

INCORPORATING PROBLEM SPECIFIC INFORMATION IN 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 an approach to incorporating domain knowledge into genetic algorithm search. We use genetic algorithms to attack system configuration design problems; specifically, the structural design and optimization of trusses. Since there exists a large amount of domain knowledge on this problem, we describe the incorporation of this knowledge for guiding genetic search. We outline the problem, the heuristics used, and the encoding of the problem in light of available domain knowledge. Preliminary results point toward the effectiveness of combining genetic algorithms with knowledge-based systems.

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.

In search there is a tradeoff between *flexibility* — applicability in different domains, and *speed* — the number of candidate designs searched through be-

fore finding the correct one. This tradeoff usually corresponds to a balance between exploration of the search space versus exploitation of a particular area. When viewed from search's perspective, knowledge-based systems tend to use domain knowledge to reduce the search space and quickly find the correct solution with little exploration. Examples of such systems used in engineering design (our problem domain) can be found in [8]. 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. Such systems are fast, but not flexible.

Genetic algorithms (GAs) are at another end of the search spectrum. They are stochastic, parallel search algorithms based on the mechanics of natural selection, the process of evolution [4]. 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 (domain independent) and robust, exhibiting the adaptiveness and graceful degradation of biological systems. However, like other robust search algorithms, they often require a large number of iterative steps before converging on a candidate solution.

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. One of the issues faced by such systems is in interfacing the genetic search component with the knowledge-base. Striking a good balance between exploration and exploitation is essential to the design of hybrid systems.

In this context, we describe a system that uses do-

main knowledge to suggest good starting points for 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.

The next section provides brief introductions to genetic algorithms and truss design. Sections 3 and 4 describe our representation and the effect of genetic operators on candidate designs. Initial results reported in Section 5 suggest that genetic algorithms augmented by domain knowledge can be used in system configuration problems to provide, feasible and useful designs. The last section presents conclusions and suggests questions for future research.

2 GAs AND TRUSS DESIGN

2.1 Genetic Algorithms

Genetic algorithms, originally developed by Holland, model natural selection, the process of evolution [4]. They 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 [7].

Conceptually, GAs use the mechanisms of natural selection in evolving individuals that, over time, adapt to an environment. 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.

A population of candidate designs (phenotypes) usually encoded as bit strings (genotype) is evaluated and modified by the probabilistic application of the genetic operators of selection, crossover, and mutation from one generation to the next. Evaluation of each string which encodes a candidate design is based on a fitness function that is problem dependent. 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 selects relatively fitter individuals for recombination and concentrates search in the area specified by fitter individuals (exploitation). Mutation insures against the permanent loss of genetic material and with very low probability flips a bit in the genotype. Crossover is the main exploration operator. 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 the chosen points. Figure 1 shows how crossover mixes information from two parent strings,

producing offspring made up of parts from both parents.

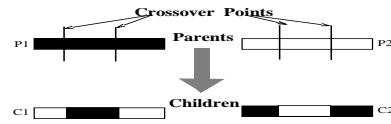


Figure 1: two point crossover.

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 [2]. In terms of truss design, genetic algorithms can be expected to combine good parts of candidate structures to generate improved structures. GA theory also states that binary encodings provide the maximal number of building blocks and thus guide us in our choice of encoding. However, incorporating domain knowledge in binary encodings is less intuitive than in more “natural” encodings of a problem.

2.2 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 2). 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 service requirements. 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.

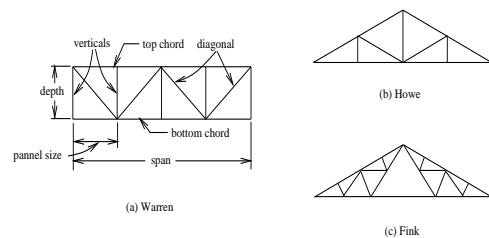


Figure 2: Three Different Types of Trusses

Traditionally, given the geometry and topology, the truss member cross sectional areas are optimized using mathematical optimization techniques. This is

however, very slow for large problems. 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 [2]. Jenkins also applies a GA to optimize truss geometry and member cross sectional areas [5]. To our knowledge, the only reported attempt on optimization of truss topology is described in Cagan and Mitchell’s work [1], in which simulated annealing (a search algorithm) is used to search through possible truss topologies. Once again, a problem with this approach is that it takes significant computational time, especially if the truss is statically indeterminate. Domain knowledge restricting the search space would be especially useful in such problems.

3 ENCODING ISSUES

We only consider planar trusses in this study. 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 [3]. 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 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.

4 REPRESENTATION

We incorporate problem knowledge into a GA by generating the initial population based on (in our case) engineering knowledge. Given a span, we first calculate the truss depth. Design heuristics indicate that good values for truss depth lie between 1/12 and 1/8 of the span. Next, the panel size is set to be approximately the same as the depth. This also determines the number of panels. Members are now generated to connect nodes such that the profile is, with high probability, triangularized. Each member’s cross sectional area is generated randomly in the range between 0.1 and 6.5 square inches. 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.

$$\text{total weight} + p_1 \sum_{\text{all members}} |\text{stress}_{\text{allowed}} - \text{stress}_{\text{actual}}| + p_2 |\text{deflection}_{\text{allowed}} - \text{deflection}_{\text{actual}}|$$

Here $\text{stress}_{\text{allowed}}$ and $\text{stress}_{\text{actual}}$ are the allowed and the actual stress for each truss member and $\text{deflection}_{\text{allow}}$ and $\text{deflection}_{\text{actual}}$ 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 thus penalized. The objective function returns a value that is minimized to drive genetic search in the direction of feasible, minimal weight solutions.

4.1 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.

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 and their connected members. This lamarkian process results in genetic search through the space of possible top-chord node heights, connectivity, and cross sectional areas. Figure 3 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 (and all attached members of that node) that came from “parent 2” and vice versa. Thus,

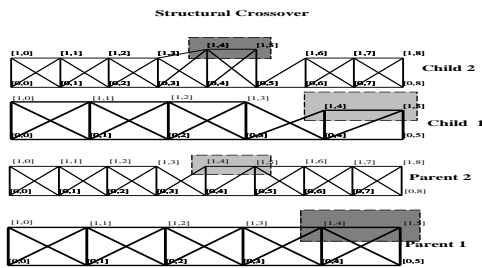


Figure 3: Structural truss crossover, [1, 4] and [1, 5] exchanged.

structural crossover has the effect of changing the topology of trusses in the population. Panel sizes are not subject to crossover and are fixed (heuristically) for an individual. Truss depths (geometry) depend on the heights of nodes on the top chord. Top chord node heights are encoded as bit-strings along with member cross sectional areas and are subject to two-point crossover. All these changes are restricted to be within the topological, geometrical and cross sectional space spanned by the initial population. The mutation operator allows the space to change from that spanned by the initial population. Since the initial population was generated nonrandomly, mutation takes on a greater significance in maintaining diversity. It adds and deletes *non-randomly* chosen members and nodes, and randomly perturbs member cross sectional areas. Like crossover there are two kinds of mutation operators. Random mutation works on the bit string encoded cross sectional areas and top-chord node heights with the usual low probability. Structural mutations that add/remove members are biased toward removing members with low stresses and adding members to nodes with large displacements. Top chord nodes can be removed through random mutation although there is no provision for adding nodes.

5 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 trusses¹. The population size for the genetic algorithm was 50. 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.

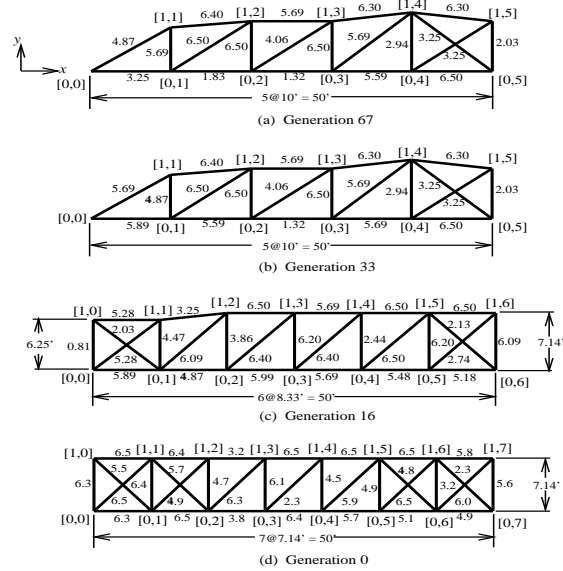


Figure 4: Evolution of Trusses

Figure 4 shows several intermediate truss designs

¹This simplification restricts the trusses to have an even number of panels, but does not affect the validity or feasibility of our methodology.

(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. The rate of convergence is depicted in figure 5 where we plot weight in pounds versus generation number. The genetic algorithm quickly reduces the weight and finds the best solution for the run in the 67th generation. This is typical behavior for the runs that we observed; note that the weight decrease may not be monotonic.

Relaxing some constraints (reducing the amount of domain knowledge used) can lead to different, and perhaps more surprising, structures. Post processing of solutions produced by GAs is another issue. Space constraints force us to elide these interesting issues for now. With the imposed constraints we can make an estimate on the reduction in size of the search space. Not encoding the x-coordinates of nodes alone reduces the search space by between 2^{40} to 2^{70} , if we assume 5 bits per x-coordinate. This must be balanced against the possibility of missing out on any feasible solutions in that area of the search space. This is a tradeoff that needs to be explored.

6 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

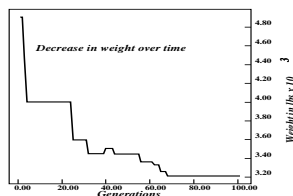


Figure 5: Convergence behavior, best truss weighed 3214 pounds.

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. However, relaxing the constraints and expanding the search space may lead

to more exploration and perhaps better and more innovative solutions [6]. We are currently exploring this tradeoff.

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.

References

- [1] J. Cagan and W. J. Mitchell. A grammatical approach to network flow synthesis. In J. Gero, editor, *Preprints of the IFIP WG 5.2 Workshop on Formal Design Methods for Computer-Aided Design*, pages 153–166, Tallinn, Estonia, 16–19 June 1993.
- [2] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, MA, 1989.
- [3] J. Grefensttete and C. Ramsey. Case-based initialization of genetic algorithms. In Stephanie Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 84–91, San Mateo, California, 1993. Morgan Kaufman.
- [4] J. Holland. *Adaptation In Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, 1975.
- [5] W.M. Jenkins. Towards structural optimization via the genetic algorithm. *Computers and Structures*, 40(5):1321–1327, 1991.
- [6] S. J. Louis. *Genetic Algorithms as a Computational Tool for Design*. PhD thesis, Department of Computer Science, 1993. Indiana University, Bloomington.
- [7] Gregory J. E. Rawlins. Introduction. In Gregory J. E. Rawlins, editor, *Foundations of Genetic Algorithms-1*, pages 1–12. Morgan Kaufman, 1991.
- [8] C. Tong and D. Sriram. In Christopher Tong and Duvvuru Sriram, editors, *Artificial Intelligence in Engineering Design, Vol. I, II, III*. Academic Press, Inc., 1992.