The Constrained Geometric Knapsack Problem and its Shape Annealing Solution

Jonathan Cagan

EDRC 24-86-92

The Constrained Geometric Knapsack Problem and its Shape Annealing Solution

by

Jonathan Cagan

Department of Mechanical Engineering
Carnegie Mellon University
Pittsburgh, PA 15213

Phone: (412) 268-3713
E-mail (internet): jcag+@andrew.cmu.edu

Abstract

This paper introduces a technique called *shape annealing* as a solution to an extension of the knapsack problem which we refer to as the *constrained geometric knapsack problem.* The constrained geometric knapsack problem includes geometric constraints and reduces in zero dimensions to the ordinary knapsack problem. *Shape annealing*, a variation of the simulated annealing stochastic optimization technique, is introduced to produce packing solutions. Shape annealing incorporates *shape grammars* to dictate permissible item orientations. Stochastic mutation based on a potential function creates a packing which converges toward the optimum. Results are generally sub-optimal although acceptable and the algorithm runs in polynomial time and space complexity.

Introduction

Knapsack problems have received wide attention in the literature as discussed in Martello and Toth [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 pre-specified 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 solutions which are sub-optimal but acceptable.

The following section will discuss the constrained geometric knapsack problem. Next

simulated annealing and shape grammars are discussed. The shape annealing algorithm is then introduced and applied to a simple geometric knapsack problem with an exponentially large number of possible solutions.

1. The Constrained Geometric 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), and each which can be instantiated as many times as necessary, 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:

$$\begin{aligned} \text{max:} \quad & \sum_{k=1}^{n} V_k X_k, \\ \text{s.t.:} \quad & \sum_{k=1}^{n} W_k X_k \leq W_{max}, \\ & X_k \in Z^+, \end{aligned}$$

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 programing 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.

The knapsack problem considers only the weight and value components of the items, important aspects in scheduling and resource allocation problems. A more difficult problem evolves by consideration of *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 bounding the knapsack in $\Re^3$, and $S(X_{k_p})$ be the space of the k_p^{th} item in $\Re^3$. The *geometric knapsack problem* is formulated as:

$$\max: \sum_{k=1}^{n} V_k X_k,$$

$$\text{s.t.:} \sum_{k=1}^{n} W_k X_k \leq W_{max},$$

$$X_k \in Z^+,$$

$$S(X_{k_p}) \not\subset S(X_{j_r})\forall j_r \neq k_q,$$

$$\{S(X_{k_p})\}\forall k_p \subseteq S_{total}.$$

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 three dimensions, can be seen as a problem where objects are placed in a container, and the items settle into a tight packing. For two dimensions, 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 of the dough (the closest planar packing is desired). For one dimension, the problem is like laying out pieces of wire on a given line. Finally, for zero dimensions the problem reduces to the knapsack problem because the bounded space becomes irrelevant. Thus the geometric problem is much harder to solve than the traditional knapsack problem and thus certainly remains NP-hard.

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, 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 orientation 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_{ij} 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:

$$\text{max:} \sum_{k=1}^{n} V_k X_k,$$

$$\text{s.t.:} \sum_{k=1}^{n} W_k X_k \leq W_{max},$$

$$X_k \in Z^+,$$

$$S(X_{k_p}) \propto S(X_{j_r}) \forall j_r \neq k_q,$$

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

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

$$O_{ij} = O_{ijq}.$$

The last two constraints state that there are i pre-defined 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. In practice the constraints need to specify enough geometric information to model the relative orientations. Also, it should be noted that the last constraint will actually be a disjunction of orientations O_{ijq}. Section 3 will give an example of these orientations.

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 they do not take geometry into account. We introduce a technique called *shape annealing* to determine a good solution to this constrained problem. Shape annealing is an extension of the simulated annealing stochastic optimization technique of Kirkpatrick, *et al.*, [3]. Instead of using simulated annealing to determine values for variables in a pre-defined space, shape annealing actually *generates* the configuration solution. To do this, the X_i items are represented as *shapes* and their orientation relative to each other is specified by *shape grammars* (section 3) as introduced by Stiny [4].

Shape annealing is shown to produce a good solution to this combinatoric problem based on a time and space complexity polynomial in the number of items, n. Shape

annealing was originally presented by Cagan and Mitchell [2] in the architecture literature to propose a method to evaluate and control the generation of shapes. In this paper we show how the algorithm can be used to solve the constrained geometric knapsack problem.

2. Simulated Annealing

Simulated annealing is a stochastic optimization technique which has been demonstrated to solve continuous (*e.g.*, [5-7]) or ordered discrete (*e.g.*, [3, 8]) optimization problems of fixed structure. Kirkpatrick, *et al.*, [3] developed the simulating annealing algorithm based on the Monte-Carlo technique by Metropolis, *et al.*, [9], 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 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 at a highly random state while at lower temperatures they reach a stable minimum energy state.

With simulated annealing, a feasible solution, s_1, is randomly selected and the "energy" (*i.e.*, the objective) at that state, E_{s1}, is evaluated. A different feasible state, s_2, is then selected and the solution objective is evaluated to E_{s2}. If $E_{s2} > E_{s1}$, then s_2 becomes the new solution state. If $E_{s2} \leq E_{s1}$, then there is a probability based on the temperature that the new state will be accepted anyhow. Acceptance is determined by the probability calculation:

$$\Pr\{E_{s2}\} = \frac{e^{-\frac{E_{s2}}{T}}}{Z(T)},$$

where Z(T) is a normalization factor. A random number, r, between 0 and 1 is generated and compared with $\Pr\{E_{s2}\}$. If $r < \Pr\{E_{s2}\}$ then the new state is accepted anyhow; 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. In this case, 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 determine the global optimum [10]. Because sufficient time cannot be guaranteed for large problems, simulated annealing is used to search only for a good solution and the precise globally optimal solution is often sacrificed.

Generally 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 has been found.

3. 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 which map 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 [11], Mughul gardens [12], prairie houses in the style of Frank Lloyd Wright [13], Greek meander patterns [14], and suburban Queen Anne Houses [15].

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 important issue in shapes and their grammars is that of *emergence*; by representing a shape as a set of maximal lines, new shapes that were not originally modelled can emerge from the intended shape description. Issues of emergence are not relevant to the current application and will not be pursued further. It should be realized, however, that from a formal description of shape, this issue is important.

An example shape grammar, which will be employed as an example in section 5, is shown in Figure 1 from Mitchell [16]. 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.

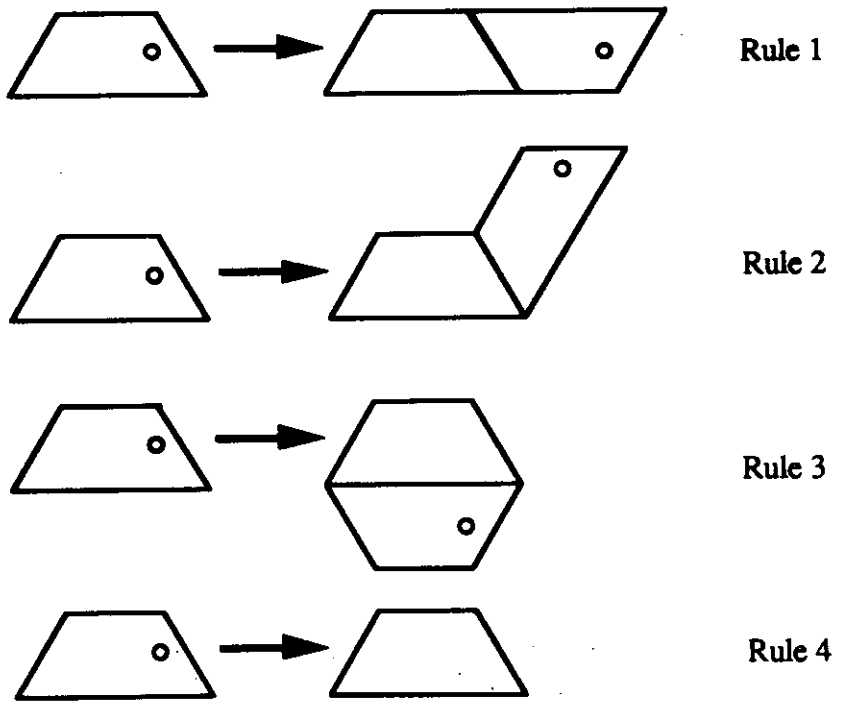

Figure 1 Example Shape Grammar (from Mitchell [16]).

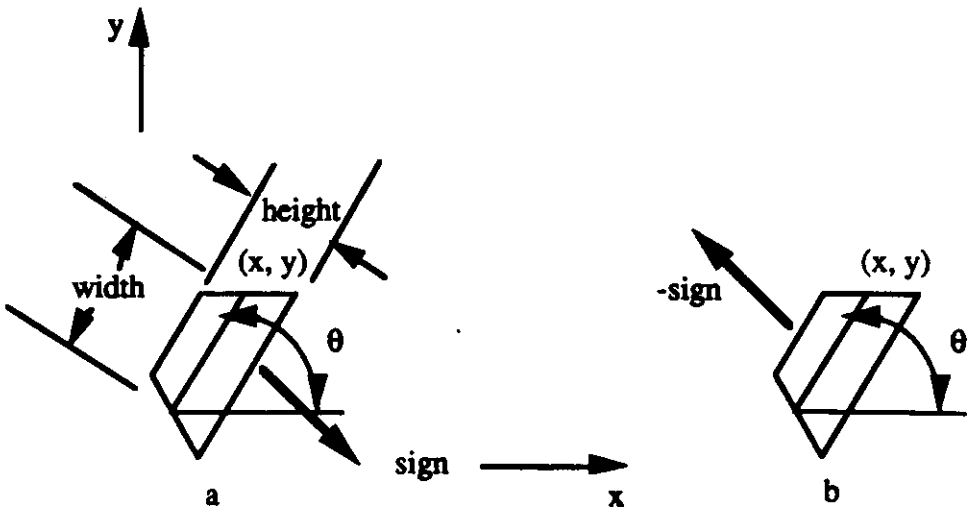

Figure 2 Description of half-hexagon.

This shape grammar can be represented by orientation functions, O^*, which give all necessary geometric information about x-y coordinate and angle, θ, locations. Here, there is only one class of elements ($X_k = X_1$) and there are three orientations for Rules 1, 2 and 3. In reference to the orientations described in section 1, these orientations are respectively called O_{111}, O_{112}, O_{113}, 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 based on the midline of the half-hexagons:

O_{111} = O^*(1, 0., width*cos(θ), width*sin(θ), -1, 0.),
O_{112} = O^*(1, -sign*60., width*cos(θ), width*sin(θ), 1, 0.),
O_{113} = O^*(1, 0., height*sin(θ), height*cos(θ), -1, 0.),

where the 6-tuple orientation function O^* indicates for each rule: a sign orientation change prior to the x-y adjustment, an angle update prior to the x-y adjustment, the x-adjustment, the y-adjustment, a sign orientation change after the x-y adjustment, and an angle update after the x-y adjustment. Width indicates the length of the mid-line and height indicates the short distance of the half-hexagon. For example, O^*(1, -sign*60., width*cos(θ), width*sin(θ), 1, 0.) for rule O_{112} means to multiply a sign variable by 1 and add -sign*60° to the orientation angle; next adjust the x-position by width times cos(θ) and the y-position by width times sin(θ); when finished, multiply the sign variable by 1 and make no adjustments on angle orientation. This keeps track of x and y positions, angle, and an

orientation sign. Figure 2a shows a half-hexagon with indication 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).

We propose to represent knapsack items as shapes. The legal orientation of those shapes with respect to each other for the constrained geometric knapsack problem are represented by the shape grammar. Thus a very simple geometric representation can generate all constrained geometric knapsack configurations. This is important; we provide an environment in which a simple and complete shape grammar must be 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 independent of the engineer's understanding of the global intricacies. The remaining issue, then, is how to generate a good solution to the problem.

4. Shape Annealing

As presented in Cagan and Mitchell [2], shape annealing utilizes the simulated annealing algorithm to determine whether a randomly selected shape rule should be applied at a given configuration state. Given a current configuration state, an eligible rule is selected and applied to the pattern. 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, based on the temperature profile, determines whether to accept it or not. If the rule violates a constraint then the old state is maintained as the current.

By only applying additive rules (*i.e.*, those rules which add a new piece 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 back itself out of these local solutions by reversing the previous rule. For every additive rule, there exists a subtractive opposite which removes the piece from the solution; if the subtractive rule is fired immediately following its companion additive rule (called *rule reversal*) then the effect of the additive rule is removed.

```
Begin SHAPE ANNEAL
    T = 1.
    Define initial state;
    Evaluate state;
    While T > 0 Do Begin
        success = 0;
        For mutations = 1 to n Do Begin     /* at each temperature mutate n 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 acceptable
                    Then Begin
                        state = temp_state;
                        success = success + 1;
                    End
                End
            End
            If success > limit Then break;
        End
        If success = 0 Then break;          /* no improvement...
                                               solution found */
        T = T*reduction_factor;
    End
End
```

Figure 3 Original Shape Annealing Algorithm.

The original shape annealing algorithm is given in Figure 3. The number of outer loop iterations, t, and the the number of inner loop iterations, m, need to be determined for the specific problem based on convergence of good solutions. The total number of loops will be $k = (t)(m)$. Note that in the algorithm, the shape grammar is assumed to have been defined, and thus any any applicable rule satisfies the orientation relations. Further, the step which checks for constraint violations verifies that the items do not overlap and do not cross the bounded knapsack space.

Shape annealing is a sort of Monte-Carlo technique in that there is no continuum of solutions and thus the convergence proof of Lundy and Meese [10] is not valid. However, the algorithm is different than traditional Monte-Carlo 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 [17] 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 as well as double the probability that the most recent piece would be removed. In that application, the packing factor increases by 15% from the original algorithm of Cagan and Mitchell.

It is interesting to note that the size of the mutation space is not reduced at each iteration; this again deviates from the normal simulated annealing approach but is required because there is no contour from which to move between neighboring configurations. In actuality the size of the configuration space *increases* while the algorithm runs; however, the increase is controlled and the algorithm finds a way to generate a maximal packing from an exponentially large number of possible configurations.

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. Space complexity is thus O(n+r); for n>>r, the space complexity becomes O(n).

The worst-case time complexity would have the algorithm run without convergence (k specified steps), where at each step one item is generated 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 generation of the n^{th} item at each iteration:

$$O(k(n+b)).$$

The best-case would generate the final solution in n iterations with one new item being generated and tested at each of the n iterations plus one temperature iteration (m) 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:

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

an upper bound on this ratio with n(n+b) >> m would be:

$$\frac{k(n+b)}{n(n+b)} = \frac{k}{n};$$

Thus the ratio and all bounds are polynomial in n. For large k, the bounds and ratio are dominated by the number of iterations. Note, however, that in practice the shape annealing algorithm is not guaranteed to find the global optimum; rather it only generates a good solution. By letting the algorithm run a sufficient time, our experience shows the solutions to be quite good.

In this approach to constrained geometric knapsack packing, each item of the knapsack is considered a geometric *shape* and is modeled with the properties of a shape as described by Stiny [4]. Utilizing the shape annealing algorithm, a good solution to this problem can be generated in polynomial time with minimal storage capacity which readily satisfies all geometric constraints. 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, only that each fits within a given set of orientations without occupying the same space. However, in the shape annealing solution order is specified, possibly restricting the solution, but guaranteeing that item orientation is valid.

5. Example

As an example application of the shape annealing solution to the constrained geometric knapsack problem, consider the 2-dimensional problem of packing as many half-hexagons as possible into a pre-defined 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 discussed in section 3. 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 the problem formulation but do not affect the algorithm. Also, there is only one class of elements ($X_k = X_1$) and there are three orientations for the pieces with respect to each other. For purposes of illustration, we consider all spaces to be of a constant area (25 units) and the half hexagon has a long base of one unit and a short base of one half a unit. Rule 4 is given for completeness but is actually only applied after the final solution is found.

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

$$\text{max:} \quad \sum_{p=1}^{n} X_{1_p},$$

$$\begin{aligned}
\text{s.t.:}\ & X_1 \in Z^+, \\
& \text{midline}\left(X_{1_q}\right) \neg\cap \ \text{midline}\left(X_{1_r}\right) \forall 1_r \neq 1_p, \\
& \text{midline}\left(X_{1_p}\right) \neg\cap \ \text{boundary}\left(S_{\text{total}}\right), \\
& O_{111} = O^*(1,\ 0.,\ \text{width}*\cos(\theta),\ \text{width}*\sin(\theta),\ -1,\ 0.), \\
& O_{112} = O^*(1,\ -\text{sign}*60.,\ \text{width}*\cos(\theta),\ \text{width}*\sin(\theta),\ 1,\ 0.), \\
& O_{113} = O^*(1,\ 0.,\ \text{height}*\sin(\theta),\ \text{height}*\cos(\theta),\ -1,\ 0.), \\
& O_{11} = O_{111} \vee O_{112} \vee O_{113},
\end{aligned}$$

where the orientations O_{111}, O_{112}, and O_{113} are described in section 3, and 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_{11q} (the symbol "v" indicates the disjunctive *or*). 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 intersection of each midline with the boundary surfaces; in this example we assume the boundary to consist of any two-dimensional closed surface. Figure 4 shows a rectangular boundary space (S_{total}) and the x-y coordinate system.

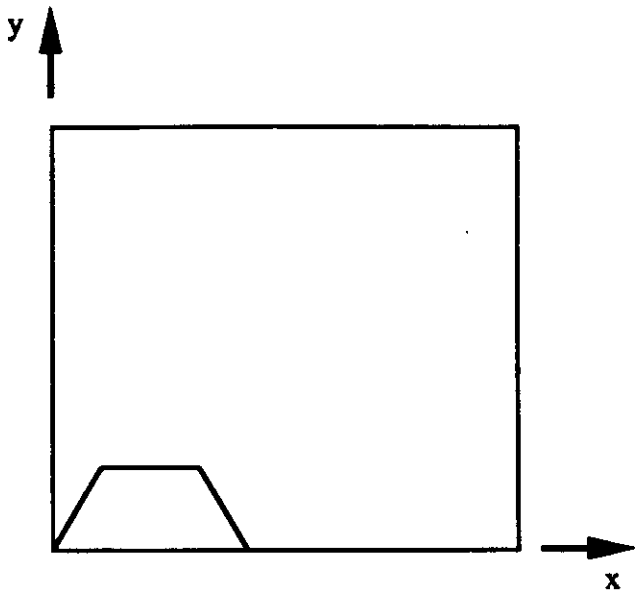

Figure 4 Description of possible boundary space.

For this example, all boundaries will be 25 square units. Typical results of the shape annealing algorithm on various 2-dimensional spaces are shown in Figures 5-7. The fill pattern indicates which rule was applied: the light fill indicates rule 1, the black fill indicates rule 2, and the medium fill indicates rule 3. For Figures 5 and 6, the packing starts with a piece (I) in the bottom left corner. For Figure 7 the packing starts toward the center of the circle. There is no guarantee that these solutions are optimal; rather the algorithm is run for a period of time and convergence is reached.

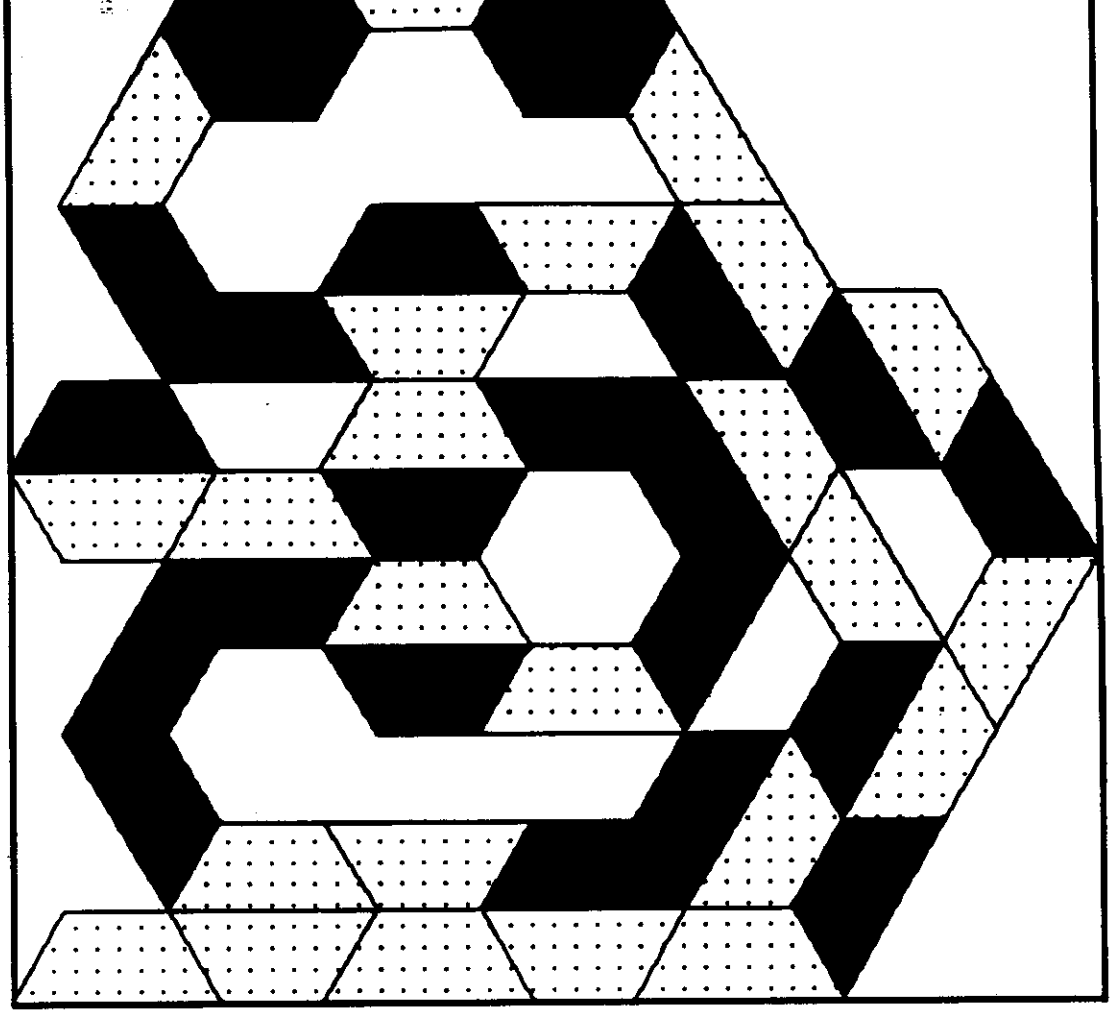

Figure 5 Resulting packing for 5X5 square of 47 pieces.

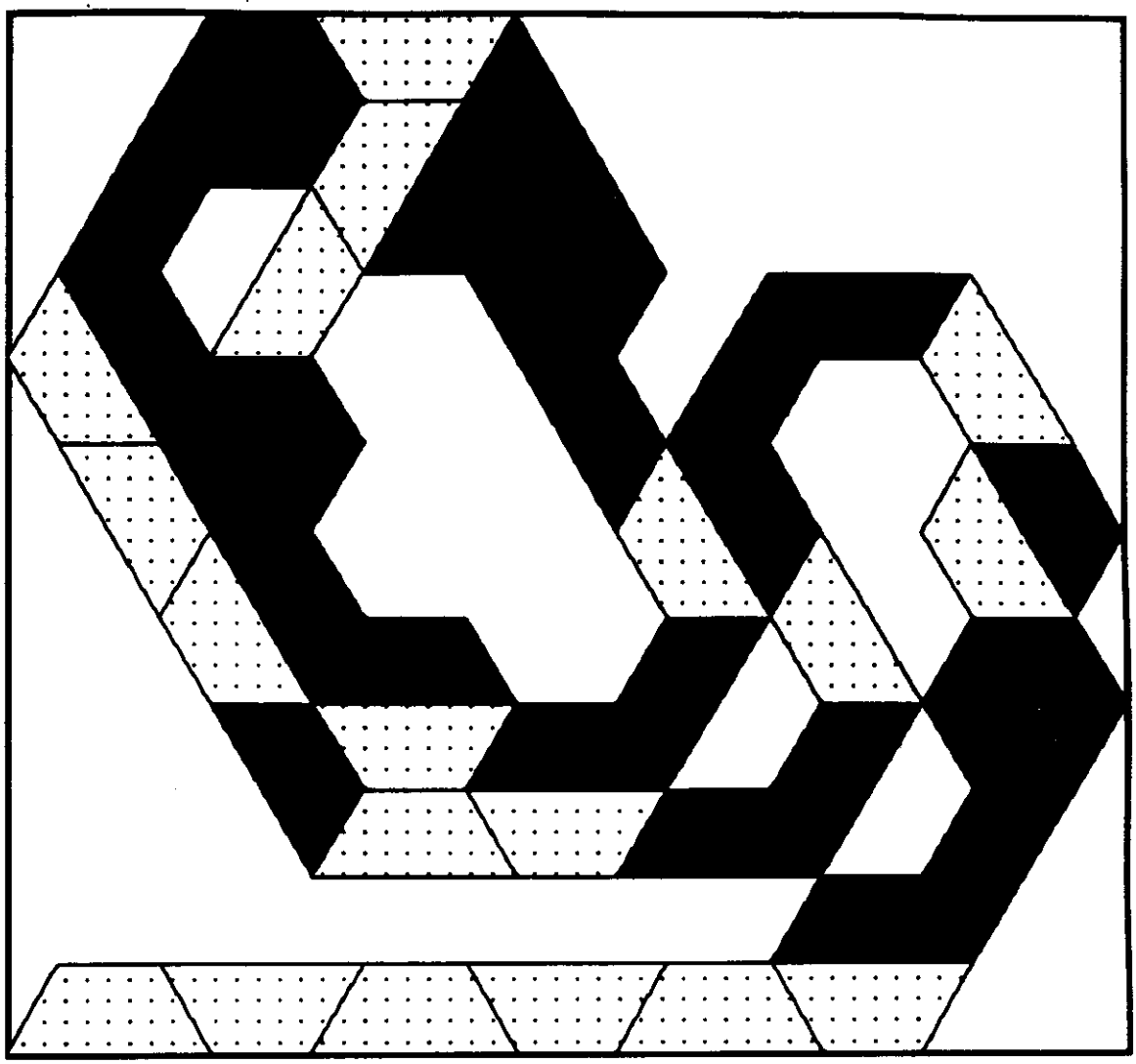

Figure 6 Resulting packing for 5X5 square of 45 pieces.

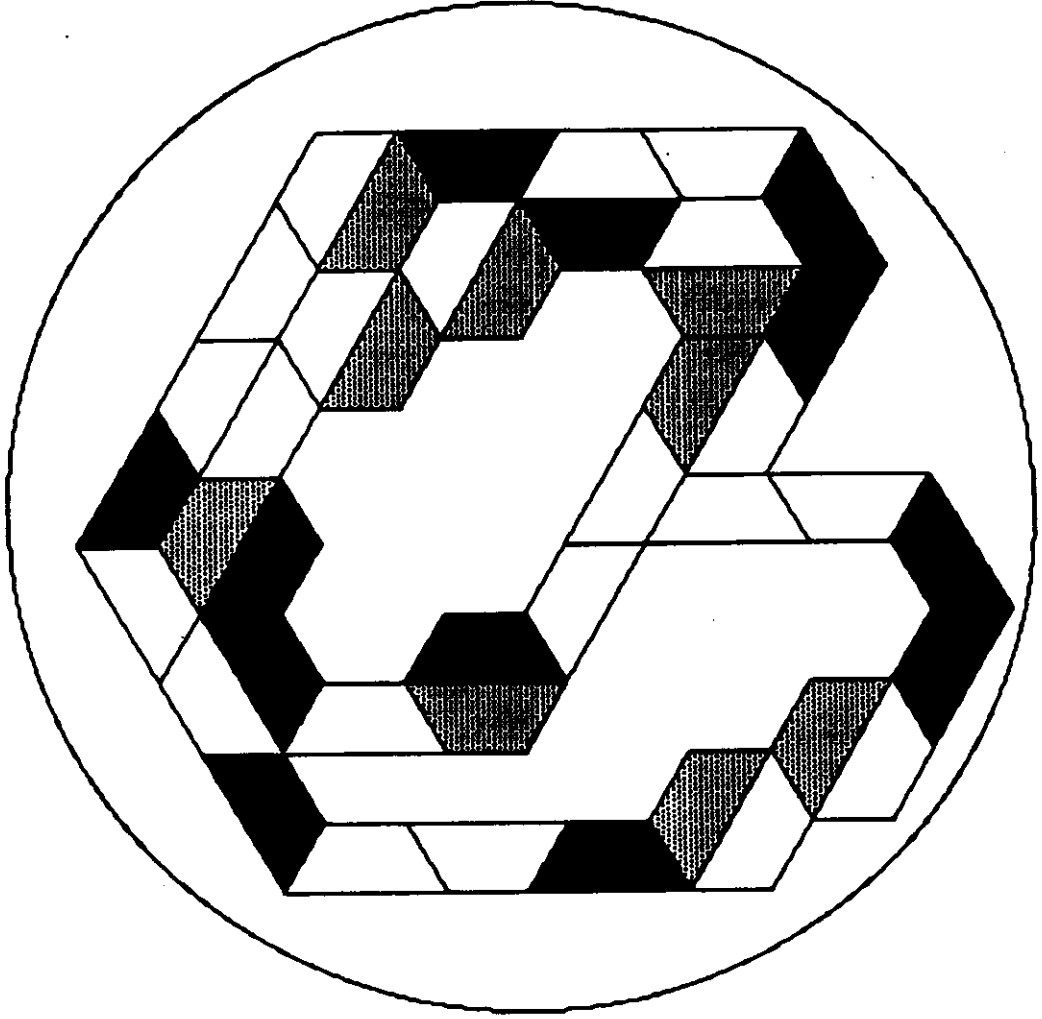

Figure 7 Resulting packing for circle of equivalent area to Figures 5 and 6 of 42 pieces.

6. 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 constrains the traditional knapsack problem with space and orientation constraints. Shape annealing runs in polynomial time and space, generating a sub-optimal but acceptable solution.

Shape annealing solves the constrained geometric knapsack problem in polynomial time and space complexity in number of items; the time is essentially dependent on the temperature profile (number of iterations) and number of pieces generated. The algorithm is computationally efficient in space. There is no need to maintain a trace of generated rules; only the current state and proposed modification at each step need 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, where a maximal solution is considered one in which a local change of adding or removing a piece will not improve the solution. In the example shown, a superior solution may be identified; again the global minimum is not guaranteed, but rather a good solution is generated algorithmically and a general method is presented which is useful with far more complicated shape grammars. A more formal analysis must still be performed to determine how *good* the shape annealing solutions are.

The use of shapes to model items and shape grammars to model their orientation makes for a general algorithm and simple solution approach. We have illustrated application in two dimensions; however, in theory, the algorithm is equally valid for three dimensions. Shape annealing can be used to solve many traditional and non-traditional layout problems such as the tiling problem, as well as find application in factory and process layout.

7. Acknowledgements

The author would like to thank Steve Cosares and Bill Mitchell for their important discussions about this work, and Steve for his comments on this manuscript. This work was partially sponsored by the Engineering Design Research Center at Carnegie Mellon University, an NSF sponsored research center.

8. References

[1] Martello, S., and P. Toth (1990), *Knapsack Problems - Algorithms and Computer Implementations*, John Wiley & Sons, New York.

[2] Cagan, J., and W.J. Mitchell (1991), "Optimally Directed Shape Generation by Shape Annealing", *accepted in: Environment and Planning B*, 1991.

[3] Kirkpatrick, S., C.D. Gelatt, Jr., and M.P. Vecchi (1983), "Optimization by Simulated Annealing", *Science*, **220**(4598):671-679.

[4] Stiny, G. (1980), "Introduction to Shape and Shape Grammars", *Environment and Planning B*, 7:343-351.

[5] Jain, P., P. Fenyes, and R. Richter (1990), "Optimal Blank Nesting Using Simulated Annealing", *Proceedings of: ASME Design Automation Conference: Advances in Design Automation - 1988* (Ravani, ed.), 2:109-116.

[6] van Laarhoven, P.J.M., and E.H.L.Aarts (1987), *Simulated Annealing: Theory and Applications*, D.Reidel Publishing Co.

[7] Cagan, J., and T.R. Kurfess (1991), "Optimal Design for Tolerance and Manufacturing Allocation", *EDRC Report 24-67-91*, Engineering Design Research Center, Carnegie Mellon University, Pittsburgh, PA 15213.

[8] Jain, P., and A.M. Agogino (1990), "Theory of Design: An Optimization Perspective", *Mech. Mach. Theory*, **25**(3):287-303.

[9] Metropolis, N., A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller (1953), *J. Chem Phys.*, **21**: 1087-1091.

[10] Lundy, M., and A. Meese (1986), "Convergence of an Annealing algorithm", *Mathematical Programming*, **34**:111-124.

[11] Stiny, G., and W.J. Mitchell (1978), "The Palladian Grammar", *Environment and Planning B*, 5:5-18.

[12] Stiny, G., and W.J. Mitchell (1980), "The Grammar of Paradise: on the Generation of Mughul Gardens", *Environment and Planning B*, 7:209-226.

[13] Koning, H., and J. Eizenberg (1981), "The Language of the Prairie: Frank Lloyd Wright's Prairie Houses", *Environment and Planning B*, 8:295-323.

[14] Knight, T.W. (1986), "Transformations of the Meander Motif on Greek Geometric Pottery", *Design Computing*, 1:29-67.

[15] Flemming, U. (1987), "More than the Sum of the Parts: the grammar of Queen Anne Houses", *Environment and Planning B*, 14:323-350.

[16] Mitchell, W.J. (1990), *The Logic of Architecture*, MIT Press, Cambridge, MA, p.143.

[17] Cagan, J., and G. Reddy (1991), "An Improved Shape Annealing Algorithm for Optimally Directed Shape Generation", to be presented at: *The Second Conference on Artificial Intelligence in Design '92*, Pittsburgh , PA, June 8-11, 1992.