Hindawi
Mathematical Problems in Engineering
Volume 2022, Article ID 2677457, 9 pages
https://doi.org/10.1155/2022/2677457

Research Article

A Topology Graph Algorithm Based on Lattice-Valued Logic to Solve Satisfiability Problems

Jieqing Tan[1] and Yingjie Li[2]

[1]*School of Computer and Information, Hefei University of Technology, Hefei, Anhui 230071, China*
[2]*Faculty of Data Science, City University of Macau, Macau 99078, China*

Correspondence should be addressed to Yingjie Li; liyingjie0928@163.com

Received 29 November 2021; Accepted 31 December 2021; Published 27 January 2022

Academic Editor: Giuseppe D'Aniello

Automatic reasoning is a very challenging research area in the field of artificial intelligence. Automatic inference based on lattice-valued logic is an important basic research content in the field of intelligent information processing. It is of great academic significance to deal with incomparability, linguistic valued information, and formal verification of the correctness of corresponding software and hardware systems in various applications. Based on the extensive work of α-resolution automated reasoning in linguistic truth-valued lattice-valued logic, this paper discusses the structure and its α-resolvability of the generalized literal. For further improving the efficiency, a topology graph deduction is proposed as well as its soundness, completeness, and universal algorithm. The use of distributed computing and resource sharing is the core idea of this deduction. In the decision process, as long as one satisfiability solution is found by subtask, the decision result can be given directly, shortening the decision time. The algorithm reduces the time complexity from factorial level to square.

1. Introduction

Satisfiability problem (SAT) is the cornerstone of computational complexity theory and a fundamental problem in many areas as the first NPC problem [1]. It plays a prominent role in many domains of computer science and artificial intelligence due to its significant importance in both theory and applications [2]. The SAT problem is fundamental in solving many practical problems in combination optimization [3], statistical physics [4], circuit verification [5], and computing theory [6], and SAT algorithms have been widely used to solve real-world applications, such as computer algebra systems [7], core graphs [8], real-time scheduling [9], gene regulatory networks [10], automated verification [11], model-based diagnosis (MBD) [12], and machine induction [13]. Therefore, it is of great significance to study SAT problem and improve its solving speed.

At present, most mainstream SAT solvers are based on Davis–Putnam–Logemann–Loveland (DPLL) algorithm framework [14]. In the past two decades, many heuristic algorithms [15] and program implementation technologies [16] based on DPLL had been proposed by scholars. The most popular solving framework of conflict-driven clause learning (CDCL) [17] had been formed. And CDCL-based solvers [18] had also developed by leaps and bounds. In the framework of DPLL algorithm, the size of search space is directly determined by the choice of decision arguments. For decades, many heuristics branching decision algorithms have been proposed by scholars. These algorithms manly include Jeroslow–Wang (JW) [19], Bohn [20], maximum occurrences in minimization (MOM) [21], dynamic literal individual sum (DLIS) [22], variable state independent decaying Sum (VSIDS) [15], learning rate elasticity (LRB) [23], and some improved algorithms. The core idea of these algorithms is to add an evaluation operator to each decision text. Then, the maximum or minimum score of the text is selected for the new branch direction each time. After each conflict, a large amount of CPU and memory resources is consumed to recalculate the score of the affected argument in these algorithms.

The Davis–Putnam–Logemann–Loveland (DPLL) algorithm [14] was proposed in 1962, which primarily employed the unit propagation rule, pure-literal rule, and

split rule to search the solution space by a depth-first search. However, due to the particularity of the SAT problem, the DPLL algorithm has an exponential time complexity in the worst case. The international SAT competitions organized by research institutions around the world were first held in 1996, and the 22nd competition will be held in Lisbon. The SAT competitions have greatly boosted the solving efficiency of SAT solvers and have also promoted the wider application of SAT problems in practice. At present, most of the state-of-the-art SAT complete algorithms are derived from DPLL algorithm, the most important of which is the conflict-driven clause learning (CDCL) algorithm [24], an extension of the DPLL algorithm, by involved effective search techniques, such as the learnt clause scheme, heuristic decision strategy, restart techniques, and lazy data structures. In 1996, the GRASP solver [24], which is the prototype of the CDCL algorithm, introduced nonsynchronous backtracking and a learnt clause scheme and reduced the searching space to a large extent. In 1997, the head/tail list data structure was employed in the SATO solver [25] and vastly enhanced the efficiency of Boolean constraint propagation (BCP). In 2001, the Chaff solver [15] was integrated into the watched-literal data structure and the low-cost variable state independent decaying sum (VSIDS) decision strategy. In 2002, the BerkMin solver [26] added a learned clause database deletion strategy to avoid memory explosion because of an increasing number of learnt clauses. In 2004, the Zchaff solver [27] improved the implementation of programming based on the Chaff solver, further increasing the solution efficiency. In 2005, the Minisat solver [28], with a strong capability of the search space by optimizing the code structure, and a majority of solvers that entered the SAT competition later were based on Minisat. In 2009, the author of the Glucose solver [29] proposed a new learnt clause reduction method based on the literals blocks distance (LBD), and a dynamic restart strategy, the Glucose solver, got the first prize of the application group in the 2011 SAT competition. The Lingeling solver [30], which was also based on the CDCL algorithm, respectively, won the 2013 and 2014 SAT competition about the application group. In 2016, Liang et al. designed the MapleCOMSPS solver [23, 31], which utilized a new decision branching method—learning rate branching (LRB)—and got the first prize of the main-track group in the 2016 SAT competition and the second prize of the main-track group in 2017, respectively. In 2017, the solver Maple_LCM_Dist [32] got the first prize of the main-track group in the 2017 SAT competition, which used a learnt clause minimization approach, and lots of competing solvers were based on it in the 2018 SAT competition [33].

Now logic is widely used in computer science in logic circuit design, program design analysis, security protocol verification, and artificial intelligence. Logic systems not only provide language tools for knowledge representation, but also provide mechanized algorithms for knowledge reasoning. The research results in this area are the theoretical basis for realizing the automation of calculation and reasoning. In addition, the rapid development of computer science, especially artificial intelligence, has provided a broad background and realistic demand for the theoretical research and practical application of logic. Lattice-valued logic is a very important nonclassical logic, which can describe not only the information of complete order, but also the uncertain information of noncomplete order (i.e., noncomparability).

On the basis of lattice logic deduction, topology deduction algorithm is proposed in this paper. Conflict analysis, backtracking, restart mechanism, and adding evaluation operator for each decision text are no longer used in this algorithm. According to the deduction principle of lattice-valued logic, the single-layer branch topology is generated by each subtask in this algorithm. The results of each subtask are summarized by the main task. And a shared library file is generated by the main task. The topology view is generated by each topology subtask based on the shared library file. As long as the satisfying solution is obtained by any topological subtask, the whole mission will be stopped. This method greatly improves the solving efficiency. It reduces time complexity from factorial level to quadratic level.

2. Methods

2.1. Preliminaries. Rules of machine learning refer to narrowly define rules. It usually refers to semantically clear and can describe the objective laws or domain concepts implied by data distribution. It can be written as "IF...THEN..." logical rules of form. Logical rule learning is to learn a set of rules from the training data that can be used to distinguish unknown data. Compared with the "black box model" such as neural network and support vector machine, rule learning has better interpretability. In addition, mathematical logic has a strong ability of expression and the vast majority of human knowledge can be described and expressed concisely through mathematical logic. The ability to abstract the description of logical rules has significant advantages when dealing with some highly complex AI tasks. In some systems, where there may be an infinite number of possible answers, abstract expression or reasoning based on logical rules is a great convenience.

Each rule in the rule set can be viewed as a submodel. The rule set is an integration of these submodels. When the same examples are overwritten by multiple rules with different discriminant results, a conflict is said to have occurred. The method of conflict resolution is called conflict resolution. Common conflict resolution strategies include voting, sorting, meta-rule, and so on. In terms of the expressive ability of formal language, rules can be divided into propositional rules and first-order rules. The propositional rule is a simple declarative sentence composed of atomic propositions and logical connectives "$\wedge$," "$\vee$," "," and "$\longrightarrow$." First-order rules are atomic formulas that describe properties or relationships of things.

The goal of logical rule learning is to produce a rule set that covers as many samples as possible. The most direct way to learn rules is sequential covering. For each rule learned on the training set, the training sample covered by that rule is removed. Then, the remaining training samples were used to form a training set and the process was repeated. Because only one part of the data is processed at a time, it is also

known as a divide-and-conquer strategy. The approach based on exhaustive search is not feasible when there are many attributes and candidate values due to combinatorial explosion. Generative testing and data-driven methods are two common strategies for generating rules. The former is much more robust to noise than the latter. So it is often used in propositional rules. The latter are often used in complex tasks such as first-order rule learning in the hypothesis space. Considering only one optimal literal at a time during rule generation is often too greedy and tends to fall into local optimality. To mitigate this problem, some relatively modest measures can be adopted, such as cluster search.

Rule generation is essentially a greedy search process. It requires mechanisms to mitigate the risk of overfitting. The most common practice is pruning, including pre-pruning and post-pruning. Rule set performance for adding or removing literals is usually evaluated based on some kind of performance metric. Then to determine if pruning is needed, pruning can be carried out by means of statistical significance tests. For example, when the Clark and Niblett (CN2) algorithm preprunes, it is assumed that the prediction based on the rule set must be significantly better than the prediction based on the posterior probability distribution of the training sample set. For ease of calculation, Likelihood Ratio Statistics (LRS) is used for CN2. Let m_+ and m_- represent the number of positive and negative examples in training sample set, respectively. $\hat{m}_+$ and $\hat{m}_-$, respectively, represent the number of positive and negative cases covered by the rule set. Then,

$$\mathrm{LRS} = 2 \cdot \left(\hat{m}_+ \log_2 \frac{(\hat{m}_+ / \hat{m}_+ + \hat{m}_-)}{(m_+ / m_+ + m_-)} + \hat{m}_- \log_2 \frac{(\hat{m}_- / \hat{m}_+ + \hat{m}_-)}{(m_- / m_+ + m_-)} \right). \tag{1}$$

In practical tasks with a large amount of data, CN2 algorithm usually stops only when LRS is very large. The most common post-pruning strategy is error-reducing pruning. The basic method is to divide the sample set into training set and verification set. Multiple rounds of pruning are performed after learning rule set R from the training set. Pruning operations include deleting a literal, deleting a literal at the end of a rule, deleting multiple literals at the end of a rule, and deleting the whole rule. It is often better to combine pruning with some other postprocessing to optimize a rule set. Take the famous rule learning algorithm RIPPER for example. Its generalization performance exceeds that of many decision tree algorithms. And it learns faster than most decision tree algorithms. The trick is to combine pruning with postprocessing optimization.

Definition 1 (see [34]). Let $(L, \vee, \wedge, O, I)$ be a bounded lattice with an order-reversing involution "$'$," $\longrightarrow : L \times L \longrightarrow L$ be a mapping. $(L, \vee, \wedge, ', O, I)$ is called an LIA if the following conditions hold for any $x, y, z \in L$:

(I_1) $x \longrightarrow (y \longrightarrow z) = y \longrightarrow (x \longrightarrow z)$,
(I_2) $x \longrightarrow x = I$,
(I_3) $x \longrightarrow y = y' \longrightarrow x'$,
(I_4) $x \longrightarrow y = y \longrightarrow x = I$ implies $x = y$,
(I_5) $(x \longrightarrow y) \longrightarrow y = (y \longrightarrow x) \longrightarrow x$,
(L_1) $(x \vee y) \longrightarrow z = (x \longrightarrow z) \wedge (y \longrightarrow z)$,
(L_2) $(x \wedge y) \longrightarrow z = (x \longrightarrow z) \vee (y \longrightarrow z)$.

Definition 2 (see [34]). (Łukasiewicz implication algebra on a finite chain L_n). Let L_n be a finite chain, $L_n = \{a_i | 1 \le i \le n\}$, and $a_1 < a_2 < \cdots < a_n$. Define for any $a_i, a_j \in L_n$, $a_i \vee a_j \in a_{\max\{i,j\}}$, $a_i \wedge a_j \in a_{\min\{i,j\}}$, $(a_i)' = a_{n-i+1}$, $a_i \longrightarrow a_j = a_{\min(n-i+j,n)}$; then $(L_n, \vee, \wedge, ', \longrightarrow, a_1, a_n)$ is an LIA.

Definition 3 (see [35]). Let $L_n = \{a_1, a_2, \cdots, a_n\}$, $a_1 < a_2 < \cdots < a_n$, $L_2 = \{b_1, b_2\}$, $b_1 < b_2$, $(L_n, \vee, \wedge, ', \longrightarrow, a_1, a_n)$ and $(L_2, \vee, \wedge, ', b_1, b_2)$ be two Łukasiewicz implication algebras. The Hasse diagram of $L_n \times L_2$ is depicted in Figure 1, and for any $(a_i, b_j), (a_k, b_l) \in L_n \times L_2$, define $(a_i, b_j) \vee (a_k, b_l) = (a_i \vee a_k, b_j \vee b_l)$, $(a_i, b_j) \wedge (a_k, b_l) = (a_i \wedge a_k, b_j \wedge b_l)$, $(a_i, b_j)' = (a_i \wedge b_j)(a_i, b_j) \longrightarrow (a_k, b_l) = (a_i \longrightarrow a_k, b_j \longrightarrow b_l)$. Then, $(L_n \times L_2, \vee, \wedge, ', \longrightarrow, (a_1, b_1), (a_n, b_2))$ is an LIA, denoted by $L_{n\times 2}$.

Definition 4 (see [35]). Let $\mathrm{AD}_n = \{h_1, h_2, \ldots, h_n\}$ be a set with n linguistic modifiers and $h_1 < h_2 < \cdots < h_n$, $\mathrm{MT} = \{f, t\}$ be a set of meta truth values, and denote $L_{V(n\times 2)} = \mathrm{AD}_n \times \mathrm{MT}$. Define a mapping g as g: $L_{V(n\times 2)} = L_n \times L_2$, $f < t$

$$g(h_i, mt) = \begin{cases} (a_i', b_1), & mt = f, \\ (a_i, b_2), & mt = t. \end{cases} \tag{2}$$

Then, g is bijection, and its inverse mapping is denoted as g^{-1}. For any $x, y \in L_{V(n\times 2)}$, define $x \vee y = g^{-1}(g(x) \vee g(y))$ x, $x \wedge y = g^{-1}(g(x) \wedge g(y))$, $x' = g^{-1}(g(x))$, $x \longrightarrow y = g^{-1}(g(x) \longrightarrow g(y))$; then, $L_{V(n\times 2)} = (L_{V(n\times 2)}, \vee, \wedge, ', \longrightarrow, (h_n, f), (h_n, t))$ is called a linguistic truth values, and g is an isomorphic mapping from $L_{V(n\times 2)}$ to $L_{n\times 2}$.

Definition 5 (see [22]). Let X be a set of propositional variables and $T = L \cup \{', \longrightarrow\}$ be a type with $ar(') = 1, ar(\longrightarrow) = 2$ and $ar(a) = 0$ for every $a \in L$. The propositional algebra of the lattice-valued propositional calculus on the set X of propositional variables is the free T algebra on X and is denoted by $\mathrm{LP}(X)$.

Remark 1. Particularly, when the field with valuation of $\mathrm{LP}(X)$ is an $L_{V(n\times 2)}$, this specific $\mathrm{LP}(X)$, i.e., $L_{V(n\times 2)}P(X)$, is a linguistic truth-valued lattice-valued propositional logic system. Similarly, the truth-valued domain of $L_nP(X)$ is a Łukasiewicz implication algebra L_n.

Definition 6 (see [22]). A valuation of $\mathrm{LP}(X)$ is a propositional algebra homomorphism γ: $\mathrm{LP}(X) \longrightarrow L$.

Implication is a basic logical connective in all kinds of logical systems, which plays an important role. Implication operators have been discussed in many articles. And they have been taken as the research object in a large number of researches on approximate reasoning in recent years.

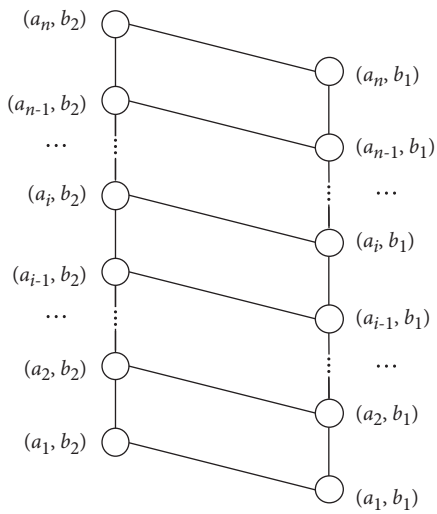


Figure 1: Hasse diagram of $L_n \times L_2$.

However, most of the researches focus on semantics, but few on construction. Therefore, the concept of fuzzy implication algebra was introduced by Wu [36]. It was an algebraic abstraction of implication connectives in [0,1]-valued logic. In order to describe incomparability and axiomatize implication operators, the concept of lattice implication algebra was proposed by Y. Xu in 1993 by combining lattice and implication algebra. The properties of lattice implication algebras were then studied by him and his students. Taking lattice implication algebra as the true range, several lattice-valued logic systems were established. They mainly included lattice-valued propositional logic system $\mathrm{LP}(X)$ and corresponding lattice-valued first-order logic system $\mathrm{LF}(X)$, dynamic hierarchical L-type lattice-valued propositional logic system L_{vpl}, and corresponding L-type lattice-valued first-order logic system L_{vfl}. The lattice-valued propositional logic system L_{cp} is obtained by improving $\mathrm{LP}(X)$ in order to discuss the problem. On this basis, Xu et al. [37] discussed the problem of *LP(X)* and *LF(X)* resolution, proposed the concept of α-resolution principle, and proved the reliability and completeness of α-resolution principle. The α-resolution principle and automatic inference were further studied by Liu et al.

Resolution principle [38] is one of the most important methods to judge the unsatisfiability of logical formulas. Because of its reliability and completeness, it is widely used in modern automatic theorem proving systems. In order to improve reasoning efficiency, many scholars reduce the clauses involved in resolution from the perspective of restricting clauses and literals and at the same time ensure its completeness, such as ordered linear resolution, semantic resolution, locking resolution, and other logical deduction methods.

2.2. Algorithm. Suppose an undirected graph $G = \langle V, E \rangle$, where V is a nonempty finite set, called a vertex set, whose elements are called vertices or nodes. E is a finite subset of the unordered product $V\&V$, called an edge set. Its elements are called undirected edges, or edges for short.

Suppose a clause set $S = \{C_1, C_2, \ldots, C_n\}$, where C_1 to C_n are corresponding to each node in undirected graph V_1 to V_n. If there are resolution literals between two clause sets, add an undirected edge to the nodes represented by these two clauses, and then add the first resolution literal to the edge.

In graph theory, to find satisfying solutions by logical deduction is to find whether there are Hamiltonian paths in the undirected graph. Before finding the Hamiltonian path, it is unknown whether there are edges between nodes. Choosing the starting node is the key problem of many algorithms. Some algorithms choose a short clause as the start node. Some algorithms choose a clause with fewer branches as the starting node. Some algorithms calculate the correlation or weight of literals or clauses according to the strategy and select the clause with high correlation or heavy weight as the starting node. Some algorithms try to extend local satisfiability solutions to global satisfiability solutions. Branching decision has been adopted by most of these algorithms and require frequent backtracking and restarts mechanisms.

Utilizing distributed computing is the core idea of this paper. It is very suitable for us to find satisfying solutions. The algorithm is described as follows.

Step 0 (Initiation). Given a set of clauses $S = \{C_1, C_2, \ldots, C_n\}$, where n is the length of the clause set, there is a conjunctive relationship between clause sets.

Step 1. To start n subtasks and determine the starting clause for each subtask.

Step 2. In each subtask, find a match for the literal in the starting clause according to the resolution deduction. And generate a one-layer branch topology according to the matching results.

Step 3. To start a general task, which collects the topology result of each subtask and forms a shared library file. Then end n subtasks.

Step 4. To start n topology subtasks. Then generate multilayer topology according to the shared library file. To judge whether there is a Hamiltonian path in the topology.

Step 5. If any topology subtask finds a Hamiltonian path, provide feedback to the general task. The general task notifies all subtasks to terminate the task. It can be determined that there is a satisfiability solution and the clause set is satisfiable.

Step 6. If all topology subtasks end, and there is no Hamiltonian path, then it can be determined that there is no satisfiability solution and the clause set is unsatisfiable.

2.3. Examples and Results. Given a set of clauses $S = \{C_1, C_2, C_3, C_4, C_5, C_6, C_7\}$, there is a conjunctive relationship between clause sets. Literals in these clauses read as follows:

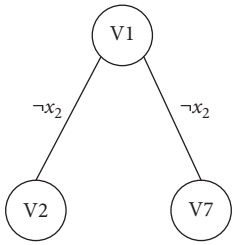


Figure 2: A one-layer branch topology of C_1.

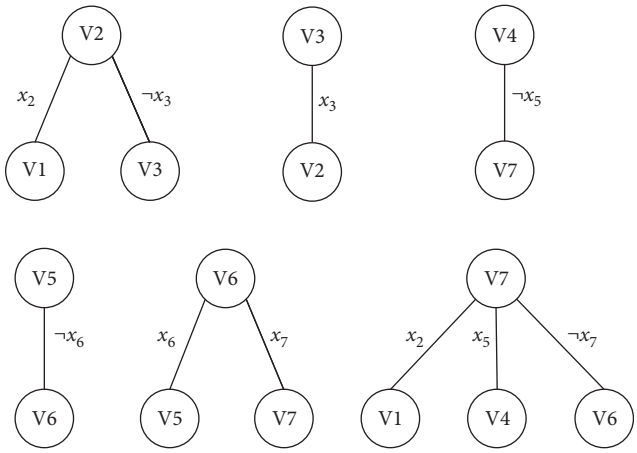
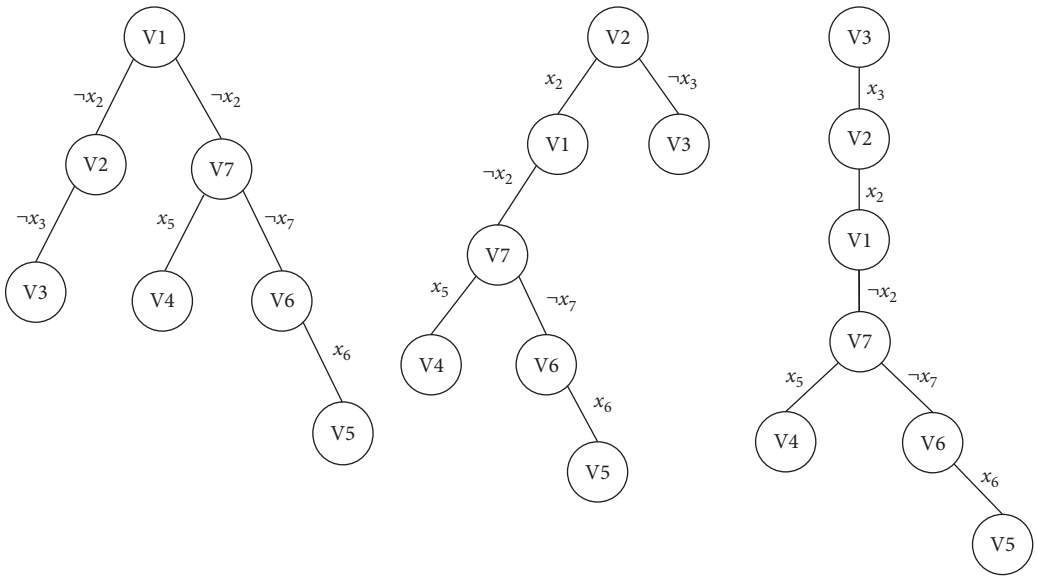


Figure 3: The one-layer branch topology of other clauses.



(a) (b) (c)

Figure 4: Continued.

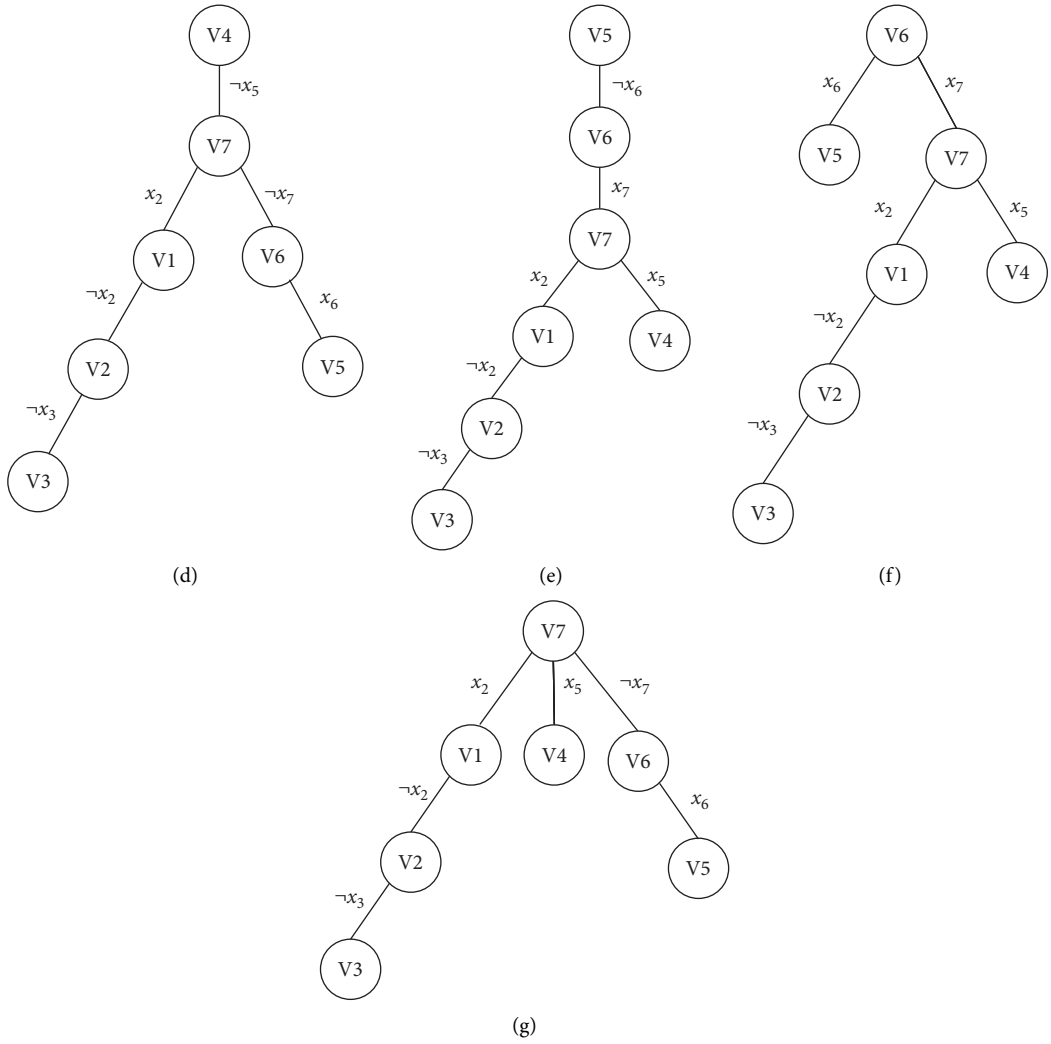


Figure 4: The multilayer topology of subtasks. (a) Topology of P_1; (b) topology of P_2; (c) topology of P_3; (d) topology of P_4; (e) topology of P_5; (f) topology of P_6; (g) topology of P_7.

$$C_1 = x_1 \vee x_2$$
$$C_2 = x_2 \vee x_3 \vee x_4$$
$$C_3 = x_1 \vee x_3$$
$$C_4 = x_4 \vee x_5 \vee x_8$$
$$C_5 = x_4 \vee x_6 \vee x_9$$
$$C_6 = x_6 \vee x_7$$
$$C_7 = x_2 \vee x_5 \vee x_7 \vee x_{10}.$$

Given a set of tasks $T = \{T_1, T_2, \ldots, T_7\}$, to start seven tasks from T_1 to T_7, take subtask T_1 as an example. Clause C_1 is chosen as the starting one. According to the principle of resolution, to find match literals of x_1 and x_2, respectively, C_2 and C_7 are identified as resolution clauses. Then a one-layer branch topology of C_1 is generated (see Figure 2). Literals on the edge represent first ones involved.

The branch topology of other clauses can be obtained by the same method (see Figure 3).

The general task collects the topology result of each subtask and forms a shared library file. Subtasks from T_1 to T_7 can be ended. Given a set of topology tasks $P = \{P_1, P_2, \ldots, P_7\}$, to start seven topology tasks from P_1 to P_7, take topology subtask P_1 as an example. V_1 is chosen as the starting node. According to the shared library file, the multilayer topology of P_1 is generated. The multilayer topology of other subtasks can be obtained by the same method (see Figure 4).

In the judgment process, if there is a satisfying solution in the result, the multilayer topology should be pruned further. According to the principle of resolution, literals on the upper and lower edges of the middle layer node cannot be matched. Remove branches that do not meet the criteria. The pruned multilayer topology is shown (see Figure 5).

According to the pruned topology, we can judge whether there is a satisfiability solution. If there is no satisfiability solution, the local maximum length satisfiability solution can be obtained. Sets of $\{C_1, C_7, C_6, C_5\}$ and $\{C_4, C_7, C_6, C_5\}$ are the local maximum length satisfiability solution in this example. Sets of $\{x_2, x_2, x_7, x_7, x_6, x_6\}$ and $\{x_5, x_5, x_7, x_7, x_6, x_6\}$ are the local maximum length satisfiability literals in this example.

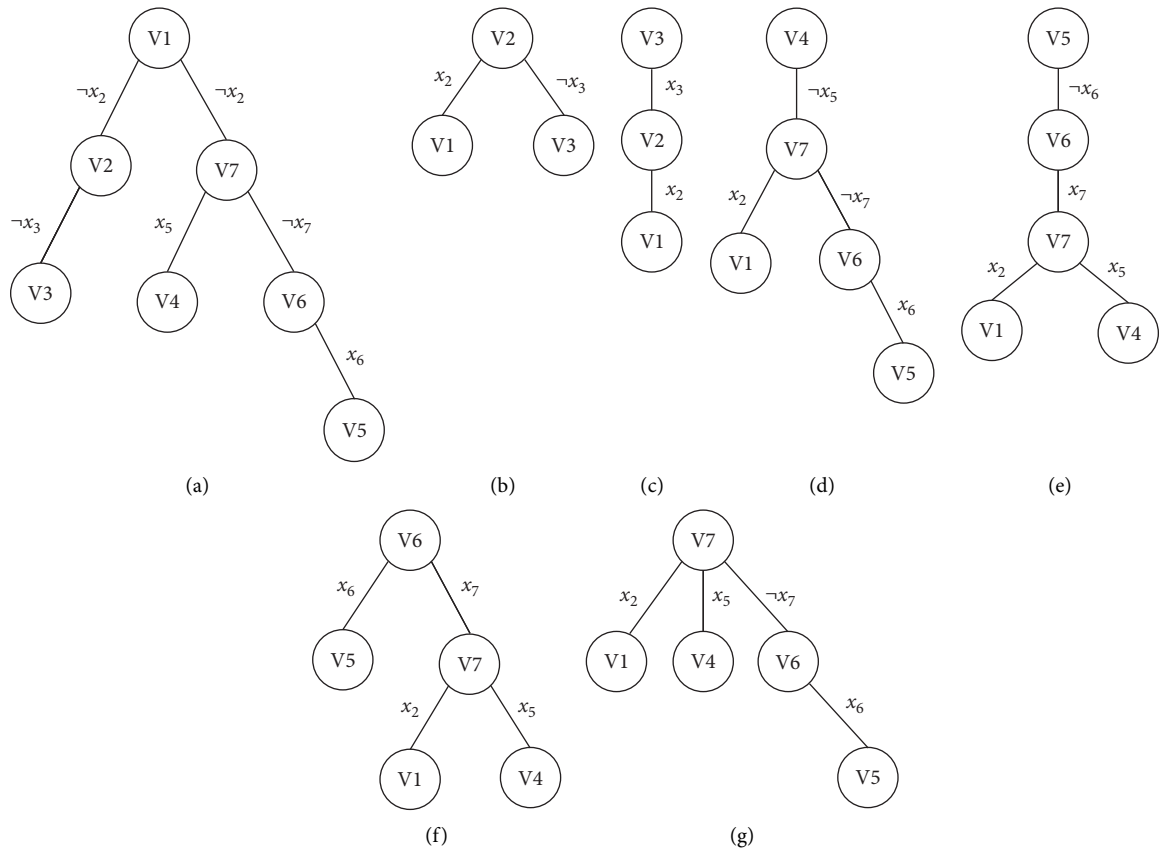


Figure 5: The pruned multilayer topology. (a) Pruned topology of P_1; (b) pruned topology of P_2; (c) pruned topology of P_3; (d) pruned topology of P_4; (e) pruned topology of P_5; (f) pruned topology of P_6; (g) pruned topology of P_7.

3. Conclusion and Discussion

Classical logic is the basis of reasoning mechanism in artificial intelligence, fuzzy control, and other fields. It adopts formal methods to describe uncertain reasoning problems. At present, nonclassical logic theory is in the stage of rapid development, and there are many problems to be solved, which limit the further successful application of nonclassical logic theory in control engineering. Based on the latest research results in this field, some problems of nonclassical logic are studied, specifically, lattice-implication algebra, lattice-valued logic, and lattice-valued model theory. In this paper, a topology graph algorithm based on the principle of lattice-valued logic is presented. It provides a basis of solving the satisfiability solutions to some clause sets. The further research will be concentrated on discussing how to combine it with other traditional resolution methods and algorithms, and analyzing the complexity of related algorithms. Formal verification is a direct application of automated reasoning in software engineering; how to apply this method to reason and validate the validity of programs which are represented by logical formulation is worthy of studying. Transportation control systems are safety critical systems. The formal verification of software and hardware in intelligent transportation system and the related high confidence software in some specific areas are also deserving research topics. Hence how to translate the related procedures into formulas in some specific logic systems and further to validate their correctness due to our existing resolution method is an important direction.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that there are no potential conflicts of interest in this study.

Authors' Contributions

All the authors have seen and approved the manuscript to be published.

Acknowledgments

This paper was supported by the National Natural Science Foundation of China under Grant no. 62172135.

References

[1] S. A. Cook, "The complexity of theorem-proving procedures," in *Proceedings of the Third Annual ACM Symposium on*

Theory of Computing, pp. 151–158, Shaker Heights, OH, USA, May 1971.

[2] D. Achlioptas, "Random satisfiability," *Handbook of Satisfiability*, vol. 185, p. 245, 2009.

[3] A. Biere, "PicoSAT essentials," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 4, no. 2–4, pp. 75–97, 2008.

[4] A. Douglass, A. D. King, and J. Raymond, "Constructing SAT filters with a quantum annealer, Lecture Notes in Computer Science," in *Proceedings of the International Conference on Theory and Applications of Satisfiability Testing*, pp. 104–120, Springer, Austin, TX, USA, September 2015.

[5] Y. Diao, X. Wei, T. K. Lam, and Y. L. Wu, "Coupling reverse engineering and SAT to tackle NP-complete arithmetic circuitry verification in~ O (# of gates)," in *Proceedings of the 2016 Twentyfirst Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 139–146, IEEE, Macao, China, January 2016.

[6] L. Yin, F. He, W. N. Hung, X. Song, and M. Gu, "Maxterm covering for satisfiability," *IEEE Transactions on Computers*, vol. 61, no. 3, pp. 420–426, 2010.

[7] C. Bright, I. Kotsireas, and V. Ganesh, "Applying computer algebra systems with SAT solvers to the Williamson conjecture," *Journal of Symbolic Computation*, vol. 100, pp. 187–209, 2020.

[8] B. König, M. Nederkorn, and D. Nolte, "CoReS: a tool for computing core graphs via SAT/SMT solvers," *Journal of Logical and Algebraic Methods in Programming*, vol. 109, Article ID 100484, 2019.

[9] W. Liu and C. Xiao, "An efficient technique of application mapping and scheduling on real-time multiprocessor systems for throughput optimization," *ACM Transactions on Embedded Computing Systems*, vol. 15, no. 4, pp. 1–25, 2016.

[10] A. Deshpande and R. K. Layek, "Fault detection and therapeutic intervention in gene regulatory networks using SAT solvers," *Biosystems*, vol. 179, pp. 55–62, 2019.

[11] M. Ouimet and K. Lundqvist, "Automated verification of completeness and consistency of abstract state machine specifications using a sat solver," *Electronic Notes in Theoretical Computer Science*, vol. 190, no. 2, pp. 85–97, 2007.

[12] X. Zhao, L. Zhang, D. Ouyang, and Y. Jiao, "Deriving all minimal consistency-based diagnosis sets using SAT solvers," *Progress in Natural Science*, vol. 19, no. 4, pp. 489–494, 2009.

[13] V. I. Ulyantsev and F. N. Tsarev, "Extended finite-state machine induction using SAT-solver," *IFAC Proceedings*, vol. 45, no. 6, pp. 236–241, 2012.

[14] M. Davis, G. Logemann, and D. Loveland, "A machine program for theorem-proving," *Communications of the ACM*, vol. 5, no. 7, pp. 394–397, 1962.

[15] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: engineering an efficient SAT solver," in *Proceedings of the Thirtyeight Annual Design Automation Conference*, pp. 530–535, Las Vegas, NV, USA, June 2001.

[16] L. Ryan, *Efficient Algorithms for Clause-Learning SAT Solvers*, Doctoral Dissertation, Thesis, School of Computing Science)/ Simon Fraser University, Burnaby, Canada, 2004.

[17] J. Marques-Silva, I. Lynce, and S. Malik, "Conflict-driven clause learning SAT solvers," *Frontiers in Artificial Intelligence and Applications*, vol. 185, pp. 133–182, 2021.

[18] G. Audemard and L. Simon, "Glucose in the SAT 2014 competition," *Proceedings of SAT Competition*, vol. 2014, 2014.

[19] R. G. Jeroslow and J. Wang, "Solving propositional satisfiability problems," *Annals of Mathematics and Artificial Intelligence*, vol. 1, no. 1–4, pp. 167–187, 1990.

[20] M. Buro and H. K. Büning, *Report on a SAT Competition*, Fachbereich Math.-Informatik, Univ. Gesamthochschule, Paderborn, Germany, 1992.

[21] J. W. Freeman, *Improvements to Propositional Satisfiability Search Algorithms*, Doctoral dissertation, University of Pennsylvania, PA, USA, 1995.

[22] J. P. Marques-Silva and K. A. Sakallah, "GRASP: a search algorithm for propositional satisfiability," *IEEE Transactions on Computers*, vol. 48, no. 5, pp. 506–521, 1999.

[23] J. H. Liang, V. Ganesh, P. Poupart, and K. Czarnecki, "Learning rate based branching heuristic for SAT solvers," in *Proceedings of the International Conference on Theory and Applications of Satisfiability Testing*, pp. 123–140, Springer, Bordeaux, France, July 2016.

[24] J. P. S. Marques and K. A. Sakallah, "Grasp-A new search algorithm for satisfiability," in *The Best of ICCAD*, pp. 73–89, Springer, Boston, MA, USA, 2003.

[25] H. Zhang, "SATO: an efficient propositional prover," in *Proceedings of the International Conference on Automated Deduction*, pp. 272–275, Springer, Townsville, North Queensland, Australia, July 1997.

[26] E. Goldberg and Y. Novikov, "BerkMin: a fast and robust SAT-solver," *Discrete Applied Mathematics*, vol. 155, no. 12, pp. 1549–1561, 2007.

[27] Y. S. Mahajan, Z. Fu, and S. Malik, "Zchaff2004: an efficient sat solver," in *Proceedings of the International Conference on Theory and Applications of Satisfiability Testing*, pp. 360–375, Springer, Vancouver, BC, Canada, May 2004.

[28] N. Eén and N. Sörensson, "An extensible SAT-solver," in *Proceedings of the International Conference on Theory and Applications of Satisfiability Testing*, pp. 502–518, Springer, Santa Margherita Ligure, Italy, May 2003.

[29] G. Audemard and L. Simon, "Predicting learnt clauses quality in modern SAT solvers," in *Proceedings of the Twenty-first International Joint Conference on Artificial Intelligence*, Pasadena, CA, USA, June 2009.

[30] A. Biere, "Lingeling and friends entering the SAT Challenge," *Proceedings of SAT Challenge*, vol. 2012, pp. 33-34, 2012.

[31] J. H. Liang, H. G. V.K., P. Poupart, K. Czarnecki, and V. Ganesh, "An empirical study of branching heuristics through the lens of global learning rate," in *Proceedings of the International Conference on Theory and Applications of Satisfiability Testing*, pp. 119–135, Springer, Melbourne, Australia, August 2017.

[32] M. Luo, C. M. Li, F. Xiao, F. Manya, and Z. Lü, "An effective learnt clause minimization approach for CDCL SAT solvers," in *Proceedings of the Tweentysixth International Joint Conference on Artificial Intelligence*, pp. 703–711, Melbourne, Australia, August 2017.

[33] W. Chang, Y. Xu, and S. Chen, "An adaptive strategy for tuning duplicate trails in SAT solvers," *Symmetry*, vol. 11, no. 2, 2019.

[34] Y. Xu, D. Ruan, K. Qin, J. Liu, and L. V. Logic, "An alternative approach to treat fuzziness and incomparability," *Studies in Fuzziness and Soft Computing*, vol. 132, 2003.

[35] Z. Pei, D. Ruan, J. Liu, and Y. Xu, "Linguistic values based intelligent information processing: theory, methods, and applications," *Atlantis Computational Intelligence Systems*, Berlin, Germany, 2010.

[36] W. M. Wu, "Fuzzy implication algebras," *Fuzzy systems and Mathematics*, vol. 1, pp. 510–513, 1990.

[37] Y. Xu, D. Ruan, E. E. Kerre, and J. Liu, "α-resolution principle based on lattice-valued propositional logic LP (X)," *Information Sciences*, vol. 130, no. 1–4, pp. 195–223, 2000.

[38] C. L. Chang and R. C. T. Lee, *Symbolic Logic and Mechanical Theorem Proving*, Academic Press, Cambridge, MA, USA, 2014.