World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:3, No:1, 2009

Optimal Solution of Constraint Satisfaction Problems

Jeffrey L. Duffany

***Abstract*—**An optimal solution for a large number of constraint satisfaction problems can be found using the technique of substitution and elimination of variables analogous to the technique that is used to solve systems of equations. A decision function $f(A)=\max(A^2)$ is used to determine which variables to eliminate. The algorithm can be expressed in six lines and is remarkable in both its simplicity and its ability to find an optimal solution. However it is inefficient in that it needs to square the updated A matrix after each variable elimination. To overcome this inefficiency the algorithm is analyzed and it is shown that the A matrix only needs to be squared once at the first step of the algorithm and then incrementally updated for subsequent steps, resulting in significant improvement and an algorithm complexity of $O(n^3)$.

***Keywords*—**Algorithm, complexity, constraint, np-complete.

I. Introduction

SUPPOSE there are n objects each of which are incompatible with some subset of the other n-1 objects. The problem is to partition all n objects (x_i) 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 the adjacency matrix (A) of ones and zeros which summarizes the compatibility of each object (variable xi) with every other object (variable xj). A one in the (i,j) element of the A matrix indicates incompatibility between variables xi and xj while a zero represents compatibility. This problem is usually referred to as an ILP (Integer Linear Program)[1] and has the same form as a system of equations Ax=b (Fig. 1).

$\mathbf{Ax=b}$	systems of equations
$\mathbf{Ax \leq b}$	systems of inequalities
$\mathbf{Ax=\lambda x}$	eigenvalue problem
$\mathbf{Ax \neq x}$	systems of inequations

Fig. 1 Fundamental Mathematical Problems

Fig. 1 lists four fundamental problems of mathematics all of which involve the simultaneous solution of more than one equation, inequation[9][10] or inequality. Each of these problem formulations involve a matrix A which which make the general statement of the problem very concise.

J. L. Duffany is with the Electrical and Computer Engineering Department, Universidad del Turabo, Gurabo, PR 00778 USA (e-mail: jduffany@suagm.edu).

The first three of the problems formulations in Fig. 1 are quite well known and have been widely studied and analyzed in the scientific mathematical and engineering literature. The fourth problem formulation $Ax \neq x$ (system of inequations) [9][10] is not nearly as well known as the first three since the alternative representation of the Integer Linear Program Ax=b, x integer, is normally used. The motivation for creating a separate formulation is that the compatibility problem may be better characterized by using a logical or set-theoretical formulation as opposed to trying to fit it into an existing formulation which is inherently arithmetic. The $Ax \neq x$ representation requires that addition in matrix multiplication be interpreted as a logical union and the $\neq$ symbol be interpreted as the symbol for not being a member of a set. In other words the value of variable x_i is not equal to that of any member of some subset of the other n-1 variables, that subset selected by values of row i of the A matrix which equal 1.

One reason for having a separate formulation for inequations is that there are many subtle differences between the two. Table I shows that a system of inequations usually has a large number of suboptimal solutions and always has at least one optimal solution. The number of optimal solutions is related to how constrained the system is. A system of inequations that has more than one optimal solution is called underconstrained. If there is exactly one optimal solution and the removal of a single constraint increases the number of optimal solutions then the system is perfectly constrained. If there is only one optimal solution and there is at least one constraint whose removal does not increase the number of optimal solutions then the system is called overconstrained. A system of equations can have 0, 1 or an infinite number of solutions. This represents an important difference between equations and inequations since a system of inequations always has at least one optimal solution.

II. Constraint Satisfaction Problems

Finding an optimal solution for a system of inequations is in general an NP-hard[5][7] problem. Finding an optimal solution for a system of inequations for solution cardinality k*=3 is NP-complete[6]. An algorithm that can determine an optimal solution for a system of inequations can be used to solve a wide variety of important problems spanning across many diverse fields from artificial intelligence[12] to bioinformatics[11] including many problems known as constraint satisfaction problems [12].

TABLE I
COMPARISON OF EQUATIONS AND INEQUATIONS

	Number of Solutions (q)		
Equations	q=1 (equations=unknowns)	q= ∞ (equations<unknowns)	q=0 (equations>unknowns)
Inequations	q=1 (perfectly constrained)	1 < q < ∞ (underconstrained)	q=1 (overconstrained)

Some constraint satisfaction problems explicitly maximize or minimize an objective function while others do not. An example of the latter is the n-queens problem [12] where n queens are placed on an nxn chessboard in such a way that no two of them are in the same row, column or diagonal. For this problem any solution that satisfies the constraints is a valid solution. If any valid solution can be found before all the possibilities are searched it is known with certainty that the problem has been solved and the algorithm can terminate at that point.

Another category is the class of constraint problems that also maximize or minimize an objective function. In this case it is generally not possible to know with any certainty if an optimal solution has been found without checking every possibility. An example of this type of problem is the system of inequations[3][10]. The algorithm for solving a system of inequations can be viewed as a search of the feasible solutions. Each time a feasible solution vector is generated the objective function is calculated. A solution vector (s) is a mapping of each variable xi into an integer si such that 1<= si <= k while ensuring that si ≠ sj when A[i,j] = 1. The set of all solution vectors can be represented by a tree with n-k*+1 levels representing all solutions of cardinality k*, k*+1,...n. The total number of solutions equals the number of nodes in the decision tree and grows exponentially with n. One way to find an optimal solution s* is to check all the possibilities.

III. SUBSTITUTION OF VARIABLES

All feasible solutions to a system of inequations can also be generated by substitution and elimination of variables in a manner analogous to finding a solution to a system of equations. It is also similar to gaussian elimination except that arithmetic addition is replace by logical OR. Two variables xi and xj can substitute for one another only if the inequation xi ≠ xj is not present in the system, in other words A[i,j]=0. To solve a system of inequations using this technique two variables xi and xj are chosen and set equal to one another xi = xj by mapping their constraints onto one another (logical OR). Since the two variables have been made equal one can be substituted for the other and one of them can be eliminated (substitution and elimination of variables).

A decision function f(A) can be used to choose a specific xi and xj to combine. This decision function f(A) can be considered a global decision function if it takes into account every possible combination (i.e., for which A[i,j]=0). Perhaps the simplest decision function is f(A)=max(A). Since all pairs of variables that can be combined have A[i,j]=0 the decision function f(A)=max(A) will combine with equal probability any legal pair of variables xi and xj, i ≠ j and A[i,j]=0. This can be called the ambivalent decision function which is equivalent to choosing i and j at random. This procedure is repeated recursively and the final solution vector is determined by the standard method of back substitution.

Another decision function is f(A)=max(A^2). This decision function maximizes the number of shared constraints between every possible pair of variables that can be combined (i.e., where A[i,j]=0). The idea is illustrated by Fig. 2 which considers the block diagonal form of the A matrix which is also known as a complete k-partite system (any system of inequations can be derived by removing constraints from some complete k-partite system). Considering each variable's constraints as a row of the A matrix the number of shared constraints can be calculated as xi * xj (where * represents the vector product). Performing this operation across all ij leads to matrix multiplication and the decision function f(A)=max(A^2).

In Fig. 2 the maximum value of A^2 is 5 and it can be shown that in this case (as in almost all cases) combining the pair of variables corresponding to max(A^2) leads to an optimal solution. A simple algorithm for solving systems of inequations based on the decision function f(A)=max(A^2) is given in Fig. 3 as algorithm *ineq*.

```
ineq(A)
ij<->max(A2)
if ij={} return A
xi=xi|xj
A=A[-j,-j]
ineq(A)
```

Fig. 3 The ineq algorithm

The ineq algorithm starts with a solution vector s which has an initial value of s = (1,2,3,...n). The algorithm squares the adjacency matrix A and finds the maximum value of A^2 [i,j] for pairs of variables that can be combined (i.e., A[i,j]=0). It then combines variables xi and xj by taking the constraints that are in xj but not in xi and adding them to xi (xi=xi|xj) where | = logical OR. Then it updates the solution vector s[j]=s[i] and eliminates variable xj (i.e., remove row and column j from A as represented by the line A=A[-j,-j]). The matrix A is reduced by one in dimension each time a variable is eliminated.

$$A = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 \end{bmatrix} \quad A^2 = \begin{bmatrix} 4 & 4 & 4 & 2 & 2 & 2 & 2 \\ 4 & 4 & 4 & 2 & 2 & 2 & 2 \\ 4 & 4 & 4 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 5 & 5 & 3 & 3 \\ 2 & 2 & 2 & 5 & 5 & 3 & 3 \\ 2 & 2 & 2 & 3 & 3 & 5 & 5 \\ 2 & 2 & 2 & 3 & 3 & 5 & 5 \end{bmatrix}$$

Fig. 2 Block Diagonal Form of the A Matrix and Corresponding A^2 Matrix

The difference with gaussian elimination is that the algorithm ineq uses logical OR instead of addition and uses a decision function f(A)=max(A^2) to determine[i,j]. Note that ineq is recursive and stops when there is no longer any variables to combine. A system of dimension n=100, optimal solution cardinality k*=3 requires the decision function (for example f(A) = max(A^2)) to make 97 consecutive correct decisions to find an optimal solution in n-k* (i.e., O(n)) iterations. This would establish the complexity of ineq at O(n^4) since squaring the A matrix is O(n^3) and this must be done O(n) times. However it is shown in the next section that the A matrix only needs to be squared on the first iteration and then updated for each iteration thereafter.

Any decision function that can identify two variables in the same equivalence class with probability=1 will always find an optimal solution s* using the ineq algorithm. For systems of n=100 variables the decision function f(A)=max(A^2) can determine two variables in the same equivalence class of an optimal solution s* with approximately 99.8% accuracy across the entire system space[3]. As a result tests have shown [3] the ineq algorithm can directly solve 90% of systems of inequations of n=100 variables. By using the ineq algorithm multiple times the success rate can be brought close to 100%[2,3,4]. The number of iterations required depends on the constraint density[2] and could be of the order of several hundred for systems of dimension n=100[2].

IV. Complexity Analysis

The ineq algorithm in Fig. 3 involves the calculating the square of a matrix (O(n^3)), finding the maximum (O(n^2)) and the substitution and elimination of variables (O(n)). The complexity is therefore dominated by the complexity of squaring the A matrix. This is relatively inefficient since the elimination of a variable only adds relatively few constraints to another variable which is a relatively small change to the A matrix. As a result the complexity if the ineq algorithm has previously [2,3,4] been estimated at O(n^4). However it can be shown that the overall complexity can be reduced to O(n^3) by updating the A^2 matrix instead of performing the full matrix multiplication at each iteration.

Representing the new A' matrix as the old A matrix plus an incremental matrix ΔA allows A'^2 to be represented as in equation (1):

$$A'^2 = (A + \Delta A)^2 = A^2 + A*\Delta A + \Delta A*A + (\Delta A)^2 \quad (1)$$

It is sufficient to show that one of the three terms (for example A*ΔA) can be calculated in O(n^2) time complexity since ΔA*A is just the transpose of A*ΔA and calculation of $(\Delta A)^2$ is the same as the other two with A replaced by ΔA. Suppose that only one new constraint was added then ΔA would be all zeros except for row i and column i each of which would have a single 1. Calculation of A*ΔA copies the columns of the A matrix that correspond to these two 1's to an all zero matrix which is an O(n) operation. In the general case of combining two variables all constraints that were in variable j but not in i are added to row and column i in a matrix of all zeros to create ΔA. To calculate A*ΔA the column vector corresponding to each added constraint is copied into the appropriate column of A*ΔA and the sum of these column vectors is placed in column i. In a worst case this requires all of the values of the A^2 to be updated resulting in O(n^2) time complexity.

```
ada<-0
acol<-0
for(qq in added){
acol<-acol+a[,qq]}
ada[,i]<-acol
for(qq in added){
ada[,qq]<-a[,i]}
```

Fig. 4 Calculation of A*ΔA matrix

Fig. 4 shows code for calculating the A*ΔA matrix where "added" is the vector of added constraints, "acol" is a column vector, "ada" is A*ΔA and qq is an index variable. ΔA*A is just the transpose of A*ΔA and $(\Delta A)^2$ is calculated the same as A*ΔA with A replaced by ΔA. Since the three matrix additions are O(n^2) the computation of A'^2 from A^2 is O(n^2). A total of O(n) updates could be required to complete the ineq algorithm. In the general case therefore the complexity of the ineq algorithm can be established as O(n)*O(n^2) = O(n^3).

IV. Optimal Solution of Constraint Satisfaction Problems

The first step is to take the desired constraint satisfaction problem and convert it into a system of inequations. If the problem happens to be vertex coloring (i.e., the classic compatibility problem) then it is already in the required form. If problem has been classified as NP-complete [8] then there exists a polynomial time algorithm to convert the given problem into a system of inequations[1][6]. For the 8 queens problem the conversion can be done by creating an A matrix (64x64) with each row representing a square on the chessboard. The entire matrix is filled with 0's and then a 1 is placed in columns corresponding to chessboard squares that are incompatible with the square represented by that particular row. The second step would be to use the ineq algorithm directly or in one of its powerful variations [2,3,4] to produce a solution vector s which is shown in Fig. 5.

1	5	6	4	7	8	2	3
2	7	8	3	1	9	5	4
3	9	4	7	5	6	8	1
4	1	3	8	2	7	9	5
5	8	2	1	9	3	4	7
6	3	9	5	4	2	1	8
7	4	1	2	6	5	3	9
8	2	5	9	3	4	7	6

Fig. 5 A Solution to the 8 Queens Problem

Fig. 5 represents a vertex coloring of the chessboard squares where each of the numbers 1 through 9 represents a different color. It is equivalent to placing 64 queens of 9 different colors on the chessboard in such a way that no two queens of the same color are attacking each other. In actuality there are 8 queens of color 3, 4 and 5, 7 queens of color 1, 2, 7, 8 and 9 and 5 queens of color 6. These numbers are arranged in such a way that no number is repeated along any row, column or diagonal. If this were just a vertex coloring problem the solution vector s is the desired solution and max(s) is the solution cardinality k. In the case of the n-queens problem the s vector must be sorted to find the largest equivalence class to find the solution. For this type of constraint satisfaction it is possible to recognize immediately when an optimal solution is found. For other more general cases the constraint density of the system of inequations can then be calculated to give an estimate of the probability that an optimal solution has been found [2,3,4]. The reason for this, as shown in [2,3,4], is that there are large contiguous regions of constraint density over which the ineq algorithm has virtually a 100% success rate. It is a simple matter to calculate the constraint density for any particular problem.

V. Summary and Conclusions

An optimal solution for a large number of constraint satisfaction problems can be found using the technique of substitution and elimination of variables analogous to the technique that is used to solve systems of equations. A decision function $f(A)=\max(A^2)$ is used to determine which variables to eliminate. The resulting algorithm can be expressed in only six lines of pseudocode and is remarkable in both its simplicity and its ability to find an optimal solution. However it is inefficient in that it needs to square the updated A matrix after each variable elimination. To overcome this inefficiency the algorithm was analyzed and the importantant result was established that the matrix A in the decision function only needs to be squared once at the first step of the algorithm and then incrementally updated for subsequent steps, resulting in significant improvement and an overall algorithm complexity of $O(n^3)$. A wide variety of algorithms (such as backtracking) exist for searching for an optimal solution for constraint satisfaction problems [12]. The idea of converting these problems into of a system of inequations and solving them using substitution and elimination of variables represents a significant paradigm shift from the past and is an attempt to attain a more unified viewpoint across a broad class of important problems in many fields. Once a problem has been converted into a system of inequations the constraint density is easily calculated. The constraint density can be used to estimate the difficulty of the problem [2] and estimate the probability that an optimal solution has been found after a given amount of effort. This unified approach may shed new insight by allowing comparisons between systems of inequations and other fundamental problems in mathematics.

References

[1] C.H. Papadimitrou and K. Steiglitz, "Combinatorial Optimization: Algorithms and Complexity", Dover, ISBM 0-486-40258-4, pp. 344.
[2] Duffany, J.L., "Statistical Characterization of NP-Complete Problems", Foundations of Computer Science Conference, World Computer Congress, Las Vegas, Nevada, July 14-17, 2008.
[3] Duffany, J.L. "Systems of Inequations", 4th LACCEI Conference, Mayaguez, PR, June 21-23, 2006.
[4] Duffany, J.L. "Generalized Decision Function and Gradient Search Technique for NP-Complete Problems", XXXII CLEI Conference, Santiago Chile, August 20-23, 2006.
[5] E. Horowitz, S. Sahni, "Fundamentals of Computer Algorithms", Computer Science Press, Maryland, 1978.
[6] R. M. Karp, "Reducibility among Combinatorial Problems", In Complexity of Computer Computation, pages 85-104. Plenum Press, New York, 1972.
[7] Weisstein, E. W., "NP-Hard Problem." From MathWorld--A Wolfram Web Resource. http://mathworld.wolfram.com/NP-HardProblem.html
[8] Weisstein, E. W., "NP-Complete Problem." From MathWorld--A Wolfram Web Resource: http:// mathworld.wolfram.com/NP-CompleteProblem.html
[9] Weisstein, E. W., "Inequation." From MathWorld--A Wolfram Web Resource. http:// mathworld.wolfram.com /Inequation.html
[10] Wikipedia – Inequation page: http://en.wikipedia.org /wiki/Inequation
[11] J. Manuch, D.R. Gaur, "Fitting protein chains to cubic lattice is NP-complete", Journal of Bioinformatics and Computational Biology, Vol. 6, No. 1. (February 2008), pp. 93-106.
[12] S. Russell and P. Norvig, Ärtificial Intelligence, A Modern Approach", Chapter 5, Second Edition, 2003, Prentice Hall, ISBN 0-13-790395-2.