Domain Knowledge for Genetic Algorithms

Sushil J. Louis

Dept. of Computer Science

Mackay School of Mines

University of Nevada, Reno, NV 89557

sushil@cs.unr.edu

Fang Zhao

Dept. of Civil and Environmental Eng.

Florida International University

Miami, FL 33199

zhaof@fiu.edu

Abstract

This paper describes the encoding of domain knowledge for use by a genetic algorithm. We use the domain of system configuration design problems; specifically, the structural design and optimization of trusses to ground our discussion and results. The approach applies evolutionary principles to the optimally directed configuration design of complex structures and incorporates engineering domain knowledge into a genetic algorithm to synthesize the topology, geometry, and component properties of the structure. Preliminary results indicate that genetic algorithms with domain knowledge can generate feasible and useful designs.

Keywords: Genetic Algorithms, Domain Knowledge, Truss Design

1 Introduction

Design problems can be classified into two types based on the problem specification and expected results. *Preliminary* or conceptual design deals with structuring a design problem and reducing it to a form suitable for *detailed* design. Configuration design is an example of a preliminary design problem and as such, has to cope with uncertain and missing information as well as system complexity brought on by the large number of components and their possible interactions. Its ill-structured nature leads to a lack of rigorous and/or effective design methods for attacking this problem. Artificial intelligence techniques typically cast such ill-structured problems as search problems in a large state space of possible solutions. Since genetic algorithms were designed to effectively search large, poorly-understood spaces, we use them as the search method of choice for this research.

When considering configuration design as a search problem, the number of possible solutions is usually too large for exhaustive search. Knowledge-based systems rely on expert or experiential knowledge to narrow the search space to a manageable size. Examples of such systems can be found in (Tong and Sriram, 1992). This approach works well in areas where such knowledge is encodable, but there are two disadvantages: 1) it is often difficult to extract and encode relevant design knowledge and 2) knowledge applicable in one domain may not be applicable in another domain.

Optimization techniques work well for well-understood problems (Radford and Gero, 1988). However, many problems are not well-understood and may not meet the preconditions necessary for the application of mathematical techniques. In addition, the problem size may be so large that the time taken to find a solution is not practical.

Genetic algorithms (GAs) are stochastic, parallel search algorithms based on the mechanics of natural selection, the process of evolution (Holland, 1975). GAs were designed to efficiently search large, non-linear, poorly-understood search spaces where expert knowledge is scarce or difficult to encode and where traditional optimization techniques fail. They are flexible (not brittle) and robust, exhibiting the adaptiveness and graceful degradation of biological systems. As such, GAs appear well suited for searching the large, poorly-understood spaces that arise in design problems.

We seek to combine some of the speed and accuracy of knowledge based systems with the robustness and flexibility of genetic algorithm search by incorporating domain knowledge in a genetic algorithm

We chose the structural design and optimization of trusses as the application domain for three reasons. First, truss design involves the determination of the topology and geometry of a truss system as well as the properties of the truss members. Second, the structure of a truss only consists of axially loaded truss members with the major property of truss members being their cross sectional areas. Thus, truss design has the essential characteristics of a complex system while being simple enough to allow an understanding of both the interactions of different design elements and the effectiveness of the methodology. We use a genetic algorithm augmented with engineering heuristics to design the topology, geometry, and member properties of trusses while minimizing the weight and maintaining feasibility. The third and last reason was that we could use a finite element analysis program to provide the feedback required by the GA.

The next section provides a brief introduction to search in design problems and the use of domain knowledge. Section 3 describes genetic algorithms and truss design. Section 5 describes our representation, encoding of domain knowledge, and the effect of genetic operators on candidate designs. Initial results reported in Section 6 suggest that genetic algorithms can be used in system configuration problems to provide feasible and useful designs. The last section discusses the results and suggests questions for future research.

2 Design and Search

Casting design as a search problem, a formulation with a well-established heritage in artificial intelligence research, a design process searches through a state space of possible structures for one or more structures that perform a specified function. Points in the state space represent candidate designs, and one or more of these points defines a goal state representing a design that meets specifications.

In search there is a tradeoff between *flexibility* — applicability in different domains, and *speed* — the number of candidate designs searched through before finding the correct one. Current design systems tend to depend on domain-specific knowledge and favor speed over flexibility. Such *knowledge-based systems* therefore do not perform well when domain knowledge is scarce but perform extremely well on the domains they were designed for. At the other extreme are enumerative or exhaustive search algorithms that trade flexibility for speed. They perform about equally well (or badly) across a wide variety of domains, but are usually prohibitively slow.

In between these extremes of random and/or exhaustive search and no search, every other search algorithm makes assumptions of varying kinds about search spaces. These assumptions correspond to knowledge about the space and may be correct, incorrect, or misleading, implicit or explicit, and known or unknown, when used for searching a particular space. This knowledge is exploited to guide exploration, speeding up search by generating a search bias manifested in the set of generated solutions.

Genetic algorithms belong to a class of algorithms, called blind search algorithms, which make the assumption that there is enough knowledge to compare two solutions and tell which is better (Rawlins, 1991). They make do with little domain knowledge, make few assumptions about the search space, and use domain-independent operators for generating candidate points in a state space.

A hybrid system that combines the best aspects of knowledge-based systems (speed) with the robustness of genetic algorithm search (domain-independence) would therefore be more useful, especially in engineering domains where domain knowledge is available but where such knowledge is usually not enough to solve a problem to optimality. Engineers use heuristics (rules of thumb) to help solve design problems. This knowledge is usually relatively easy to capture or encode, and helps to cut down the search space size. In this context, we describe a system that uses domain knowledge to guide a genetic algorithm that looks for feasible solutions in the domain of structural design and optimization of trusses. These solutions can then be improved through post processing by an engineer.

3 Genetic Algorithms and Truss Design

3.1 Truss Design

A truss is a structure that consists of a set of long, slender members arranged in the shape of one or more triangles (see figure 1). The members are assumed to be connected by frictionless pins and external loadings are applied only at the truss joints. Truss design usually starts with a given distance that the structure has to span and the loading conditions. Next, the designer determines the depth and overall profile of the truss (geometry), the number of members and their arrangement (topology), and the shape and areas of member cross sections (component properties) such that the truss will be able to resist the given loads while meeting the service requirements (specifications). The goal of truss optimization is to maximally utilize the material to result in the lightest structure while satisfying all the design, manufacturing, and other physical constraints.

Traditionally, truss optimization is performed using mathematical optimization techniques for a truss with a fixed configuration. That is, given the geometry and topology, the truss member cross sectional areas are optimized. The task, although well formulated, involves intensive computation. Using mixed integer optimization techniques, limited topology optimization may be achieved based on a set of alternative fixed configurations. An example of applying GAs to truss member optimization is reported in (Goldberg, 1989). Jenkins also applies a GA to optimize truss geometry and

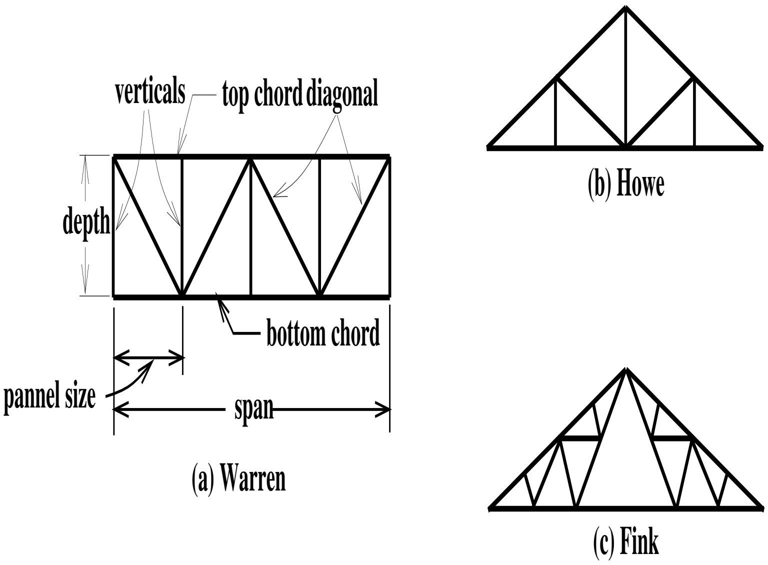


Figure 1: Three Different Types of Trusses

member cross sectional areas (Jenkins, 1991). In his study the topology is fixed and geometry is defined by a set of parameters such as member lengths and their inclined angles. To our knowledge, the only reported attempt on optimization of truss topology is described in Cagan and Mitchell's work (Cagan and Mitchell, 1993), in which simulated annealing (a search algorithm) is used to search through possible truss topologies. A truss topology is first generated by a shape grammar producing a triangularized shape. The geometry is then adjusted to satisfy problem constraints (the locations of supports and loads), followed by shape optimization based on, for example, stress, buckling, and manufacturing constraints using traditional optimization techniques. Based on the utility of the resultant shape the simulated annealer searches for another shape and the whole process is repeated. One problem with this approach, as pointed out by the authors, is that a complete shape optimization is performed at each step. This takes significant computational time, especially if the truss is statically indeterminate. The internal force distribution in a statically determinate truss, or any such structure in general, may be calculated independently of the members' properties while its calculation in a statically indeterminate structure requires information about members' properties, which are themselves the targets of optimization.

Genetic algorithms have also been used in shape design (Gero et al., 1994; Watabe and Okino, 1993). Although shape grammars provide a powerful representation technique, we chose a different representation for reasons of computational complexity and because we found it difficulty to encode heuristic (engineering design) knowledge. Our representation is described in Section 5 while the next subsection introduces genetic algorithms.

3.2 Genetic Algorithms

Genetic algorithms, originally developed by Holland, model natural selection, the process of evolution (Holland, 1975). Conceptually, GAs use the mechanisms of natural selection in evolving individuals that, over time, adapt to an environment. They can also be considered a search process, searching for better individuals in the space of all possible individuals. In practice, individuals represent candidate designs in a state space of possible designs, while the environment provides a measure of "fitness" that helps identify better individuals. Genetic algorithms are a robust, parallel search process requiring little information to search effectively. Goldberg's book provides a large list of application areas (Goldberg, 1989).

As already noted, the motivational idea behind GAs is natural selection implemented through selection and recombination operators. A population of candidate designs (usually encoded as bit strings) is modified by the probabilistic application of the genetic operators from one generation to the next. The basic algorithm where $P(t)$ is the population of strings at generation t, is given below.

$t = 0$

initialize $P(t)$

evaluate $P(t)$

```
while (termination condition not satisfied) do
begin
    select P(t+1) from P(t)
    recombine P(t+1)
    evaluate P(t+1)
    t = t + 1
end
```

Evaluation of each string which encodes a candidate design is based on a fitness function that is problem dependent. This corresponds to the environmental determination of survivability in natural selection. For our problem, a finite element analysis program calculates the total weight, node displacements, and member stresses, providing the information necessary for evaluating a design.

Selection is done on the basis of relative fitness and it probabilistically culls from the population those designs that have relatively low fitness. Recombination, which consists of mutation and crossover, imitates sexual reproduction. Mutation, as in natural systems, is a very low probability operator and just flips a specific bit. Crossover in contrast is applied with high probability. It is a structured yet stochastic operator that allows information exchange between candidate solutions. Two point crossover is implemented by choosing two random points in the selected pair of strings and exchanging the substrings defined by that point. Figure 2 shows how crossover mixes information from two parent strings, producing offspring made up of parts from both parents. We note that this operator, which does no table lookups or backtracking, is very efficient because of its simplicity. In summary, selection concentrates search in promising areas of the search space, while crossover and mutation provide new points in the search space (solutions) to evaluate. We can think of a genetic algorithm as a population of information exchanging hill climbers.

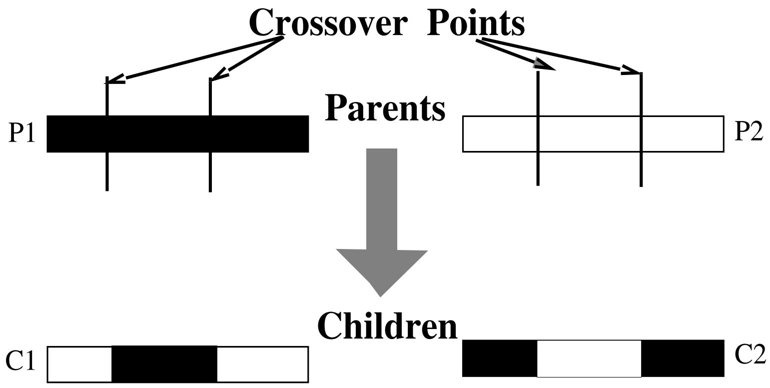


Figure 2: Crossover of the two parents A and B produces the two children C and D.

Holland's fundamental theorem of genetic algorithms, the *schema theorem*, leads to the building block hypothesis which states that genetic algorithms work near-optimally by *combining* certain types of *building blocks* corresponding to partial solutions or designs (Goldberg, 1989). In terms of truss design, genetic algorithms can be expected to combine good parts of candidate structures to generate improved structures.

The next section describes our methodology, the representation of the space of possible trusses, and how knowledge is encoded in the representation and genetic operators. The genetic algorithm searches this space for a feasible truss that optimizes weight, nodal displacement, and member stresses.

4 Methodology

We only consider planar trusses with (initially) rectangular profiles. As will be shown later the converged solution's profile may no longer be rectangular. A truss' geometry and topology may be defined by a set of joints, or to borrow from the terminology of finite element analysis, a set of *nodes*. A set of members connect these nodes. The set of nodes and their locations defines the geometry of a truss, while the set of members and the nodes they connect defines the topology.

There are three approaches to using domain knowledge in genetic search. We can place this knowledge 1) in the initial population 2) in the encoding of the genotype, and 3) in the genetic operators of crossover and mutation.

Initializing the genetic algorithm with individuals that our domain knowledge tells us are highly fit usually results in quicker convergence and/or better adaptability in changing environment (Grefensttete and Ramsey, 1993). However, the reduced diversity that may result can lead to premature convergence. To alleviate this problem we use high crossover and mutation probabilities.

Domain knowledge in the form of constraints on the range of variables is usually placed in the encoding of the genotype. In addition, ensuring that only viable offspring are produced is a task that depends on both the encoding and the recombination operators. In our task we use bit strings to encode member cross sections and top chord node heights. The topology and the rest of the geometry are encoded as more complex data structures with correspondingly different and "smarter" crossover and mutations operators. Instead of randomized crossover and mutation, these "smarter" operators use domain knowledge to guide crossover and mutation and usually lead to quicker convergence.

Note that as more domain knowledge is used to guide search, the less robust the algorithm becomes. Ideally we would like to start with a robust algorithm and using domain knowledge tune it to the domain.

The next section provides more detail on our representation and engineering heuristics used to seed the initial population.

5 REPRESENTATION

The first engineering heuristic that we encode indicates that good values for truss depth lie between 1/12 and 1/8 of the span for Warren trusses. Providing some leeway, we initialize the population by randomly generating trusses with depths between 1/14 and 1/6 of the span. Thus we determine initial truss depths with the following formula

$$\text{truss depth} = \frac{\text{span}}{\text{rnd}(6, 14)}$$

where $\text{rnd}(x, y)$ generates a random number between x and y inclusive.

The panel size heuristic suggests that panels be of approximately the same size as the depth. Thus trusses in the initial population have approximately equal depths and panel sizes.

Members are now generated to connect nodes such that the profile is, with high probability, triangularized. Triangularization makes for stable trusses. If we consider a rectangular truss to be made of two chords, top and bottom, we start with the leftmost node in the top chord and generate between 2 and 5 members to connect to adjacent nodes. Step 0 in Figure 3 shows that 3 members have been generated from node $[1, 0]$ to nodes $[0, 0]$, $[0, 1]$, $[1, 1]$. There is already one triangular structure, but one member ($[1, 0] - [1, 1]$) has been left dangling. In the next step, we connect node $[1, 1]$ with other nodes. This time we have generated (randomly) 4 members connecting to the nodes $[0, 0]$, $[0, 1]$, $[0, 2]$, $[1, 2]$. We continue in this way until reaching the rightmost node, in this case node $[1, 5]$. At this point, if a node does not form the vertex of a triangle, we add bracing members to triangularize the structure. This results in the truss structure shown at the bottom of figure 3.

Each member's cross sectional area is generated randomly in the range between 0.1 and 6.5 square inches. This procedure repeats until enough individual are generated to fill the initial

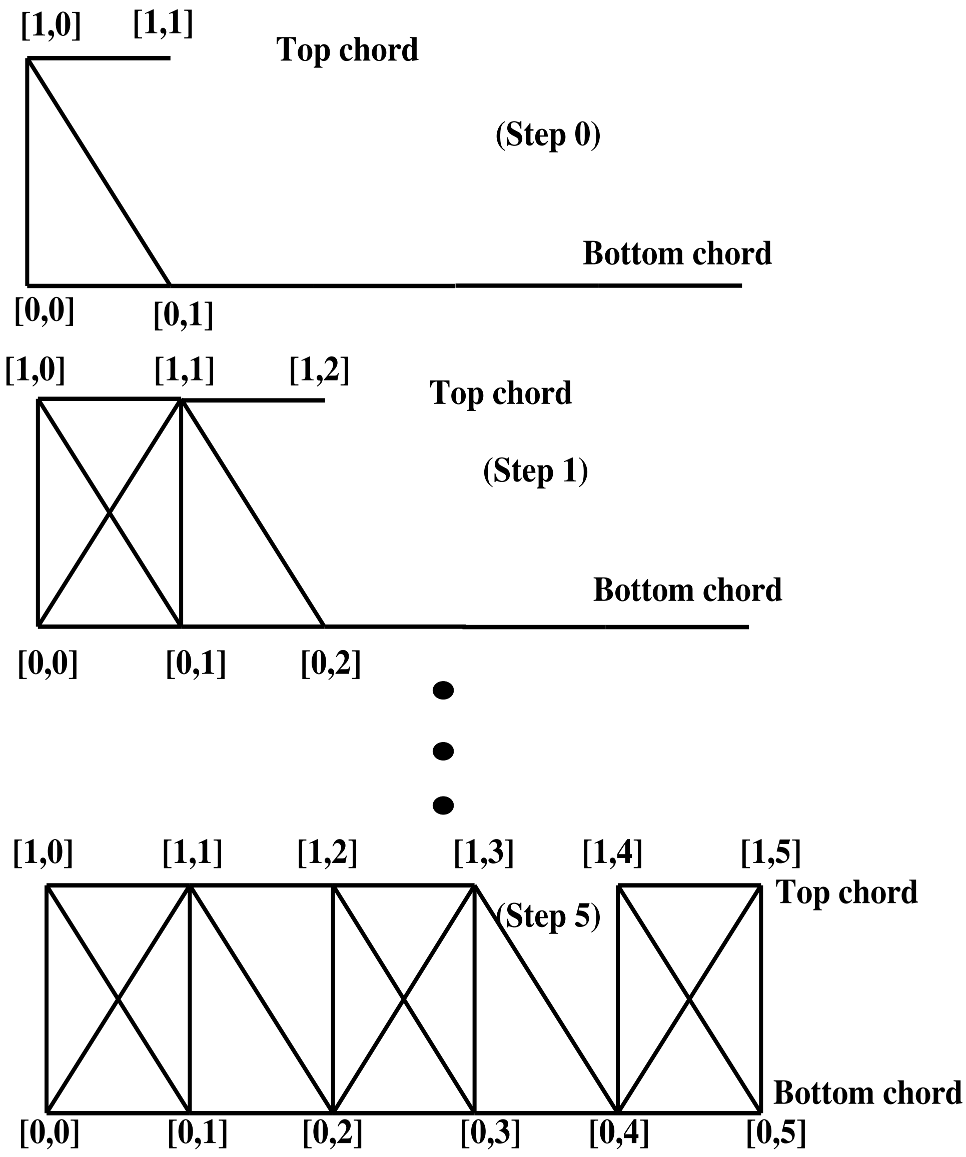


Figure 3: Generating a truss in the initial population.

population. The genetic algorithm then searches for a structure that satisfies design constraints and minimizes the objective function described below. For simplicity, only the most important design constraints, which are constraints on stress in the truss members and deflection of the truss, are used.

$$\text{total weight} \quad + p_1 \sum_{allmembers} |\text{stress}_{allowed} - \text{stress}_{actual}|$$

$$+ p_2 \, |\text{deflection}_{allowed} - \text{deflection}_{actual}|$$

Here $\text{stress}_{allowed}$ and stress_{actual} are the allowed and the actual stress for each truss member and $\text{deflection}_{allow}$ and $\text{deflection}_{actual}$ are the allowed and actual vertical deflections, at a predetermined truss node. p_1 and p_2 are penalty weights. Excessive stresses, excessive nodal displacements, and underutilization of material are penalized. Generated trusses that disobey hard constraints on nodal displacements and member stresses are more heavily penalized than those that satisfy these constraints. For example, overstressed members are penalized more than understressed members. Thus, p_1 takes a larger value if $\text{stress}_{actual} > \text{stress}_{allowed}$ as does p_2 if $\text{deflection}_{actual} > \text{deflection}_{allowed}$. $p1$ and $p2$ will take a smaller value if $\text{stress}_{actual} < \text{stress}_{allow}$ or $\text{deflection}_{actual} < \text{deflection}_{allow}$. The smaller values penalize the truss for wasted material. Ideally, an optimal solution will be of minimum weight with $\text{stress}_{actual} = \text{stress}_{allow}$ and $\text{deflection}_{actual} = \text{deflection}_{allow}$, since maximum utilization of material implies that all members are stressed to the design limits and the truss has just enough stiffness to satisfy the deflection requirement. However, it has been shown that optimal solutions may not always exist using the full stress design criterion (Save and Prager, 1985).

Figure 4 presents some of the trusses produced at initiation. The objective function returns a fitness value that is minimized to drive the genetic search in the direction of feasible, minimal weight solutions.

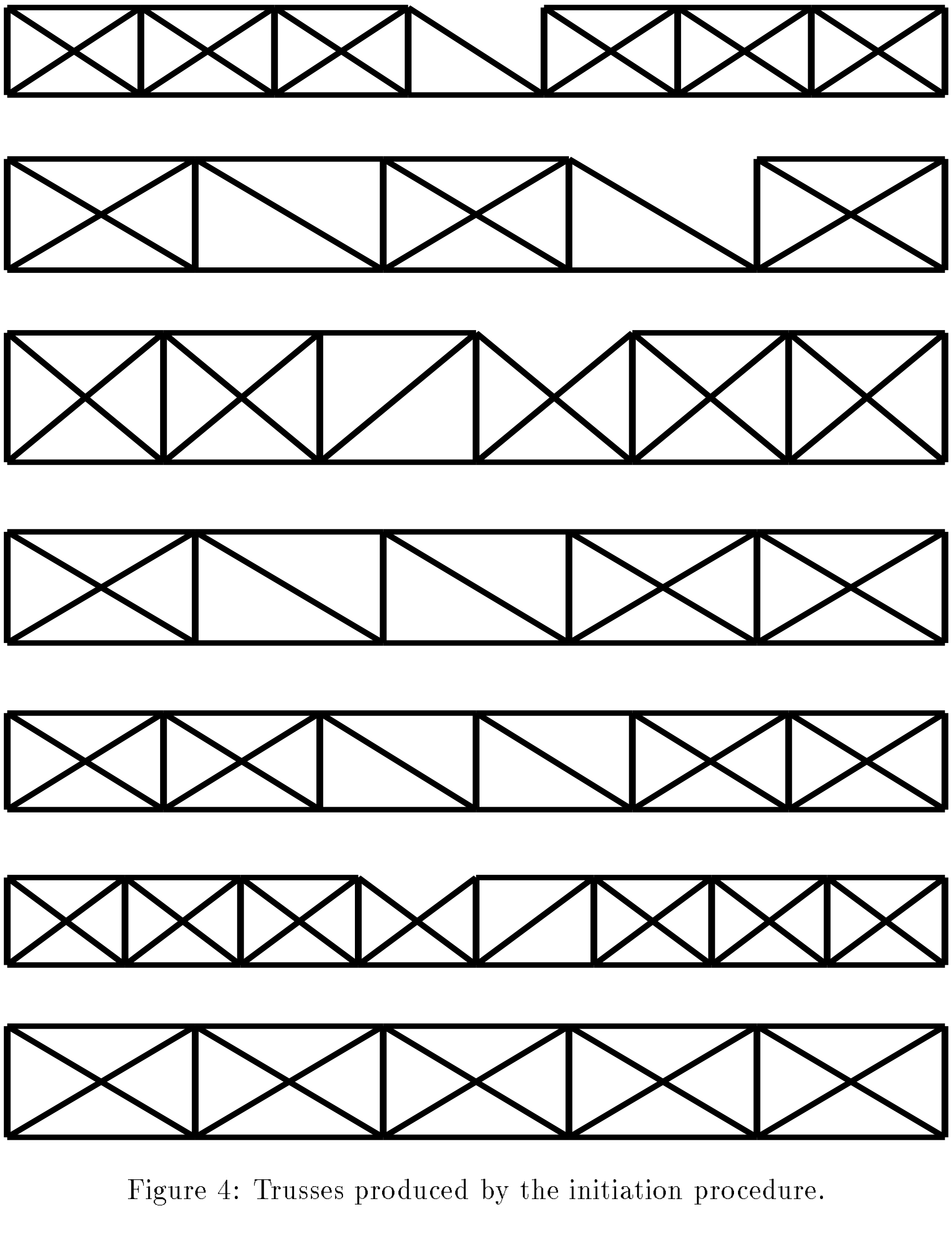


Figure 4: Trusses produced by the initiation procedure.

5.1 Representation and the Effect of Genetic Operators

Unlike traditional genetic algorithm representations that emphasize bit strings, we use a mixed representation, employing bit strings as well as more complex structures. This makes it easier to add heuristic constraints to the problem and reduces the size of the search space. Panel sizes are not subject to crossover and are fixed for an individual limiting exploration of this parameter to within the space spanned by the initial population.

Selection in genetic algorithms concentrates search in a promising area and is the exploitation operator. We discuss selection in the next section.

We want the crossover and mutation operators to generate new structures and guide exploration of the search space of possible truss structures. Starting with a statically determinate truss (triangularized), we need to be able to change the topology, geometry, and member cross sections in order to explore the space.

Exploring member cross sections and truss depths is fairly straightforward. We store these parameters as binary strings. Crossover and mutation as defined in the section on genetic algorithms suffice to explore the space of possible cross sections and truss depths.

Although changing the depth of nodes explores some of the topological space, we do not change the geometry. We define new crossover and mutation operators which are not simple bit string exchanges but in keeping with the spirit of genetic algorithms, have the effect of exchanging information.

Since crossover in genetic algorithms combines good substructures to improve solutions, we implement *structural crossover* by randomly selecting a contiguous set of nodes on the top chord and exchanging them as well as the members connected to them. Figure 5 depicts the result of crossing over the two structures on the bottom. The nodes to be crossed over are labeled $[1, 4]$ and $[1, 5]$ (shaded) and exchanging these nodes results in the structures labeled “child 1” and “child 2.”

Intuitively, "child 1" is a copy of "parent 1" except for the nodes that came from "parent 2" and vice versa.

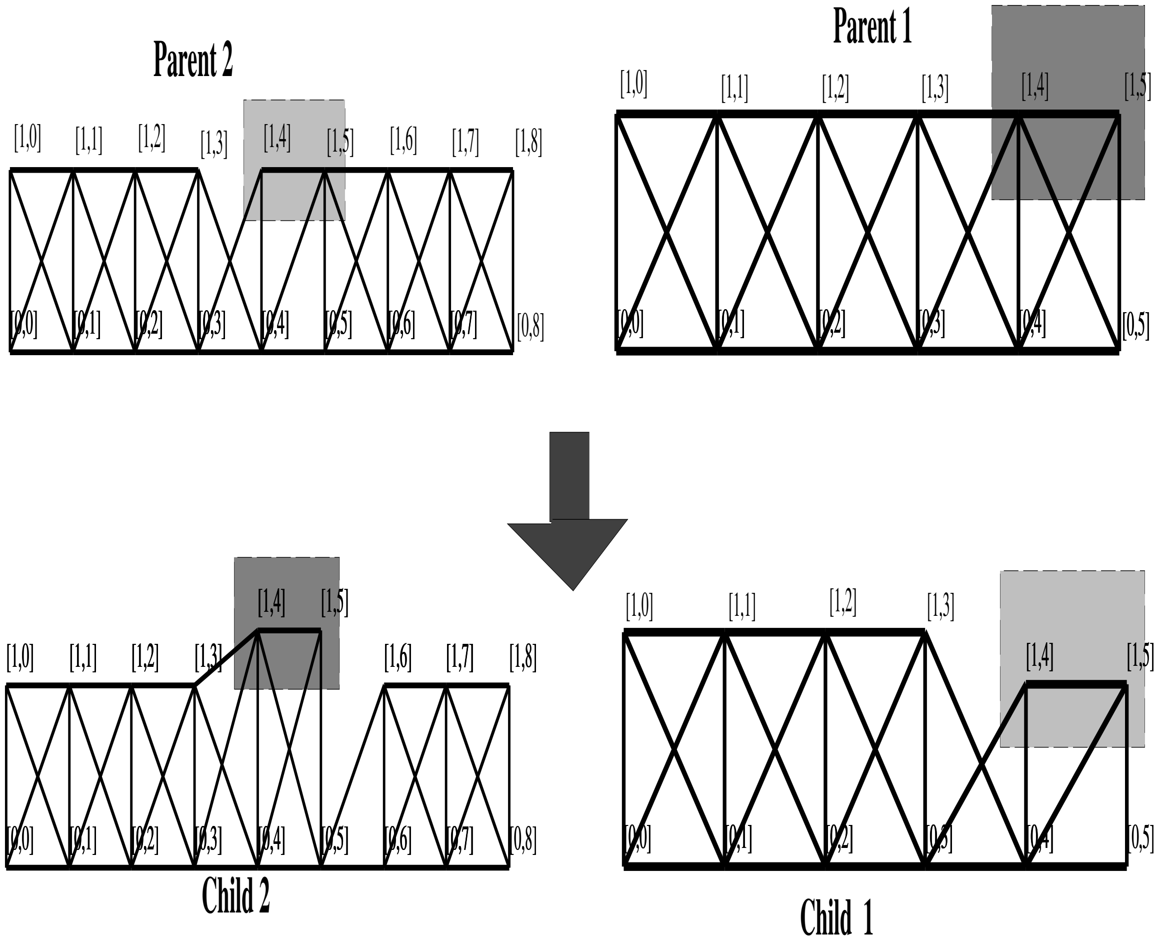


Figure 5: Structural crossover of trusses. Nodes $[1,4]$ and $[1,5]$ are exchanged.

Thus, structural crossover has the effect of changing both the topology and geometry of trusses in the population. Since member cross sectional areas are exchanged along with the members attached to a node, the cross sectional areas change in this way as well. *Structural mutation* adds and deletes *non-randomly* chosen members and nodes, and randomly perturbs member cross sectional areas. We encode engineering heuristics in the structural mutation operator by biasing it toward removing members with low stresses and adding members to nodes with large displacements. The probability of a member's removal due to structural mutation is inversely proportional to the stress on that member while the probability of a member being added to a node is directly proportional to the displacement. Top chord nodes can be removed through random mutation although there is no provision for adding nodes.

Recombination of this kind may generate trusses that are statically indeterminate (not triangularized). Every offspring is therefore checked and, if needed, additional members are added to brace and triangularize the truss.

6 Results

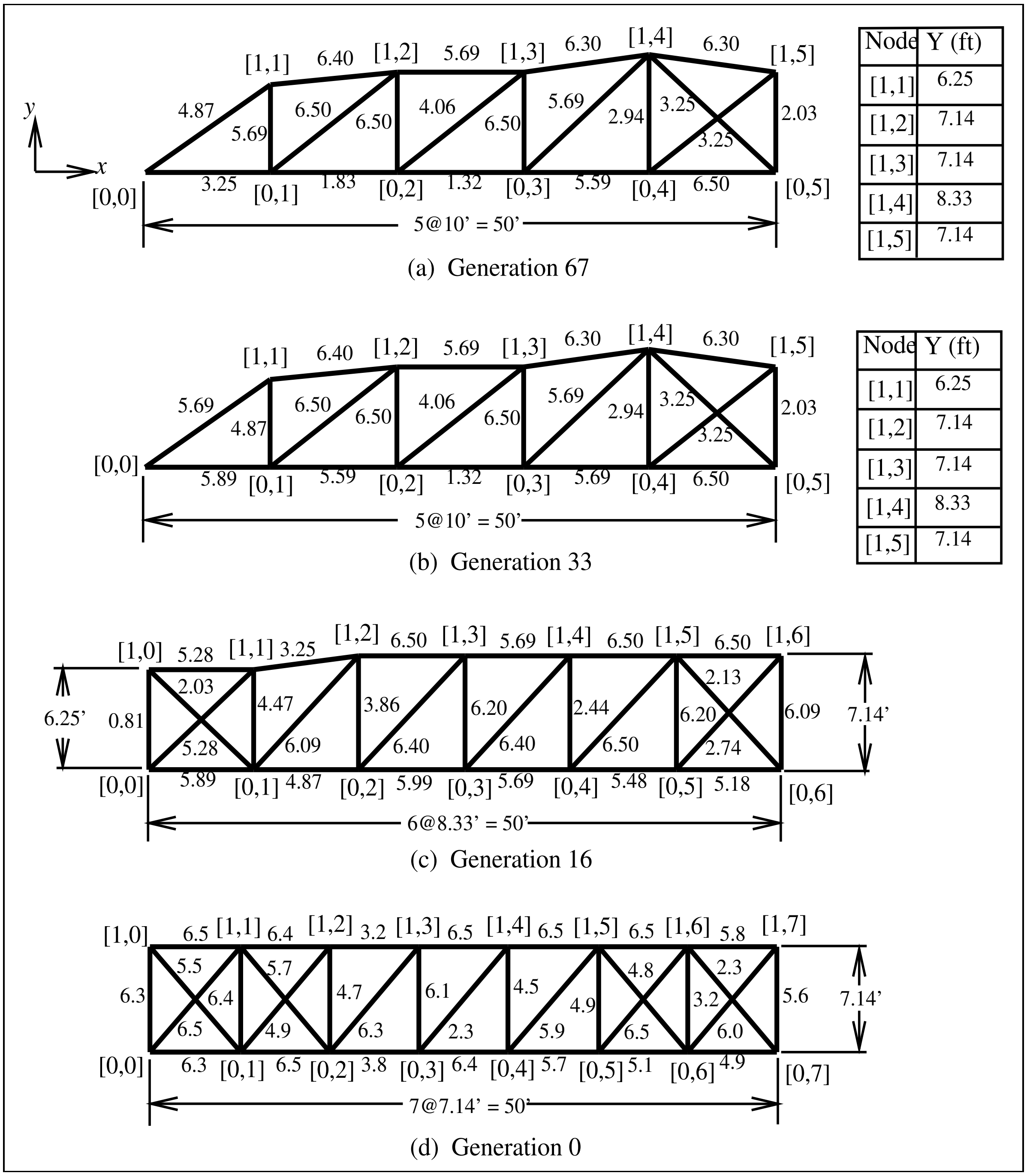
The test problem presented in this section is the design of a truss with a span of 100 feet and an evenly distributed vertical load that has an intensity of 600 pounds per linear foot. Since the loading is symmetric, for simplicity, we designed only for half trusses. This simplification restricts the trusses to have an even number of panels, but does not affect the validity or feasibility of our methodology. We assume the following:

1. Material is A36 steel with Young's modulus $E = 29^6$psi
2. Truss members have circular cross sections
3. Allowable deflection = 1/240 of span, which is 5 inches
4. Allowable tensile stress = $22,000$ psi
5. Allowable compressive stress is determined according to the American Institute of Steel Construction (AISC) "Specification for the Design, Fabrication and Erection of Structural Steel for Buildings." The allowable compressive stress for A36 steel is a function of the l/r ratio of a truss member in compression, where l is the length of the truss member and r the least radius of gyration (Merritt, 1983). Depending on the magnitude of l/r, the allowable stress ranges

between 3730 psi (for $l/r = 200$) and 21160 psi (for $l/r = 10$), and is larger for a shorter member with a larger r and smaller for a longer member with a smaller r. It is assumed that for $l/r > 200$, the allowable stress is zero.

Some typical configurations of the initial GA population were already shown in figure 4. The population size for the genetic algorithm was 50 and it ran for a 100 generations. We used the CHC elitist selection strategy in which for a population of size N, the offspring double the size of the population and the N best are chosen for the next generation from the $2N$ population of parents and offspring. Since elitism increases selection pressure and because the heuristic initialization of the population reduces diversity we used a crossover probability of 1.0 and a mutation probability dependent on the mutation cite. The probability of mutating a bit in cross sectional area of a member or height of a node was 0.07 and the probability of adding/removing a member was proportional to the relative stress on the member (removal), or displacement at a corresponding node (addition). Empirical testing established these parameter values as better than the more traditional values of 0.66 and 0.001 for crossover and mutation probabilities. The figures shown below are the best results from more than 20 genetic algorithm runs with different random seeds and parameters.

Figure 6 shows several intermediate truss designs (half trusses) produced by the GA, in which the best was found at generation 67. Numbers next to truss members are their cross sectional areas in square inches. A typical rate of convergence is depicted in figure 7 where we plot weight in pounds versus generation number. We use this graph of the genetic algorithm's performance instead of the average over many runs since a designer will be more interested in the best result rather than an average. The genetic algorithm quickly reduces the weight and finds the best solution for the run in the 67^{th} generation. This is typical behavior for the runs that we observed. Table 1 summarizes some of the major structural characteristics of the design solutions shown in figure 6.



Node	Y (ft)
[1,1]	6.25
[1,2]	7.14
[1,3]	7.14
[1,4]	8.33
[1,5]	7.14

(a) Generation 67

Node	Y (ft)
[1,1]	6.25
[1,2]	7.14
[1,3]	7.14
[1,4]	8.33
[1,5]	7.14

(b) Generation 33

(c) Generation 16

(d) Generation 0

Figure 6: Some solutions produced by a genetic algorithm run

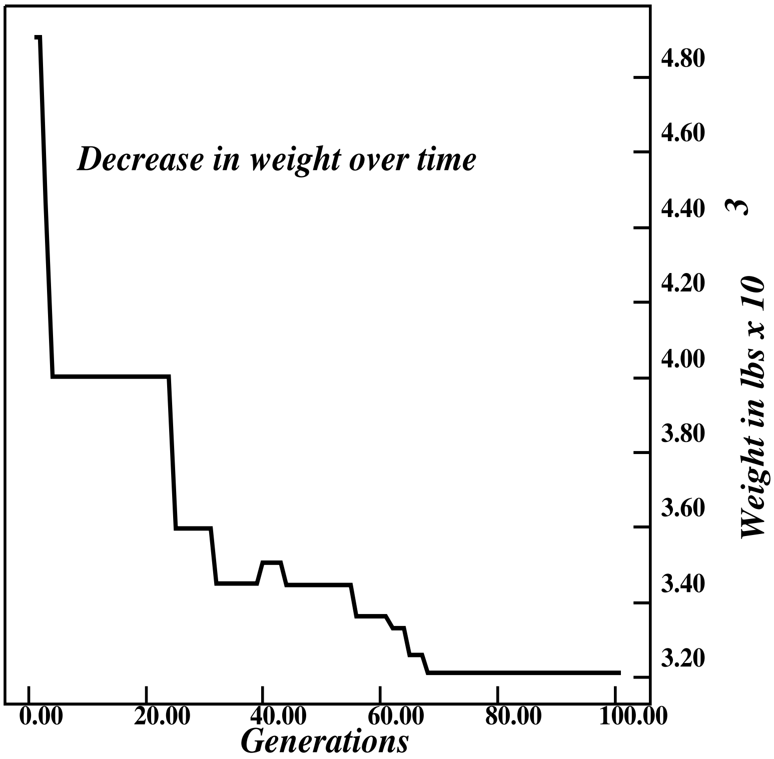


Figure 7: Convergence behavior, best truss weighed 3214 pounds.

[Table 1 here]

In the table, column two gives the maximum depth of each truss while column three provides panel size and column four the total weight. Column five to seven report maximum, minimum, and average *utility factors*, which are defined as the ratios of the actual stress to the allowable stress in truss members. The last column displays the maximum displacement. Truss (e) (generation 0) is not feasible because of the large utility factor, indicating overstressing in the truss. The design stress requirement used here allows truss members to be overstressed by 3%, which means a maximum utility factor of 1.03. All other trusses are feasible solutions, with utility factors smaller that 1.03 and the maximum displacement, which occurs at the rightmost bottom node, smaller than the 5 inches permitted. It appears that design constraint on stresses plays a much more dominant role than the deflection constraint does.

The results indicate that the evolution of the population has brought changes in their topology, geometry, as well as cross sectional areas. Truss envelopes have evolved from rectangles to curves, which, in general, agree with the shape of the moment diagrams of the trusses (see figure 8). In trusses (a) - (c), the depth is reduced at the center of the trusses (at node $[1,5]$). We note that if the depth remains 8.33 feet as at node $[1,4]$, the truss will be lighter while all the the design constraints are satisfied. The maximum depth of trusses (a) - (c) range from 7.14 to 8.33 feet, which is approximately 1/14 to 1/13.5 of the span. Since average utility factors reflect the efficiency of material use, another indication of improvement in the population is given by the increase in average utility factors over time.

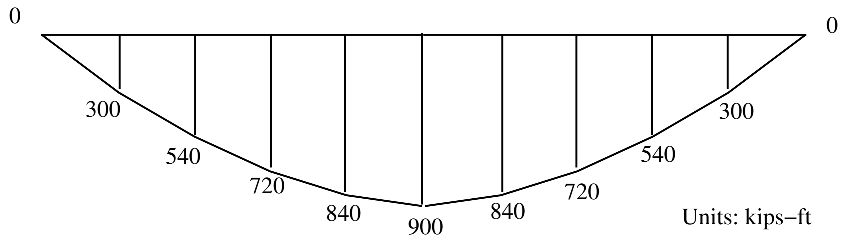


Figure 8: The shape of moment diagram of trusses (a) - (c)

We would like to point out that genetic algorithms do not *guarantee* optimal solutions. Different populations sizes or probabilities of crossover can result in the genetic algorithm finding other feasible solutions, some with lower weights. Using Cagan and Mitchell's terminology, genetic algorithms are optimally *directed* (Cagan and Mitchell, 1993) and can be counted on to improve upon their initial population. Genetic algorithm results *point* toward good regions of the search space and allow designers to use their knowledge and expertise to improve upon solutions generated through genetic search. For example, if the depth of truss (a) in figure 6 at node $[1,5]$ is changed to 8.33 feet and the two members connecting nodes $[0,5]$, $[1,4]$ and $[0,5]$, $[1,5]$, are removed, the truss weight will be reduced to 3021 pounds. However, the GA has, as expected, quickly reduced the size of the design space and produced some (possibly good) solutions that allow a designer to apply design knowledge and experience to improve them. Letting the genetic algorithm take care of identifying a promising region in the search space for post-processing by an engineer saves time and reduces cost.

Figure 9 shows the best truss configuration weighing 3171 pounds and found in generation 74, for another run of the genetic algorithm. Figure 10 plots its convergence behavior to highlight the fact that the decrease in weight need not be monotonic. The GA is optimizing a function that includes weigth as only one out of three criteria.

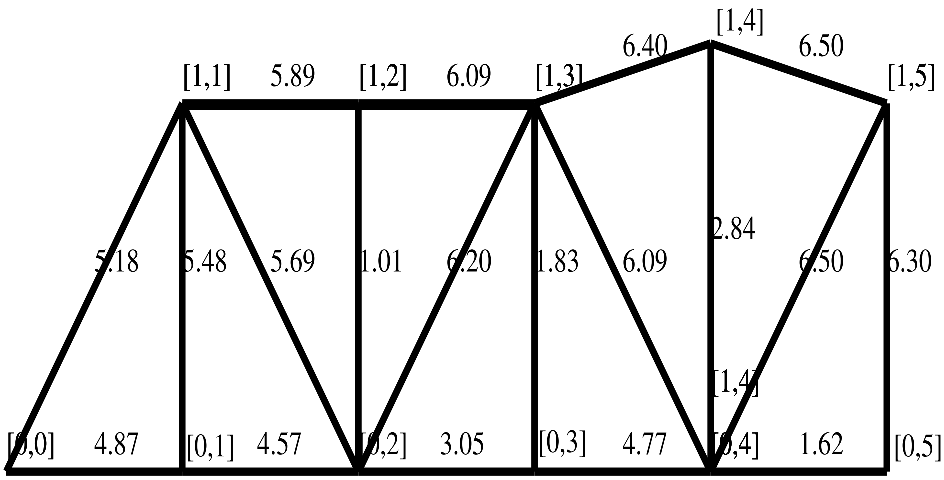


Figure 9: The best truss configuration of another genetic algorithm run, found in generation 74.

As noted earlier, many design heuristics and constraints have been incorporated into the genetic algorithm. These design heuristics and constraints greatly affect both the size of the design solution space and the GA's convergence rate. Relaxing some constraints (reducing the amount of domain knowledge used) can lead to different, and perhaps more surprising, structures. For example, figure 11 shows the best configuration produced in another run where we allow diagonal members to cross two panels.

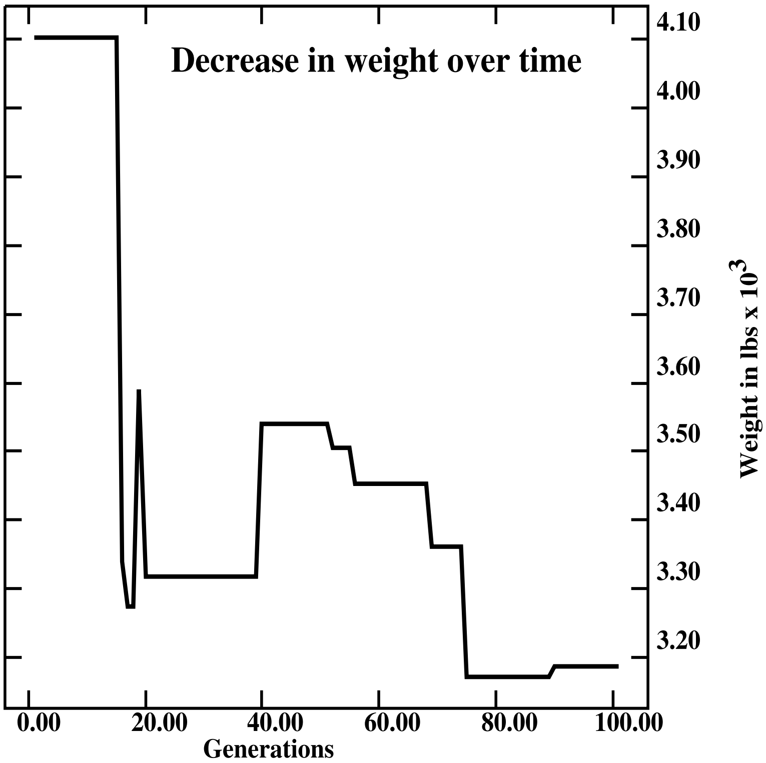


Figure 10: Convergence behavior of genetic algorithm run for the truss in the previous figure, note the non-monotonic decrease in weight.

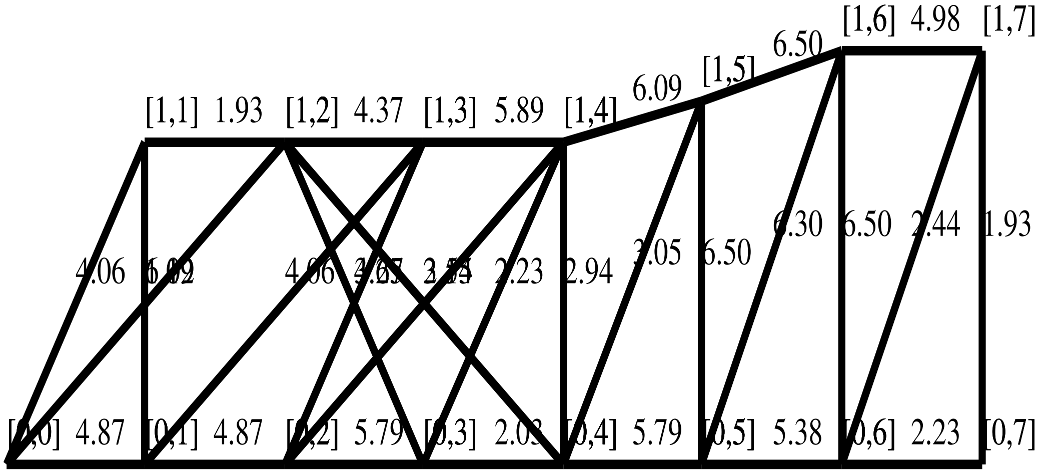


Figure 11: The best truss configuration of a less constrained genetic algorithm run, found in generation 63. Diagonal member were allowed to cross panels.

7 Discussion and Conclusions

Many engineering problems require both domain knowledge and search/optimization techniques for their solution. Hybrid systems that use robust search algorithms when confronted with a problem outside the scope of their knowledge, bring up the issue of how best to incorporate this knowledge in guiding robust search. This paper studied the problem of incorporating domain knowledge in genetic algorithm search for the configuration design and optimization of trusses.

We used domain knowledge to initialize the genetic algorithm's population and to guide the application of crossover and mutation. This substantially reduces the search space and speeds convergence. The genetic algorithm is able to produce feasible, useful solutions in approximately twenty minutes on a high-end workstation. Broadly speaking, the GA first settles on a topology and geometry, and spends the rest of the time optimizing member cross sections.

This paper explores an essential aspect of interfacing genetic search with knowledge-bases. As a next step, we plan to use case-based reasoning to store genetic algorithm solutions and use these stored cases to initialize the population.

The representation used in this paper allows easy incorporation of design heuristics and constraints. This leads to a smaller search space and therefore fast convergence times. However, relaxing the constraints and expanding the search space may lead to more exploration and perhaps better and more innovative solutions (Louis, 1993). We are currently exploring this tradeoff. One of the problems in our representation is that crossover can generate structures that are not viable; in other words, structures that cannot bear the load or that are simply not realistic. This causes the finite element analysis program to abort. Our current solution is to patch up such structures by adding bracing members. A representation that guarantees viable offspring would be better. Shape grammars (Stiny and Gips, 1978) offer an alternative representation and have been used by

Cagan and Mitchell.

In the future, we plan to investigate the influence of various program parameters on the results, which include the penalty weights for overstressing and understressing, the probability of crossover and mutation, and parameters reflecting design heuristics and constraints. We also plan to develop different representations which are more easily manipulated by the genetic algorithm as well as being more efficient. Along with representations, we can try different search algorithms such as simulated annealing or other stochastic hill-climbers and explore the encoding of domain knowledge for these algorithms. Finally, we hope to extend the methodology to other types of structural system design as well as system configuration design in different domains.

References

Cagan, J. and Mitchell, W. J. (1993). A grammatical approach to network flow synthesis. In Gero, J., editor, *Preprints of the IFIP WG 5.2 Workshop on Formal Design Methods for Computer-Aided Design*, pages 153–166, Tallinn, Estonia.

Gero, J. S., Louis, S. J., and Kundu, S. (1994). Evolutionary learning of novel grammars for design improvement. *Artificial Intelligence in Engineering Design, Analysis and Manufacturing*, 8:83–94.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, Reading, MA.

Grefensttete, J. and Ramsey, C. (1993). Case-based initialization of genetic algorithms. In Forrest, S., editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 84–91, San Mateo, California. Morgan Kauffman.

Holland, J. (1975). *Adaptation In Natural and Artificial Systems.* The University of Michigan Press, Ann Arbour.

Jenkins, W. (1991). Towards structural optimization via the genetic algorithm. *Computers and Structures*, 40(5):1321–1327.

Louis, S. J. (1993). *Genetic Algorithms as a Computational Tool for Design.* PhD thesis, Indiana University, Bloomington. Department of Computer Sciences.

Merritt, F. S. (1983). *Standard Handbook for Civil Engineers.* McGraw-Hill, New York, NY.

Radford, A. D. and Gero, J. S. (1988). *Design by Optimization in Architecture, Building, and Construction.* Van Nostrand Reinhold Company, New York.

Rawlins, G. J. E. (1991). Introduction. In Rawlins, G. J. E., editor, *Foundations of Genetic Algorithms-1*, pages 1–12. Morgan Kauffman.

Save, M. and Prager, W., editors (1985). *Structural Optimization, Volume 1: Optimality Criteria.* Plenum Press, New York, NY.

Stiny, G. and Gips, J. (1978). *Algorithmic Aesthetics: Computer Models for Criticism and Design in the Arts.* University of California Press, Berkeley and Los Angeles, California.

Tong, C. and Sriram, D. (1992). Introduction. In Tong, C. and Sriram, D., editors, *Artificial Intelligence in Engineering Design*, pages 1–53. Academic Press, Inc.

Watabe, H. and Okino, N. (1993). A study on genetic shape design. In Forrest, S., editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 445–450, San Mateo, California. Morgan Kauffman.

Truss	**max. D (ft)**	**Panel Size (ft)**	**W (lbs)**	**max. UF**	**min. UF**	**avg. UF's**	**max. Displ (in)**
(a)	8.33	10.0	3214	1.02	0.01	0.47	1.2
(b)	8.33	10.0	3451	0.99	0.02	0.39	1.14
(c)	7.14	8.33	4004	0.93	0.006	0.33	1.23
(d)	7.14	7.14	4906	1.37	0.03	0.30	1.31

Table 1: The structural characteristics of the trusses