Identifying the Shortest Path in Large Networks using Boolean Satisfiability

Fadi A. Aloul
Computer Engineering Department
American University of Sharjah
faloul@aus.edu

Bashar Al Rawi
Computer Engineering Department
American University in Dubai
Bashar.alrawi@mymail.aud.edu

Mokhtar Aboelaze
Dept. of Computer Science & Eng.
York University
aboelaze@cs.yorku.ca

Abstract — **Today, most routing problems are solved using Dijkstra's shortest path algorithm. Many efficient implementations of Dijkstra's algorithm exist and can handle large networks in short runtimes. Despite these advances, it is difficult to incorporate user-specific conditions on the solution when using Dijkstra's algorithm. Such conditions can include forcing the path to go through a specific node, forcing the path to avoid a specific node, using any combination of inclusion/exclusion of nodes in the path, etc. In this paper, we propose a new approach to solving the shortest path problem using advanced Boolean satisfiability (SAT) techniques. SAT has been heavily researched in the last few years. Significant advances have been proposed and has lead to the development of powerful SAT solvers that can handle very large problems. SAT solvers use intelligent search algorithms that can traverse the search space and efficiently prune parts that contain no solutions. These solvers have recently been used to solve many problems in Engineering and Computer Science. In this paper, we show how to formulate the shortest path problem as a SAT problem. Our approach is verified on various network topologies. The results are promising and indicate that using the proposed approach can improve on previous techniques.**

I. Introduction

The Internet has started in the late 1970, ever since its inception, the traffic, the nodes, and the users are growing at an unprecedented pace. Today the Internet handles completely different types of traffic compared to the 1970's and 1980's Internet. Link bandwidth and Internet traffic are continuously increasing. Routing protocols are constantly being proposed and improved in order to handle the constant changes in the traffic, link bandwidth, and the required quality of service in today's Internet.

Internet routing is dominated by link state routing protocols such as the OSPF [21]. In this protocol, routers exchange link state information with neighboring routers, then they calculate the shortest path tree by using shortest path algorithms such as Dijkstra's shortest path algorithm [15]. This algorithm is one of the most widely used, and one of the earliest algorithms for Internet routing. Fault Torrance link state protocol was introduced in [29] without using flooding to send changes to link states. Scalability has been considered to be a major problem especially in the Internet Exterior Routing Protocol [6].

A compact routing scheme for internet like graphs was introduced in [23]. Their protocol has a near optimal memory requirements per node. Modifications to the shortest path algorithm were introduced in [25] and are known as the widest-shortest path and shortest-widest path algorithms. In the widest-shortest path algorithm, the algorithm computes the shortest path(s), if there is more than one, the path with the maximum reservable bandwidth is chosen. In the shortest-widest path, the widest path(s) are calculated, if there is more than one, the one with the minimum path is chosen.

A minimum interference algorithm was introduced in [22]. this algorithm finds a path that minimizes the expected future interference between the traffic requesting the path and the existing traffic. An alternate path routing algorithm was proposed in [33]. In this algorithm, several routes are identified and the algorithm finds the links that are responsible for congestion. The traffic is routed through the paths that have the least number of such links.

The shortest path algorithm is very effective in finding the shortest path between two nodes. However more factors could be taken into consideration in routing than minimizing the source/destination path. Some of these factors include: (1) optimizing the use of the links bandwidth, (2) distributing the traffic all over the network, (3) minimizing the number of hops, (4) avoiding specific nodes or any combination of nodes, and (5) redirecting existing traffic to satisfy the above mentioned criterias.

The routing problem using the shortest path algorithm has very efficient algorithms. However, when we generalize the problem and include many constraints, it becomes an NP-complete problem [33]. The objective of this paper is to formulate the routing problem as a Boolean satisfiability (SAT) instance and explore the possibility of using advanced SAT techniques to solve the routing problem.

Recently, SAT have been shown to be very successful in solving complex problems in various Engineering and Computer Science applications. Such applications include: Formal Verification [7], FPGA routing [28], Power Optimization [4], etc. SAT has also been extended to a variety of applications in Artificial Intelligence including other well known NP-complete problems such as graph colorability, vertex cover, hamiltonian path, and independent sets [12]. Despite SAT being an NP-Complete problem [11], many researchers have developed powerful SAT solvers that are able of handling problems consisting of thousands of variables and millions of constraints. Briefly defined, the SAT problem consists of a set of Boolean variables and a set of constraints expressed in product-of-sum form. The goal is to identify an assignment to the variables that would satisfy all constraints or prove that no such assignment exists.

In this paper, we present a SAT-based approach to solving the routing problem in computer networks. We show how to formulate the problem as a SAT instance. We report results using randomly-generated network topologies. Initial results indicate the effectivity of the proposed approach. The proposed approach is *complete* and is guaranteed to identify the shortest path. The approach also allows user-specific conditions to be easily added to the problem, which was not as easy to add in previous approaches.

This paper is organized as follows. Section II provides a general overview of SAT. Section III shows how to formulate the routing problem as a SAT instance. Experimental results are presented and discussed in Section IV. Finally, the paper is concluded in Section V.

II. Boolean Satisfiability

The last few years have seen significant advances in Boolean satisfiability (SAT) solving. These advances have lead to the successful deployment of SAT solvers in a wide range of problems in Engineering and Computer Science. Given a set of Boolean variables and a set of constraints expressed in product-of-sum form, the goal is to find a variable assignment that satisfies all constraints or prove that no such assignment exists. The term "Satisfiability" emerges from that fact that we are asked to find a satisfying assignment, while the term "Boolean" comes from the fact that such assignment consists of only *true* or *false* variable states.

The SAT problem is usually expressed in conjunctive normal form (CNF). A CNF formula φ on n binary variables $x_1, \ldots, x_n$ is the conjunction (AND) of m clauses $\omega_1, \ldots, \omega_m$ each of which is a disjunction (OR) of one or more literals, where a literal is the occurrence of a variable or its complement. A formula φ denotes a unique n-variable Boolean function $f(x_1, \ldots, x_n)$ and each of its clauses corresponds to an implicate of f [20]. Clearly, a function f can be represented by many equivalent CNF formulas. We will refer to a CNF formula as a *clause database* and use "formula" and "CNF formula" interchangeably. Finally, a variable x is *monoform* if it is possible to write a CNF formula for the function f in which all literals on x are either exclusively x or $\bar{x}$.

A variable x is said to be *assigned* when its logical value is set to 0 or 1 and *unassigned* otherwise. A literal l is a *true* (*false*) literal if it evaluates to 1 (0) under the current assignment to its associated variable, and a *free literal* if its associated variable is *unassigned*. A clause is said to be *satisfied* if at least one of its literals is true, *unsatisfied* if all of its literals are set to false, *unit* if all but a single literal are set to false, and *unresolved* otherwise. A formula is said to be satisfied if all its clauses are satisfied, and unsatisfied if at least one of its clauses is unsatisfied. In general, the SAT problem is defined as follows: Given a Boolean formula in CNF, find an assignment of variables that satisfies the formula or prove that no such assignment exists.

In the following example, the CNF formula:

$$\varphi = (a \vee b)(\bar{b} \vee c)(\bar{a} \vee c) \tag{1}$$

consists of 3 variables, 3 clauses, and 6 literals. The assignment $\{a = 1, b = 0, c = 0\}$ violates the third clause and unsatisfies φ, whereas the assignment $\{a = 1, b = 0, c = 1\}$ satisfies φ. Note that a problem with n variables will have 2^n possible assignments to test. The above example with 3 variables has 8 possible assignments.

Despite the SAT problem being NP-Complete [11], there have been dramatic improvements in SAT solver technology over the past decade. This has lead to the development of several powerful SAT algorithms that are capable of solving problems consisting of thousands of variables and millions of constraints. Such solvers include: GRASP [26], zChaff [27], and Berkmin [18]. In the next three sections, we describe the basic SAT search algorithm, recent extensions to the SAT solver input, and the use of hardware with SAT.

A. Backtrack Search

Most modern complete SAT algorithms can be classified as enhancements to the basic Davis-Logemann-Loveland (DLL) backtrack search approach [14]. The DLL procedure performs a search process that traverses the space of 2^n variable assignments until a satisfying assignment is found (the formula is satisfiable), or all combinations have been exhausted (the formula is unsatisfiable). It maintains a *decision tree* to keep track of variable assignments and can be viewed as consisting of three main engines: (1) *Decision* engine that makes *elective* assignments to the variables, (2) *Deduction* engine that determines the consequences of these assignments, typically yielding additional *forced* assignments to, i.e. implications of, other variables, and (3) *Diagnosis* engine that handles the occurrence of conflicts, i.e. assignments that cause the formula to become unsatisfiable, and backtracks appropriately.

Recent studies have proposed the use of the *conflict analysis* procedure in the diagnosis engine [26]. The idea is whenever a conflict is detected, the procedure analyzes the variable assignments that cause one or more clauses to become unsatisfied. Such analysis can identify a small subset of variables whose current assignments can be blamed for the conflict. These assignments are turned into a *conflict-induced clause* and augmented with the clause database to avoid regenerating the same conflict in future parts of the search process. In essence, the procedure performs a form of learning from the encountered conflicts. Today, conflict analysis is implemented in almost all SAT solvers [18, 26, 27].

B. More Expressive Input

Restricting the input of SAT solvers to CNF formulas can restrict their usage in various domains. Therefore, researchers have focused on extending SAT solvers to handle stronger input representations. Specifically, SAT solvers [3, 10, 16, 30] have recently been extended to handle pseudo-Boolean (PB) constraints which are linear inequalities with integer coefficients that can be expressed in the normalized form [3] of:

$$a_1x_1 + a_2x_2 + \ldots + a_nx_n \geq b \tag{2}$$

where $a_i, b \in Z^+$ and x_i are literals of Boolean variables. Note that any CNF clause can be viewed as a PB constraint, e.g. clause $(a \vee b \vee c)$ is equivalent to $(a + b + c \geq 1)$.

PB constraints can, in some cases, replace an exponential number of CNF constraints. They have been found to be very efficient in expressing "counting constraints" [3]. Furthermore, PB extends SAT solvers to handle *optimization* problems as opposed to only *decision* problems. Subject to a given set of CNF and PB constraints, one can request the minimization (or maximization) of an objective function which consists of a linear combination of the problem's variables.

$$\sum_{i=1}^{n} a_i x_i \tag{3}$$

This feature has introduced many new applications to the SAT domain. Recent studies has also shown that SAT-based optimization solvers can in fact compete with the best generic integer linear programming (ILP) solvers [3, 10].

C. Hardware-Based SAT Solvers

Note that SAT solvers can be implemented in hardware. Several studies proposed the use of FPGA reconfigurable systems to solve SAT problems [1, 36]. Hardware solvers could be a standalone or as an accelerator where the problem is partitioned between the hardware solver and the attached computer using software. Many different architecture were proposed to solve SAT problems in hardware. Linearly connected set of finite state machines, control unit, and deduction logic was proposed in [36]. The authors in [36] implemented their algorithm on Xilinx XC4028 FPGA. While in [1], the authors proposed a technique for modeling any boolean expression. Their objective is to set the function output to 1. A backtrack algorithm is used to propagate the output back to the input and finding an assignment of the inputs to satisfy a logical 1 at the output.

The authors in [13] proposed an architecture for evaluating clauses in parallel. In their architecture, the clauses are separated into a number of groups and the deduction is performed in parallel. Then the results are merged together to allow the assignment to the variables.

A software/hardware solver for SAT was introduced in [31]. In their approach, they minimized the hardware compilation time which greatly reduces the total time to solve the problem. They also implemented their solver on FPGA.

III. Problem Formulation

In this paper we are interested in using advanced SAT solvers to identify the shortest path between two nodes in a network grid. To illustrate our approach, lets consider the network in Figure 1. In the figure, each node is labeled by an upper-case letter, and each link is marked by (x,n) where x is the name of the link and n is a positive integer that represents the weight, i.e. cost, of the link. Node *I* and *H* are the *source* and

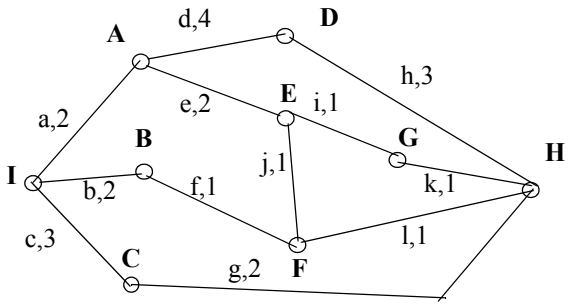

Fig. 1. An example of a network with 9 nodes and 12 edges. Upper-case letters represent nodes. Lower-case letters represent edges. Each edge is associated with an integer representing its weight

destination nodes, respectively. The objective is to find a path from *I* to *H* that minimizes the total path cost (sum of weights of all links in the path).

Two sets of *variables* are defined for the problem:

- A Boolean variable is defined for each node. A value of 1 (0) for each variable indicates that the corresponding node is (is not) included in the optimal path from the source node to the destination node.
- A Boolean variable is also defined for each link. Again, a value of 1 (0) for each variable indicates that the corresponding edge is (is not) included in the optimal path from the source node to the destination node.

In the above example, 21 variables are declared, 9 of which represent the nodes (*A, B, C,..., I*) and 12 represent the edges (*a, b, c,..., l*). The following set of *constraints* are generated:

- For both source and destination nodes, only one of the neighboring edges will be part of the path. This can be expressed using two PB constraints in the above example:

$$a + b + c = 1 \text{ and } h + k + l + g = 1 \quad (4)$$

- All other nodes (except the source and destination nodes) will either be (i) part of the path or (ii) not part of the path. In the first case, *exactly* two edges connected to that node will be part of the path. In the second case, none of the edges connected to the node will be part of the path. This can be expressed using a PB constraint. In the above example the PB expression for node *A* is as follows:

$$2\overline{A} + a + d + e = 2 \quad (5)$$

If node *A* within the path, that means *A* is true. Hence $\overline{A} = 0$ and the expression reduces to $a + d + e = 2$. The only way to satisfy the expression is to set two edges to true, i.e. making them part of the path. If node *A* is not within the path, then *A* is set to *0*. Hence $\overline{A} = 1$ and the expression reduces to $a + d + e = 0$. The only way to satisfy this expression is to set all three edges to 0, i.e. none of the edges are part of the path. Similar PB constraints are generated for the other nodes as follows:

$$2\overline{B} + b + f = 2 \; ; \ldots ; \; 2\overline{G} + i + k = 2 \quad (6)$$

The above two sets of constraints guarantee that a complete path will be formulated from the source node to the destination node. To minimize the total cost of the path, a PB objective function consisting of all edge variables is created as follows:

$$min(2a + 2b + 3c + 4d + f + 2g + 3h + i + j + k + l) \quad (7)$$

In general, the minimization could be represented as

$$minimize\left(\sum_{\forall i} weight_i \times var_i\right) \quad (8)$$

where $weight_i$ and var_i represent the cost and variable of edge *i.*

By formulating the problem as such, we can do more than finding the minimum cost path. We can incorporate any restrictions that we can think of in the resulting path. For example, by adding the PB constraint $A = 1$, we are forcing node *A* to be part of the minimal cost path. Similarly, we can exclude node *A* from the solution by adding the PB constraint $A = 0$.

We can also add dependencies between nodes. For example, we can force one of two nodes, e.g. *J* and *B*, to exist in the resulting path. This can be expressed by adding the following two CNF constraints:

$$(J \vee B)(\overline{B} \vee \overline{J}) . \quad (9)$$

We can also force certain nodes to be in the path only if a specific node is. For example, we can force nodes *B, C,* and *D* to be part of the solution if and only if node *A* is. This is expressed using the following set of CNF constraints:

$$(\overline{A} \vee B)(\overline{A} \vee C)(\overline{A} \vee D)(A \vee \overline{B})(A \vee \overline{C})(A \vee \overline{D}) \quad (10)$$

Note that the complexity of converting the graph into a SAT problem is $O(v + e + k)$, where *v* is the number of nodes, *e* is the number of edges, and *k* is the number of graph restrictions (e.g. (9)).

IV. Experimental Results

In this section, we evaluate the use of SAT solvers in identifying the shortest path in a network. The routing problem was encoded as a SAT instance as shown in Section III. Topology generation has been an active area of research [2, 9, 17]. Therefore, we decided to use the BRITE topology generator [8] to produce different random topologies to test our approach. BRITE can produce multiple generation models and can assign links attributes such as bandwidth and delay.

We created networks of different sizes with the number of nodes ranging from 20-500 and number of links per node ranging from 2-5. Nodes are placed randomly in a plane with a side of 1000 units. We considered the weight of the link as the Euclidean length of the link (we can choose any weight factor but choose the distance since it is already generated by the topology simulator). The topology model is Waxman model [34] with parameters $\alpha = 0.15, \beta = 0.2$.

The network is stored in a text file and passed to a PERL script that converts it into a SAT-encoded problem. The SAT problem is then solved by advanced SAT solvers. For our experiments, we used the PBS solver [3, 5]. PBS is a new solver than can handle both CNF and PB constraints and can solve decision and optimization problems. It implements the latest enhancements in the SAT domain and can solve optimization problems using a linear-based or binary-based search scheme. Both schemes have shown competitive performance on various optimization instances that consists of CNF-only or CNF/PB constraints. The experiments were conducted on a Pentium Xeon 3.2 Ghz machine, equipped with 4 GBytes of RAM, and running Linux. The runtime limit was set to 1000 seconds.

Table 1 lists the runtime results for the routing benchmarks. The table lists the name of the instance, the runtime in seconds of PBS using a linear-based and binary-based search, and the size of the shortest path. The name of the instance of the form *X_Y_u_A_B* indicates that the instance has *X* nodes and the number of links per node is *Y*. For each grid two random nodes *A* and *B* are selected as the source and destination nodes. A "*" in the PBS shortest-path value column indicates that PBS didn't complete the search process because it exceeded either the allowed runtime or memory limits. In such a case, the size of the shortest path detected so far is shown. Several observations are in order:

TABLE 1. Experimental results on various size network grids using the PBS4 SAT Solver. (S. Path = Shortest Path)

Instance Name	PBS4			
	Binary Search		Linear Search	
	Time	S. Path	Time	S. Path
20_2_u__17_12	0	700	0	700
20_3_u_10_9	0	112	0	112
20_5_u_3_0	0	189	0.03	189
50_2_u__8_45	0	309	0.03	309
50_3_u__15_13	0.01	123	0.09	123
50_5_u__22_20	0.28	821	9.13	821
100_2_u__98_56	0.07	1145	0.89	1145
100_3_u__3_91	0.06	590	1.42	590
100_5_u__50_25	0.03	400	0.31	400
500_2_u__103_309	2.69	1521	25.38	1521
500_2_u__248_483	0.8	984	7.63	984
500_2_u__254_391	0.48	909	2.65	909
500_2_u__345_119	2.63	1873	>1000	6102*
500_3_u__164_158	3.12	863	>1000	4740*
500_3_u__197_189	0.54	761	265.19	761
500_3_u__212_366	47.3	782	347.94	782
500_3_u__307_177	225	1361	>1000	8543*
500_5_u__11_453	29	658	>1000	7504*
500_5_u__160_453	3.62	524	>1000	4903*
500_5_u__311_49	148	685	>1000	5596*
500_5_u__369_466	254	580	>1000	3908*

- PBS was able to identify the shortest path in *all* reported cases using the binary-base search scheme.
- The linear search scheme is not as competitive as the binary search scheme for these instances.
- The larger the network grid, the longer is the search runtime.
- The approach is complete and is guaranteed to find the shortest path given enough time and memory resources.

V. CONCLUSION

In this paper, we presented a new approach to detecting the shortest path between two nodes in large networks using advanced Boolean satisfiability (SAT) solvers. We show how to formulate the shortest path problem as a SAT instance. The approach was tested on a number of networks of various sizes and showed promising results. The presented approach is *complete* and will find the shortest possible path. One of the advantages of the new approach is the ability to add user-specific constraints that can restrict the existence of certain nodes and edges in the resulting path.

REFERENCES

[1] M. Abramovici, and D. Saab "Satisfiability on Reconfigurable Hardware," in *Proc. of the Int'l Workshop on Field Programmable Logic and Application*, 448-456, 1997.

[2] W. Aiello, F. Chung, and L. Lu "A Random Graph Model for Massive Graphs," in *Proc. of Symposium on Theory of Computing*, 171-180, 2000.

[3] F. Aloul, A. Ramani, I. Markov, and K. Sakallah, "Generic ILP Versus Specialized 0-1 ILP: An Update," in *Proc. of* ICCAD, 450-457, 2002.

[4] F. Aloul, S. Hassoun, K. Sakallah, and D. Blaauw, "Robust SAT-Based Search Algorithm for Leakage Power Reduction," in *Proc. of* PATMOS, 167-177, 2002.

[5] B. Al-Rawi and F. Aloul, PBS4 SAT Solver, 2005. Available at: *http://www.eecs.umich.edu/~faloul/Tools/pbs4*

[6] T. Bass, "Internet exterior routing protocol development, problems, issues, and misconception," in *IEEE Networks*, 11(4), 50-55, 1997.

[7] A. Biere, A. Cimatti, E. Clarke, M. Fujita, and Y. Zhu, "Symbolic Model Checking using SAT procedures instead of BDDs," in *Proc. of* DAC, 317-320, 1999.

[8] BRITE Topology Generator, 2005. Available at *http://www.cs.bu.edu/brite*

[9] K. Calvert, M. Doar, and E. Zegura "Modeling Internet Topology," in *IEEE Trans. on Communications,* 35(6), 160-163, 1997.

[10] D. Chai and A. Kuehlmann, "A Fast Pseudo-Boolean Constraint Solver", in *Proc. of* DAC, 830-835, 2003.

[11] S. A. Cook, "The Complexity of Theorem Proving Procedures," in *Proc. of the ACM Symposium on the Theory of Computing*, 151-158, 2004.

[12] N. Creignou, S. Kanna, and M. Sudan, "Complexity Classifications of Boolean Constraint Satisfaction Problems", *SIAM Press*, 2001.

[13] A. Dandalis, and V. K. Prasanna, "Run-time Performance Optimization of an FPGA Based Deduction Engine for SAT Solvers," in *ACM Trans. on Design Automation of Electronic Systems*, 7(4), 547-562, October 2002.

[14] M. Davis, G. Longman, and D. Loveland "A Machine Program for Theorem Proving," in *Journal of the ACM*, 5(7), 394-397, 1962.

[15] E. W. Dijkstra, "A Note on Two Problems in Conexion with Graphs," in *Numerische Math,* vol. 1, 269-271, 1959.

[16] H. Dixon and M. Ginsberg, "Inference Methods for a Pseudo-Boolean Satisfiability Solver," in *Proc. of* AAAI, 635-640, 2002.

[17] M. Doar "A Better Model for Generating Test Networks," in *Proc. of IEEE Globecom*, 86-93, November 1996.

[18] E. Goldberg and Y. Novikov, "BerkMin: A Fast and Robust SAT-solver," in *Proc. of* DATE, 142-149, 2002.

[19] E. Goldberg, M. Prasad, and R. Brayton. "Using SAT for Combinatorial Equivalence Checking," in *Proc. of* DATE*,* 114-112, 2001.

[20] J. Hayes, "Introduction to Digital Logic Design," *Addison-Wesley*, 1993.

[21] C. Huitema, "*Routing in the Internet Second Edition*," Prentice Hall, 2000.

[22] M. Kodialam, and T. V. Lakshman "Minimium interference routing with applications to MPLS traffic engineering" in *Proc. of* INFOCOM*,* Vol. 2, 884-893, 2000.

[23] D. Krioukov, K. Fall, and X. Yang, "Compact Routing on Internet-Like Graphs," in *Proc. of* INFOCOM*,* 2004.

[24] T. Larrabee "Test Pattern Generation Using Boolean Satisfiability," in *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*. 11(1), 4-15, January 1992.

[25] Q. Ma and P. Steenkiste, "On Path Selection for Traffic with Bandwidth Guarantees," in *5th IEEE Int'l Conference on Network Protocols,* 1997.

[26] J. Marques-Silva and K. Sakallah "GRASP: A search algorithm for propositional satisfiability," in *IEEE Trans. on Computers*, 48(5), 506-521, 1999.

[27] K. Moskewicz, C. Madigan, Y. Zho, and S. malik "Chaff: Engineering an efficient SAT solver," in *Proc. of* DAC*,* 503-535, 2001.

[28] G. Nam, F. A. Aloul, K. A. Sakallah, and R. Rutenbar, "A Comparative Study of Two Boolean Formulations of FPGA Detailed Routing Constraints," in *Proc. of* ISPD, 222-227, 2001.

[29] P. Narvaez, K.-Y. Siu, and H.-Y. Tzeng, "Fault-Tolerant Routing in the Internet Without Flooding," in *Dependable Network Computing*, Kluwer Academics, 193-206, 2000.

[30] H. Sheini and K. Sakallah, "Pueblo: A Modern Pseudo-Boolean SAT Solver," in *Proc. of* DATE, vol. 2, 684-685, 2005.

[31] I. Skliarova and A. Ferrari "A Software/Reconfigurable Hardware SAT Solver," in *IEEE Trans. on Very Large Scale Systems*, 12(4), 408-419, April 2004.

[32] A. Smith, A. Veneris, M. Ali, and A. Viglas. "Fault Diagnosis and Logic Debugging Using Boolean Satisfiability," in *IEEE Trans. on Computer Aided Design of Integrated Circ. and Systems*. 24(10), 1606-1621, 2005.

[33] S. Subranmanian and V. Muthukumar, "Alternate Path Routing Algorithm for Traffic Engineering in the Internet," in *Proc. of* ITCC, 367-372, 2003.

[34] B. Waxman, "Routing of Multipoint Connections," in *IEEE Journal of Selected Areas in Communications,* 6(9), 1617-1622, December 1988.

[35] R. Wood, and R. Rutenbar "FPGA Routing and Routability Estimation via Boolean Satisfiability," in *IEEE Trans. on Very Large Scale Integration Systems,* 6(1), 222-231, June 1998.

[36] P. Zhong, M. Martonosi, P. Ashar, and S. Malik "Using Reconfigurable Computing to Accelerate Boolean Satisfiability," in *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, 18(6), 861-868, 1999.