the problem that the half hexagons, each of equal value, can only orient themselves in one of three different configurations. These configurations are specified by the shape grammar illustrated in *Figure 1*. This shape grammar can be represented by orientation functions, O^*, which give all necessary geometric information about xy coordinate and angle, θ, locations. Here, there is only one class of items ($X_k = X_1$) and there are three orientations for Rules 1, 2 and 3. In reference to the orientations described in the fourth section, these orientations are respectively called O_{111}, O_{211}, O_{311}, where items of class X_1 can be related to each other by orientation relation type O_1, O_2, O_3. We formulate this grammar on the basis of the midline of the half hexagons:

$$O_{111} = O^*(1, 0., \mathit{width} * \cos(\theta), \text{width} * \sin(\theta), -1, 0.)$$

$$O_{211} = O^*(1, -\text{sign} * 60., \text{width} * \cos(\theta), \text{width} * \sin(\theta), 1, 0.)$$

$$O_{311} = O^*(1, 0., \mathit{height} * \sin(\theta), \text{height} * \cos(\theta), -1, 0.)$$

where the 6-tuple orientation function O^* indicates for each rule a sign orientation change prior to the xy adjustment, an angle update prior to the xy adjustment, the x adjustment, the y adjustment, a sign orientation change after the xy adjustment, and an angle update after the xy adjustment. Width indicates the length of the midline and height indicates the short distance of the half hexagon. For example, $O^*(1, -\text{sign} * 60., \text{width} * \cos(\theta), \text{width} * \sin(\theta), 1, 0.)$ for rule O_{211} means to multiply a sign variable by 1 and add $-\text{sign} * 60°$ to the orientation angle; next adjust the x position by adding width times $\cos(\theta)$ and the y position by adding width times $\sin(\theta)$; when finished, multiply the sign variable by 1 and make no adjustments to the angle orientation (see Rule 2 in *Figure 1*). This keeps track of x and y positions, angle, and an orientation sign. *Figure 2a* shows a half hexagon with indications of sign direction, angle orientation, (x, y) positioning, height, and width. *Figure 2b* shows the same item but with the opposite sign orientation (i.e. $-$ sign). Note that in this example the only label information required is the location of where the next rule can be applied, which is included in the x and y positions, angle, and orientation sign.

Given this language, a countably infinite number of shapes can be generated. Note the difference between this constrained geometric knapsack problem and a traditional knapsack problem: geometry adds a level of complexity which makes the standard approximations ineffective. In this particular problem, the values of the items (V_i) are all 1 and the weight constraint does not become an issue; note that weight, geometry, and any other problem constraints are readily incorporated into

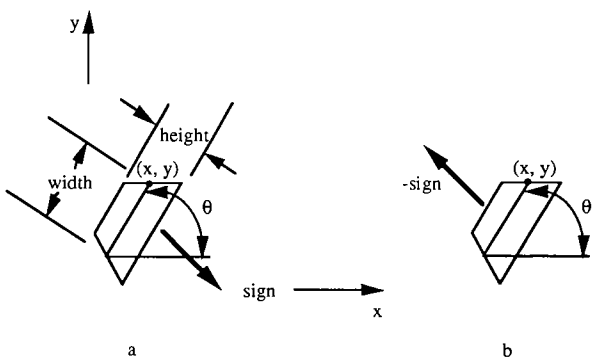


Figure 2 Description of half hexagon

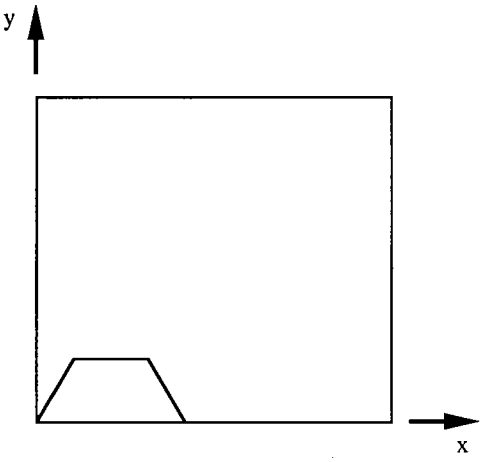


Figure 3 Description of possible boundary space with initial half hexagon

the problem formulation but do not affect the algorithm. Also, there is only one class of items ($X_k = X_1$) and there are three orientations for the items with respect to each other. Rule 4 given for completeness but is actually only applied after the terminating solution is found.

We formulate this problem as an optimization problem with geometric constraints based on the midline of the half hexagons as follows.

Maximize

$$\sum_{p=1}^{n} N(X_{1_p})$$

subject to

$$N(X_1) \in Z^+$$

$$\text{midline}(X_{1_p}) \cap \text{midline}(X_{1_r}) = \varnothing \qquad \forall\, r \neq p$$

$$\text{midline}(X_{1_p}) \cap \text{boundary}(S_{\text{total}}) = \varnothing$$

$$O_{111} = O^*(1,\ 0.,\ \text{width} * \cos(\theta),\ \text{width} * \sin(\theta),\ -1,\ 0.)$$

$$O_{211} = O^*(1,\ -\text{sign} * 60.,\ \text{width} * \cos(\theta),\ \text{width} * \sin(\theta),\ 1,\ 0.)$$

$$O_{311} = O^*(1,\ 0.,\ \text{height} * \sin(\theta),\ \text{height} * \cos(\theta),\ -1,\ 0.)$$

$$O_{11} = O_{111} \vee O_{211} \vee O_{311}$$

where O_{11} is an orientation relation function which specifies that items in class X_1 will orient themselves via one of the three orientations O_{q11}. In addition, the items cannot overlap as specified by midline(X_{1_p}) as the location of the midline of the half hexagon X_{1_p}. Finally, the items must fit, via midline(X_{1_p}), within the specified boundary, S_{total}, by checking the intersection of each midline with the boundary surfaces†; in this example we assume the boundary to consist of any 2D closed surface. *Figure 3* shows a rectangular boundary space (S_{total}) and the xy coordinate system.

For this example, all spaces S_{total} are considered to be of a constant area (25 square units) and the half hexagon has a long base of one unit and a short base of one-half of a unit. Typical results of the shape annealing algorithm for various 2D spaces are shown in *Figures 4–6*. The fill pattern indicates which rule was applied: the light fill indicates Rule 1, the back fill indicates Rule 2, and the medium fill indicates Rule 3. For *Figures 4* and *5*, the packing starts with a half hexagon (I) in the bottom left corner as illustrated in *Figure 3*. For *Figure 6* the packing starts toward the centre of the circle.

The quality of the solutions varies. One half hexagon has an area of 0.325 square units. By area alone a possible 76 half hexagons can fit inside the 25 square unit area; however, that maximum number can never be reached in these boundaries owing to the requirements of the grammar. The algorithm, with the current annealing schedule, averages 39 half hexagons (51% coverage); the best solutions generated contain around 55 half hexagons (72% coverage). The better the annealing schedule (such as that given in Reference 21), the better the expected solution coverage.

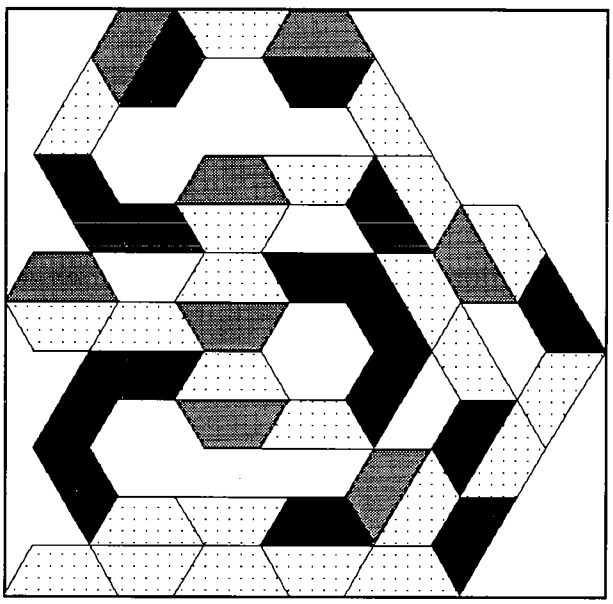


Figure 4 Resulting packing for 5 × 5 square of 47 half hexagons

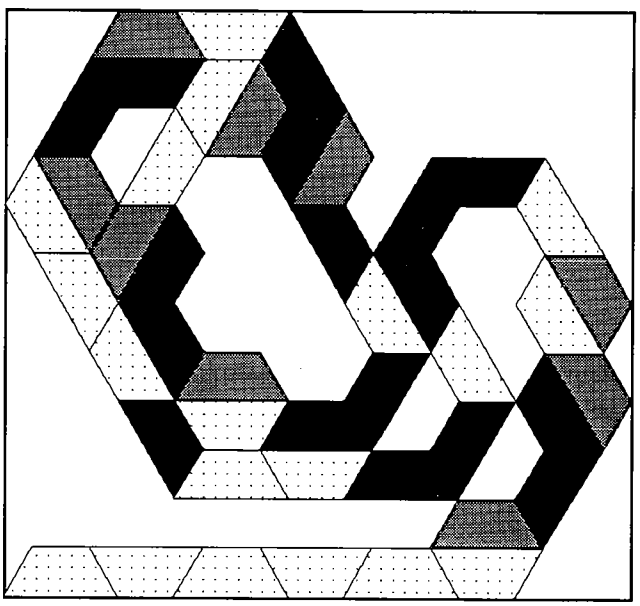


Figure 5 Resulting packing for 5 × 5 square of 45 half hexagons

†The endpoints of the midlines are permitted to intersect the boundary of S_{total}. Boundaries in the examples are shifted to compensate for these intersections. The items are assumed to be located within, and not outside, the boundary.

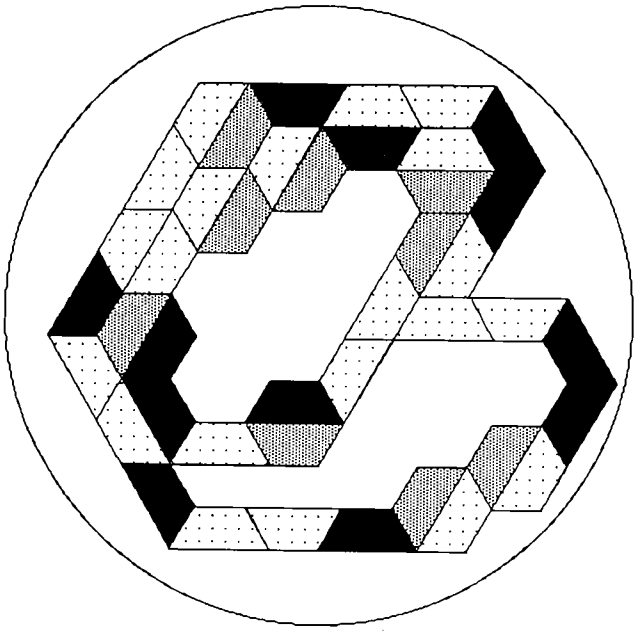


Figure 6 Resulting packing for circle, of 42 half hexagons, of equivalent area to figures in *Figures 4* and *5*

IMPLEMENTATION

Implementation of the shape algorithm is done in C on a Mac IIci and follows the pseudocode in the SHAPE-ANNEAL algorithm. The specific implementation of the half hexagon example models the half hexagons as midline segments. Each time an item is added to the layout the segment is added onto the point that models the end of the current layout and then the new layout is verified for constraint satisfaction. This requires an algorithm for line segment intersection that checks for intersection of the new segment with the region boundaries and all the other segments already laid out; intersection implies overlap of the boundary or other components.

In the example, the algorithm runs for up to 1000 temperature iterations with a maximum of 200 attempted rule applications at each temperature (at 30 successful applications the algorithm moves to the next temperature). For a solution with 40 half hexagons, the ratio of worst-case to best-case time is O((200 000 steps)/(40 steps)) = O(5000). However, the algorithm actually runs in an average 13 000 steps (a factor of 325 times worse than the best case).

CONCLUSIONS

We have introduced a method called *shape annealing* which combines the formalism of shape grammars with the stochastic optimization algorithm of simulated annealing. Shape annealing offers a powerful technique for solving the constrained geometric knapsack problem which restricts the traditional knapsack problem with space and orientation constraints. Shape annealing solutions are acceptable although generally not provably optimal.

Shape annealing, as presented here, generates serially configured solutions to the constrained geometric knapsack problem in polynomial time and space complexity in terms of the number of items; the time is essentially dependent on the temperature function (total number of iterations) and number of rules applied. The algorithm is computationally efficient in storage space. There is no need to maintain a trace of generated rules; only the current state and proposed modification at each step need to be stored. The algorithm itself will backtrack out of a configuration if required. Optimal solutions are not guaranteed in practice; rather, a maximal solution is consistently generated with a general method and arbitrarily complicated shape grammars. Future research must perform analyses to determine how *good* the shape annealing solutions are. Further, we believe a more sophisticated annealing schedule would produce better solutions.

The use of shapes to model items which must be laid out and shape grammars to model their orientation makes for a general algorithm and simple solution approach. We have illustrated the application to the constrained geometric knapsack problem in 2D; however, in theory, the algorithm is equally valid for 3D. Shape annealing can solve many traditional and nontraditional layout problems such as the tiling problem, as well as factory, process, and product layout. If the objective function is modified to pack a fixed number of items on the basis of geometric constraints, as represented through the grammar, then the approach is applicable to the layout of machines; if the objective function is modified to optimize the performance of the solution configuration, then design of mechanical structures is feasible through the method (see References 22 and 23 for a description of this work in progress).

ACKNOWLEDGEMENTS

The authors would like to thank Steve Cosares, Bill Mitchell, and Linda Schmidt for their important discussions about this work and manuscript. The author is grateful to the US National Science Foundation for supporting this work under grant DDM-9258090.

REFERENCES

1 Martello, S and Toth, P *Knapsack Problems – Algorithms and Computer Implementations* John Wiley, USA (1990)
2 Cagan, J and Mitchell, W J 'Optimally directed shape generation by shape annealing' *Environ. & Planning B* Vol 20 (1993) pp 5–12
3 Kirkpatrick, S, Gelatt Jr, C D and Vecchi, M P 'Optimization by simulated annealing' *Science* Vol 220 No 4598 (1983) pp 671–679
4 Stiny, G 'Introduction to shape and shape grammars' *Environ. & Planning B* Vol 7 (1980) pp 343–351
5 Jain, P, Fenyes, P and Richter, R 'Optimal blank nesting using simulated annealing' *Trans. ASME: J. Mech. Des.* Vol 114 (1992) pp 160–165
6 van Laarhoven, P J M and Aarts, E H L *Simulated Annealing: Theory and Applications,* Reidel, USA (1987)
7 Cagan, J and Kurfess, T R 'Optimal tolerance allocation over multiple manufacturing alternatives' *Adv. Des. Autom.* DE-Vol 44-2 (1992) pp 165–172
8 Cohn, J M, Garrod, D J, Rutenbar, R A and Carley, L R 'KOAN/ANAGRAM II: new tools for device-level analog placement and routing' *IEEE J. Solid State Circuits* Vol 26 No 3 (1991) pp 330–342
9 Jain, P and Agogino, A M 'Theory of design: an optimization perspective' *Mech. Mach. Theor.* Vol 25 No 3 (1990) pp 287–303
10 Metropolis, N, Rosenbluth, A, Rosenbluth, M, Teller, A and Teller, E 'Equations of state calculations by fast computing machines' *J. Chem. Phys.* Vol 21 (1953) pp 1087–1091

11 Lundy, M and Mees, A 'Convergence of an annealing algorithm' *Math. Program.* Vol 34 (1986) pp 111–124

12 Stiny, G and Mitchell, W J 'The Palladian grammar' *Environ. & Planning B* Vol 5 (1978) pp 5–18

13 Stiny, G and Mitchell, W J 'The grammar of paradise: on the generation of Mughul gardens' *Environ. & Planning B* Vol 7 (1980) pp 209–226

14 Koning, H and Eizenberg, J 'The language of the prairie: Frank Lloyd Wright's prairie houses' *Environ. & Planning B* Vol 8 (1981) pp 295–323

15 Knight, T W 'Transformations of the meander motif on Greek geometric pottery' *Des. Comput.* Vol 1 (1986) pp 29–67

16 Flemming, U 'More than the sum of the parts: the grammar of Queen Anne houses' *Environ. & Planning B* Vol 14 (1987) pp 323–350

17 Mitchell, W J *The Logic of Architecture* MIT Press, USA (1990) p 143

18 Cagan, J and Reddy, G 'An improved shape annealing algorithm for optimally directed shape generation' *in* Gero, J S (Ed.) *Artificial Intelligence in Design '92* Kluwer, Netherlands (1992) pp 307–324

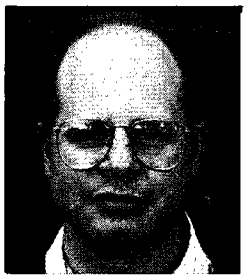
19 Mount, D M and Silverman, R 'Packing and covering the plane with translates of a complex polygon' *J. Algorithms* Vol 11 (1990) pp 564–580

20 Yanasse, H H, Zinober, A I and Harris, R G 'Two-dimensional cutting stock with multiple stock sizes' *J. Oper. Res. Soc.* Vol 42 No 8 (1991) pp 673–683

21 Huang, M D, Romeo, F and Sangiovanni-Vincentilli, A 'An efficient cooling schedule for simulated annealing' *Proc. 1986 IEEE Int. Conf. CAD IEEE* (1986) pp 381–384

22 Reddy, G and Cagan, J 'Optimally directed truss topology generation using shape annealing' *Adv. Des. Autom.* DE-Vol 65-1 (1993) pp 749–759

23 Szykman, S and Cagan, J 'Automated generation of optimally directed three dimensional component layouts' *Adv. Des. Autom.* DE-Vol 65-1 (1993) pp 527–537



Jonathan Cagan received a BS and an MS in mechanical engineering from the University of Rochester, USA, in 1983 and 1985, and a PhD from the University of California at Berkeley, USA, in 1990. He is currently an assistant professor of mechanical engineering at Carnegie Mellon University, USA. He has been awarded the US National Science Foundation's Young Investigator Award. Dr Cagan has significant industrial experience and is a licenced Professional Engineer in the states of Pennsylvania and California, USA. His research interests include conceptual design, design layout, quality and robustness, and design optimization.