Ariel

18 **Rapid #: -3031363** **IP: 129.93.17.64**

Status	Rapid Code	Branch Name	Start Date
Pending	GZN	Main Library	12/14/2009 7:53:12 AM

CALL #:	**Q334 .I568a**
LOCATION:	**GZN :: Main Library :: stacks**
TYPE:	Article CC:CCL
JOURNAL TITLE:	IJCAI
USER JOURNAL TITLE:	International Joint Conference on Artificial Intelligence
GZN CATALOG TITLE:	IJCAI :
ARTICLE TITLE:	A new method for solving constraint satisfaction problems
ARTICLE AUTHOR:	R. Seidel
VOLUME:	1
ISSUE:	
MONTH:	
YEAR:	1981
PAGES:	338--342
ISSN:	1045-0823
OCLC #:	
CROSS REFERENCE ID:	[TN:391537][ODYSSEY:129.93.17.103/ILL]
VERIFIED:	

BORROWER:	**LDL :: Main Library**
PATRON:	**Berthe Y. Choueiry**
PATRON ID:	choueiry
PATRON ADDRESS:	
PATRON PHONE:	
PATRON FAX:	
PATRON E-MAIL:	
PATRON DEPT:	
PATRON STATUS:	
PATRON NOTES:	

University of Nebraska Lincoln

RAPID ILL

A NEW METHOD FOR SOLVING CONSTRAINT SATISFACTION PROBLEMS

Raimund Seidel

Department of Computer Science
University of British Columbia

ABSTRACT

This paper deals with the combinatorial search problem of finding values for a set of variables subject to a set of constraints. This problem is referred to as a constraint satisfaction problem.

We present an algorithm for finding all the solutions of a constraint satisfaction problem with worst case time bound $O(m*k^{f+1})$ and space bound $O(n*k^{f+1})$, where n is the number of variables in the problem, m the number of constraints, k the cardinality of the domain of the variables, and $f<n$ an integer depending only on a graph which is associated with the problem. It will be shown that for planar graphs and graphs of fixed genus this f is $O(\sqrt{n})$.

I. INTRODUCTION

Many problems in diverse fields of computer science can be formulated as constraint satisfaction problems: a number of variables (sometimes called units) are to be assigned values (labels), such that certain given constraints on subsets of these variables are satisfied. Instances of constraint satisfaction problems range from graph theory and automata theory problems, like graph colouring and automata homomorphism, to problems in AI such as scene analysis and combinatorial puzzles. For representative examples see [4]. Problems involving more general constraints are treated in [6].

Algorithms to solve these problems usually rely on backtracking and/or on some forms of relaxation methods or look-ahead operators ([1], [3], [12], [10], [2], [4], [5], [9]). All these algorithms seem to work well most of the time. But they can behave badly in some cases and do not allow a tight worst case analysis. This is not surprising as Montanari [11] showed that the general constraint satisfaction problem is NP-complete.

One of the reasons why backtrack algorithms have a potentially bad behaviour is the fact that they use a minimal amount of space. Observe for instance, that a simple exhaustive backtrack search has an exponential running time but uses only a linear amount of space. One can view the relaxation methods and look-ahead operators proposed in the literature as attempts to invest in space in order to save time.

In the following sections we develop an algorithm which invests in space heavily. Using the terminology of backtrack search and search trees one can say that the saving in time is achieved by identifying initial segments of a search tree which are effectively identical, that is, they differ only on variables which do not constrain the remaining uninstantiated variables. However, we found it advantageous to formulate our algorithm not in terms of backtrack search but using the concept of dynamic programming. Thus in contrast to other methods, our algorithm permits easy analysis of its time and space complexity.

II. THE PROBLEM

A constraint satisfaction problem (CSP) can be defined as follows: given is a set of variables $X_1,\dots,X_n$ and associated with each variable X_i a domain D_i of values. Furthermore, on some subsets of the variables constraints are given, limiting possible value tuples for those variables. A solution of a CSP is an n-tuple of values $(a_1,\dots,a_n) \in D_1 x \dots x D_n$, which simultaneously satisfies all given constraints. The complete set of solutions of a CSP is the subset of $D_1 x \dots x D_n$, comprising exactly all the solutions. A CSP is called unsatisfiable if its complete set of solutions is empty.

For our purposes all domains are finite. We also assume that the domains of all n variables are of equal cardinality, $|D_i|=k$ for $i=1,\dots,n$. We shall see later on, that this assumption is just a convenience for the sake of analysis and by no means vital to the algorithm to be proposed.

Furthermore we will restrict our attention to CSPs involving only binary constraints. This restriction seems more critical. But it will be seen that the method to be presented can be applied to general CSPs without much modification.

Montanari [11] pointed out that a CSP only involving binary constraints can be represented by a graph. Let us call it the constraint graph. Each of its vertices corresponds to a variable. Two vertices are adjacent iff there is a constraint between the corresponding two variables. In the following we will feel free to call a vertex a variable or vice versa, or to identify edges with constraints.

III. A SIMPLE EXAMPLE

Let us look at the following example. Let us assume we have a CSP involving 10 variables and 19 binary constraints, and it can be represented by the constraint graph given in figure 1. Let C_{ij} be the constraint between variables X_i and X_j.

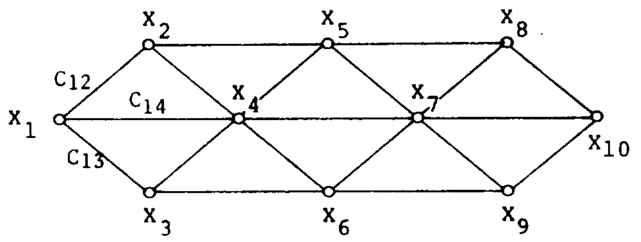


Figure 1

We can find out whether such a CSP is satisfiable in the following way:
Create a ternary constraint relation C_{123} for X_1, X_2, and X_3 which comprises all value triples for those variables allowed by C_{12} and C_{13}. Next, using C_{123} construct a ternary constraint relation C_{234} which comprises all value triples for X_2, X_3, and X_4 which permit a value for X_1, such that C_{12}, C_{13}, C_{14}, C_{24}, and C_{34} are satisfied. Note that, as indicated in figure 2 by the shaded lines, X_2, X_3, and X_4 "cut" the constraint graph and thus all of the influence of X_1 on the CSP is subsumed by C_{234} and therefore X_1, C_{14}, C_{24}, C_{34}, and C_{123} are of no importance any more. (This is indicated by the dashed lines in figure 2.) Also note that if C_{234} is empty the CSP must be unsatisfiable.

In the same manner, we construct using C_{234} a ternary relation C_{345} for X_3, X_4, and X_5, which comprises all value triples allowed by C_{34} and C_{45}, and permits values for X_1 and X_2. Again, if C_{345} is empty the CSP must be unsatisfiable and we can stop. Otherwise we continue in the same way and construct ternary relations C_{456}, C_{567}, C_{678}, C_{789} and C_{8910}. If any of those relations is empty, the CSP is unsatisfiable. If C_{8910} is nonempty we can generate a general solution for the CSP by using the created ternary constraints and instantiating the variables in the reverse order to the order in which they were discarded.

If each of the variables X_i can assume k different values, then any of the ternary relations above can have at most k^3 elements. Therefore only k^4 combinations of value triples and values need to be considered for the construction of a new ternary relation. Hence a CSP representable by such a graph can be decided in $O(k^4)$ steps. Note that this worst case complexity is completely independent of the specific instances of the constraint relations C_{ij}.

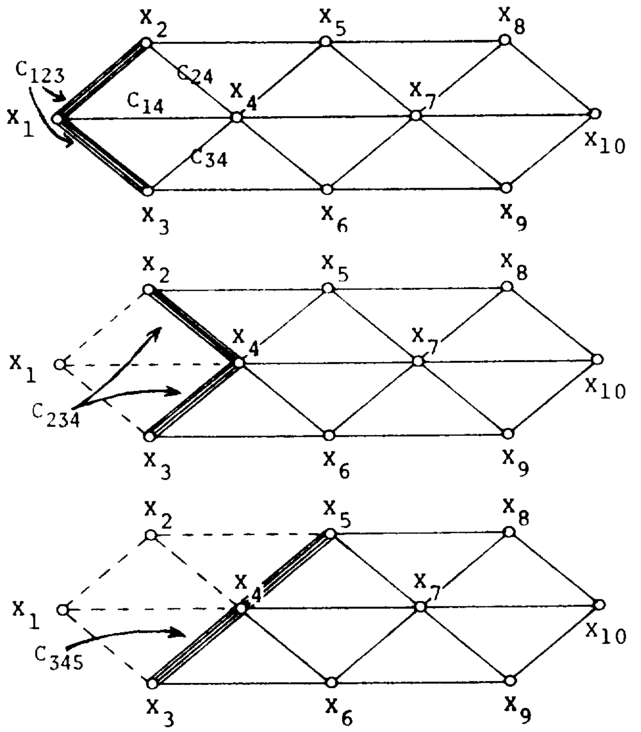


Figure 2

IV. THE INVASION PROCEDURE

In order to describe, how the method outlined above can be generalized so that it can be applied to an arbitrary constraint graph, we need a few definitions.

Given an n-vertex graph G, call a sequence $\{G_i\}$, i=1,...,n, of induced subgraphs of G, where the number of vertices in G_i is i and G_i is a subgraph of G_{i+1}, an <u>invasion of G</u>. We call the set F_i of vertices of G_i which are adjacent to vertices not in G_i the <u>front of G_i</u>. Vertices in G_i-F_i are called <u>conquered vertices</u>. The <u>front length f_i of G_i</u> is the number of vertices in F_i. <u>The front length of an invasion</u> is the maximum front length of the subgraphs G_i involved. An invasion of a graph G is called <u>optimal</u> if its front length is not greater than the front length of any other invasion of G.

Given an invasion $\{G_i\}$, i=1,...,n, of a constraint graph G, we claim that satisfiability of a corresponding CSP can be decided by the procedure outlined below:

> For i=1 to n inductively find all value tuples for the front vertices F_i of G_i which are consistent with one of the allowed value tuples of F_{i-1}. If there are no such tuples, stop and report the CSP unsatisfiable.

The correctness of this algorithm follows by induction using two observations:

Let $<X_1,\ldots,X_i>$ be the i-tuple of the vertices in G_i arranged in a way such that $X_1,\ldots,X_{f_i}$ are the vertices in F_i.

i) The conquered vertices $X_{f_i+1},\ldots,X_i$ are not involved in constraints with variables not in G_i.

ii) $<a_1,\ldots,a_{f_i}>$ is made a valid labeling tuple for the vertices in F_i if and only if there are values $a_{f_i+1},\ldots,a_i$ for the conquered vertices $X_{f_i+1},\ldots,X_i$ such that $<a_1,\ldots,a_i>$ is a valid labeling for the vertices in G_i, i.e. $<a_1,\ldots,a_i>$ satisfies all constraints within G_i.

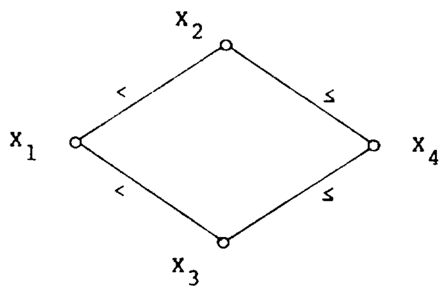


Figure 3

V. HOW TO CONSTRUCT ALL SOLUTIONS OF A CSP

Given an invasion $\{G_i\}$, i=1,...,n, for the constraint graph of a CSP, the above procedure just answers the question whether the CSP is satisfiable. But it can be improved to render a graph which represents the complete set of solutions of the CSP. Call this graph the solution graph of a CSP with respect to invasion $\{G_i\}$. This solution graph is a circuit-free directed multigraph with labeled arcs. (To avoid confusion let us use the terms node and arc for the solution graph, and the terms vertex and edge for the constraint graph.) It has two distinguished nodes such that the set of directed paths between these two nodes corresponds one to one with the complete set of solutions of the CSP.

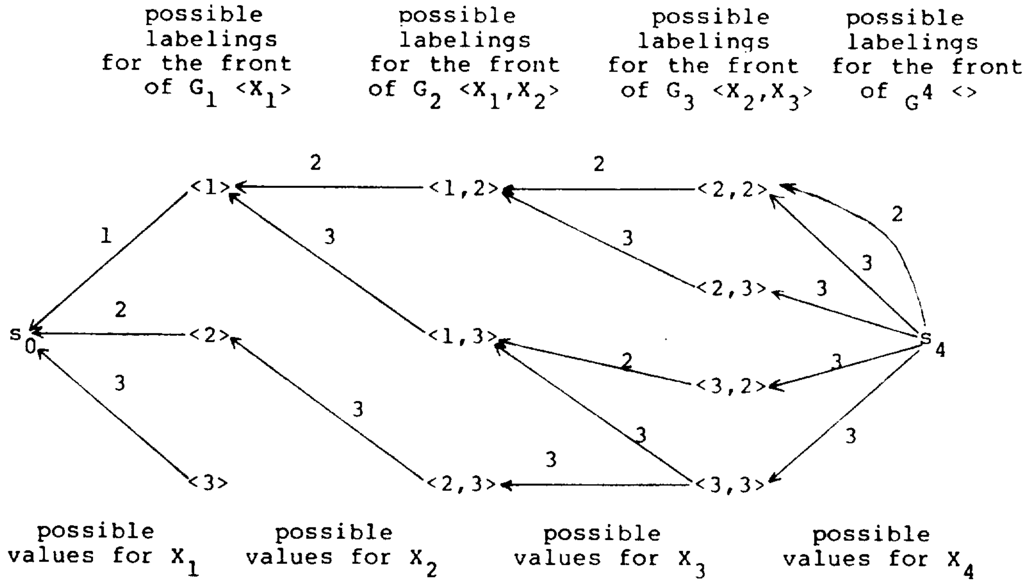
Consider the following simple example: four variables $X_1,\ldots,X_4$ are given; each variable is to be assigned an integer between 1 and 3 such that the constraints $X_1<X_2$, $X_1<X_3$, $X_2\le X_4$, and $X_3\le X_4$ hold. Figure 3 shows the constraint graph G for this CSP, Figure 4 shows the solution graph with respect to the invasion $\{G_i \mid G_i$ is the subgraph of G induced by the vertices X_j , $j\le i\}$. The labels along the directed paths from s_4 to s_0 represent all the solutions for this CSP.

We are now ready to state an improved version of the invasion procedure which constructs the solution graph of a CSP with respect to a given invasion.

Assume a CSP with n variables involving only binary constraints is given. For its constraint graph an invasion $\{G_i\}$, i=1,...,n, is given. Let G_0 be the empty set and let X_i be the single vertex of G_i-G_{i-1}.

Furthermore let S_i , i=1,...,n, be disjoint sets of nodes of the solution graph. The elements of S_i shall be named by labeling tuples of the front vertices F_i of G_i. (By convention let there be only one labeling tuple for an empty set of variables. Thus, if F_i is empty, S_i contains at most one element, let us call it s_i.)

```
Initially $S_i$ is empty for i=1,...,n.
$S_0$ is set to $\{s_0\}$.
For i=1 to n do:
 For each value tuple $a$ in $S_{i-1}$ and for each
 value c of $X_i$ whose combination satisfies all
 constraints between the vertices in $F_{i-1}$ and
 $X_i$ do:
    Let $b$ be the resulting value tuple for the
    vertices in $F_i$.
    Set $S_i$ to the union of $S_i$ and $\{b\}$.
    Construct an arc from $b$ to $a$ and label it
    with c.
```



possible labelings for the front of G_1 $<X_1>$
possible labelings for the front of G_2 $<X_1,X_2>$
possible labelings for the front of G_3 $<X_2,X_3>$
possible labelings for the front of G_4 $<>$
$<1>$
$<2>$
$<3>$
s_0
$<1,2>$
$<1,3>$
$<2,3>$
$<2,2>$
$<2,3>$
$<3,2>$
$<3,3>$
s_4
1
2
3
possible values for X_1
possible values for X_2
possible values for X_3
possible values for X_4

Figure 4

Claim:
1. The above procedure yields a circuit-free directed multigraph.
2. The tuple $<a_1,\dots,a_n>$ is a solution of the CSP if and only if there is a directed path from s_n to s_0 whose arc label sequence is $a_n,\dots,a_1$.

Proof:
1. The solution graph is circuit-free because for all i and j, $i \neq j$, S_i and S_j are disjoint and there are only arcs from nodes in S_i to nodes in S_{i-1}. It is possible to have more than one arc between a node in S_i and a node in S_{i-1} in the case that X_i is not a front vertex of G_i.

2. This statement follows from the inductive argument that the set of paths between a node $<a_1,\dots,a_{f_i}>$ in S_i and s_0 represents exactly all the solutions of the CSP restricted to G_i which have $a_1,\dots,a_{f_i}$ as instantiation of the front vertices of G_i.

Q.E.D.

It is now natural to ask about the complexity of this procedure. The following theorem gives an answer to this question. It is assumed that it takes constant time to determine whether two variables satisfy a common constraint.

Theorem:
Given is a CSP involving m binary constraints on n variables. Each of the variables can take on k different values. For the constraint graph of the CSP an invasion $\{G_i\}$, i=1,...,n, with front length f is given.
The above algorithms produces a solution graph of the CSP in time at most $O(m*k^{f+1})$ and uses space at most $O(n*k^{f+1})$.
Proof:
S_{i-1} can not have more than $k^{f_{i-1}} \leq k^f$ elements. Thus at most k^{f+1} combinations are possible between elements of S_{i-1} and values of X_i. Therefore there can not be more than k^{f+1} arcs from nodes in S_i to nodes in S_{i-1}. So the algorithm uses space $O(n*k^{f+1})$.
For each i at most f binary constraints need to be checked for each of the at most k^{f+1} combinations between elements in S_{i-1} and values of X_i. Therefore there are not more than $n*f*k^{f+1}$ checks. But overall there exist only m constraints. Thus only $m*k^{f+1}$ checks are necessary, and the algorithm uses time $O(m*k^{f+1})$.

Q.E.D.

Implementation of this procedure should be straightforward. But one should carefully select the data structure to represent the sets S_i so that set insertion and set enumeration can be done quickly, but no excessive amount of space is used. The actual usefulness of this procedure will of course depend heavily on the front length of the invasion used, and on the actual sizes of the sets S_i to be constructed.

VI. THE GENERAL CSP

So far we have looked only at CSPs involving binary constraints. Can our procedure also handle general CSPs with constraints involving more than two variables? The changes and generalisations necessary to answer this question positively should be obvious: we only need to generalize the notion of a constraint graph; variables correspond again to vertices, and two vertices are adjacent if the corresponding two variables are involved in some common constraint. With this definition of a constraint graph only a few modifications in bookkeeping are required so that the invasion procedure can be applied to general CSPs.

VII. FINDING A GOOD INVASION

In order to make efficient use of the algorithm of the last section, one needs "good" invasions, that is invasions with small front length. But good invasions do not exist for all graphs. Consider a complete n-vertex graph: each of its invasions has front length n-1. Furthermore there are n! invasions for an n-vertex graph, but no good algorithm is known to select an optimal or almost optimal invasion. But if we restrict our attention to the class of planar graphs, we can exhibit an algorithm which computes an invasion with front length $O(\sqrt{n})$. Similar algorithms exist for the classes of graphs of fixed genus. But we will concentrate on planar graphs. The importance of this class is illustrated by the fact that for instance most CSPs arising in A.I. vision involve planar constraint graphs.

In the construction of the invasion of a planar graph we will make use of a planar separator theorem by Lipton and Tarjan [7]:

> Let G be an n-vertex planar graph. The vertices of G can be partitioned into three sets A, B, C, such that no edge joins a vertex in A with a vertex in B , neither A nor B contains more than 2n/3 vertices, and C contains no more than $\sqrt{8n}$ vertices.

Lipton and Tarjan also exhibit an algorithm which finds such a partition in O(n) time. In [8] they show how this theorem can be extended to graphs of arbitrary genus.

In the previous section we formally defined an invasion of a graph as a sequence of induced subgraphs. It should be clear that each invasion of a graph G induces a numbering on the vertices of G, and vice versa. Thus if the vertices of G are numbered $X_1,\dots,X_n$, then $\{G_i \mid G_i$ is the subgraph of G induced by the vertices X_j, with $j \leq i\}$, i=1,...,n, is clearly an invasion of G. So finding an invasion for a graph is equivalent to finding the corresponding numbering of its vertices.

In the following we specify a divide and conquer type procedure INVADE-PLANAR-GRAPH which numbers the vertices of a planar graph. We shall mean by "invade S starting with i", where S is a subset of the n vertices of a graph and i an integer, $1 \leq i \leq n$, that each integer between i and $i+|S|-1$ is assigned to one of the vertices in S.

```
INVADE-PLANAR-GRAPH (G,1)
where INVADE-PLANAR-GRAPH (G,i) is:
    If there are no more than 4 vertices in G,
      invade them starting with i.
    Otherwise, using Lipton's and Tarjan's method,
      partition the vertices of G into three sets
      A, B, C, such that there are no edges between
      vertices in A and vertices in B.
      Invade C starting with i.
      INVADE-PLANAR-GRAPH ($G_A$, $i+|C|$).
      INVADE-PLANAR-GRAPH ($G_B$, $i+|C|+|A|$).
      ($G_A$ and $G_B$ are the subgraphs induced by A and
      B respectively.)
```

<u>Claim</u>:
Given a planar n-vertex graph G, the above procedure yields an invasion for G whose front length f is smaller than $16*\sqrt{n}$.
<u>Proof</u>:
Let $\{G_i\}$, i=1,...,n, be the invasion induced by the numbering of the vertices achieved by the above procedure. Let f_i denote the front length of G_i for all i. Let A, B, C be the three sets into which the vertices of G are partitioned.
For all $i \leq |C|$, f_i must be less than $|C| \leq \sqrt{8n}$.
For all i, $|C| < i \leq |C|+|A|$, f_i must be less than $|C|+f_A$, where f_A denotes the front length of the invasion of the subgraph induced by A.
For all i, $|C|+|A| < i \leq n$, f_i must be less than $|C|+f_B$ (with f_B defined as f_A), because there are no edges between vertices in A and B and thus all the vertices in A must be conquered.
Thus the following inequality holds:

$$f \leq \sqrt{8n} + \max(f_A, f_B)$$

Using the fact that neither A nor B contains more than 2n/3 vertices, we can derive the following recursive relation for the front length:

$$f(n) = n \qquad \text{for } n \leq 4$$

$$f(n) \leq \sqrt{8n} + f(|2n/3|) \qquad \text{otherwise}$$

It can easily be shown that f(n) is bounded from above by $c\sqrt{n}$, where c is not greater than 16. Thus the front length f is smaller that $16*\sqrt{n}$. Q.E.D.

This result leads immediately to the following
<u>Corollary</u>:
A CSP with n variables, m constraints, k values for each variable, and a planar constraint graph can be solved in time $O(m*k^{1+16\sqrt{n}})$ and space $O(n*k^{1+16\sqrt{n}})$.

VIII. CONCLUSION

We have described new algorithms to decide satisfiability, or to compute the complete set of solutions of a given CSP. The worst case complexities of the algorithms depend heavily on the structure of the constraint graph of the CSP, so we do not claim that our method will be efficient for all CSPs. But we could show that for CSPs with planar constraint graphs this algorithm leads to a considerable improvement in the asymptotic worst case complexity of the problem. It remains to be seen how the proposed algorithms will behave in practical applications.

ACKNOWLEDGEMENTS

I would like to thank Alan Mackworth for bringing this problem to my attention. I also want to thank Alan, as well as David Kirkpatrick and Derek Corneil for helpful discussions and valuable comments. Finally I want to thank an anonymous referee who pointed out the relation between the invasion method and standard backtrack search.

REFERENCES

[1] E.C.Freuder, "Synthesizing Constraint Expressions," Communications of the ACM, vol.21, no.11 (1978), pp.958-966.
[2] E.C.Freuder, "A Sufficient Condition for Backtrack Free Search," to appear in JACM.
[3] J.Gaschnig, "Experimental Case Study of Backtrack vs. Waltz-type vs. New Algorithms for Satisficing Assignment Problems," Proc. 2nd CSCSI Conf. (1978), pp. 268-277.
[4] R.M.Haralick and L.G.Shapiro, "The Consistent Labeling Problem: Part I," IEEE Trans. Pattern Analysis and Machine Intelligence, vol. PAMI-1, no.2 (1979), pp. 173-184.
[5] R.M.Haralick and L.G.Shapiro, "The Consistent Labeling Problem: Part II," IEEE Trans. Pattern Analysis and Machine Intelligence, vol. PAMI-2, no.3 (1980), pp. 193-203.
[6] J.L.Lauriere, "A Language and a Program for Stating and Solving Combinatorial Problems," Artif.Intelligence, vol.10, no.1 (1978), pp. 29-127.
[7] R.J.Lipton and R.E.Tarjan, "A Separator Theorem for Planar Graphs," SIAM J.Appl.Math., 36 (1979), pp. 177-189.
[8] R.J.Lipton and R.E.Tarjan, "Applications of a Planar Separator Theorem," SIAM J.Comput., vol.9, no.3 (1980), pp. 615-627.
[9] J.J.McGregor, "Relational Consistency Algorithms and their Applications in Finding Subgraph and Graph Isomorphism," Info.Sci. 19 (1979), pp. 229-250.
[10] A.K.Mackworth, "Consistency in Networks of Relations," Artif.Intelligence, vol.8, no.1 (1977), pp. 99-118.
[11] U.Montanari, "Networks of Constraints: Fundamental Properties and Applications to Picture Processing," Info.Sci. 7 (1974), pp. 95-132.
[12] D.L.Waltz, "Understanding Line Drawings of Scenes with Shadows," in P.H.Winston (ed.) <u>The Psychology of Computer Vision</u>, McGraw-Hill, New York, 1975, pp. 19-91.