A NEW PERSPECTIVE OF PARAMODULATION COMPLEXITY BY SOLVING 100 SLIDING BLOCK PUZZLES

Ruo Ando[1]and Yoshiyasu Takefuji[2]

[1]National Institute of Informatics, 2-1-2 Hitotsubashi,
Chiyoda-ku, Tokyo 101-8430 Japan
[2]Musashino University Faculty of Data Science 3-3-3 Ariake, Koto-Ku,
Tokyo 1358181, Japan

ABSTRACT

*This paper gives complete guidelines for authors submitting papers for the AIRCC Journals. A sliding puzzle is a combination puzzle where a player slides pieces along specific routes on a board to reach a certain end configuration. In this paper, we propose a novel measurement of the complexity of 100 sliding puzzles with paramodulation, which is an inference method of automated reasoning. It turned out that by counting the number of clauses yielded with paramodulation, we can evaluate the difficulty of each puzzle. In the experiment, we have generated 100 * 8 puzzles that passed the solvability checking by countering inversions. By doing this, we can distinguish the complexity of 8 puzzles with the number generated with paramodulation. For example, board [2,3,6,1,7,8,5,4, hole] is the easiest with score 3008 and board [6,5,8,7,4,3,2,1, hole] is the most difficult with score 48653.Besides, we have succeeded in obverse several layers of complexity (the number of clauses generated) in 100 puzzles. We can conclude that the proposed method can provide a new perspective of paramodulation complexity concerning sliding block puzzles.*

KEYWORDS

Automated reasoning complexity, paramodulation, given-clause algorithm, generated clauses, 8 puzzles, OTTER.

1. INTRODUCTION

A sliding puzzle (also called a sliding block puzzle) is a combination puzzle where a player slides pieces along specific routes on a board to reach a certain end configuration (state).The pieces are usually numbered and sometimes may be imprinted with colours, patterns, and sections of a large picture. In nature, sliding puzzles are two-dimensional, even if encaged marbles or three-dimensional tokens facilitate the sliding.

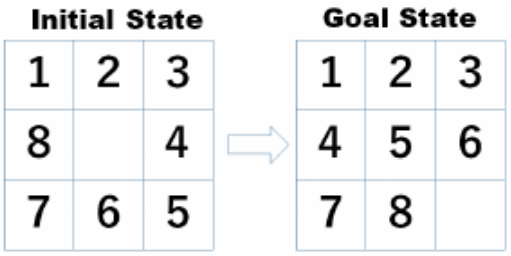

Figure 1. Initial state and goal state of 8 puzzle

DOI: 10.5121/ijaia.2021.12604

In sliding puzzles, a player is prohibited from lifting any piece of the board. This constraint separates sliding puzzles from rearrangement puzzles. Consequently, discovering routes opened up by each move with the two-dimensional confines of the board is an interesting point of solving sliding block puzzles. Figure 1 shows the example of a sliding puzzle. The puzzle has 9 square slots on a square board. The first eight slots have square pieces. The 9th slot is empty. Historically, Noyes Chapman invented the oldest type of sliding puzzle, the fifteen puzzle, in 1880.Folklore tells us that in 1886, puzzle master Sam Loyd offered a one-thousand dollar prize if anyone could swap tile 14 and 15 and return the other tiles to their original slots.

Sliding block can be represented as the permutation. A permutation of a set S is a bijection from S onto itself. If the set we permuting is A = {1,2, ..., n}, it is often convenient to represent a permutation σ as follows:

$$\sigma = \begin{Bmatrix} 1, & 2, & 3, & \ldots & \\ \sigma(1), & \sigma(2), & \sigma(3), & \ldots, & \sigma(n) \end{Bmatrix} \qquad (1)$$

For instance, consider the set A = {1,2,3,4,5,6}. Then the permutation π,

$$\pi = \begin{Bmatrix} 1, & 2, & 3, & 4, & 5, & 6 \\ 4, & 1, & 5, & 2, & 3, & 6 \end{Bmatrix} \qquad (2)$$

Send 1 to 4, 2 to 1, 3 to 5, fixes, or leaves unchanged, element 6.

This paper is organized as follows. In section 2, we introduce the basic method and tool for our proposed method. In section 3, we present the detail illustration of paramodulation. In section 4, we present the inference rules for solving sliding block puzzle. Also, we discuss how to generated the samples of sliding block puzzles and measure the complexity. In section 5, we show the numerical results and summary of experiment of solving 100 sliding block puzzle. After the related work of section 6, we proceed to conclusion and further work.

2. OTTER

2.1. OTTER and Its Clause Sets

The theorem prover OTTER (Organized Techniques for Theorem-proving and Effective Research) has been developed by W. McCune as a product of Argonne National Laboratory. OTTER is based on earlier work by E. Lusk, R. Overbeek, and others [12].By the research efforts of [8][9][11] for certain classes of problem, OTTER is widely regarded as the most powerful automated deduction system. OTTER adopts the given-clause algorithm and implements the set of support strategy [13].

In the given-clause algorithm, all retained clauses are divided into two sets. The first set is called the set of support (SOS). Thus, OTTER starts with the retention of a set of support, including all of the chosen input clauses. During the run, the initial set of support and the clauses which are generated are retained. The second set is the usable list. The usable list is not included in the initial set of support at the beginning phase of reasoning. The usable list is the clauses that were once in the set of support but have already been picked up as the focus of attention for deducing additional clauses. More technically, in detail, OTTER maintains four lists of clauses in the reasoning process.

[1] Usable. This list works as a rule by keeping clauses that are available to make inferences.
[2] SoS. Clauses are regarded as facts. Set of support are not used to make inferences. They are kept to participate in the search.
[3] Passive. They are specified to be used only for forward subsumption and unit conflict. Therefore, the passive list does not participate in the search.
[4] The passive list does not change from the start of the reasoning process as a fixed input.
[5] Demodulators. Demodulators are used to rewrite newly inferred clauses with equalities.

In this paper, mainly, we focus on the size of the set of support list. Set of support is an important indicator for introspecting the reasoning process.

2.2. Given Clause Algorithm

OTTER adopts a given-clause algorithm in which the program attempts to use any and all combinations from axioms in a given clause. In other words, the clauses' combinations are generated from given clauses that have been focused on.

Algorithm 1. Given Clause Algorithm

Algorithm 1 Given clause algorithm

Input: SOS, Usable List
Output: Proof

```
1:  while until SoS is empty do
2:    choose a given clause G from SoS;
3:    move the clause g to Usable List;
4:    while  c_1, ..., c_n in Usable List do
5:      while R(c_1, ..c_i, G, c_{i+1}, ..c_n)exists do
6:        A ⇐ R(c_1, ..c_i, G, c_{i+1}, ..c_n);
7:        if A is the goal then
8:          report the proof;
9:          stop
10:       else {A is new odd}
11:         add A to SoS X
12:       end if
13:     end while
14:   end while
15: end while
```

n line 2, given clause G is extracted from SoS (Set of Support).Line 4 and 5 are a loop to use any given clause and Usable List combinations. In detail, [7][8] discuss the basic framework of given clause algorithm. To put it simply, the given clause algorithm consists of the following steps.

[1] Pick up a clause (called the given clause) from the set of support.
[2] Add the given clause to the usable list.
[3] Applying the inference rule or rules in effect infer all clauses which are generated from the given clause (one parent) and the usable list (other parents).
[4] Process newly inferred clause.
[5] Append each inferred new clause to the SoS. This clause is not discarded as a result of processing. Exactly, this is done in the course of processing the newly generated clause.

In a nutshell, the reasoning program chooses a clause from the clauses focused on in the support set. The selected clause is called a focal clause or given clause.

Definition of given clause. The reasoning program chooses a clause on which to focus from among those in the set of support, where the choice is based on various criteria, such as the weight of the clause. The chosen clause is based on various criteria from among those in the clause. The selected clause is called the ``focal clause'' (formerly the ``given clause''). The algorithm under discussion permits the focal clause to be considered by whatever inference rules are being used, where the remaining clauses required by the inference rule are selected from the usable list but not from the set of support. An equality literal is a literal whose predicate is to be interpreted as meaning ``equal''. The inference rule yields clause C from clauses A and B that are assumed to have no variables in common when A contains positive equality literal and B contains a term which unifies with one of the arguments of that equality literal.

3. PARAMODULATION

Paramodulation is a powerful method of equational reasoning. It is the method based on resolution refutations that include equality. In the view of equational reasoning, paramodulation is a generalization of equality substitution. For example, if the equality of s = t and s occurs in the sentence S, paramodulation can replace sin any of the occurrences. Also, on the rule of s = t, if t occurs in S, then reasoning program can replace s with t.

$$\frac{C \vee AD \vee \neg B}{(C \vee A)\sigma} \quad if\sigma = mgu(A, B) \qquad (3)$$

Here, mgu (A, B) denotes a most general unifier of A and B, and factoring:

$$\frac{C \vee A \vee B}{(C \vee A)\sigma} \quad if\sigma = mgu(A, B) \qquad (4)$$

Definition of paramodulation. Equality means if a literal whose predicate is to be represented as equal. The inference rule of paramodulation yields clause C between clauses A and B, which are assumed to have no variables in common. Also, if A contains positive equality literal and B contains a term which unifies with one of the arguments, C is yielded by paramodulation.

In the equations above (3)(4), Clause A is called the from clause, clause B is called the into clause, and clause C a paramount. Corresponding to the syntax of OTTER, for example, given the following two clauses, from the first into the second.

EQUAL(a,b).
Q(a).
yields the clause
Q(b).

EQUAL(sum(x,0), x).
P(sum(sum(a,0), b),c).
Paramodulation yields
P(sum(a,b),c).

as a paramount, from

Q(g(f(g(x)))).
EQUAL(g(a),b).
the clause

Q(g(f(b))).

is yielded by adopting paramodulation. For a second example, given the following two clauses, from the first into the second.

In general, paramodulation is intended to be utilized, along with resolution, for theorem proving in first-order theories with equality. Concerning the implementation of OTTER, in paramodulation, two parents and a child are processed. The parent clauses contain the equality applied for the replacement. Thus, the parent clauses are divided into two: from parent and from clause. If equality comes from the literal, the side of equality unifies with the term, which is replaced with the from the term. The replaced term is called the into the term. The literal containing the replaced term is also called the into literal. Also, the parent containing the replaced term is called the into the parent or clause.

Algorithm 2. Checking the solvability of N puzzles

Algorithm 2 Checking the solvability of N puzzles

Input: Board$[x_1, x_2, ..., x_n, hole]$
Output: SOLVABLE or UNSOLVABLE

1: $Board[X|XS] = Board[x_1, x_2, ..., x_n, hole]$
2: **while** XS in $Board[XlXS]$ is empty **do**
3: **for** i in XS **do**
4: statements..
5: **if** $(X \neq XS[i])$ **then**
6: $counter[i] + +$
7: **end if**
8: **end for**
9: **end while**
10: $line = check(Board[...] \subseteq hole)$
11: $sum = 0$
12: **for** i to n **do**
13: $sum+ = counter[i]$
14: **end for**
15: **if** $(line + sum\%2 == 0)$ **then**
16: $flag =$ SOLVABLE
17: **else**
18: $flag =$ UNSOLVABLE
19: **end if**

4. METHODOLOGY

4.1. Setting OTTER's rule set

As we discussed before, the primary inference mechanism of OTTER is based on the given-clause algorithm. Given-clause algorithm can be viewed as a simple implementation of the set of support strategies. OTTER maintains four lists of clauses: usable, SoS, demodulator, and passive. In our case, we cope with two kinds of clauses: usable and SoS.

Horizontal sliding from row[i] to row[i+1] is represented as follows.

list(usable).
EQUAL(l(hole,l(n(x),y)),l(n(x),l(hole,y))).
end_of_list.

$$\sigma = \left\{ \begin{array}{cccc} 1 & hole & 2 & 3 \\ 1 & 2 & hole & 3 \end{array} \right\} \qquad (5)$$

Vertical sliding from row[i] to row[i+4] is represented as follows.

list(usable).
EQUAL(l(hole,l(x,l(y,l(z,l(u,l(n(w),v)))))),
l(n(w),l(x,l(y,l(z,l(u,l(hole,v)))))).
end_of_list.

$$\sigma = \left\{ \begin{array}{ccccccccc} 1 & hole & 2 & 3 & 4 & 5 & 6 & 7 \\ 1 & 2 & 3 & 4 & 5 & hole & 6 & 7 \end{array} \right\} \qquad (6)$$

4.2. Checking the solvability of N puzzles

In general, to check the solvability of N puzzles, the number of inversions of each number of N slots is calculated. For example, if we have the board configuration board [2,3,6,1,7,8,5,4, hole](5,2,8,4,1,7, hole, 3,6), the number of inversions are as follows:

[1] 2 precedes 1 - 1 inversions
[2] 3 precedes 1 - 1 inversion
[3] 6 precedes 1, 5, 4 - 3 inversions
[4] 1 precedes none - 0 inversions
[5] 7 precedes 5, 4 - 2 inversions
[6] 8 precedes 5, 4 - 2 inversions
[7] 5 precedes 4 - 1 inversions
[8] 4 precedes none - 0 inversions

Total inversions 1+1+3+0+2+2+1+0 = 10 (Even Number) So this puzzle configuration is solvable.

On the other hand, it is not possible to solve an instance of 8 puzzles if a number of inversions are odd in the input state.

Algorithm 2 shows the procedure for checking the solvability of N puzzles. At lines 2 to 9, the number of inversions of each slot is counted. These figures are counted up at lines 11 to 14.Finally, the sum is checked if it is an even or odd number at lines 15 to 19.

Algorithm 3. Incrementing the number of generated clauses

Algorithm 3 Incrementing the number of generated clauses

```
1:  while given clause is NOT NULL do
2:     index_lits_clash(giv_cl);
3:     append_cl(Usable, giv_cl);
4:     if splitting() then
5:        possible_given_split(giv_cl);
6:     end if
7:     infer_and_process(giv_cl);
8:     giv_cl = extract_given_clause();
9:     track(the_number_of_generated_clauses);
10: end while
```

4.3. Incrementing the number of generated clauses

The main loop for inferring and processing clauses and searching for a refutation operates mainly on the lists usable and SoS.

[1] Choose appropriate given_clause in SoS;
[2] Move given_clause from list(SoS) to list(usable)
[3] Infer and process new clauses using the inference rules set.
[4] Newly generated clause must have the given_clause.
[5] Do the retention test on new clauses and append those to list(SoS).

The main loop is depicted in Algorithm 3.In line 9, the number of generated clauses is incremented. After line 8 of picking up the clause from a set of support, we can record the current size of the set of support. By doing this, we can obtain the plot with # puzzles and the number of generated clauses of the Y-axis, as shown in the next section.

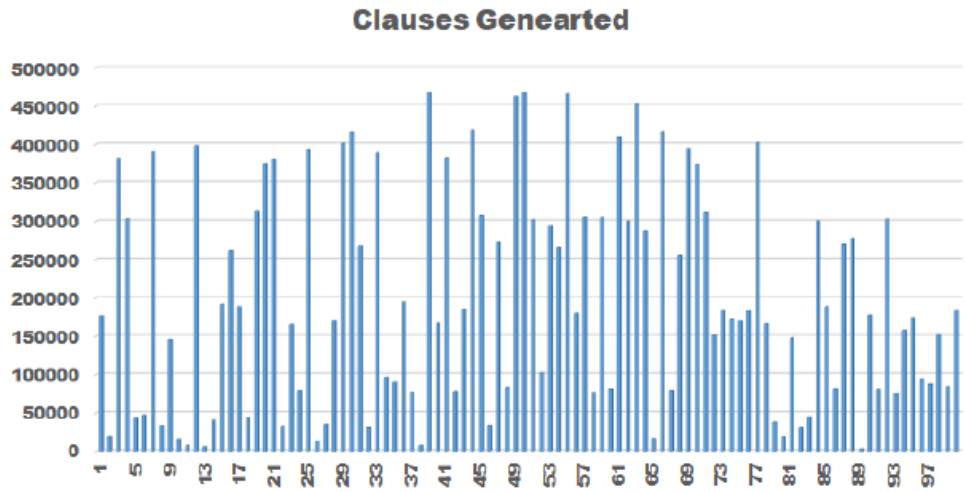

Figure 2. Clauses generated by paramodulation. X-axis is the number 8 puzzles. Y-axis is the number of generated clauses.

5. EXPERIMENTAL RESULTS

In the experiment, we have generated 100 sliding puzzles with size 8 * 8.All generated configurations of 8 puzzles are solvable. For each puzzle, we have measured the number of generated clauses with the procedures shown in Algorithm 2.For simplicity, we have generated the configuration of the first eight slots with random integers ranging from 1 to 8and fixed 9th slot to hole, as shown on the left side of Table I.

The number of generated clauses with paramodulation ranges from 3008 (2,3,6,1,7,8,5,4, hole) to 468453 (6,5,8,7,4,3,2,1, hole).In the view of complexity of reasoning process, the configuration [(6,5,8,7,4,3,2,1, hole)] is 155.73 times harder to solve than the configuration [(2,3,6,1,7,8,5,4, hole)].

Table 1. Initial board states and the complexities of paramodulation

Initial state	clauses generated
2,3,6,1,7,8,5,4,hole	3008 (easiest)
2,4,3,8,7,6,5,1,hole	31344
2,5,3,8,6,1,7,4,hole	272413
6,5,8,7,4,3,2,1,hole	468453 (the most difficult)

Figures 2 and 3 show the number of clauses generated with paramodulation, which could be described as paramodulation complexity. In both graphs, X-axis is the number 8 puzzle. Y-axis is the number of generated clauses. For yielding Figures 2 and 3, we have generated 100 * 8 puzzles which are solvable as discussed in section4.2.We have observed a large variance in Figure 2.Besides, Figure 3 depicts the plot sorted by the number of clauses where the puzzle numbers are shuffled.

Curiously, in the sorted graph in Figure 3, the number of clauses is not increasing linearly.

Instead, the number of clauses generated with paramodulation is increased drastically around X-axis 22, 38, 62, 79, and 97.Consequently, we can conclude that there are several layers of complexity in Figure 3.Also, in each layer, the number of clauses generated is increasing linearly.

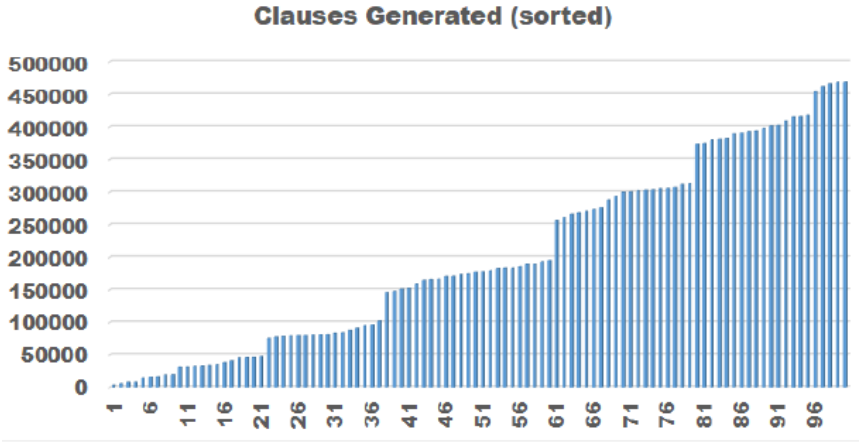

Figure 3. Clauses generated by paramodulation after sorting. X-axis is the number 8 puzzles. Y-axis is the number of generated clauses.

6. RELATED WORK

Archer [1] firstly discusses an algorithmic analysis of 15 puzzles. In [1], a summary of all possible permutations of slots attained by moving the black block from cell i to cell j affecting the permutation sigma[i][j].Howe [2] proposes two approaches in the two kinds of viewpoints: the properties of permutations and graph theory. Ariyanto [3] proposes the new sliding puzzle made with several additional rules from M13 puzzle. Calabro [4] proposes O(n^2) time algorithm for deciding the time when the initial tie configuration of the n * n puzzle game is solvable. Conrad [5] discusses 15 puzzles and Rubik cube as permutation puzzles. Bischoff [6] adopts reinforcement learning to solve 15-puzzle.Ando [14] applies hot list strategy [15] for faster paramodulation-based viral code detection. Takefuji proposes the application of paramodulation to the translator of Common Lisp [17]. Ando and Takefuji apply hotlist strategy based on paramodulation for faster graph coloring [16].

Paramodulation originated as the development of resolution [18], one of the main computational methods in first-order logic, see [19].For improving resolution-based methods, the study of the equality predicate has been particularly important since reasoning with equality is well-known to be of the great importance of mathematics, logic, and computer science. Dan Carson and Larry Wos developed a resolution-based theorem prover they called P1, which stands for ``Program 1". P1 is the founder of OTTER and includes basic strategy of OTTER including the set-of-support strategy [13], unit preference [20] and paramodulation.P1 is the first implementation of the theorem prover where Wos's invention of the paramodulation inference rule [21] is experimented.RW1, which stands for ``Robinson-Wos 1" which is the product from the collaboration of Wos and George Robinson.RW1 adopts the paramodulation inference rule, as well as demodulation. Also, RW1 is based on the concept by Knuth and Bendix, who independently formulated paramodulation and demodulation in the view of a complete set of reductions in their 1970 paper [22].

7. CONCLUSION

This paper proposes a novel method for providing a new perspective of paramodulation complexity by solving 100 sliding block puzzles (8 puzzles). Paramodulation is designed based on the concept of generalization of a substitution rule for equality. We have counted the number of clauses generated with paramodulation as the complexity of each sliding block puzzle, as shown in Algorithm 3.As a result, a wide range of complexity of 100 solvable eight puzzles has been measured. For example, board [2,3,6,1,7,8,5,4, hole] is the easiest with score 3008 and board [6,5,8,7,4,3,2,1, hole] is the most difficult with score 48653.

There have been many research efforts on the measurement and evaluation of the complexity of sliding block puzzles. However, the method for coping with the complexity of the puzzle in the aspect of the computation cost in automated reasoning has never been proposed. We have succeeded in figuring out more computational methods for comparing the difficulty of 100 * 8 puzzles with the help of automated reasoning. Besides, we have observed several layers of complexity (the number of clauses generated) in 100 puzzles, as shown in Figure 3.We can conclude that the proposed method can provide a new perspective of paramodulation complexity concerning sliding block puzzles.

For further work, we are aiming to apply this research for the hybrid of algorithmic modules providing formal reasoning, search and abstraction capabilities with geometric modules providing informal intuition and pattern-recognition capabilities. The output of Figure 3 in section 6 can enables us to make labelled image data of the initial state of sliding block. For example, there are 6 plateaus in the plots of Figure 3. Then we can generate 6 categories of images of sliding block puzzles. Deep learning algorithms such as convolutional neural networks can recognize these images with 6 labels as the level of difficulties. A necessary transformational development that we can expect in the field of machine learning is a move away from model that perform purely pattern recognition and can only achieve local generalization, toward models capable of abstraction and reasoning that can achieve extreme generalization.

REFERENCES

[1] Archer, A. F. A Modern Treatment of the 15 Puzzle. The American Mathematical Monthly 106, 793-799, 1999. Web.

[2] Gizem, Aksahya&Ayese, Ozcan (2009) *Coomunications& Networks*, Network Books, ABC Publishers.

[3] Tom Howe, Two Approaches to Analyzing the Permutations of the 15 Puzzle \\ https://www.whitman.edu/Documents/Academics/Mathematics/2017/

[4] Prihardono Ariyanto and Kenichi Kawagoe, ``New Sliding Puzzle with Neighbors Swap Motion",in Proc of International Symposium on Computational Science, Kanazawa, Japan February 2015
[5] Chris Calabro, (2005), Solving the 15-Puzzle.
[6] Keith Conrad, The 15-Puzzle (and Rubik Cube), on-line notes, 2016
[7] Bastian Bischoff, Duy Nguyen-Tuong, HeinerMarkert, Alois C. Knoll: Solving the 15-Puzzle Game Using Local Value-Iteration. SCAI 2013: 45-54
[8] John K. Slaney, Ewing L. Lusk, William McCune: SCOTT: Semantically Constrained Otter System Description. CADE 1994: 764-768
[9] William McCune: Skolem Functions and Equality in Automated Deduction. AAAI 1990: 246-251
[10] William McCune:Experiments with Discrimination-Tree Indexing and Path Indexing for Term Retrieval. J. Autom. Reason. 9(2): 147-167 (1992)
[11] Ross A.Overbeek, An implementation of hyper-resolution, Computers \& Mathematics with Applications Volume 1, Issue 2, June 1975, Pages 201-214
[12] Gizem, Aksahya&Ayese, Ozcan (2009) *Coomunications& Networks*, Network Books, ABC Publishers.
[13] Ewing L. Lusk, William McCune:Tutorial on High-Performance Automated Theorem Proving. CADE 1990: 681
[14] Ewing L. Lusk, William McCune, Ross A. Overbeek:ITP at Argonne National Laboratory. CADE 1986: 697-698
[15] Larry Wos, George A. Robinson, Daniel F. Carson: Efficiency and Completeness of the Set of Support Strategy in Theorem Proving. J. ACM 12(4): 536-541 (1965)
[16] Ruo Ando: Faster Parameter Detection of Polymorphic Viral Code Using Hot List Strategy. ICONIP (1) 2008: 555-562
[17] Larry Wos, Gail W. Pieper: The Hot List Strategy. J. Autom. Reason. 22(1): 1-44 (1999)
[18] Ruo Ando, YoshiyasuTakefuji, "Hot List Strategy for Faster Paramodulation based Graph Coloring", WSEAS TRANSACTIONS ON COMPUTERS, Issue 7, Volume 5, pp1596-1599, July 2006
[19] Y. Takefuji and M. Dowell, "A Novel Approach to a Rule-Based General Purpose Program Translator Using Paramodulation: Case Study of A Franz-To-Common Lisp Translator," Knowledge-Based Systems, 1, 2, 90-93, March 1988.
[20] J. A. Robinson. A machine-oriented logic based on the resolution principle. Journal of the Association for Computing Machinery, vol. 12 (1965), pp. 23-41.
[21] L. Bachmair, H. Ganzinger Resolution theorem provingvA. Robinson, A. Voronkov (Eds.), Handbook of Automated Reasoning, vol. I, Elsevier Science, Amsterdam, The Netherlands (2001), pp. 19-99
[22] Gizem, Aksahya&Ayese, Ozcan (2009) *Coomunications& Networks*, Network Books, ABC Publishers.