Formula Dissection: A Parallel Algorithm for Constraint Satisfaction

John H Reif
Department of Computer Science
Duke University
Durham, NC

Simon Kasif
Department of Biomedical Engineering
Boston University
44 Cummington St. Boston, MA 02215

Deepak Sherlekar
Virage logic Corporation

Abstract

Many well-known problems in Artificial Intelligence can be formulated in terms of systems of constraints. The problem of testing the satisfiability of propositional formulae (SAT) is of special importance due to its numerous applications in theoretical computer science and Artificial Intelligence. A brute-force algorithm for SAT will have exponential time complexity $O(2^n)$, where n is the number of Boolean variables of the formula. Unfortunately, more sophisticated approaches such as resolution result in similar performance in the worst case. In this paper we present a simple and relatively efficient parallel divide-and-conquer method to solve various subclasses of SAT. The dissection stage of the parallel algorithm splits the original formula into smaller subformulae with only a bounded number of interacting variables. In particular, we derive a parallel algorithm for the class of formulae whose corresponding graph representation is planar. Our parallel algorithm for planar 3-SAT has worst-case performance of $2^{O(\sqrt{n})}$ on a PRAM (parallel random access model) computer. Applications of our method to constraint satisfaction problems are discussed.

Keywords: Boolean formula, combinatorial search, parallel algorithm, graph, dissection

1 Introduction

Many of the well-known problems in Artificial Intelligence can be formulated as systems of symbolic constraints [1, 2, 3, 4]. Much research has been directed towards developing effective search methods to solve the problem in general and for domain specific applications ([1, 2, 3, 4, 5, 6, 7]). The majority of the solutions utilize local constraint propagation techniques (discrete relaxation) to achieve global consistent solutions. Relaxation techniques have been used extensively in the context of image understanding and interpretation ([2, 3, 4, 7, 8, 9]) as well as planning, natural language analysis and common sense reasoning[1].

Since the constraint propagation procedures appear to operate locally, it was believed that relaxation techniques had a natural parallel implementation. However it was shown in [10] that local consistency is in some sense inherently sequential (in the worst case). This result motivated us to examine possible parallel approaches to constraint solving with "good" worst-case performance. A natural approach is divide-and-conquer. Namely, decomposing the system of constraints into smaller systems and solving each independently. However, the number of interacting constraints arising from the obvious decomposition causes a combinatorial explosion. In this paper we propose a technique that exploits the structure of the constraint graph to reduce the size of the search sphere.

The problem of testing the satisfiability of propositional formulae (SAT) is an important instance of constraint satisfaction systems because of its special role in numerous applications. SAT has been used extensively in theoretical computer science to demonstrate the intractability (NP-Completeness) of many problems [11]. In Artificial Intelligence, SAT is used as a basic building block of many automated deduction systems. The time complexity of brute-force algorithms is exponential (of order 2^n) where n is the number of Boolean variables. Unfortunately, more sophisticated approaches such as resolution result in a similar performance in the worst case [12, 13].

The central role played by SAT in such diverse and important applications makes it a natural candidate for presenting any approach to solving constraint satisfaction problems. Our approach is therefore presented in the context of solving many subclasses of SAT. It can be applied to constraint systems that arise in computer vision. In fact the research presented in this paper was originally motivated by its possible application to the problem of recognizing trihedral scenes in computer vision. Previous approaches to the problem have used the consistent labeling method. The time complexity of this approach is known to be exponential in the worst case (see the discussion in [14]. A recent result due to Kirousis and Papadimitriou [15] established the NP-Completeness of the problem by transformation from Planar 3-SAT. *Planar 3-SAT* is the problem of testing the satisfiability of Boolean formulae in 3CNF form whose graph representation is planar. This result motivated us to carefully examine this class of formulae, and exploit the structure of the graphs representing them. Such an approach has been used with success to solve significant practical subclasses of other constraint satisfaction problems (e.g., see [16]).

We assume a shared memory random access model for a parallel computer known as the *PRAM (parallel random access model)*. This PRAM computation model is an extension of the conventional sequential computer model known as the *RAM(random access model)*. Both these models allow for a con-

stant number of memory registers, as well as a random-access memory, when each memory location is indexed from its integer address, and can be written or read from in one time step. We assume that each memory and register location can only hold Boolean value or an integer whose number of bits is logarithmic in the number of inputs. Also, both these models allow the computer to execute operations, which include Boolean and as well as basic arithmetic (addition and multiplication) operations. However, the PRAM model allows for synchronous, parallel execution of these operations. The *processor bound* of a PRAM computation is the maximum number of operations that are executed in parallel, the *time bound* is the total number of synchronous steps executed in the computation, and the *work bound* is the total number of operations executed in the computation. We assume the (CREW-PRAM) [17] variant of the PRAM model that no parallel writes are executed simultaneously at the same memory or register location.

In this paper we present a simple and relatively efficient parallel method to solve various subclasses of SAT. These classes are defined by the representation of propositional formulae by graphs (see section 2). The importance of one such class, viz. Planar 3-SAT, has already been mentioned above. Although Planar 3-SAT has already been shown to be NP-Complete (see [18]), applications of separator decompositions of planar graphs [19] will be applied to considerably decrease the work to $(2^{O(\sqrt{n})})$ in the worst case. Our algorithm is highly parallel, and its implementation on a PRAM has $O(\log^3 n)$ time complexity and $(2^{O(\sqrt{n})})$ processor complexity.

A formal definition of the graph representation of propositional formulae is given in Section 2. Intuitively, the graph G of a formula F is a bipartite graph whose 2 sets of nodes correspond respectively to the variables and clauses of F. A 'variable node' and a 'clause node' are connected iff the variable occurs in the clause. The graph thus constructed can be thought of as a constraint graph where the assignment of values to any variable node is constrained by the assignment of values to other variable nodes, which are connected to the same clause node. Our algorithms use divide-and-conquer, where the dissection step repeatedly splits the graph into smaller subgraphs with only a bounded number of interacting constraints. The use of separator theorems (see next section) allows us to partition certain classes of sparse graphs into two components of roughly equal size by removing only a few vertices. This allows us to contain the exponential growth generated by interacting constraints.

The worst-case work complexity of our algorithms is $(2^{O(\sqrt{n})})$ for Planar 3-SAT. This result is encouraging since $2^{c\sqrt{n}}$ grows significantly slower than 2^n for modest c. In fact, it may be considered to be subexponential for inputs less than 10^6 (see discussion in Section 5). More generally, for formulae represented by a class of graphs having an n^ϵ separator theorem, $\epsilon < 1$, our algorithms have worst case complexity of $2^{O(n^\epsilon)}$. An n^ϵ separator theorem allows us to partition a graph into two roughly equal parts by removing $O(n^\epsilon)$ vertices. [20].

The outline of the paper is as follows. Section 2 contains the preliminary background necessary to understand the constructions that follow. Specifically, we provide formal definitions of the graph representations of propositional formulae and of the subclasses of propositional satisfiability studied in this paper. We also include definitions of the graph separator theorems used in our algorithms.

In section 3 we develop and describe our algorithm for testing satis-

fiability of propositional logic formulae and analyze its performance. Sections 4 and 5 discuss relevant literature and summarize the main results reported in the paper.

2 Preliminary Notations

2.1 Graphs and Graph Separators

We assume the reader is familiar with standard graph theoretical concepts and definitions [21]. The basic terminology concerning separator theorems is reviewed below. Our primary interest is in classes of graphs that are easily separable. A graph G is said to be easily separable if it can be partitioned into two subgraphs of approximately equal size by removal of very few vertices. To be able to apply our algorithm inductively we also need that the subgraphs thus created are also easily separable in the same sense. Theorems that prove classes of graphs to be easily separable are called separator theorems. A separator theorem is formally defined below:

Definition 4: Let S be a class of graphs. The class S has an $f(n)$ *separator theorem* if S is closed under the subgraph relation and there exists constants $\alpha < 1, \beta > 0$, such that for any n-vertex graph G in S having nonnegative vertex costs summing up to no more than 1, the vertices of G can be partitioned into 3 sets A, B, C such that no vertex in A is adjacent to a vertex in B, neither A nor B have total cost exceeding α, and C contains no more that $\beta f(n)$ vertices.

For our purposes we need a somewhat stronger definition of a separator theorem given below:

Definition 5: Let S be a class of graphs having an $f(n)$ separator theorem with $\alpha = \frac{1}{2}$. then S is said to have a *strong* $f(n)$ *separator theorem.*

It is possible to derive a strong separator theorem from a weak separator [20]. The strong separator theorems resulting from the above technique are given below for two different values of $f(n)$ below.

Lemma 2.1: [20] If a class of graphs S has an n^α separator theorem, $0 < \alpha < 1$, then S has a strong n^α separator theorem. If S has a $log^k n$ separator theorem, $k \geq 0$, then S has a strong $log^{k-1} n$ separator theorem.

A well known separator theorem is the $\sqrt{n}$ separator theorem for planar graphs [22], which is reproduced below.

Lemma 2.2 [22] Let G be any n-vertex planar graph having nonnegative vertex costs summing to no more than 1. Then the vertices of G can be partitioned into three sets A, B and C, such that no edge joins a vertex in A with a vertex in B, neither A nor B have total costs exceeding $\alpha = \frac{1}{2}$, and C contains no more than $2\sqrt{2}\sqrt{n}/(1 - \sqrt{2/3})$ vertices.

Djidjev [23] has improved the size of C by a constant factor. However, in our analysis, we shall be interested in only the order of the size of C.

2.2 The Satisfiability Problem for Propositional Logic Formulae

Let $V = \{v_1, \cdots, v_n\}$ be a set of Boolean variables. If v is a variable in V then v and $\bar{v}$ are called *literals* over V. An *interpretation* Φ of V is a $1-1$ mapping from V to $\{0, 1\}$ (alternatively, TRUE,FALSE). We say that $v \in V$

is 'true' under Φ if $\Phi(v) = 1$: otherwise v is 'false'. A literal v is *true* under interpretation Φ iff the v is a variable and is 'true', or is the complement $\bar{v}$ of a variable v which is false. A *clause* C over V is a set of literals over V. It denotes a disjunction of the literals comprising it. It is *satisfied* by an interpretation Φ iff at least one of the literals in it is true under Φ. We then say that Φ *satisfies* C.

A Boolean formula is in *Conjunctive Normal Form (CNF)* [24] if it is expressed as the conjunction of a set of clauses $F = \{C_1, \cdots, C_m\}$. F is *satisfiable* iff there exists an interpretation Φ that simultaneously satisfies all the clauses in F.

The question of whether there is an interpretation Φ that satisfies a given set of clauses is known as the *satisfiability problem (SAT)*. There are numerous known methods to solve SAT (see [24]), of which resolution is the most popular one. SAT was shown to be NP-Complete by Cook. Thus it is considered unlikely that the problem will have general subexponential solutions. In fact, for the resolution method, it has been shown that there are clauses for which proofs of unsatisfiability are exponentially long [12, 13].

A CNF Boolean formula $F = \{C_1, \cdots, C_m\}$ is said to be a *k-conjunctive normal form (k-CNF formulae)* iff none of its clauses contains more than k literals. The problem of testing the satisfiability of a formula in k-CNF (k-SAT) is known to be NP-Complete for $k \geq 3$ [11]. However, 2-SAT has a simple linear work algorithm. Moreover, it is in NC, the class of problems solvable in polyalgorithmic time on a PRAM using a polynomial number of processors [25, 26, 27]. In this paper we develop PRAM algorithms for a restricted class of 3-CNF formulae. Recall that the PRAM model we assume (CREW-PRAM), which allows concurrent reads but prohibits concurrent writes [17].

2.3 3-CNF formulae and Their Graph Representation

Let $F = \{C_1, \cdots, C_m\}$ be a 3-CNF formula over a set of variables V. Then $H_F = (V, F)$ denotes the hypergraph of F. Its vertices are the variables in V and every hyperedge corresponds to a unique clause C_i in F . A vertex corresponding to a variable v_i is incident with a hyperedge C_i iff either v_i or $\bar{v_i}$ occurs in the clause C_i. An example of a formula in 3-CNF and its hypergraph representation is given in Figure 1.

An alterative representation of the formula $F = \{C_1, \cdots, C_m\}$ is in terms of a bipartite graph $G_F = (V', E)$, with vertex set V' which is the union of the set of variables V and the clauses of F, and with edge set $E = \{(c_i, v_j) : v_j \in C_i \text{ or } \bar{v_j} \in C_i\}$. Note that this representation is congruent with the hypergraph representation, with every hyperedge replaced by a unique vertex along with the k arcs joining it with the 'variable-vertices' that were linked by the hyperedge.

The representation of formulae by graphs allows us to deal efficiently with formulae corresponding to graphs that are easily separable in the sense explained in Section 2.1. Partitioning the graph of a formula by removal of variable-vertices yields subgraphs which correspond to formulae that do not share any variables. this leads to an efficient divide and conquer method developed in the next section.

3 An Efficient Parallel Algorithm for Subclasses of 3-SAT

In this section we develop a divide-and-conquer algorithm for classes of 3-CNF formulae representable by bipartite graphs that are easily separable. The restriction of 3-SAT that we consider are not necessarily amenable to a polynomial time solution. In fact, the restriction of 3-SAT to formulae representable by planar bipartite graphs, viz. Planar 3-SAT (or P3-SAT) is also NP-Complete (see [18]). However, the complexity of the sequential implementation of our algorithm for P3-SAT is $O(n^2 2^{c\sqrt{n}})$ for a fixed constant c, which is significantly better than $O(2^n)$. The parallel implementation of our algorithm on the PRAM runs in polylog time using $O(n^2 2^{c\sqrt{n}})$ processors.

3.1 Inspiration from the Davis & Putnam Procedure

Davis & Putnam in their original procedure [28] used a divide and conquer principle in which a Boolean formula F in CNF form is represented (with respect to a variable v), as a conjunction of three CNF formulae (written as sets of clauses) S_0, S_1, and S_2 such that: (i) S_0 is free of variable v (ii) variable v occurs only positively in S_1, and (iii) variable v occurs only negatively in S_2. In that case we can delete variable v from S_1 and S_2 to obtain S_1' and S_2' respectively. Then F is unsatisfiable iff the two sets of clauses $S_0 \cup S_1'$ and $S_0 \cup S_2'$ are both unsatisfiable (see [24]). The Davis & Putnam rule is more restrictive than our method and consequently may be applied to a smaller class of problems.

There have been numerous attempts to cast propositional and first-order deduction in terms of graph rewriting techniques [29, 30, 31]. To the best of our knowledge none of the methods above used the separability properties of a graph to obtain efficient satisfiability procedure.

3.2 Informal Description of Our Parallel Formula Dissection

Let F be a set of clauses over V. Let v be some variable in V. Assume that we can partition F into two sets of clauses F^1 and F^2. Let $F^1_{v=0}$, $F^1_{v=1}$, $F^2_{v=0}$, and $F^2_{v=1}$ be the clauses generated from F^1 and F^2 by instantiating all occurrences of v in F^1 and F^2 to 0 or 1 respectively. Then

$$F = F^1_{v=1} \wedge F^2_{v=1} \vee F^1_{v=0} \wedge F^2_{v=0}$$

Hence F is satisfiable iff at least one of the two disjuncts above is satisfiable. The formulae comprising the two disjuncts do not share any variables, allowing computation of their satisfiability to be done in parallel.

This technique may be generalized to the following divide and conquer method. Let F^1 and $F^2 = F - F^1$ be two subsets of F sharing a set of variables $V' = \{L_1, L_2, \cdots, L_k\} \subseteq V$. For each of the 2^k possible interpretations of V, create an instance of F^1 and an instance of F^2 under the interpretation. Now the satisfiability of F can be tested by testing the satisfiability of these $2 * 2^k$ formulae, and then efficiently composing the results of the local tests.

Let the values of F^1, F^2, under the 2^k interpretations of V' be denoted F_i^1, F_i^2, respectively for $0 \leq i \leq 2^k - 1$. Then F is satisfiable iff at least one of both $F_i^1 \wedge F_i^2$, $0 \leq i \leq 2^k - 1$ is satisfiable. The satisfiability of the new formulae is tested by recursive application of the above steps until they are in a form that is amenable to an efficient solution by other means. For a parallel (sequential) algorithm, this implies performing the recursive step until the formula is in 2-CNF form (which is also known as *Horn form*, and as previously mentioned can be efficiently solved by known algorithms).

Note: The above technique is feasible if the set V' is small in size and can be computed efficiently. If the bipartite graph G_F of the formula F belongs to a class of easily separable graphs, then this set can be computed efficiently. However, most separator theorems in the literature would provide a separator set containing both variable and clause vertices that partition G_F. The separator we desire can be obtained by merely replacing the clause vertices in the separator set by all the variable vertices adjacent to them. For 3CNF formulae, the size of the resulting separator is at most 3 times the size of the original separator.

3.3 Our Parallel Formula Dissection Algorithm

Our parallel Formula Dissection (FD) Algorithm is given below. As mentioned in Section 2.2, the algorithm is described on the CREW-PRAM model of parallel computation.

Algorithm FD
INPUT:
A 3-CNF formula F represented by its bipartite graph G_F, where G_F belongs to a family separable graphs.

OUTPUT:
If F is satisfiable output '1' else output '0'.

1 If F is in 2-CNF apply the algorithm for 2-CNF. Return '1' if F is satisfiable and '0' otherwise.

2 Let the formulae F^1 and F^2 result from splitting G_F by its separator S. Let $|S| = J$.

3 For each interpretation Φ_i, $0 \leq i \leq 2^J - 1$, of S

4 Return $\bigvee_{i=0}^{2^J-1} \left(FD(F_i^1) \wedge FD(F_i^2)\right)$

END

The algorithm FD creates is an implicit tree-like calling structure. The base case, Step 1, is performed at the bottom level of the recursion. The results are sent upwards, where 'AND' and 'OR' operators are applied as required. See Fig. 2. Step 2 of the algorithm can make use of a preprocessing step which precomputes the entire separator tree.

The correctness of step 5 follows readily from the discussion in Section 3.1. The correctness of the algorithm can be established using induction on the depth

of the recursion. Thus we have the following theorem.

Theorem 1

Agorithm FD is correct.

3.4 Analysis

We shall carry out our analysis for graphs satisfying two kinds of separators: n^{α}-separators (for some constant α, where $0 < \alpha < 1$), and log^{k} n-separators (for some constant $k \geq 0$). The input consists of a set of $O(n)$ clauses. The number of variables is $O(n)$, which is the worst case for our algorithm. The separators are assumed to be strong separators (as defined above) to simplify the analysis. Let the computation of the separator of any n vertex graph belonging to the class of graphs under consideration require $O(\log^{2} n)$ time using $O(n^{3})$ processors. This is certainly true for planar and outerplanar graphs. then precomputing the separator tree takes $O(\log^{3} n)$ time using $O(n^{3})$ processors. The test for satisfiability of a ground formula in 3CNF takes $O(\log n)$ time using n processors. Thus given enough processors to handle all formulae generated, the algorithm can be shown to run in $O(\log^{3} n)$ time. The analysis below examines the number of processors required by our algorithm for the two kinds of separators listed above. The processor complexity of our PRAM algorithm is the same as the time complexity of its sequential counterpart. This can be readily established from the proof of the processor complexity in the analysis below using the fact that both 2-SAT and Horn satisfiability can be solved in linear time.

Theorem 2: 3-SAT for formulae represented by graphs having an n^{α} separator, $0 < \alpha < 1$, can be computed by a PRAM in time $O(\log^{3} n)$ using $O(n^{2}2^{O(n^{\alpha})})$ processors.
Proof: For the n^{α}-separator, the number of formulae N_i at depth i is given by:

$$N_i = N_{i-1}\left(2 * (2^{c(n/2^{i-1})^{\alpha}})\right) \text{ for some constant } c > 0.$$

If the recursion bottoms out at depth L, the number of formulae is at most $2^{L}2^{cn^{\alpha}}$. Thus the number of processors required is $O(n2^{L}2^{cn^{\alpha}})$. For $L = \log n$, this yields a processor complexity of $O(n^{2}2^{O(n^{\alpha})})$ time.

Planar graphs satisfy a $\sqrt{n}$-separator theorem ([22]). Thus we have
Corrolary 2.2: P3-SAT can be solved in $O(\log^{3} n)$ time using $O(n^{2}2^{O(\sqrt{n})})$ processors.

Naturally, we are not seriously considering using $O(n^{2}2^{O(\sqrt{n})})$ processors. The structure of our algorithm allows us to solve P3-SAT in time $O(n^{2}2^{O(\sqrt{n})}) \times (\log^{3} n)/P$ using P processors. Recall, this is worst case analysis. In many cases the actual complexity of our algorithm is better (see next section).

Theorem 3: 3-SAT for formulae represented by graphs having a strong $\log^{k} n$ separator can be computed on a PRAM in $O(\log^{3} n)$ time using $O\left(n^{2}2^{O(\log^{k+1} n)}\right)$ processors.

Proof: Proceed on the same lines as the proof of Theorem 1. The number

of formulae at depth L of the recursion is:

$$N_i = 2N_{i-1}2^{c\log^k(n/2^{i-1})} \text{ for some constant } c > 0,$$

which solves to $n \quad 2^{c\log^{k+1} n}$ formulae at depth $\log n$. Hence we need only $O\left(n^2 2^{c\log^{k+1} n}\right)$ processors.

For formulae represented by graphs having a strong 1-separator, that is they can be separated by removing only a constant number of vertices, the processor complexity is $O(n^{c+2})$. Thus for these formulae we need a polynomial number of processors to acheive logarithmic parallel time.

4 Discussion

In this paper we presented a simple but relatively efficient procedure for testing satisfiability of many classes of propositional formulae. as mentioned in the introduction, the ideas presented here are also applicable to solving other systems of constraints such as those arising in computer vision and connectionist networks. The basic idea is similar to the construction described in the paper and is based on the observation that the 2D projections in the polyhedra discussed in [7] generate plane graphs. Thus, using an algorithm similar to FD we can obtain a separation of the graph into two subgraphs by removing relatively few vertices. the removed vertices correspond to the interactive constraints in the interpretation. A similar observation was made in [32] where a sequential algorithm for constraint satisfaction was described that was also based on the principle of graph separators. Our method and the method described in [32] are related to the theoretical construction described in [19]. We have extended this construction to other problems and demonstrated its inherent parallelism.

There are several reasons to believe that the class of clauses represented by 'easily separable graphs' is quite general and may lead to efficient practical implementations.

1. All separable graphs are known to be sparse; that is, they have only a linear number of edges.
2. For $n \leq 10^5, \sqrt{n} \leq \log^k n$ for $k \geq 2$ Thus, for this range of n, our algorithms for planar graphs achieves subexponential $O\left(n^2 2^{c\log^2 n}\right) = O(n^{\sqrt{c}\log n})$ performance.
3. Whenever one variable in a clause is evaluated to I by an interpretation Φ, the entire clause may be deleted from the copy of the formula created by this interpretation.
4. On average, the number of variables in the separator set C, is likely to be much smaller that $O(\sqrt{n})$.

Thus it is likely that the actual complexity of the algorithm on typical problems occurring in practice will be significantly better than the worst case given in the analysis here. Recall that the worst case complexity of existing methods

is exponential. The technique presented in this paper may have immediate practical consequences if our analysis as presented in Section 3 carries over to the currently realizable massively parallel machines. Since our algorithms use divide-and-conquer, it appears that communication will not be a problem on other architectures that support tree-like computations [33, 34].

References

[1] Winston, P.H., *"Artificial intelligence"*, *Third Addition*, Addison-Wesley, (1992).

[2] Tsang, E. "Foundations of Constraint Satisfaction," *Academic Press, New York*, 1993.

[3] Dechter, R. "Constraint processing," *Morgan Kaufmann, San Fransisco, CA*, 2003.

[4] Apt, K. 'Principles of constraint programming," *Cambridge University Press, Cambridge, UK*, 2003.

[5] Steele, G. and Sussman, G., *"Constraints"*, *M.I.T. AI Memo*, 502 (November 1978).

[6] Sussman, G. and Stallman, R., *"Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis"*, *MIT AI Memo*, 380 (1976).

[7] Waltz, D., *"Understanding Line Drawings of Scenes with Shadows"*, *in The Psychology of Computer Vision*, Winston, P.H. (editor), McGraw-Hill, New York, (1975), 19-92.

[8] Mackworth, A.K., *"Consistency in Networks of relations"*, *Artificial Intelligence*, 8 (1977), 99-118.

[9] Rosenfeld, A., Hummel, R. and Zucker, S., *"Scene labelling by relaxation operations"*, *IEEE Trans. Syst. Man Cybern*, SMC-6 (1976), 420-433.

[10] Kasif, S., *"On the Parallel Complexity of Some Constraint Satisfaction Problems"*, *Proc. of AAAI-86*, (1986), 349-353.

[11] Garey, M.R. and Johnson, D.S. *"Computers and Intractability: A guide to NP-Completeness."*, *Freeman and Company*, San Francisco, (1979).

[12] Haken, A., *"The Intractability of Resolution."*, *PhD Thesis, University of Illinois Urbana-Champaign*, (1984).

[13] Urquhart, A., *"Hard Examples of Resolution"*, *Journal of ACM*, ?,1 (January 1987), 209-219.

[14] Mackworth, A. and freuder, E., *"The Complexity of Some Polynomial Network Consistency Algorithms for Constraint Satisfaction"*, *Artificial Intelligence*, 25 (1985) 65-74.

[15] Kirousis, L.M. and Papadimitriou, C.H., *"The Complexity of Recognizing Polyhedral Scenes"*, *Proc. 26th Annual IEEE Symposium on Foundations of Computer Science*, (1985), 175-185.

[16] Pan, V. and Reif J., *"Fast and Efficient Solutions of Linera Systems"*, *Proc. 17th Annual ACM Symposium on Theory of Computing*, (1985), 143-152.

[17] Fortune, S. and Wyllie, J., *"Parallelism in Random Access Machines"*, *Proceedings of the tenthAnnual ACM Symposium on Theory of Computing*, San Diego, CA, (1978), 114 - 118.

[18] Lichtenstein, D., *"Planar Formulae and their Uses"*, *SIAM Journal on Computing*, 11, 2 (1982), 329-343.

[19] Lipton, R.J and Tarjan, R.E., *"Applications of a Planar Separator Theorem"*, *SIAM Journal on Computing*, 9, (3) (Aug 1980), 615-627.

[20] Ullman, J.D., *"Computational Aspects of VLSI"*, *Computer Science Press* Rockville, MD, (1984).

[21] Harary, F., *"Graph Theory"*, *Addison Wesley*, (1969).

[22] Lipton, R.J and Tarjan, R.E., *"A Separator Theorem for Planar Graphs"*, *SIAM Journal on Applied Mathematics*, 36, (1979), 177-189.

[23] Djidjev, H.N., *"On the Problem of Partitioning Planar Graphs"*, *SIAM Journal of Algebraic and Discrete Methods*, 3, 2 (June 1982), 229-240.

[24] Chang C.L. and Lee R.C.T., "Symbolic Logic and Mechanical Theorem Proving," *Academic Press, New York*, 1979.

[25] Ja'Ja', J. and Simon J., *"Parallel Algorithms in Graph Theory: Planarity Testing"*, *SIAM Journal on Computing*, 11, 2 (May 1982), 314-328.

[26] Karp R.M. and Widgerson, A., *"A Fast Parallel Algorithm for the Maximal Independent Set Problem"*, *Proc. 16th Annual ACM Symposium on the Theory of Computing*, (1984), 266-272.

[27] Luby, M., *"A Simple Parallel Algorithm for the Maximal Independent Set Problem"*, *SIAM Journal on Computing*, 15 (4), (1986) 1036 - 1055.

[28] Davis M. and Putnam H., *"A Computing Procedure for Quantification Theory"*, *Journal of ACM*, 7, 3 (1960), 201-215.

[29] Chang C.L. and Slagle J.R., *Using Rewiring Rules for Connection Graphs to Prove Theorems*, *Artificial intelligence*, 12, 2 (August 1979).

[30] Kowalski, R., *"A Proof Procedure Using Connection Graphs"*, *J. ACM*, 22 (1975), 572-595.

[31] McKay, D.P. and Shapiro, S.C., *"Using Active Connection Graphs for Reasoning with Recursive Rules"*, *IJCAI-81*, (August 1981), 368-374.

[32] Seidel, R., *"A New Method for Solving Constraint satisfaction Problems"*, *Proceedings of the international Joint Conference on AI*, (1981), 338-342.

[33] Leiserson, C.E., *"Fat-Trees: Universal Networks for Hardware-Efficient Supercomputing"*, *IEEE Transactions on Computers*, C-84, 10 (Oct 1985), 892-901.

[34] Sherlekar, D.D., *"Graph Separator-Based Dissection methods in VLSI and Algorithms"*, *Doctoral Dissertation, University of Maryland, College park*, (1987).

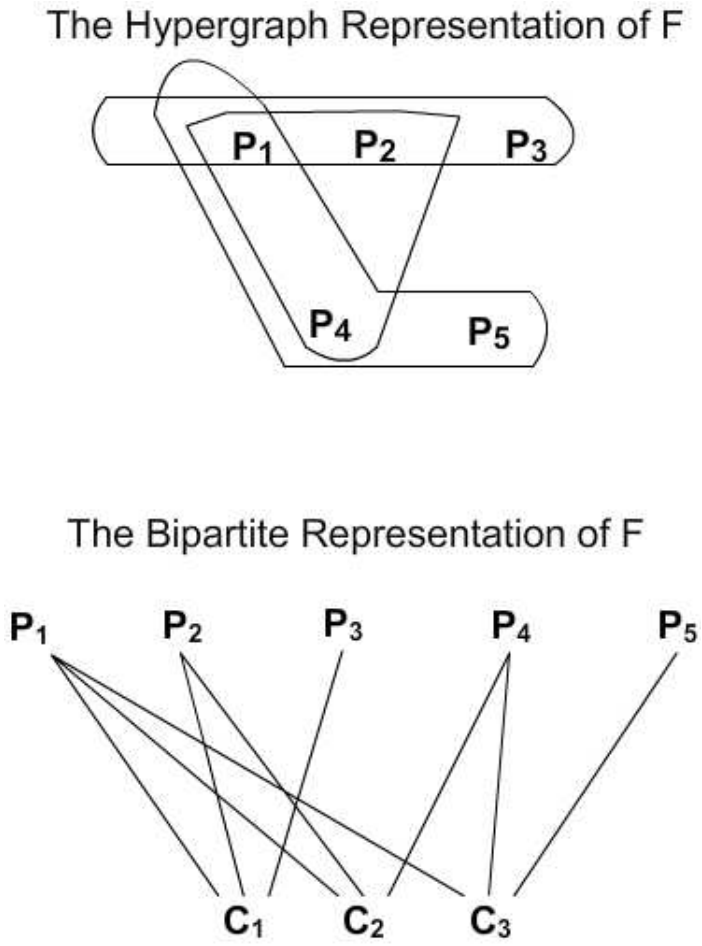

Figure 1: Graph Representations of a Propositional Formula.
$F = (P_1, P_2, P_3), (P_1, \bar{P}_2, P_4), (\bar{P}_1, P_4, \bar{P}_5)$

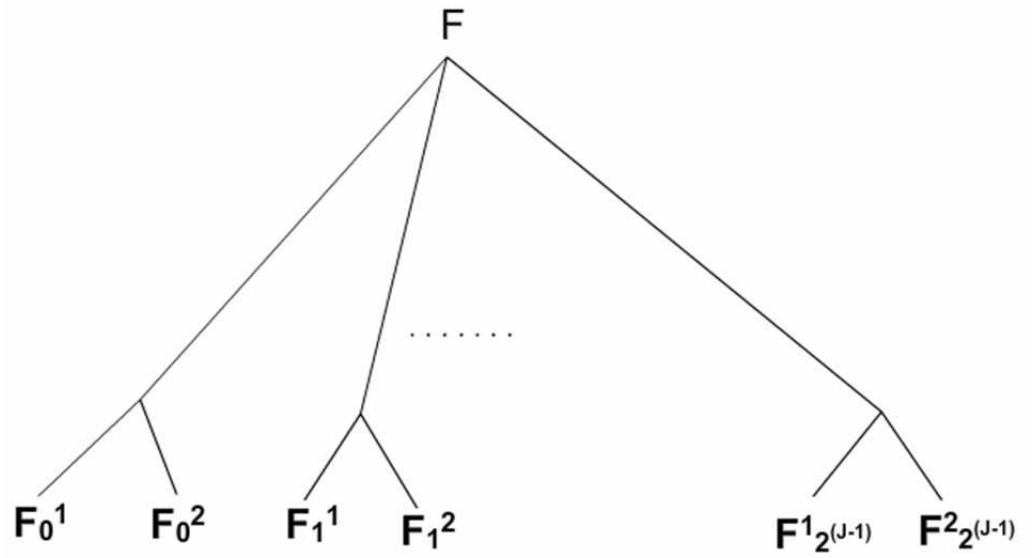

Figure 2: The AND/OR Tree Generated by one Iteration of Algorithm FD.