# Generalized Decision Function and Gradient Search Technique for NP-Complete Problems

**Jeffrey L. Duffany, Ph.D.**
Universidad del Turabo
Gurabo, PR USA 00778
jduffany@suagm.edu

## Abstract

*A method is described for solving np-complete problems by using a power-series expansion to represent a generalized decision function. The coefficients of the terms in the power-series are then used in a gradient search to find an optimal solution. Several factors are investigated including the effect of the number of terms used in the decision function, the initial region of space selected for the search and the overall amount of computation required.*

**Keywords:** algorithm, complexity, np-complete, inequation

## 1 Introduction

The compatibility problem [1] is one of the best known and classic examples of np-complete problem. The problem can be represented as a nxn matrix A of ones and zeros with each row or column of the matrix representing one of the n variables $(x_1,x_2,, \ldots x_n)$ each of which is either compatible or incompatible with each of the other variables. A solution can be represented as a vector $s = (s_1,s_2,, \ldots s_n)$ where each $s_i$ is an integer. This is known to be an np-complete and np-hard problem [5] [6][7]. There are generally a large number of feasible solutions to this class of problem but a solution s* is called optimal only if the number of distinct elements (k) in the solution vector s is minimum over the set of all solutions {s}. Any single incompatibility can be represented by an inequation[8]. The general compatibility problem or "system of inequations" as it is referred to in [2] can be represented by a symmetric square matrix A of 1s and 0s with 1 representing a constraint or incompatibility and a 0 representing a compatibility or lack of constraint. The matrix A is a representation of the system of inequations or the incompatibility between each of the n variables $x_i \neq x_j$. As described in [2], a system of inequations can also be represented by inequation (1):

$$Ax \neq x \tag{1}$$

An algorithm for solving the general compatibility problem of equation (1) in exponential time is given in reference[9]. Many algorithms for solving this problem in polynomial time have been studied (see[10] for a representative example). The typical non-exponential algorithm is relatively fast but does not have a high success rate especially when the system dimension (n) increases. Recently a method has been introduced that is slower than these algorithms but increases the success rate close to 100% in polynomial time[2] even for rather large systems. This current investigation is an extension of that work.

Analogous to systems of equations, systems of inequations can be solved using substitution and elimination of variables[2]. A decision function is used to decide which pair of variables $x_i$ and $x_j$ to combine or "associate" with one another. This process is repeated recursively until it is not possible to combine any more variables. It has been shown [2] that a number of decision functions can be intuitively derived from powers of the A matrix and its complement $\overline{A}$. In this current investigation, a generalized decision function based on a power series expansion of the A matrix is derived. A grid search technique is then developed using the coefficients of the terms in the expansion and the performance of the technique is evaluated. The grid search is then used as a starting point for a gradient search technique. This leads to a method called the equivalence class subset gradient search method. The significance of this investigation is that the solution methods described can be performed in polynomial time and it is well known that a solution for one np-complete problem can be used to solve any other np-complete problem[3][4][7].

## 2 Algorithm

A solution algorithm for systems of inequations will first be stated[2]. Using a decision function f(A) an algorithm can be derived[2] for solving systems of inequations as shown in Figure 1. The algorithm is recursive. It takes as input an nxn matrix A and reduces it to an (n-1)x(n-1) system, calling itself to solve the reduced system.

```
ineq(A)
ij⇔ f(A)
if {ij}={ } return s
xi=xi|xj
A=A-xj
ineq(A)
```

Figure 1. Algorithm for solving systems of inequations

The decision function f(A) identifies a single pair of variables $x_i$ and $x_j$ which are likely to be in the same equivalence class of an optimal solution. For example using f(A) = max($A^2$) could be used to identify a single pair of variables. These two variables are combined using the logical "OR" operator which represents the union of the constraints of $x_i$ and $x_j$. The algorithm steps through a series of feasible solutions starting with the trivial solution s=(1,2,3…n). Each time the size of the matrix is reduced by one the number of equivalence classes in the solution vector s is reduced by one. The subtraction operator (-) is used to symbolically represent the elimination of the variable xj from the system (i.e., the removal of row j and column j from the matrix A). The algorithm terminates when the set {ij} is empty.

The success rate of the algorithm depends on the decision function f(A) as shown in Table 1. For example using f(A) = max($A^2$) for a system of inequations of n=100 variables it is seen that the algorithm success rate is 90%. The success rate drops off rapidly at first for small n and then appears to level out. For decision function f(A)=min($A^2$) the success rate is much lower(40.8%). The decision function that chooses entirely at random f(A)=A has a success rate (55.7%) which is between f(A)=max($A^2$) and f(A)=min($A^2$). The decision function f(A)=A is called random because aij=0 for all pairs of variables that can be combined.

| f(A) | | n=10 | 20 | 30 | 50 | 100 | 200 | 300 | 500 |
|---|---|---|---|---|---|---|---|---|---|
| $A^2$ | max | 99.5 | 98.6 | 96.8 | 93.2 | 90.0 | 87.6 | 86.1 | 87.4 |
| $A^2$ | min | 49.5 | 44.3 | 41.4 | 40.6 | 40.8 | 41.7 | 43.3 | 41.5 |
| $\overline{A}A$ | max | 59.8 | 56.8 | 53.5 | 52.4 | 51.8 | 62.4 | 62.8 | 71.5 |
| $\overline{A}A$ | min | 99.9 | 98.4 | 96.5 | 91.3 | 89.2 | 88.8 | 88.3 | 87.7 |
| $\overline{A}^2$ | max | 90.4 | 80.9 | 73.7 | 66.1 | 63.4 | 60.2 | 58.5 | 62.0 |
| $\overline{A}^2$ | min | 64.8 | 63.0 | 66.6 | 67.6 | 69.1 | 75.3 | 74.7 | 81.5 |
| $A$ | ---- | 72.4 | 61.2 | 57.3 | 56.8 | 55.7 | 57.9 | 56.9 | 56.5 |

Table 1 Success rate for decision functions f(A) (random A matrix sample size=5000)

## 3 Powers of the A matrix

The solution space of each of the individual decision functions in Table 1 largely overlap one another. However they do not completely overlap and use of more than one decision functions will generally yield a higher success rate than use of a single decision function. The difference is typically on the order of a few percent; using all the decision functions in Table 1 would for example increase the success rate for n=500 from 87.4% using f(A)=max($A^2$) alone to about 90%[2]. It is desirable to find a single decision function or set of decision functions that covers 100% of the system space {A}. A starting point for investigating this possibility is to consider the powers of the A matrix to see if any of them have resolving ability for solving systems of inequations (the resolving ability of a decision function can be defined as having a success rate better than random guessing, i.e., f(A)=A). Table 2 shows the result of this calculation.

| $f(A^p)$ | | n=10 | 20 | 30 | 50 | 100 | 200 | 300 | 500 |
|---|---|---|---|---|---|---|---|---|---|
| $A^2$ | max | 99.5 | 98.6 | 96.8 | 93.2 | 90.0 | 87.6 | 86.1 | 87.4 |
| $A^3$ | max | 71.4 | 74.6 | 77.3 | 77.5 | 79.8 | 75.8 | 79.9 | 80.2 |
| $A^4$ | max | 99.8 | 97.4 | 94.3 | 90.2 | 87.9 | 85.4 | 87.5 | 84.5 |
| $A^5$ | max | 78.6 | 82.3 | 81.5 | 80.3 | 81.8 | 81.3 | 83.3 | 86.9 |
| $A^6$ | max | 99.8 | 97.2 | 94.1 | 89.4 | 85.6 | 84.2 | 84.0 | 83.1 |
| A | ---- | 72.4 | 61.2 | 57.3 | 56.8 | 55.7 | 57.9 | 56.9 | 56.5 |
| $A^2$ | min | 49.5 | 44.3 | 41.4 | 40.6 | 40.8 | 41.7 | 43.3 | 41.5 |
| $A^3$ | min | 85.3 | 68.7 | 60.0 | 50.2 | 48.7 | 49.7 | 48.0 | 46.0 |
| $A^4$ | min | 54.5 | 45.6 | 41.6 | 41.6 | 42.0 | 40.8 | 39.9 | 42.3 |
| $A^5$ | min | 80.1 | 59.8 | 52.4 | 50.4 | 45.8 | 46.8 | 47.8 | 46.4 |
| $A^6$ | min | 54.1 | 46.3 | 44.4 | 43.6 | 44.9 | 40.6 | 43.9 | 42.6 |

Table 2. Success rate for decision functions $f(A^p)$ (random A matrix sample size=5000)

Table 2 shows that the maximum of the even powers of the A matrix tend to have the highest success rate especially for systems less than dimension n=500. The success rate drops off from nearly 100% for small systems of dimension n=10 to somewhat less than 90% for n=500, although the rate of decrease appears to start leveling off for n>=200. The minimum of the even powers perform worse than selecting pairs at random f(A)=A. The maximum of the odd powers do not perform as well as the even powers for small n but instead of decreasing trend slightly upwards in success rate as system dimension increases. The difference in performance between odd and even powers of A may be related to dependencies between the variables known as cycles. A cycle is a dependency between a chain of variables starting at variable i and ending at the same variable i. The ij entry in the matrix $A^p$ represents the number of cycles of length p that will be created if variables i and j are associated. An even cycle can potentially be reduced to a chain of 2 variables while an odd cycle can only be reduced at best to a chain of 3 variables. This could be one reason why there is a difference between decision functions which maximize even cycles and decision functions which maximize odd cycles. It is not clear what is the asymptotic value of success rate as the dimension of the system increases but it is shown in [2] that $f(A)=\max(A^2)$ can solve systems of inequations of arbitrary dimension $n \to \infty$. It is somewhat remarkable that a single decision function such as $f(A)=\max(A^2)$ can be found that leads to an optimal solution in such a high percentage of cases. For example to solve this problem with n=500 and k=3 a total of 500-3 = 497 consecutive correct decisions have to be made.

## 4 Generalized Decision Function

The results in Tables 1 and 2 point show that the powers of the A matrix can be used to solve systems of equations but the combined result of all the individual decision functions does not cover the entire system space {A}. Another possibility would be to use a linear combination of the powers of the A matrix as shown in equation (2).

$$f(A) = c_1 A + c_2 A^2 + c_3 A^3 + c_4 A^4 + \ldots \quad (2)$$

The first term of (2) can be dropped as it is always zero for all pairs of variables {ij} that can be combined. A grid or gradient search can be performed over the remaining $c_i$ values for a given A matrix to find a set of decision functions which yields an optimal solution. Another technique is to use simulation over a large number of random A matrices to see if there is any particular set of $c_i$ values that generally work better than others. This could provide a starting point for the grid or gradient search and reduce the amount of computation required on average. After a fair amount of trial and error, the result of applying a simulation technique showed a strong peak of success rate for coefficient values $c_1=0$, $c_2=1$, $c_3 = -1/n$, $c_4 = 1/n^2$, $c_5 = -1/n^3$, etc., where n is the dimension of the A matrix, as given in equation (3).

$$f(A) = A^2 - n^{-1}A^3 + n^{-2}A^4 - n^{-3}A^5 + n^{-4}A^6 + \ldots \quad (3)$$

This expression for f(A) can also be represented in closed form as shown in equation (4). The form in equation (3) can be obtained from (4) by multiplying by $n^2$ and by dropping the I and A terms which are zero for any two pairs of variables that can be combined.

$$f(\mathrm{A}) = (I + n^{-1}A)^{-1} = I - n^{-1}A + n^{-2}A^2 - n^{-3}A^3 + n^{-4}A^4 - \ldots \quad (4)$$

Another observation is that the matrix $n^{-1}A$ is very close to a markov matrix M where the transition probability to any state other than the state that the system is currently in is equiprobable. The $n^{-1}A$ matrix can be converted to a markov matrix M by adding to the main diagonal the number of zeros in each row ($q_i$) divided by n as given in (5):

$$\mathrm{M} = \mathrm{n}^{-1}\mathrm{A} + \mathrm{n}^{-1}\mathrm{Iq} \quad (5)$$

Where q is the vector of the number of zeros in each row. It is not clear why there might be any relationship between a markov chain and the decision function to an np-complete problem. Further investigation so far has not found any additional analytical or intuitive insight along this line of reasoning. The result is a generalized decision function based on equation (3) but modified to include variable coefficients as shown in equation (6):

$$f(A) = A^2 - c_1 n^{-1} A^3 + c_2 n^{-2} A^4 - c_3 n^{-3} A^5 + c_4 n^{-4} A^6 + \ldots \quad (6)$$

Equation (6) represents a generalized decision function for solving systems of inequations or any other np-complete problem since any np-complete problem can be converted into a system of inequations. Note the first term in the series is $A^2$ which is one of the basic decision functions as shown in Table 1. In practice only a finite number (m) of terms of f(A) are used so the generalized decision function can also be represented as in equation (7).

$$f(A) = A^2 + \sum_{i=1}^{m} c_i (-1)^i n^{-i} A^{i+2} \quad (7)$$

Preliminary investigation indicates there may be some relationship between the convergence or divergence of this series and the ability of the decision function to find an optimal solution. However further investigation is required before any conclusion can be drawn in this regard.

## 5 Grid Search

It is seen from Table 1 that the decision function f(A) = max($A^2$) can be used to find an optimal solution for approximately 90% of the entire system space {A} for dimension n=100 variables. The amount of computational effort is seen from Figure 1 to involve O(n) calculations of the square of the A matrix (O($n^3$)) for a total computational effort of O(n)* O($n^3$) = O($n^4$).

A grid search involves multiple solutions of the basic algorithm of Figure 1 plus additional calculations of higher powers of the A matrix. As a starting point a grid search was first carried out with the first 3 terms of equation (6) as shown in equation (8):

$$f(A) = A^2 - c_1 n^{-1} A^3 + c_2 n^{-2} A^4 \quad (8)$$

Note that this decision function will solve all the same systems that f(A)=max($A^2$) will solve if the coefficients ($c_1$=$c_2$=0) are included in the grid. The variables $c_1$ and $c_2$ were used in a grid search of the first quadrant and $c_i$ values were tried for integer values between 0 and 10: 0<=$c_1$<=10 and 0<=$c_2$<=10. Figure 2 shows a typical result for a system of dimension n=100 variables and optimal solution cardinality k=20. The decision function f(A)= max($A^2$) was unable to solve this particular problem as seen by the lower left hand corner of the diagram ($c_1$=$c_2$=0). Similarly coefficient values $c_1$=$c_2$=1 did not yield an optimal solution. However as seen in the white area of the figure that the grid search did find a set of optimal solutions as there were a number of (c1,c2) combinations that did solve the problem (such as ($c_1$=0, $c_2$=2) and ($c_1$=3,$c_2$=3)). It is typical to find more than one decision function which leads to an optimal solution for a given system of inequations (A). For example there are many non-integer sets of coefficients that lead to an optimal solution for solve this system as indicated by the region of white space in Figure 2.

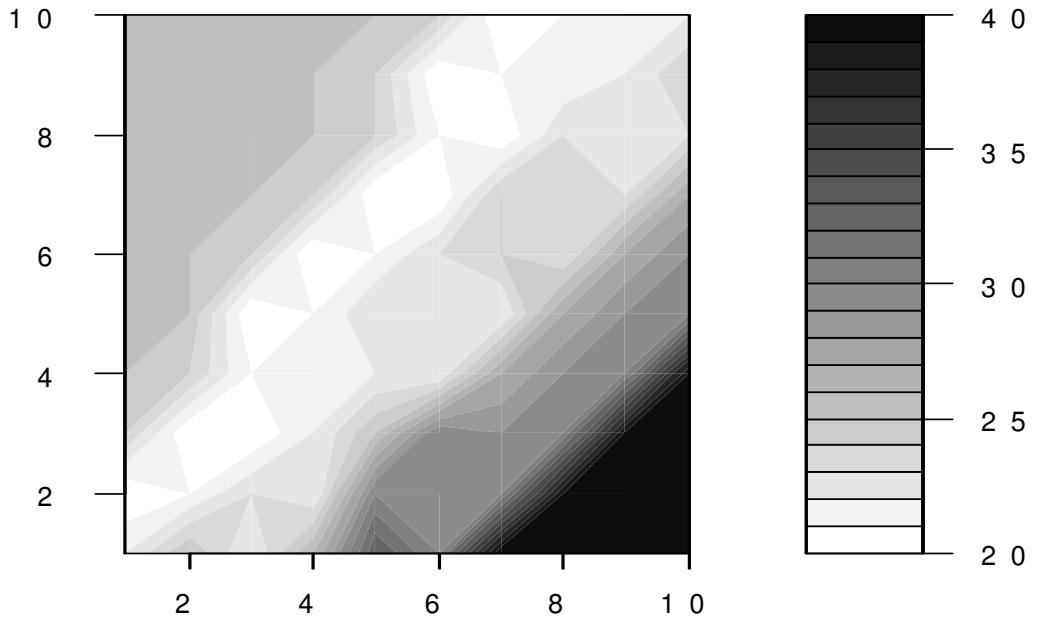


Figure 2. Grid search for matrix A of dimension n=100 and k=20 in first quadrant

A simulation of 1000 random A matrices of dimension n=100 and known solution cardinality k randomly distributed between 2 and 100 and ones density uniformly distributed between 0 and 1 for a search on the first quadrant had a solution rate of about 94.5% as compared to 90% for f(A)=max($A^2$). Since the method of Figure 1 is about O($n^4$) and the first quadrant search required 11*11=121 calculations of the basic algorithm the computational effort for this improved success rate was approximately O(n)*O($n^4$) = O($n^5$).

The next idea investigated was the extending of the grid search to all four quadrants of the ($c_1$,$c_2$) space. Figure 3 shows a typical result for a matrix A of dimension n=100 and optimal solution cardinality k=13. In this case a region of optimal solution was found in the first, second and third quadrants but not the fourth. This region of optimal solution includes the case of $c_1$=$c_2$=1 as well as many (perhaps an infinite number) of non-integer ($c_1$,$c_2$) values. Extending the search to all four quadrants quadrupled the amount of effort required to 441 calculations of the basic algorithm as compared to 121 calculations for only the first quadrant, but this can still be considered O($n^5$) complexity. A simulation using 1000 random A matrices with known solution cardinality k (2<=k<=100) resulted in a success rate for n=100 of about 96.3%. This is an improvement of two percentage points for a four-fold increase in computational effort.

Based on the results obtained thus far it us seen that a simple grid search on integers can yield a success rate for about 96% of systems of dimension n=100. There is no doubt that success rate could be further increased by using a finer grid spacing or extending the grid beyond (|$c_1$|,|$c_2$|)=(10,10). How much further the success rate can be increased using this approach is not known. However a grid search approach can become very inefficient if extended to large regions of space with increasingly fine grid spacing and at some point it becomes more efficient to use a gradient search. In the next section, two types of gradient search are investigated. One uses the solution cardinality k which is the same as in the grid search. The other uses a new parameter z which is the total number of pair associations {ij} in a feasible solution s.

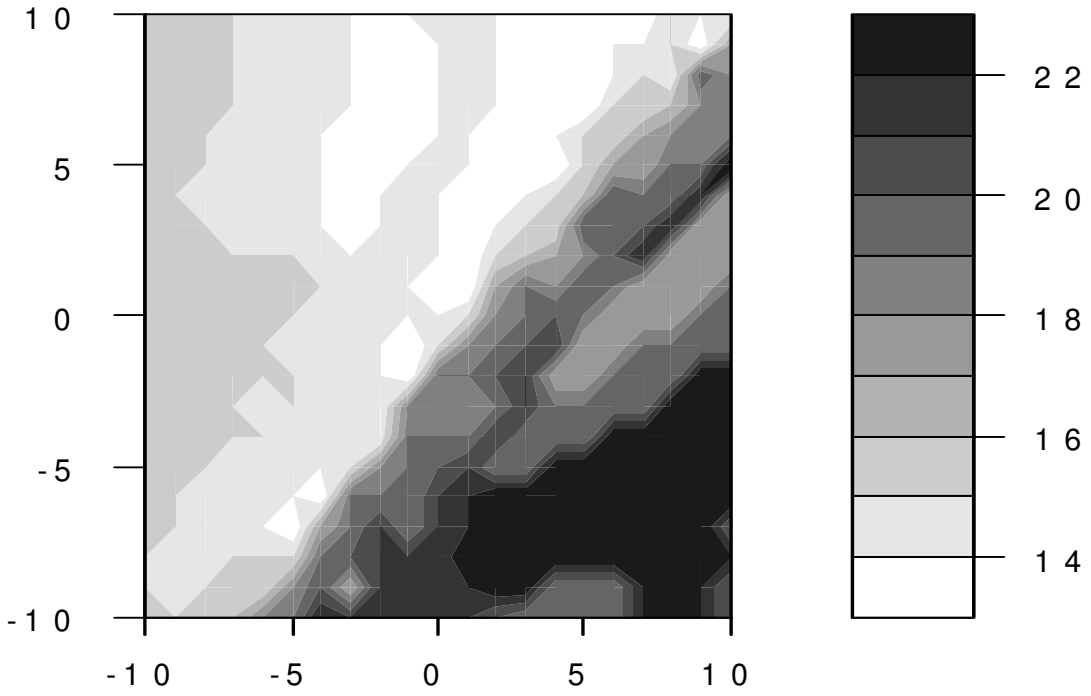


Figure 3. Grid search for matrix A of dimension n=100 k=13 over four quadrants

## 6 Gradient Search

The grid search can be used as a starting point of a technique known as a gradient search. Looking back on Figures 2 and 3 it is seen that a significant subregion of the total area covered by the grid resulted in a decision function that lead to optimal solutions. In general it will be found that some systems have relatively small regions where the decision function result in optimal solution while others have relatively large regions. This is an indication that the difficulty of a particular problem may be related to how few or how many decision functions lead to an optimal solution. To find an optimal solution for systems with small regions of optimal decision functions the gradient search can be more efficient than the grid search. The idea of the gradient search is analogous to the standard technique used with multivariable continuous functions. For example, Figure 3 shows that regions of optimal solution k*=13 are surrounded by regions of suboptimal solution k=14 which is surrounded by a suboptimal region of k=15. This gives an expectation that at least in some cases that were not solved using a grid search might have been solved by placing a finer grid in those areas which had the lowest solution cardinality. This is done to save computational effort since it is more efficient than putting a finer grid over the $(c_1,c_2)$ coefficient space.

To use a gradient search some technique such as using a convex hull could be used to define the subregion of lowest solution cardinality and then a finely spaced grid of points could be selected. Another approach would be to use the points themselves to define the space instead of first finding the boundary region of that space. One way to do this would be to choose pairs of points at random within the desired subregion and choose a random point on the line between them. This defines a new subregion on which to perform a new grid search and the process can be repeated as many times as desired. The gradient search technique was tested using the decision function in equation (9) which extends out to the sixth power of the A matrix and involves a gradient search on four variables $c_1,c_2,c_3,c_4$.

$$f(A) = A^2 - c_1 n^{-1} A^3 + c_2 n^{-2} A^4 - c_3 n^{-3} A^5 + c_4 n^{-4} A^6 \quad (9)$$

It was decided to use only the first quadrant for test purposes and to speed up the computation since it was observed that most of the systems studied had solutions in the first quadrant even if they had solutions in other quadrants. To keep the computation reasonable each of the $c_i$ coefficients was allowed to have 6 values: 0,2,4,6,8, and 10. This leads to $6^4$ = 1296 calculations of the basic algorithm of Figure 1. From this grid calculation a subregion of the 4-dimensional space was formed to further refine the search. Next 400 pairs of points in this 4-dimensional space were chosen by randomly choosing a point on the line connecting the random pairs of points. Then another subregion was defined by using the best solutions from these points and these were used to generate another 400 point subregion to further refine the search.

The performance of this gradient search algorithm was evaluated by simulation using 1000 random A matrices and the result was that an optimal solution was found in 98.1% of the cases. This was at a computational cost of 1296+400+400 = 2096 solutions of the basic algorithm. Compared to the previous two calculations it was even more computationally intensive because each decision function required raising the A matrix to the fifth and sixth powers. Considering all of these factors the gradient search had a computational complexity somewhere between $O(n^5)$ and $O(n^6)$. As might have been anticipated the computational appears to increase significantly as the solution rate approaches 100%.

## 7 Equivalence Class Subset Algorithm

The solution techniques discussed so far have been based on minimizing the solution cardinality k using either a grid or gradient search. There is however another form of gradient search which is based on a related parameter z which is the total number of pair associations {ij} in the solution s. A pair association {ij} occurs when a pair of variables xi and xj are combined into the same equivalence class. If an optimal solution vector s* is permuted such that all equivalence classes are grouped together and the corresponding A matrix is permuted accordingly, it is seen that the A matrix takes on a block diagonal form[2]. Generally speaking the lower the solution cardinality (k) the greater the total number of zeros in all of the blocks of the block diagonal form.

To illustrate the idea of the total number of pair associations refer to Figure 4 which represents the result of a grid search on a system of n=100 variables. However, instead of showing the solution cardinality (k) as in Figures 2 and 3, the number of pair associations (z) is shown. This gives a very similar result as compared to Figures 2 and 3 since optimal solutions will generally have the highest number of pair associations as represented by the white areas of Figure 4 (note that to make Figure 4 easier to compare with Figures 2 and 3 it actually shows (max(z)-z) instead of z since when z=max(z), max(z) – z = 0).

The white areas of Figure 4 also correspond to optimal solutions k* as do some of the light gray areas. As with the solution cardinality k, the total number of pair associations z follows a contour pattern with contiguous areas of optimal solution surrounded by contiguous areas of suboptimal (but close to optimal) solutions. This gives rise to another form of gradient search based on the total number of pair associations z which is equivalent to total number of zeros in the diagonal blocks of A as shown in equation (10).

$$z = \sum_{i=1}^{k} k_i^2 \quad (10)$$

where $k_i$ is the number of variables in each of the k equivalence classes. For example a solution vector s = (1,2,1,1,2,2) would correspond to an A matrix having two 3x3 diagonal blocks when the rows and columns are permuted so that all variables in the same equivalence class are grouped together for example in monotonically increasing order (s=(1,1,1,2,2,2)). In this case the parameter z =18 since $k_1=k_2=3$. Every solution in the set of feasible solutions {s} corresponds to its own z value and the highest values of z almost always corresponds to the lowest values of solution cardinality k (as illustrated in Figure 4).

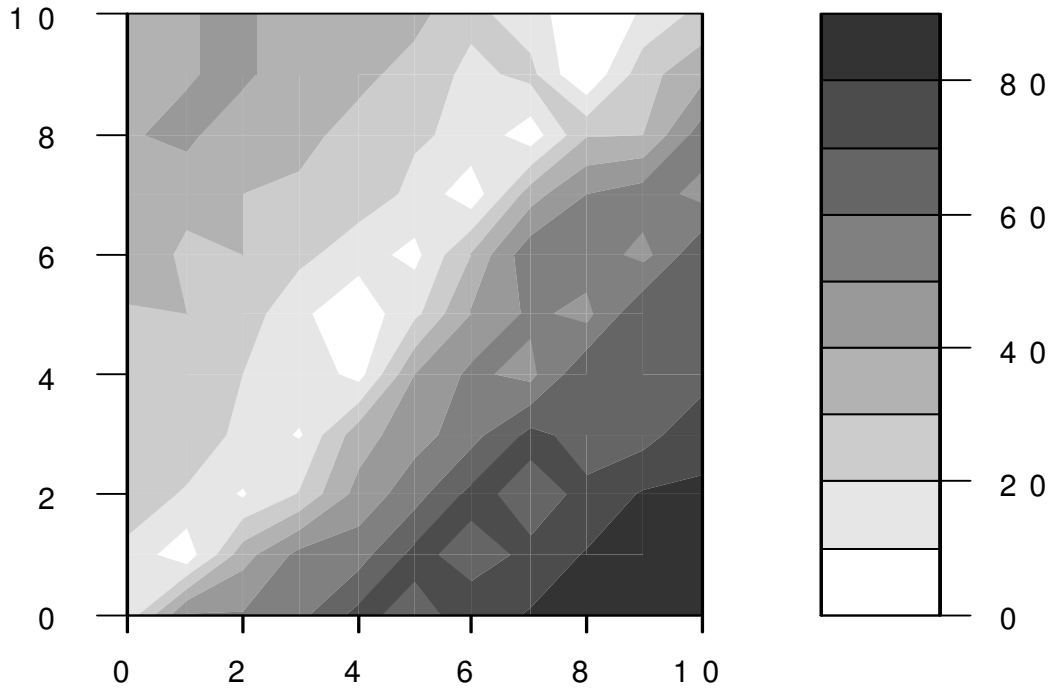


Figure 4. Total number of pair associations z for a typical A matrix (max(z)-z)

Generally speaking the optimal solutions of a particular A matrix corresponding to a system of inequations will contain more zeros in the block diagonals as compared to the suboptimal solutions. To estimate which pair associations {ij}are most likely to lead to an optimal solution an nxn pair association matrix Z can be defined as shown in equation (11).

$$Z = \begin{bmatrix} 0 & \bar{z}_{12} & \bar{z}_{13} & . \\ \bar{z}_{21} & 0 & \bar{z}_{23} & . \\ \bar{z}_{31} & \bar{z}_{32} & 0 & . \\ . & . & . & . \end{bmatrix} \qquad (11)$$

The element $z_{ij}$ of the matrix Z represents an estimate of the average number of total pair associations found when the pairs i and j are in the same equivalence class. In general, the more pair associations in a solution s means the less number of equivalence classes required. During a grid or gradient search, many feasible solutions are found. Every time any solution s is calculated, the total number of pair associations z corresponding to s is calculated and used to update the matrix Z for the set of pair associations in s. On average pair associations with the largest values in the Z matrix are more likely to lead to optimal solutions.

The resulting algorithm involves using the Z matrix to choose a maximum likelihood subset {ij} of the pair associations represented by a solution vector s to maximize the probability of finding an optimal solution. The method is called the equivalence class subset algorithm which is given in Figure 5.

```
while(condition){
A' = A({ij})
s = ineq(A')
k = max(s)
z = z(s)
if(k==k* && z>z*) s*=s, z*=z
if(k<k*) s*=s, k*=k, z*=z}
```

Figure 5. Equivalence class subset algorithm

The first step in the equivalence class subset algorithm (not shown for clarity) is to initialize s*=(1,2,…n), k*=n, and z*=n where s*, k*, and z* represent the current best solution and n is the dimension of the system. This corresponds to the trivial solution that solves any system of inequations and also the optimal solution for k=n. The while(condition) provides a stopping criterion. For example limiting the number of iterations to some value such as n creates an $O(n^5)$ algorithm while limiting the number of iterations to $n^2$ creates an $O(n^6)$ algorithm (assuming ineq(A) is $O(n^4)$). The first step of the algorithm is to take the system A matrix and create a new A' matrix by combining some subset of variables {ij} based on the matrix Z and the current best solution s*. The first time through the algorithm all values of zij in Z=0 and s*=(1,2,3…n) which means that {ij}={} so that A'=A. The solution vector s corresponding to A' is then calculated using the ineq(A) algorithm in Figure 1 and this solution is used to calculate the solution cardinality k=max(s). The total number of pair associations z=z(s) is then calculated according to equation (10). There are three possible outcomes: k=k*, k<k*, or k>k*. If the solution cardinality has not improved (k*=k) but the number of pair associations z has improved (z>z*) then the solution s is taken as the new best solution s*. If the solution cardinality has improved (k<k*) then the solution s is taken to be the new best solution s*. If the solution cardinality of s is greater than s* (k>k*) or if k=k* and z<z* then the solution s* is not updated. Before starting the next iteration the pair association matrix Z is updated by calculating the new averages for each pair association as represented by equation (11).

The idea behind the equivalence class subset algorithm is that in the majority of cases most of the decisions made by an f(A) decision function are correct. If most of the decisions are correct then choosing a subset of pair associations from even a suboptimal solution s will in many cases only contain pair associations which lead toward an optimal solution. When the subset of pair associations {ij} are associated by combining the indicated variables the matrix A is modified to form a new matrix A'. If no incorrect decisions were included in {ij} then the A' matrix will have the same optimal solution cardinality as the matrix A. If in addition the matrix A' falls into the region of convergence of one of the decision functions of the grid or gradient search then an optimal solution will be found. The larger the size of the subset, the higher the probability it is to contain an incorrect decision. The smaller the size of the subset, the less likely it is to contain an incorrect decision. The result of solving thousands of systems indicates that the subset size that leads to the optimal solution is usually less than 30% of the total number of pair associations in a solution vector s. The subset size can be chosen at random for each iteration. However, once the size of the subset is determined the method of choosing the subset needs to be decided. One simple way is to use the maximum likelihood method which is to simply rank order the elements of the matrix Z and choose the desired subset using the largest values in Z. The effect of the equivalence class subset algorithm is to simultaneously minimizing k while maximizing the parameter z. The z parameter maximization provides a way to differentiate between the large number of solutions of cardinality k>k*.

The result of a simulation using 1000 random A matrices shows that the equivalence class subset search for system of dimension n=100 having a success rate of about 99.8% for a computational complexity $O(n^6)$. Figure 5 shows a typical trajectory of the equivalence class subset algorithm for a system of n=100 variables and optimal solution cardinality k*=13. The lower line represents the sequence of k values starting at k=15 and ending an optimal solution k*=13. The dashed line represents the total number of pair associations z which it is desired to maximize (for simplicity the actual numbers are not shown for the pair associations). The method required choosing about 40 subsets from the suboptimal solution vector s(k=15) to find a better solution s(k=14) while improving the solution vector s continuously along the z gradient. After choosing 60 more subsets from s(k=14) and improving steadily along the z gradient the method found an optimal s*(k*=13) solution.

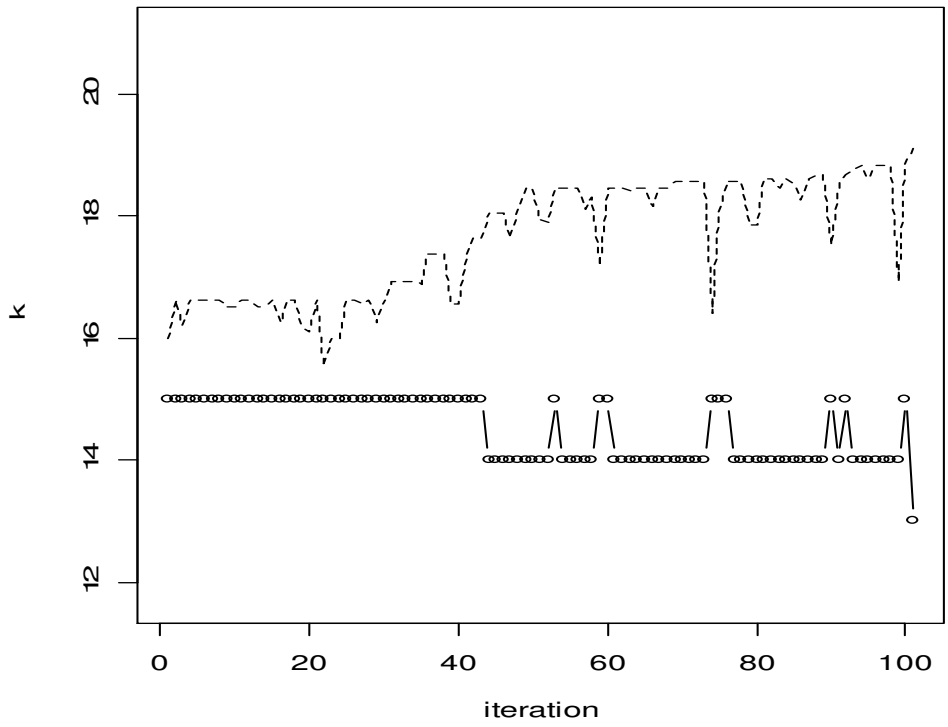


Figure 5. Typical trajectory of equivalence class subset algorithm for system with k*=13

## 8 Summary and Conclusions

A method is described for solving the compatibility problem[1][2] using a power-series expansion to represent a generalized decision function. The method is very general in that it can be applied to any system of any dimension n and solution cardinality k. The basic technique uses the coefficients of the terms in the power-series in a grid or gradient search. A two-dimensional grid search of the first quadrant was able to solve 94% of the systems of dimension n=100 for $O(n^5)$ complexity. When extended to all four quadrants the success rate was increased to 96% for a four-fold increase in complexity. A four dimensional gradient search increased the success rate to 98% at complexity somewhere between $O(n^5)$ and $O(n^6)$. Next the equivalence class subset method was tested and found to have a 99.8% success rate for $O(n^6)$ complexity. The random A matrices used to obtain the success rate estimates were chosen to be uniformly distributed across the system space S={A} (the set of all possible systems of inequations) so that the result would be a reliable and reproducible indicator of algorithm performance that could be used to compare against other algorithms. The uniform distribution was applied across both the constraint density and the solution cardinality k to ensure that the sample included a true cross-section of the entire spectrum of problem difficulty. The results were validated by generating the results for other random samples of size 1000 and observing that there were no significant deviation of the results. The high success rate of the generalized decision function leads to the question of whether there is for every A matrix (or system of inequations) at least one decision function that leads to an optimal solution. It is possible to keep adding terms to the power series expansion and to increase the breadth and depth of search in an attempt to continue increasing the success rate. These results appear to imply that there may exist some algorithm for solving np-complete problems with a success rate asymptotically approaching 100% in polynomial time. The reason for this kind of asymptotic behaviour appears to be that the problem difficulty falls into a type of continuum ranging from systems that can be solved by any decision function to others that appear to be solved by only a small number of decision functions. It was observed that there is a pattern to the location of the optimal solutions within the vast sea of suboptimal solutions. The optimal solutions tend to fall in semi-contiguous regions of space and are surrounded by regions of suboptimal solutions which are in some sense close to optimal. This makes it possible to construct algorithms such as the equivalence class subset method which can find an optimal solution with high success rate without searching all of the possibilities.

## References


[1] Clay Mathematics Institute, Millennium Prize Problems, “P vs. NP”, http://www.claymath.org/millenium/P_vs_NP, 2005.

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

[3] S. Cook, “The P vs. NP Problem”, Clay Mathematics Institute, http://www.claymath.org/millennium/P_vs_NP/Official_Problem_Description.pdf, 2005.

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

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

[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] N. Christofides, A. Mingozzi, P. Toth, and C. Sandi, editors, “Combinatorial Optimization”, pages 211-235, New York, 1970.

[10] Gomes, C.P. and Selman, B. Algorithm Portfolio Design: Theory vs. Practice *Proc. UAI-97,* Providence, RI, 1997.