STATISTICAL CHARACTERIZATION OF NP-COMPLETE PROBLEMS

Jeffrey L. Duffany, Ph.D.
Universidad del Turabo
Gurabo, PR 00778
jduffany@suagm.edu
(787) 743-7979 x4153 (office)
(787) 744-5476 (fax)

conference: FCS '08
keywords: np-complete, complexity, algorithm

ABSTRACT

NP-complete problems fall into a continuum of difficulty and an attempt is made to identify characteristics that account for this difference. The approach is based on the NP-hard compatibility problem, also known as a system of inequations, which can used as a basis for characterizing the subset of NP-complete problems. First a benchmark for difficulty is developed based on thenumber of iterations required to find an optimal solution. It is then attempted to relate the statistical properties of the problem to the difficulty level. The main area of focus is in relating the difficulty of solving a particular system to how constrained the system is. The results show that the least difficult problems are either highly overconstrained systems or highly underconstrained while the most difficult are in the region of being slightly underconstrained.

1 INTRODUCTION

This paper is concerned with the statistical characterization of NP-complete problems[4,5,7]. The approach is to start with the well-known compatibility problem[10] which is one of the best known examples of an NP-hard problem[3,6] and use that problem as a basis for studying the NP-complete problems. The compatibility problem can be stated as follows. Suppose there are n objects each of which are incompatible which some subset of the other n-1 objects. The problem is to partition all n objects into a set of k equivalence classes such that no object is incompatible with any other object in its equivalence class and where k is minimum over all possible partitionings. The problem is usually stated in terms of an adjacency matrix (A) of ones and zeros which summarizes the compatibility of each object (variable x_i) with every other object (variable x_j). A one in the (i,j) element of the A matrix indicates incompatibility between variables x_i and x_j while a zero represents compatibility. A solution vector (s) is a mapping of each variable x_i into an integer such that $1 <= x_i <= k$ while ensuring that $x_i \neq x_j$ when $A[i,j] = 1$. This problem is also known as a system of inequations[1,8,9].

One way of characterizing these systems of inequtions is to calculate the ones density (constraint density) of the A matrix. This can be defined as the ratio of the number of ones in the A matrix to the maximum possible and this quantity varies between 0 and 100 percent. Another way is to calculate the number of zeros in the block diagonal of an optimal solution (if the rows and columns of the A matrix are permuted according to an optimal solution vector (s) the result is a block diagonal form of A). Yet another way is to look at the ratio of the number of zeros outside the block diagonal to the number of zeros inside the block diagonal (z_o/z_i). For complete k-partite systems this ratio is zero. Complete k-partite systems are the least difficult of all compatibility problems as they can be solved by any algorithm with a success rate of 100%. Based on this observation, the most difficult problems might be related to the those having the highest ratio of z_o/z_i. What is needed is a method to estimate the complexity of a given problem. This can be done using an algorithm to find an optimal solution and using the number of iterations as a measure of complexity.

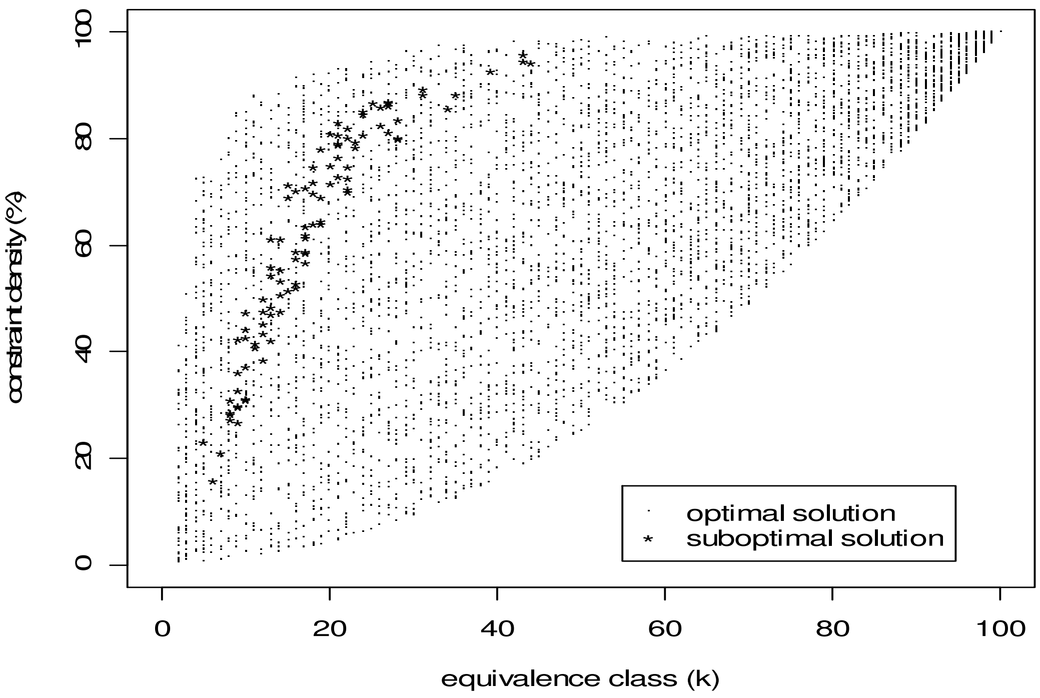

Figure 2. Layers of NP-complete problem complexity

2 Method for Estimating Complexity of NP-complete problems

It has been shown[1] that a polynomial time algorithm similar to Gaussian elimination (Figure 1) can be used to solve the compatibility problem. The algorithm starts with a solution vector which has an initial value of s = (1,2,3,...n). The algorithm squares the adjacency matrix A and finds the maximum value for pairs of variables that can be combined (i.e., A[i,j]=0). It then combines variables x_i and x_j by taking the constraints that are in x_j but not in x_i and adding them to x_i. Then it updates the solution vector s[j]=s[i] and eliminates variable x_j as in Gaussian elimination. The matrix A is reduced by one in dimension each time a variable is eliminated. The main difference with Gaussian elimination is that the algorithm ineq uses logical OR instead of subtraction and the decision function is $f(A)=\max(A^2)$. Note the algorithm ineq is recursive and stops when there is no longer any variables left to combine.

```
ineq(A)
ij<-max(A^2)
xi=xi|xj
s[j]=s[i]
A=A[-j,-j]
ineq(A)
```

Figure 1 Algorithm for solving systems of inequations

The algorithm ineq was used to solve randomly generated systems across the full range of constraint densities from 0% to 100% for 5000 systems of n=100 variables and optimal solution cardinalities ranging from 1 to 100. The result is shown in Figure 2. The points represent systems where optimal solutions were found and the asterisks represent systems where an optimal solution was not found. A system of inequations in Figure 2 is represented by its constraint density and solution cardinality.

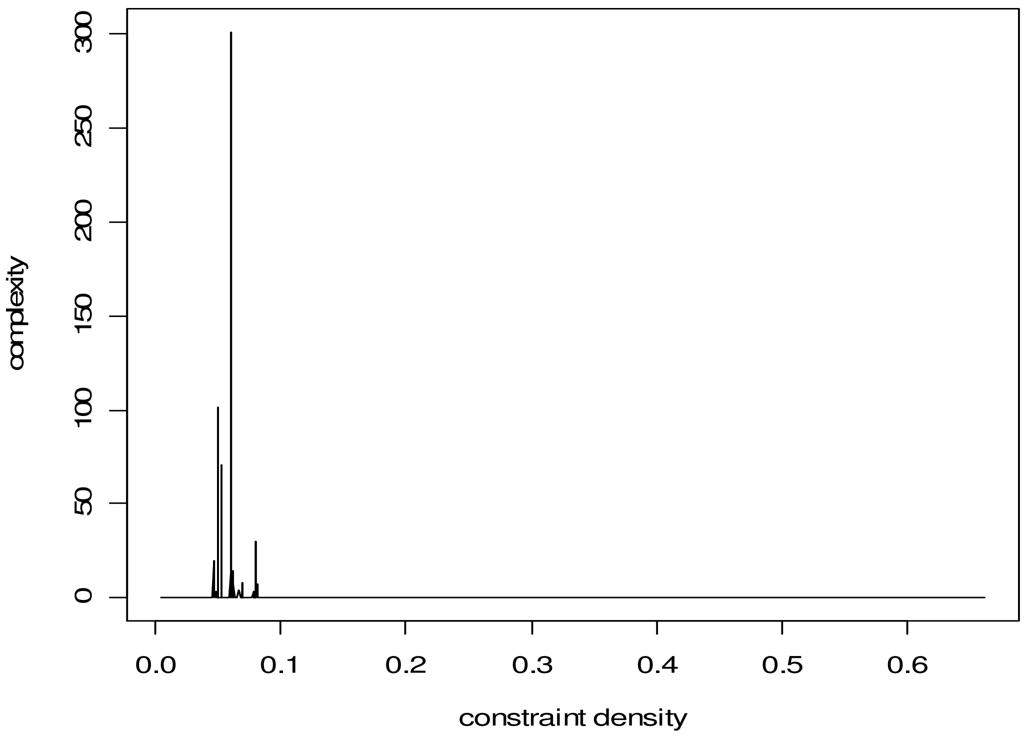

Figure 3. Complexity for optimal solution cardinality k=3 (NP-complete problems)

One way to interpret Figure 2 is that the ineq algorithm has partitioned the system space into two layers of difficulty. The more difficult problems represented by the asterisks in Figure 2 have themselves been separated into two parts, one less difficult and one more difficult. The less difficult part is solved by ineq itself and the more difficult part has been reduced to a search of the very beginning of the decision tree. The area covered by points and asterisks in Figure 2 represent the entire NP-hard compatibility problem space while the subset of NP-complete problems is represented by the vertical line at k=3 which has a range of constraint density from 0 to 2/3.

Figure 3 shows the result of using ineq on 1000 systems of k=3 and n=100 (i.e., the subset of NP-complete problems). In Figure 3 the constraint density is on the horizontal axis while the vertical axis represents problem difficulty which is also referred to as problem complexity or the number of iterations of the algorithm before reaching an optimal solution. The algorithm solved 100% of the problems although some took longer than others. The most difficult problems were located in a small range of constraint densities between about 4 and 8 percent. Outside of that region the algorithm found a solution 100% of the time in 0 iterations. Inside the region the algorithm's success rate percentage for 0 iterations dropped to about 65%. Most of systems were solved in less than about 20 iterations while one required about 300 iterations, a rather long-tailed distribution (see Figure 5 for a histogram of this distribution for a sample of 200 systems taken in the 4-8% constraint density region).

A possible explanation for this behaviour is as follows. For high constraint densities there is only one optimal solution and all systems are close in some sense to complete k-partite which are the least difficult to solve. For very low constraint densities there are a very large number of optimal solutions so it is difficult for the decision function to make a mistake. A system can be called perfectly constrained if it has one optimal solution and the removal of a single constraint would lead to more than one optimal solution. Originally it was suspected that these perfectly constrained systems were the most difficult to solve.

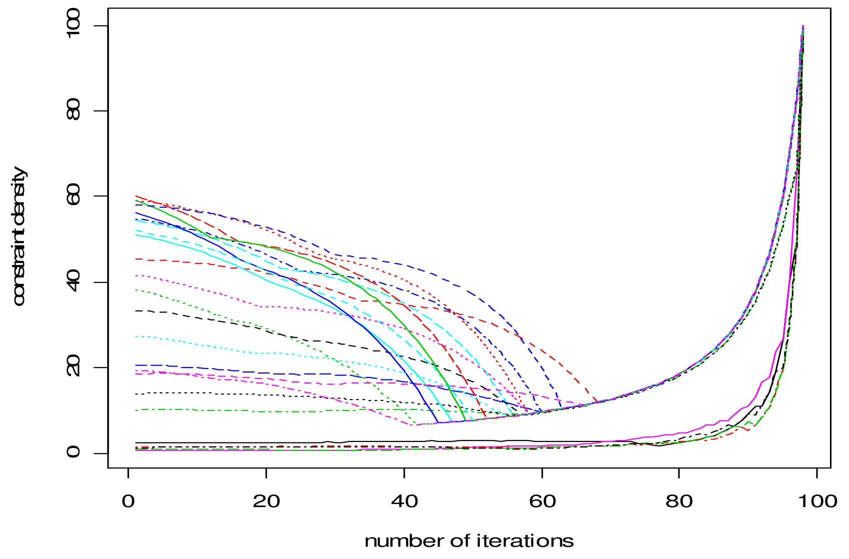

Figure 4. Ensemble of constraint density trajectories

However, tests run on systems that are difficult to solve has shown that the most difficult problems appear to be those that are not perfectly constrained but somewhat underconstrained. It is now believed that the most difficult problems may be in an area where the ratio of the number of suboptimal solutions of cardinality k+1 to optimal solutions of cardinality k reaches a maximum. Intuitively this can be explained by considering that the most difficult problems (for k=3) have a fairly low one's density (between about 4 and 8 percent). This means that average number of constraints for each variable is also very small (somewhere between 4 and 8 since n=100). In this situation it is very difficult for any decision function to determine which pair of variables to combine to create the basis of an equivalence class. The entire system looks like some sort of a "plasma" of low degree vertices each of which look like they could belong to many different equivalence classes.

It has already been stated that the ineq algorithm works by elimination of variables. Each time the matrix A is reduced in dimension by one it changes in constraint density until no more variables can be combined (where the constraint density is 100%). The constraint density is easily calculated at each step of the ineq algorithm and the result is shown in Figure 4 for an ensemble of 30 NP-complete systems of optimal solution cardinality k=3. Two different types of trajectories are seen. Below densities of 4% there are relatively few constraints (ones) in the system and eliminating variables reduces both ones and zeros about the same rate. This occurs until a certain point is reached where the constraint density increases rapidly up to a final value of 100%. For systems with constraint density above 8% the constraint density usually decreases until a certain point and then increases up to a final value of 100%. These high constraint density systems also appear to merge almost into a single trajectory when beginning their ascent phase.

The ineq algorithm works by reducing ones density in regions significantly above 8% thus leading in the direction towards theoretically easier problems. Note that for n=100, k=3 it requires the decision function $f(A) = \max(A^2)$ to make 97 consecutive correct decisions to find an optimal solution. This is represented by the number of iterations variable on the horizontal axis. Any mistake, if made, is most likely to be in the first few steps. After that it is believed that there are usually so many suboptimal solutions of cardinality k+1 that it is unlikely to make more than one error in the solution of a single problem.

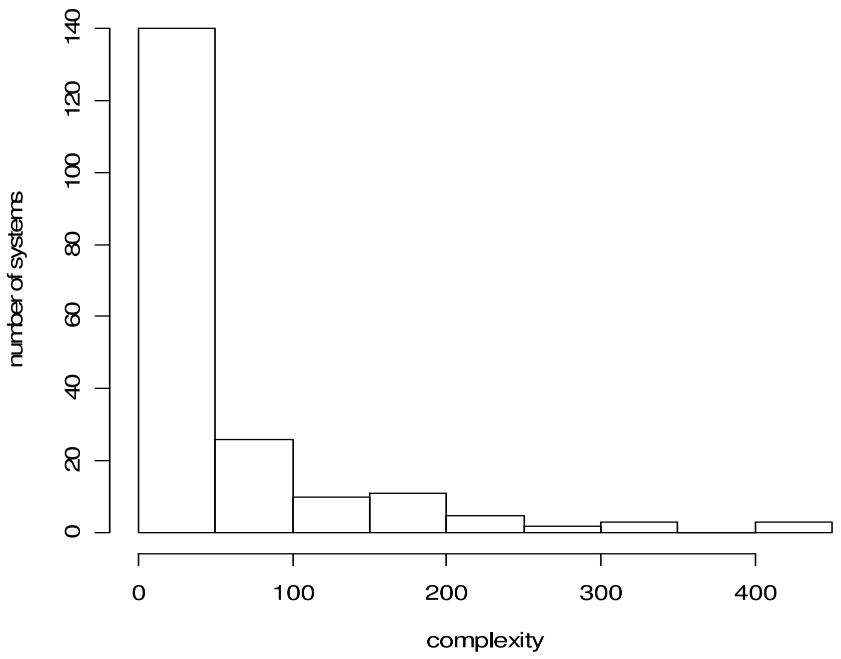

Figure 5 Complexity Distribution in 4-8% Region

There is a technique that can find which first few decisions to make to find an optimal solution with relative efficiency. It is based on the fact that inside the region of constraint density between 4% and 8% the success rate of ineq does not decrease to zero. This means that somewhere near every system A that cannot be solved directly is another equivalent system A' that can be solved directly. In many cases combining a single pair of variables is enough to move the original system to a system that is in the solution space of the ineq algorithm. This leads to the equivalence class subset algorithm[1][2] (Figure 6).

```
find a solution s to system A using ineq
choose a subset of variable associations from  s
combine these variables  to create the A' matrix
find a solution to A' using the ineq algorithm
```

Figure 6 Equivalence Class Subset Algorithm

This equivalence class subset algorithm in Figue 6 is based on the observation that any solution vector s will partition the zeros of the A matrix into two groups: those inside the block diagonal and those outside the block diagonal. Solutions with more zeros on the main diagonal are more likely to be closer to an optimal solution. Each time a solution is found the number of zeros on the block diagonal are calculated. If this number is more than any previous solution then use this solution s to take the next subset. If not continue taking subsets from the previous solution vector.

The equivalence class subset algorithm was used to focus only on the region in between the 4 and 8 percent constraint densities. A random sample of 200 systems was generated in this constraint density range and the range of complexity is shown in Figure 4. Note there is a wide variation in the complexity of the individual problems ranging from 0 iterations (no search of the decision tree) to about four hundred iterations (searching a small portion of the decision tree). The number of iterations required to find an optimal solution is therefore used as a measure of complexity for an individual problem.

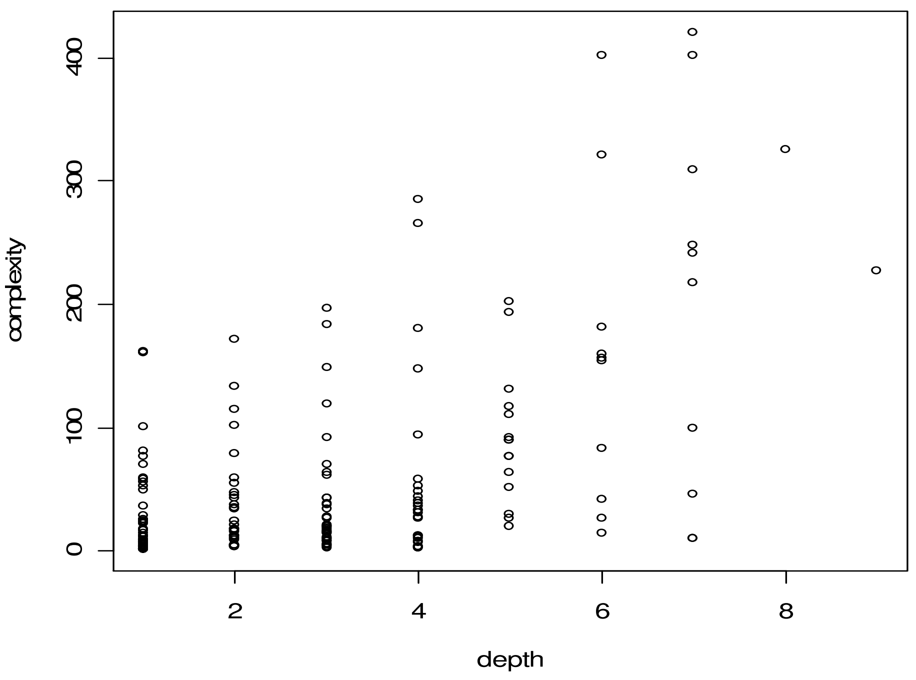

Figure 7. Search Depth vs. Solution Complexity

3 Statistical Characterization of NP-Complete Problems

One of the primary goals of this investigation is to be able to view all NP-complete problems from a broad enough perspective to understand the nature of a solution algorithm that can work over the entire range of problem complexity. This investigation attempts to characterize NP-complete problems in terms of a variety of system characteristics such as:

- Constraint density
- Number of optimal solutions
- Ratio of zeros inside to outside block diagonal
- Solution cardinality
- Number of iterations (complexity)
- Search depth

The search depth is the number of variables that were combined to transform the A matrix into an A' matrix that is in the solution space of the ineq algorithm. The solution complexity shown in Figure 5 was plotted against the corresponding search depth and the result is shown in Figure 7 above. The greatest search depth (d) was d= 9 for a system that had a solution complexity of 200 iterations. The system with the highest complexity of just over 400 iterations required a search depth of d = 7. There appears to be a trend in Figure 7 that the most difficult systems all had relatively high search depths. Note that these results should be interpreted with some caution because a solution at a certain search depth does not necessarily mean that that was the minimum search depth required for that system. There could have been an optimal solution at a lower search depth that was not found by the algorithm. A worst case complexity of 400 attempts to a depth of 9 as seen in Figure 7 represents only a tiny fraction of the billions of possible paths through the first 9 levels of the decision tree.

4 SUMMARY AND CONCLUSIONS

An investigation was performed to develop insight into the continuum of difficulty that is observed in NP-complete problems ranging from least difficult to most difficult. The result of the investigation showed that the complexity of NP-complete problems can be divided into two layers depending on whether or not a search of the first few branches of the decision tree is required. The solution algorithm involving the decision function $f(A) = \max(A^2)$ was seen to separate the problem space into two subclasses: least difficult and more difficult. In addition, it was able to separate the more difficult problems into two separate subproblems: a least difficult subproblem and a more difficult search of the first few steps of the full decision tree. Complexity of an individual problem was then defined in terms of the number of iterations it took in the search of the first few steps of the decision tree to find an optimal solution. A relationship was observed between the constraint density of a problem and its difficulty. The most difficult problems appear to be somewhat underconstrained where a decision function can easily make an error. The variation in complexity seen in NP-complete problems was also seen to be correlated to the depth of search required to find an optimal solution. Systems of high complexity which require many iterations tend to require also a high search depth which is a measure of how close or how far away that system is to some other system that is solvable by the decision function $f(A) = \max(A^2)$.

REFERENCES

1. Duffany, J.L. “Systems of Inequations”, 4th LACCEI Conference, Mayaguez, PR, June 21-23, 2006.

2. Duffany, J.L. “Generalized Decision Function and Gradient Search Technique for NP-Complete Problems”, XXXII CLEI Conference, Santiago Chile, August 20-23, 2006.

3. E. Horowitz, S. Sahni, “Fundamentals of Computer Algorithms”, Computer Science Press, Maryland, 1978.

4. S. Cook, “The P vs. NP Problem”, Journal of the ACM, Vol. 50, No. 1, January 2003, pp. 27-29, 2003 ACM 0004-5411/03/0100-0027 $5.00.

5. R. M. Karp, “Reducibility Among Combinatorial Problems”, In Complexity of Computer Computation, pages 85-104. Plenum Press, New York, 1972.

6. Weisstein, E. W., "NP-Hard Problem." From *MathWorld*--A Wolfram Web Resource. http://mathworld.wolfram.com/NP-HardProblem.html

7. Weisstein, E. W., "NP-Complete Problem." From *MathWorld*--A Wolfram Web Resource. http://mathworld.wolfram.com/NP-CompleteProblem.html

8. Weisstein, E. W., "Inequation." From *MathWorld*--A Wolfram Web Resource. http://mathworld.wolfram.com/Inequation.html

9. Wikipedia - Inequation page: http://en.wikipedia.org/wiki/Inequation

10. Clay Mathematics Institute, Millenium Prize Problems, "P vs. NP", http://www.claymath.org/millenium/P_vs_NP, 2005.