Trusses, NP-Completeness, and Genetic Algorithms

Authors:

Shannon Overbay, Department of Mathematics and Computer Science, Gonzaga University, Spokane, WA 99258-2615, overbay@gonzaga.edu
Sara Ganzerli, Department of Civil Engineering, Gonzaga University, Spokane, WA 99258-0026, ganzerli@gonzaga.edu
Paul De Palma, Department of Mathematics and Computer Science, Gonzaga University, Spokane, WA 99258-2615, depalma@gonzaga.edu
Aaron Brown, Department of Mathematics and Computer Science, Gonzaga University, Spokane, WA 99258-2615, abrown2@gonzaga.edu
Peter Stackle, Department of Mathematics and Computer Science, Gonzaga University, Spokane, WA 99258-2615, pstackle@gonzaga.edu

ABSTRACT

The optimization of large trusses often leads to a nearly optimal solution, rather than a truly optimal design. In fact, the problem space for truss optimization grows exponentially with the size of the truss. Using the method of problem reduction, this paper demonstrates that truss optimization is in the set of NP-complete problems. Hence, the only practical techniques for solving the truss problem are heuristic in nature. Genetic algorithms provide a viable solution for large trusses.

INTRODUCTION

The optimization of large trusses (TP) with discrete design variables is a form of constrained minimization problem [Rajan]. We define the problem like this:

- **Def. 1. Truss Optimization (TP)**: Find the minimum total volume, V, of a truss, T, with n members, and a set, A, of m discrete design variables (cross-sectional areas) that satisfy a set of prescribed constraints.

A truss's total volume is the sum of the length of each member times the member's cross sectional area:

- **Def. 2. Total Volume of a Truss (V)**:

$$V = (L_1 \times a_1) + (L_2 \times a_2) + (L_3 \times a_3) + \ldots + (L_n \times a_n) \quad (1)$$

where:

V is the volume of the truss
L_i is the length of member i
a_i, the cross-sectional area of member i, is selected from the set, A, of cross-sectional areas.

When the members are composed of discrete, rather than continuous, design variables, the objective function, in this case, the total volume, is not differentiable and so traditional calculus-based optimization methods will not apply. For large trusses of this type, optimization methods produce nearly optimal rather than a truly optimal design. This paper will show that the problem space for TP grows exponentially with the size of the truss. Such a problem cannot be computed in reasonable time. Though not a formal definition, the following serves the purposes of this paper:

- **Def. 3. Reasonable Time [Ganzerli, De Palma, Smith, and Burkhart]:** A computer's basic operations include arithmetic, comparison, memory retrieval and so forth. The execution time of these will vary. Suppose there is a computer that can carry out 10^6 of the least time-consuming operations per second. If the computer is run around the clock for a year, it will have carried out $< 10^{14}$ basic operations. Any problem that requires more than 10^{14} basic operations is unreasonable.

Further, this paper will show that, if a truly optimal solution can be found in time bounded above by a polynomial, so can the Traveling Sales Person problem (TSP) . The Traveling Sales Person problem tries to find the minimum cost tour of n cities. The tour visits each city exactly once and ends at the starting city. This is important because TSP is known to be NP-complete. The designation of NP-completeness indicates that a given problem is among the "hardest" problems known to computing and, further, that if any one of these problems can be solved in reasonable time, then all such problems can be solved in reasonable time [Garey and Johnson]. There are good reasons to believe that such a solution will not be found.

Since, for now at least, the minimal volume of a large truss cannot be found in reasonable time, investigators are forced to rely upon various heuristic solutions. These solutions satisfy the constraints, but it is not known whether a given heuristic solution is truly optimal. One of the heuristic methods that has proven particularly fruitful in recent years is the genetic algorithm (GA), an optimization technique loosely based on the idea of natural selection [Haupt and Haupt]. This paper demonstrates the use of GA on a TSP. TSP is presented, because it is an NP-complete problem for which upper and lower bounds can be calculated. The authors' version of GA produces solutions that fall within the bounds. The ability of GA to produce an acceptable solution to TSP, is incentive to apply GA to problems, like TP, for which the bounds are not known. The final section of the paper presents the authors' GA solution to the classic 64-bar truss with 64 independent variables.

TRUSSES AND EXPONENTIAL GROWTH

We begin this section with three definitions [Gary and Johnson, Horowitz and Sahni]:

- **Def. 4. O (order of):** $f(n)$ is $O(r(n))$ if there exists a constant, c, such that $|f(n)| \le c \times |r(n)|$ for all values of $n \ge 0$.
- **Def. 5. Polynomial Time Algorithm:** An algorithm is referred to as polynomial time if its time complexity function, $f(n)$ is $O(p(n))$ for some polynomial, p. n is the size of the input.

Informally, we say that an algorithm is polynomial time if the growth of its running time, that is, the number of basic operations that must be executed, is bounded above by a polynomial. Any algorithm not so bounded is said to be an exponential time algorithm. Since there is no integer q that, such that n^q bounds 2^n, any algorithm bounded below by 2^n is an exponential time algorithm.

- **Def. 6. Exponential Time Algorithm:** An algorithm is referred to as exponential time if its time complexity function, $f(n)$, is bounded below by 2^n.

For TP, the time complexity function, $f(n)$, is volume, V, where n is the number of members in the truss. The simplest possible truss is the one where only two values for design variables are in the search space, A. Clearly, to exhaustively search for minimum V, a program has to examine 2^n candidate solutions. The arbitrary truss has m design variables where $m \geq 2$. Since $f(n)$ for any truss is bounded below by 2^n, we conclude that searching for minimum V grows exponentially with the size of the truss. We can also say that V is $O(m^n)$.

This has serious consequences. Suppose there is a computer that can do 10^6 basic operations per second. To find a minimal truss, the volume of the truss must be computed, a polynomial time operation (see Definition 2), and the candidate minimum must be shown to be structurally sound. This can be done in a sequence of matrix manipulations [Wang] that are also polynomial time operations [Horowitz and Sahni]. Thus, the volume computation and the structural analysis require some fixed-multiple of basic operations as defined in Definition 3. Call it α. Then the hypothetical machine could compute $(1/\alpha \times 10^6)$ volume computations and structural analyses per second. Suppose, further, that there are 8 design variables and 64 independent members. The search space contains 8^{64} candidate minimum trusses, and so finding V would require more than $(6\alpha \times 10^{57})$ volume computations and structural analyses per year, far greater than the definition of reasonable. How much greater? Finding the minimum volume of a 64-bar truss with 8 design variables on this hypothetical computer would require more than 10^{34} billion years. Since the universe is only 13.7 billion years old [Seife], the civil engineer cannot be guaranteed an optimal solution. Even if the speed of the computer were increased by a factor of 1000, V is far beyond the boundary of the reasonable.

TRUSSES AND NP-COMPLETENESS

In fact, not only is TP intractable in the sense of requiring an unreasonable time to compute, if one could find a polynomial time solution for this problem, one could find a polynomial time algorithm for any problem in the class NP-complete. These problems include some of the most difficult problems that any engineer might encounter [Garey and Johnson]. We make the following claim:

- **Theorem 1:** The Truss Optimization Problem is NP-complete. (See Definition 9).

Three definitions and a theorem are required:

- **Def. 7. P [Horowitz and Sahni]:** $L \in P$ if L is a decision problem that can be solved by a deterministic algorithm in polynomial time.
- **Def. 8. NP [Horowitz and Sahni]:** $L \in NP$ if L is a decision problem that can be solved by a non-deterministic algorithm in polynomial time.

Part of the task will be to recast TP as a decision problem, that is, one that says 'YES' there is a sequence of steps that results in a successful truss or 'NO', there is not sequence of steps that results in a successful truss. It is well-known that many optimization problems can be reformulated as decision problems. The decision problem can be solved in polynomial time if and only if the corresponding optimization problem has a polynomial time solution [Horowitz and Sahni].

- **Def. 9. NP-completeness:** To show that a problem, L, is NP-complete, it is sufficient to show that: (1). L is in NP. (2). There is a polynomial time transformation from a known NP-complete problem, L', to L.

Now we are ready to prove that TP is NP-complete. We define an instance of TP by beginning with a truss of n bars with lengths $L_1, L_2, \ldots, L_n$, where L_i is the length of bar i, and a set $A = \{a_1, a_2, ..., a_m\}$ of m of possible cross-sectional areas for the bars of the truss. For simplicity, we may assume that all m areas are possible for each of the n bars and the m areas need not be distinct, since in proving NP-completeness, we are concerned with the worst case. Thus, our set of m^n candidate trusses with area set A is given by $U = \{(u_1, u_2, ..., u_n) \mid u_i \in A\}$. Additionally, we have a set of constrains and a function, g, that will determine in polynomial time whether a given truss satisfies the constraints. If it does, g returns TRUE, otherwise g returns FALSE.

Given a number k, we ask "For the given instance of TP, does there exist a truss of volume less than or equal to k for which g is true?" It is easy to see that TP$\in$ NP since a non-deterministic algorithm need only guess a set of areas for a candidate truss, then check to see if it satisfies the constraints and if its volume does not exceed k. Both calculations can be done in polynomial time since both g and the volume calculation have polynomial complexity.

Now we will show that TP is at least as hard as the known NP-complete problem, TSP by giving a polynomial transformation of each instance of TSP to an instance of TP. An instance of TSP is given by a set n of cities and a set D of $n(n-1)/2$ distances corresponding to the $n(n-1)/2$ distinct pairs of cities. Given an integer k, we ask "For the given instance of TSP, is there a tour that visits each city exactly once of total distance at most k?" For each instance of TSP, an answer of YES or NO should be returned. Note that the candidate solutions are the possible tours corresponding to the $n!$ permutations of the cities. Without loss of generality, we may assume that all tours begin at the first city, reducing the number of candidate solutions to $(n-1)!$.

Now we will map each instance of TSP to an instance of TP in polynomial time. Take a particular instance of TSP with n cities (vertices) and $n(n-1)/2$ distances (weighted edges) between cities. We build our instance of TP by choosing the $n(n-1)/2$ distances as our area set, giving us $A = \{a_1, a_2, ..., a_{n(n-1)/2}\}$. These areas are not necessarily distinct, but we will use different labels for each weight so that our set consists of $n(n-1)/2$ elements, each corresponding to exactly one edge of our TSP graph. Our set of bars will be size n, corresponding to the n edges (equivalently, n vertices) in each candidate solution of our instance of TSP. We will assign each bar length one, so that the volume of a given truss is the sum $V = 1 \times u_1 + 1 \times u_2 + ... + 1 \times u_n$ of the volumes of the n bars.

Finally, we need a function g that will determine in polynomial time whether a candidate truss satisfies the constraints. We will define g as follows: g takes a candidate truss $(u_1, u_2, ..., u_n)$ and returns TRUE if the given sequence of areas corresponds to a candidate solution of the given instance of TSP and FALSE otherwise. In other words, starting with the first city, is there a permutation of the cities so that the sequence of weighted edges on the candidate tour matches $(u_1, u_2, ..., u_n)$? This can be done in polynomial time, since we start with the first city and check to see if it is at distance u_1 from any other city (i.e. is there any edge incident with the first vertex that is labeled u_1). We would only have to check at most $(n-1)$ edges to see if u_1 is there. If not, stop checking and return FALSE. If so, continue to the second city (the other endpoint of the unique edge labeled u_1). Repeat the process, searching for an edge labeled u_2 with the second city as an endpoint. If one is found, continue, otherwise return FALSE. Keep going until FALSE is returned or until u_n is reached. If there is an edge back to the first vertex of length u_n, return TRUE, otherwise return FALSE. Since fewer than n edges need to be checked for each of the n vertices along the tour, g clearly has polynomial complexity.

To complete the proof, we must show that an instance of TSP responds YES if and only if the corresponding instance of TP responds YES. Given an integer k, suppose that an instance of TSP has a tour of distance at most k. Then, in the corresponding instance of TP, this tour of length at most k would correspond to a viable truss of volume at most k. Hence, this instance of TP would also respond YES. Similarly, if an instance of our corresponding TP resulted in a YES, then it would represent a viable truss of volume at most k. Thus, it would represent a possible tour in the given instance of TSP and would have weight (total distance) at most k. This completes the proof.

TRUSSES AND GENETIC ALGORITHMS

Since TP is an NP-complete problem, structural engineers will have to be satisfied with near-optimal as opposed to truly optimal trusses. One promising heuristic is the genetic algorithm (GA) [Haupt and Haupt]. There have been many applications of GA to problems in structural engineering over the last decade [Burns]. Some examples in the literature produced with GA compare favorably to published minimums obtained using traditional methods [Rajeev and Krishnamoorthy], [Ghasemi and Wood]. In this section, we introduce GA as a useful heuristic for intractable problems. We first apply it to TSP, then show results for the 64-bar truss with 64 different variables.

GA is based loosely on the notion of natural selection [Eiben and Smith]. Some individuals within a population possess better survival characteristics. As such they reproduce and pass these characteristics to their children. Over time, a population becomes adapted to a given environment. GA can be expressed quite compactly as a simple while-loop with calls to appropriate functions:

```
GA(Integer: parameters)
        {
            Population: population;
            population = GeneratePop(parameters);
            Sort(population);
            while(population has not converged on a good-enough solution)
                    {
                       Pair(population);
                       Mate(population);
                       Mutate(population);
                       Sort(population);
                     }
         }
```

Though there are many definitions of convergence, the authors' genetic algorithm stops when *P* % of the population is within *Q* standard deviations of the population mean. The function Pair determines which individuals in a population mate. Mate is the analog to recombinant DNA: each of two parents passes certain of its characteristics onto each of two children. GA is inclined to get stuck in local minima. To combat this, GA introduces random mutations into the population at every generation.

GENETIC ALGORITHMS AND THE TRAVELING SALES PERSON PROBLEM

It is well-known that TSP has wide application, including computer wiring, job sequencing [Garfinkel], machine tool sequencing [Tucker] and vehicle routing. Since TSP is known to be NP-complete, it is probable that an optimal solution cannot be found in reasonable time. Nevertheless, TSP is interesting, because it is possible to compute upper and lower bounds. If GA produces results that fall within these bounds, it gives confidence that GA will produce good, though probably not perfect, solutions to a problem without known upper and lower bounds.

We will now derive bounds for TSP. Suppose we have the graph shown in Figure 1.

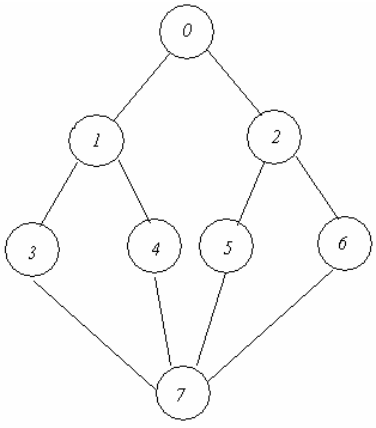

FIGURE 1
8-VERTEX GRAPH

A traversal is an ordering of vertices such that each vertex appears exactly once. The tree shown in Figure 2 is the graph from Figure 1 having undergone a depth-first traversal. Clearly, there are 8! possible traversals of this graph. A tree containing all vertices is called a spanning tree. Using Prim's algorithm [Cormen, Leiserson, and Rivest], we can construct a minimum cost spanning tree in time O(|*E*| lg |*V*|) where |*E*| is the number of edges and lg |*V*| is the log base 2 of the number of vertices. It is easy to show that this is bounded above by a polynomial, so, by Definition 5, we can compute a minimum cost spanning tree in polynomial time. The algorithm is as follows: start at any vertex, *v*. Select the least cost edge that is not already part of the tree. Continue adding edges by selecting the least-cost edge until the tree spans all the vertices in the original graph. Consider Figure 3. For simplicity, it shows only a sub-graph of the complete graph implicit in Table 1.

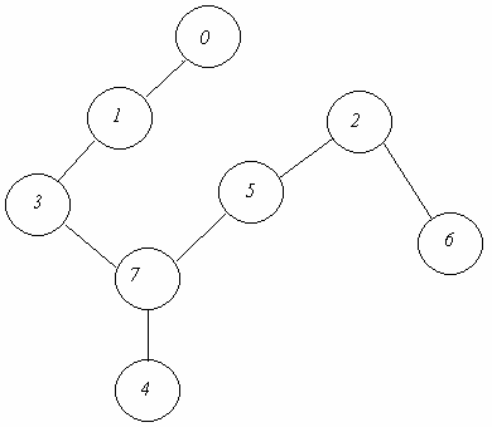

FIGURE 2
TREE RESULTING FROM DEPTH-FIRST TRAVERSAL OF GRAPH IN FIGURE 1

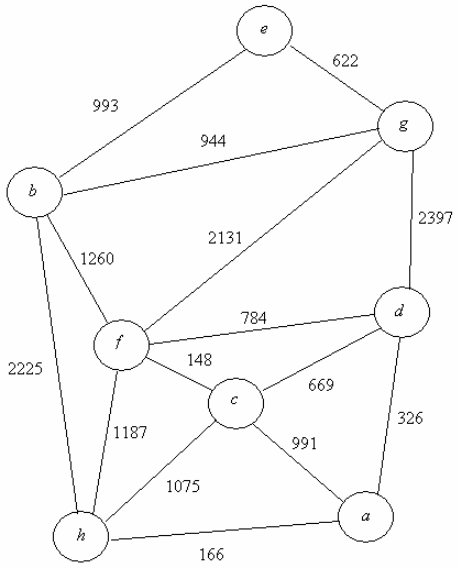

FIGURE 3
SUBGRAPH OF GRAPH IMPLICIT IN TABLE 1

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*
a	0	2080	991	326	2086	1080	2518	166
b	2080	0	1407	1883	993	1260	944	2225
c	991	1407	0	669	1840	148	2184	1075
d	326	1883	669	0	1965	784	2397	407
e	2086	993	1840	1965	0	1787	622	2265
f	1080	1260	148	784	1787	0	2131	1187
g	2518	944	2184	2397	622	2131	0	2697
h	166	2225	1075	407	2265	1187	2697	0

TABLE 1
COMPLETE TABLE OF COSTS BETWEEN NODES AND THE COST OF VARIOUS TRAVERSALS

Figure 4 shows a minimum spanning tree that results from Prim, starting at vertex *a*.

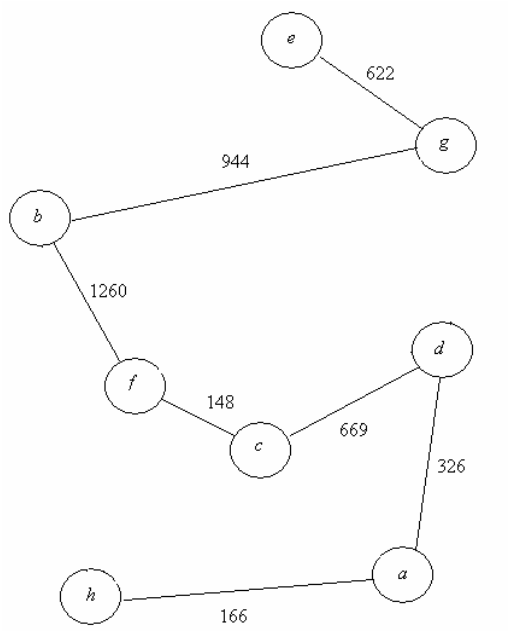

FIGURE 4
MINIMUM SPANNING TREE FOUND BY PRIM'S ALGORITHM

Next do a preorder traversal of the tree rooted at *e*, recording all vertices visited. This produces the sequence *e g b f c d a h a d c f b g e*, called a full walk. Let p^* be the optimal tour of the graph. Let *t* represent a minimum spanning tree. If *n* is the number of vertices in the original graph, the minimum spanning tree has $n - 1$ edges. Since the tour required by TSP of a graph with at least three vertices has *n* edges, $\text{cost}(t) \leq \text{cost}(p^*)$. Notice that the full walk traverses every edge exactly twice, once as the traversal descends the tree, and once as the traversal returns from the descent. If *w* represents the full walk, we can conclude that $\text{cost}(w) = 2 \times \text{cost}(t)$ and $\text{cost}(w) \leq 2 \times \text{cost}(h^*)$. So the cost of *w* is within a factor of 2 of the cost of the optimal tour.

But there is a problem that is easy to fix. *w* is not a tour since it visits some vertices more than once. Eliminate any node in the walk already visited except the first giving: *e g b f c d a h e*. Let the cost of going from vertex *u* to vertex *v* be $\text{cost}(u,v)$. Given a complete graph, one of the problem conditions, $\text{cost}(u,v) \leq \text{cost}(u,y) + \text{cost}(y,v)$. Notice

that this ordering is just the ordering of a preorder traversal. Let p be this circuit. We can conclude, then, that cost(p) ≤ cost(w) ≤ 2 × cost(p*). This is referred to as the triangle inequality of TSP [Johnson and Papadimitriou]. p is the circuit that produces the lowest known upper bound. In this example cost(p) = 6400. And, of course, cost(p*) ≤ cost(p). If GA produces a solution, s, such that cost(s) ≤ cost(p), we can conclude that cost(s) is less than or equal to the lowest known upper bound.

Table 2 shows the cost of the tour for the graph given in Figure 3.

Run	Number of Generations	Circuit Produced	Cost of Circuit (cost(s))
1	889	*h a d c f b g e*	6400
2	2579	*e g b f c d a h*	6400
3	1288	*h a d c f b g e*	6400
4	4929	*e g b f c d a h*	6400

TABLE 2
RESULTS OF 4 RUNS OF GA

Shown are the costs of s for four runs of the genetic algorithm along with the tours generated. Notice that every run of GA constructs tours that are equal to cost(p). Further, every tour is identical except for the starting vertex, suggesting that 6400 is a local minimum. Whether it is a true minimum, of course, is not known. Since the goal of this exercise was merely to show that GA performs within acceptable limits, a single mutation rate along with easy-to-implement mating and pairing algorithms were used.

THE 64-BAR TRUSS

Having shown that the Truss Optimization Problem is NP-complete and that GA is a useful technique for intractable problems, we apply GA to the 64-bar truss. See Figure 5. The truss is composed of aluminum, with a Young's modulus of 1.0×10^5 psi. Adjacent nodes are 200 in apart in the horizontal and vertical directions. Nodes 21, 22, 27, and 28 have no degrees of freedom but all other nodes have two. Two designs are proposed. In one case the design variables represent the member cross-sectional areas and constitute 64 independent variables. In the other case, the 64 variables are linked so that they are reduced to only 19 independent variables. The linking is shown in Figure 6. A set of members is represented by one variable in this case, and each member in the set will have the same value. Set 1, for example, includes members 1-3, 3-5, 2-4, and 4-6. In addition, the truss is solved using integer values. The design variables, m, are limited to the range 1 $in^2 \le m \le 32$ in^2 for integer values. The constraints on the truss are as follows: for displacement, nodes 1 (vertical) and 9 (horizontal) are limited to 10 in; for stresses, no member's stress may exceed 25 ksi in either tension or compression. The loads are imposed simultaneously.

The volumes obtained are 31840 in^3 for the 64 independent variables and 43090 in^3 for the 19 linked variables. The 64 independent variables give a more detailed description of the design. They have lower volume than the design solved with linked variables. However, the computational cost is higher for the 64 independent variables because the genetic algorithm converges at a slower rate.

The 64-bar truss was previously solved with linked variables in [Schemit and Lai]. Some variations from [Schemit and Lai] in the application of the load condition were made in this paper. The authors have also solved the 64-bar truss with the same criteria as [Schemit and Lai] and have obtained results that compare remarkably with the literature. This comparison is not presented here in detail for brevity.

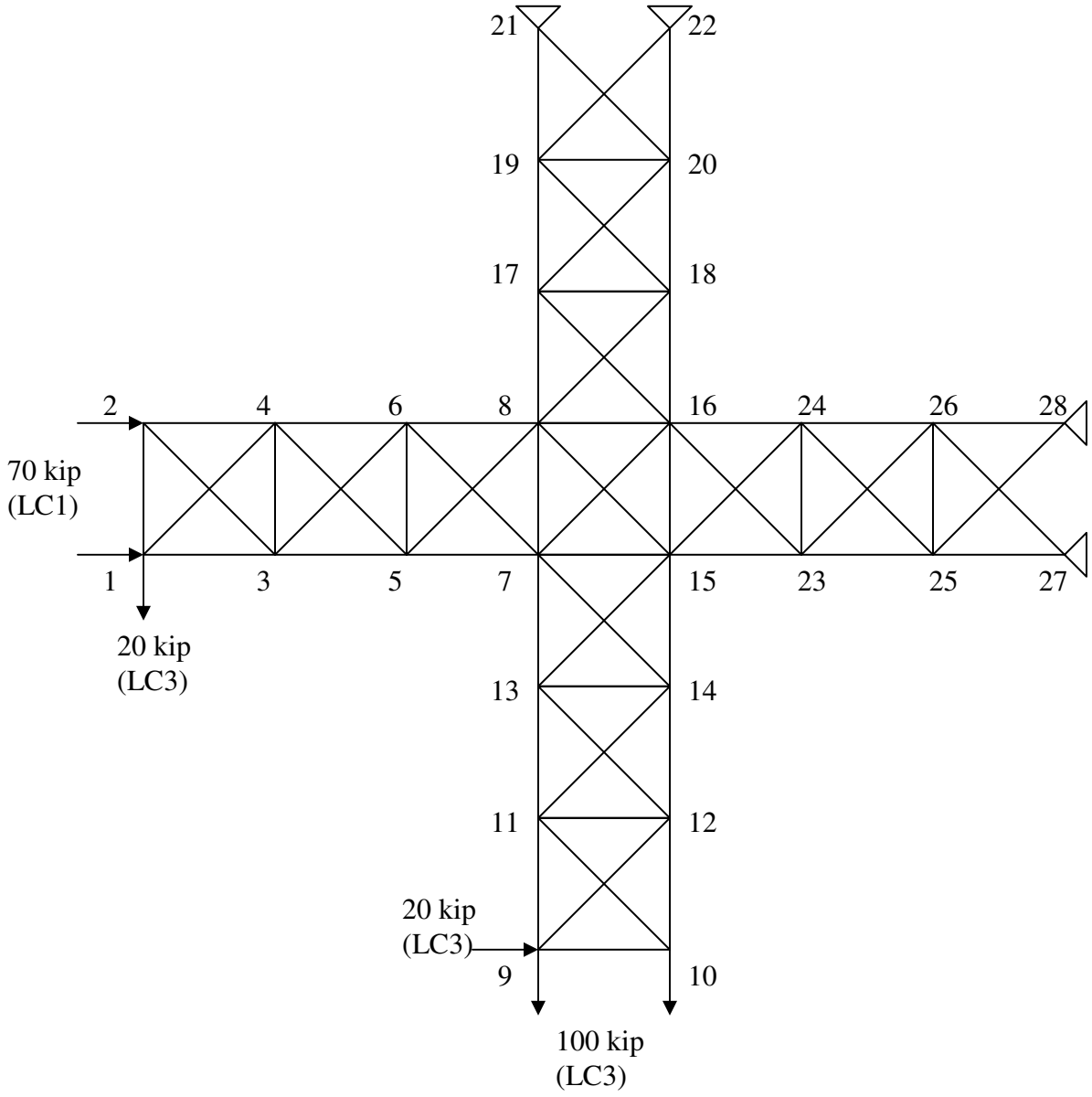

FIGURE 5
64-BAR TRUSS. NODE NUMBERING AND LOAD CONDITIONS

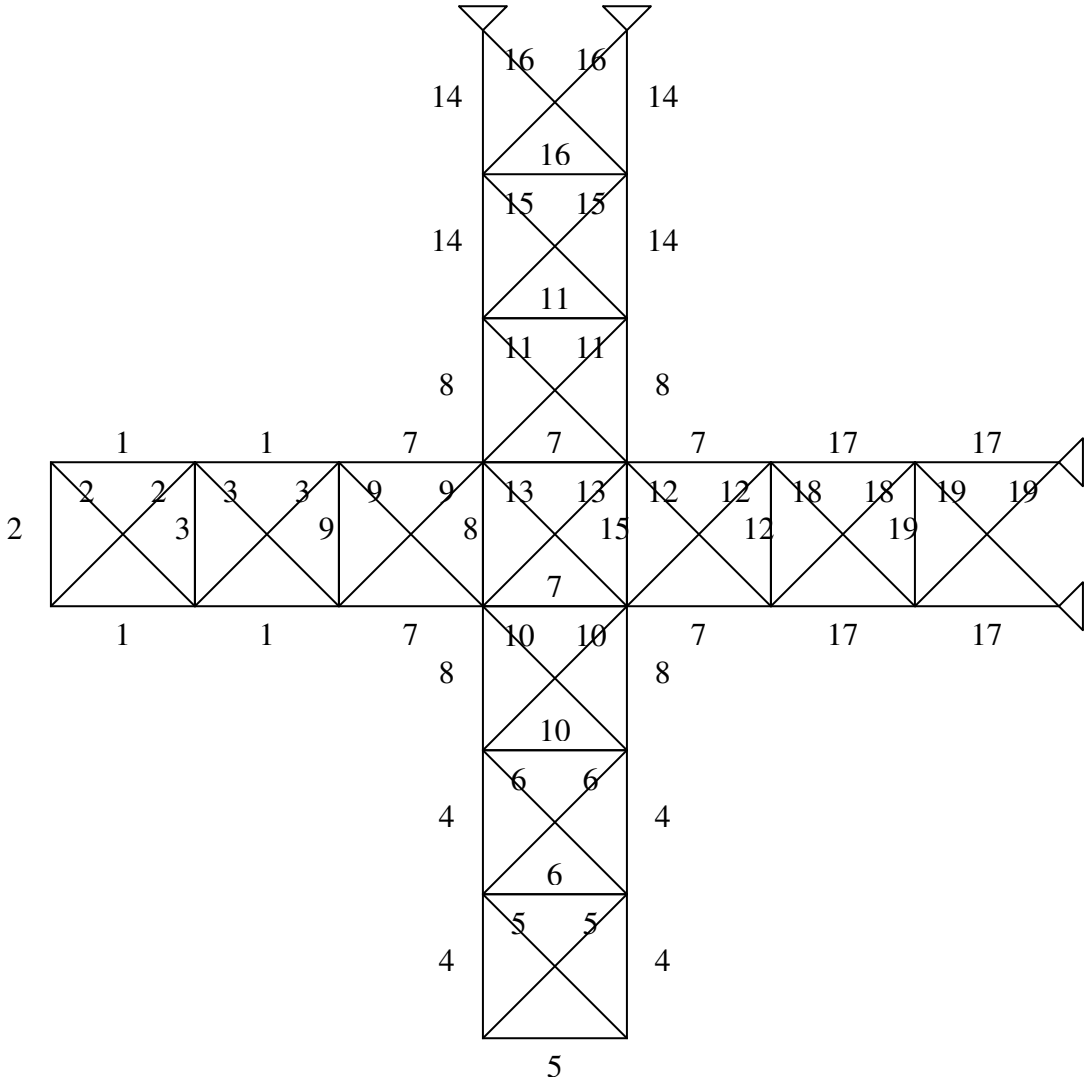

FIGURE 6
64-BAR TRUSS. DESIGN VARIABLE LINKING

CONCLUSIONS

In this paper we have demonstrated that the truss optimization problem, TP, is NP-complete. The importance of this result is that a polynomial time solution of this problem would imply a polynomial time solution to some of the most difficult problems in computing. For this reason, we assume that the probability of finding such a solution is low. Therefore, practitioners must be satisfied with heuristic methods that produce solutions that cannot be shown to be optimal. One promising heuristic is the genetic algorithm. To demonstrate the usefulness of this approach to an intractable problem of known bounds, we derived upper and lower bounds to the Traveling Sales Person problem. We demonstrated that a relatively unsophisticated implementation of GA can produce solutions within the bounds. Finally, we showed our results for a 64-bar truss. Two facts lead us to trust these results: one, they are close to published results, and two,

the technique that produced them, the genetic algorithm, also produced results within known bounds for a well-understood intractable problem, TSP. The authors have been applying GA to structural engineering considering uncertainties [Ganzerli, De Palma, Stackle, and Brown]. The demonstration that TP is NP-complete shows just why heuristic solutions like GA are important.

ACKNOWLEDGEMENT

Funding for this project has been provided by the McDonald Work Award. Robert and Claire McDonald have generously established this grant to the benefit of undergraduate students at Gonzaga University.

REFERENCES

[1] Burns, S., ed., *Recent advances in optimal structural design*, Institute of the American Society of Civil Engineers, Reston, VA, 2002.

[2] Cormen, T., H., Leiserson, C., E., Rivest, L., R., *Introduction to algorithms*, MIT Press, Cambridge, 1991.

[3] Eiben, A., E., Smith, J., E., *Introduction to evolutionary computing*, Springer, New York, 1998.

[4] Ganzerli, S., De Palma, P., Smith, J., D., Burkhart, M., F., "Efficiency of genetic algorithms for optimal structural design considering convex models of uncertainty", *Proceedings of the Ninth International Conference on Statistics and Probability in Civil Engineering*, Millpress Science Publishers, Rotterdam, NL, 2003, 1003-1010.

[5] Ganzerli, S., De Palma, P., Stackle, P., Brown, A., "Info-gap uncertainty in structural optimization via genetic algorithms", *Proceedings of the Ninth International Conference on Structural Safety and Reliabilty*, Millpress Science Publishers, Rotterdam, NL, 2005, 2325-2330.

[6] Garey, M., R., Johnson, D., S., *Computers and intractability: a guide to the theory of NP-completeness*, W. H. Freeman, San Francisco, 1979.

[7] Garfinkel, R., S., "Motivation and modeling", in *The Traveling Salesman Problem,* Lawler, E., Lenstra, J., Rinnoy Kan, R., Shmoys, D., B. (eds.), John Wiley and Sons, New York, 1985.

[8] Ghasemi, M.R., Hinton, E., and Wood, R.D., "Optimization of Trusses Using Genetic Algorithms for Discrete and Continuous Variables", *Engineering Computations (Swansea, Wales).* MCB Univ. Press Ltd., Bradford, Engl. Vol. 16, 1999, 272-301.

[9] Haupt, R., L., Haupt, S., E., *Practical genetic algorithms*, John Wiley and Sons, New York, 1998.

[10] Hopcraft, J., E., Rajeev, M., Ullman, J., D., *Introduction to Automata Theory, Languages and Computation*, Addison Wesley, Boston, 2001.

[11] Horowitz, E., Sahni, S., *Fundamentals of computer algorithms,* Computer Science Press, Rockville, MD, 1978.

[12] Johnson, D., Papadimitriou, "Performance guarantees for heuristics", in *The Traveling Salesman Problem,* Lawler, E., Lenstra, J., Rinnoy Kan, R., Shmoys, D., B. (eds.), John Wiley and Sons, New York, 1985.

[13] Rajan, S., D., *Introduction to structural analysis and design*, John Wiley and Sons, New York, 2001.

[14] Rajeev, S., Krishnamoorthy C.S., "Genetic algorithms-based methodologies for design optimization of trusses," *ASCE J. of Structural Engineering*. Vol. 123 (No 3), 1997, 350-358.

[15] Seife, C., "Illuminating the dark universe", *Science,* Vol. No 302 (5653), 2003, 2038-2039.

[16] Schemit, L., A., Lai, Y., C., "Structural optimization based on preconditioned conjugate gradient analysis methods", *International Journal for Numerical Methods in Engineering,* Vol. No 37 (6), 1994, 943-964.

[17] Tucker, A., *Applied Combinatorics*, John Wiley and Sons, New York, 2002.

[18] Wang, C., K., *Structural Analysis on Microcomputers*, Macmillan Publishing Company, New York, 1986.