THE DEVELOPMENT OF PARALLEL RESOLUTION ALGORITHMS USING THE GRAPH REPRESENTATION

Andrey Averin, Vadim Vagin

Abstract. The parallel resolution procedures based on graph structures method are presented. OR-, AND- and DCDP- parallel inference on connection graph representation is explored and modifications to these algorithms using heuristic estimation are proposed. The principles for designing these heuristic functions are thoroughly discussed. The colored clause graphs resolution principle is presented. The comparison of efficiency (on the Steamroller problem) is carried out and the results are presented. The parallel unification algorithm used in the parallel inference procedure is briefly outlined in the final part of the paper.

Keywords: Automated Reasoning, Logical inference

ACM Classification Keywords: I.2.3 Artificial Intelligence: Deduction and Theorem Proving

1. Introduction

The deductive inference procedures are broadly used in variety of fields, such as expert systems, decision support systems, deductive databases and intelligent information systems. Due to high amount of data in practical problems and the exponential growth of the search space, the efficiency of deductive procedures becomes the key factor in the development of deductive inference systems. One of the ways to improve the efficiency is the parallelization of inference procedures. We present a parallel inference procedure based on resolution principle. The connection graph representation is chosen as the basis for designing the parallel resolution procedures. Using the graph representation simplifies the parallelization of the inference process and allows to apply the different parallelization techniques such as OR, AND and DCDP parallelism. We study and implement OR-, AND- and DCDP- parallel resolution procedures, develop useful heuristics which can be used in parallel resolution procedures on connection graphs and make a comparison of the obtained results with the results of algorithms developed by other researchers. Also we describe and implement the clause graphs inference procedure. As the test task, the Schubert's Steamroller problem is examined [1]. The problem of parallelism on the term level is also investigated. The data structure for the term representation and the parallel unification algorithm using this data structure are presented.

2. Connection Graph

The connection graph method was designed by R. Kowalski [2]. A connection graph is a scheme for representing the proper first-order formulas in disjunctive normal form. Each literal is associated with a node in the connection graph. Literals in a clause are combined into a group. If the literals in two clauses form a contrary pair (P and ¬P) then there is an edge between the respective nodes of the connection graph.

Example 1.

The initial set of clauses:

1.Q(c)
2. Q(b)
3.R(x) ∨ ¬Q(y) ∨ ¬P(x)
4.P(b)
5.¬R(x) ∨ S(x,y) ∨ ¬T(x)
6.T(y) ∨ ¬B(y)
7.B(a)
8.¬F(y) ∨ ¬S(y,z) ∨ ¬B(z)
9.B(x) ∨ ¬C(x) ∨ ¬D(y)
10.D(c)
11.F(b)
12.F(c)
13.C(b)

The corresponding connection graph is shown in fig. 1.

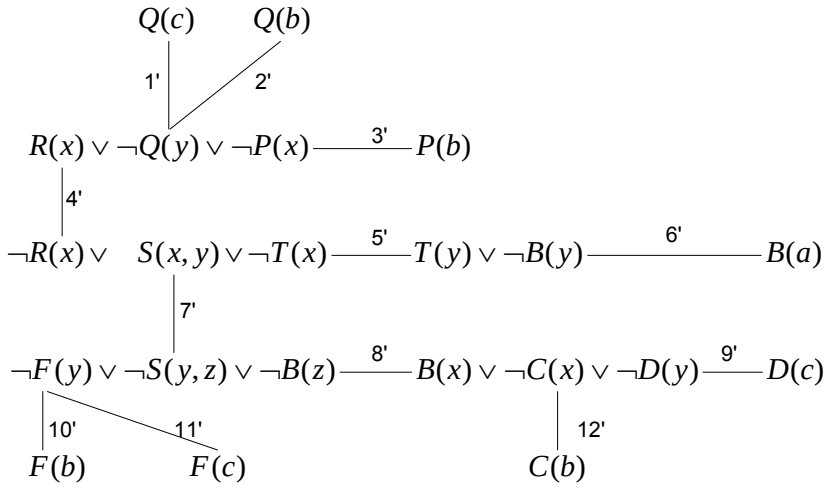

Fig.1

3. Methods of Inference on Connection Graphs

3.1 The Sequential Proof Procedure

To prove the unsatisfiability of a clause set we must generate and resolve the initial connection graph, i.e. derive an empty clause.

The main operation in connection graph refutation is the link resolution, when the resolvent is computed and added to the graph. The corresponding link is deleted and the links of the added resolvent are inserted. A pure clause is a literal group containing a node with no links. Pure clause with all its links can be easily removed from the graph without losing the completeness of the connection graph refutation procedure. Similarly, if we have a tautology clause, it also can be removed from a graph. If a resolvent on some step is a pure clause or a tautology, there is no need to insert this clause into a graph.

The refutation algorithm consists of the following steps:

1. the verification whether there is any clause in a graph or not. If there are no clauses, the algorithm terminates unsuccessfully. If there is the empty clause, then the algorithm is successfully terminated, else go to step 2;
2. if a graph does not contain any link, then the algorithm is unsuccessfully terminated, else go to step 3;
3. a link selection. The link is resolved and the resolvent is generated;
4. if an empty resolvent is obtained, then the algorithm terminates successfully, else the resolvent is inserted into the graph, its links are added, and the algorithm goes to step 2.

The fundamental problem in the connection graph refutation is the choice of suitable links by some criteria at each step of an algorithmic operation. Links are usually selected by using heuristics.

3.2 Parallel Inference on the Kowalski Connection Graph

The Kowalski connection graph can easily be used as the basis for designing parallel resolution algorithms [3]. Since the search space is complete, there is a possibility of using parallel computation strategies for enhancing the inference procedure efficiency. Parallel resolution algorithms differ from the sequential algorithm in step 3 at which a set of links satisfying certain criteria (not a single link as in the sequential procedure) is chosen and parallel resolution of all the links in this set is carried out.

3.2.1. OR-parallel Resolution on a Connection Graph

In case of OR-parallelism, the inference system associates some goal clause with the heads of clauses – possible candidates for resolution. Literals are unified and new clauses are generated. Admissible OR-links sets for the connection graph of Fig.1 are {1', 2'} and {10', 11'}.

3.2.2. DCDP-parallel Resolution on a Connection Graph

One modification of the parallel inference on the Kowalski connection graph is called DCDP parallelism (parallelism for distinct clauses) [4]. The correctness of the DCDP parallel resolution is proved in [5].

Definition 1. Clauses are said to be adjacent if there exist one or several links joining the literals of one clause with the literals of another clause.

Definition 2. A set of links joining pairs of distinct clauses is called a DCDP-link set if the clauses of every pair are not adjacent to any clause of other pairs.

To illustrate these definitions let us study the DCDP-link set for the connection graph of Fig.1. Adjacent pairs of clauses for this set are {(1), (3)}, {(2), (3)}, {(3), (4)}, etc... Thus, one of the DCDP-link sets is {1', 6', 9'}. Other examples of DCDP-link sets are: {2', 6', 12'}, {4', 12'}, {1', 6', 10'}.

3.2.3 AND-parallel Inference

Definition 3. A clause where all its links are resolved in parallel is called a SUN-clause.

Definition 4. Clauses joined with the literals of a SUN-clause are called the satellite clauses.

In AND-parallelism all links of literals of the SUN clause are resolved simultaneously. All resolvents are inserted in the graph along with all inherited links of a satellite clause. A SUN clause with all its links is removed. There is proved the correctness of the AND-parallel resolution [5]. The correct unification of separable variables under AND-parallel resolution is studied in [3].

Let us consider the choice of an AND-link set for the connection graph of fig.1. Admissible AND-link sets are, for example, {5', 4', 7'} (SUN-clause –clause (5)) and {5',6'} (SUN-clause – clause (6)). Detailed description of methods and algorithms can be found in [3,6].

4. Modification of Parallel Inference Procedures

Different heuristics can be used for choosing a link in resolving upon a connection graph. In the parallel resolution we must choose a set of links satisfying certain conditions. Note that the inference procedure becomes unsuitable if links are chosen unsuccessfully. The main principles underlying the design of heuristics are:

1. the number of literals in resolved clauses must be minimal,
2. the number of links in resolved clauses must be minimal,
3. the number of links in a literal for which the clauses are resolved must be minimal,
4. the unifiers of a resolved link must have a substitution of the type {*c*/*x*}, where *c* is a constant or a functional term, and *x* is a variable.

Further we describe the meaning of each principle in more detail.

Principle (1) simplifies a resulting connection graph, because a clause with a small number of literals, usually has a fewer number of links.

Principle (2) also simplifies a resulting connection graph, because the resolution of clauses with a small number of links yields clauses also with a small number of links.

Principle (3) prefers those links, the resolution of which yields "pure" clauses that on removing, can considerably simplify a connection graph.

The same is true for principle (4): as a result of the resolution by links containing a substitution of a variable for a constant, we obtain a clause containing constant terms. Such a clause has, at the first, a small number of links and, at the second, can be effectively used in the resolution. Principles (I)-(4) are taken into consideration in the heuristic function described below.

4.1. The Heuristic Function HI

In the heuristic function *H1* the link estimation is represented as a linear combination of estimations of the objects in the link (unifiers, clauses, and predicate literals in the link).

4.1.1 Computation of the Value of the Heuristic Function

Let *WeightLink* denote the heuristic estimation of a link. Then:

$$WeightLink = k_{clause} \cdot (Clause_{1Heur} + Clause_{2Heur}) + Uni_{Heur} \cdot k_{uni} + k_{pred} \cdot (Pred_{1Heur} + Pred_{2Heur}),$$

where $Clause_{1Heur}$, $Clause_{2Heur}$, Uni_{Heur}, $Pred_{1Heur}$ and $Pred_{2Heur}$ are the heuristic estimations for the first clause, the second clause, the link unifier, the predicate literal in the first clause of a link, and the predicate literal in the second clause of a link, respectively, and k_{uni}, k_{clause} and $k_{pred} \in [1; 100]$ are coefficients. Let us describe these symbols in more detail.

4.1.2 Heuristic Estimation of a Clause

The heuristic estimation must take into account the changes taking place in the characteristics of a connection graph during the inference (for example, changes in the number of links and the number of literals in a clause). Let us examine the heuristic estimation based on principles (1) and (2):

$$Clause_{Heur} = k_1 \cdot ClauseNumberOfLinks + k_2 \cdot ClauseNumberOfClauses,$$

where *ClauseNumberOfLinks* is the number of links in a clause, *ClauseNumberOfClauses* is the number of predicate literals in a clause; k_1, k_2 are arbitrary coefficients, chosen a priori on the basis of graph characteristics such as an average number of literals and links for the clause.

Let for example the average number of literals and links for the clause be 3.2 and 4.8, respectively. Then we can take $k_1 = 4.8/3.2 = 1.5$ and $k_2 = 1$. In this case, both principles (1) and (2) have the same weight. Characteristics may change their values during the inference. In this case, the initially chosen values of coefficients become "obsolete," and one principle gains a greater weight over the other. To avoid such a situation, the values of coefficients must be changed during the inference. Let *AverageLinkCount* and *AverageClauseLength* denote an average number of links and literals in a clause, respectively.

We can take $k_1 = AverageLinkCount/AverageClauseLength$ and $k_2 = 1$. In this case, principles (1) and (2) both gain the same weight in the course of the whole inference process. The heuristic clause estimation takes the final form:

$$Clause_{Heur} = (AverageLinkCount/AverageClauseLength) \times ClauseNumberOfLinks + ClauseNumberOfClauses.$$

4.1.3 Heuristic Estimation of the Unifier

The heuristic unifier estimation must be based on principle 4 (the unifier of a resolved link must have a substitution of the type *c/x*, where *c* is a constant or a functional term with constants and *x* is a variable) and must take into account the changes in values of the graph characteristics. The heuristic estimation of the unifier Uni_{Heur} is computed by the formula:

$$Uni_{heur} = AverageLinkCount/(1 + NumberOfConstantSubst + NumberOfFuncSubst),$$

where *NumberOfConstantsSubst* is the number of substitutions of the type *{c/x}* in the unifier, *c* is a constant term and *x* is a variable.

NumberOfFuncSubst is the number of substitutions of the type {*f/x*} m the unifier, where *f* is a functional term with constants and *x* is a variable.

4.1.4 Heuristic Estimation of a Predicate Literal

The heuristic estimation of a predicate literal must be based on principle (3) (the number of links in a literal, for which clauses are resolved upon must be minimal). The heuristic function must also take into account the changes in the graph characteristics. The heuristic estimation $Pred_{Heur}$ of a predicate literal is computed by the formula:

$$Pred_{Heur}=PredNumberOfLinks \cdot AverageClauseLength,$$

where *PredNumberOf Links* is the number of links in a predicate literal.

4.1.5 Selection of Coefficients

The coefficients k_{uni}, k_{clause} and k_{pred} are chosen experimentally. We have chosen the following ratio $k_{uni}=k_{pred}=(1/10)\cdot k_{clause}$ for coefficients. In this relationship, the greater weight is attached to principles (1) and (2). As a rule, k_{clause} is taken from the interval [10; 100] so that k_{uni} and k_{pred} lie in the interval [1; 10]. Thus, the heuristic function *H1* takes account of all principles (I)-(4). The weight of principles can be changed. The changes in the graph characteristics are also taken into account. The function *H1* enhances the efficiency of parallel inference algorithms for problems of practical complexity.

5. Deductive Inference on Clause Graphs

The deduction algorithm transforms a clause graph via two special operators – a predicate node elimination operator and a predicate node splitting operator [6,7]. They are applied to a predicate vertex depending on whether the node has multiarcs or not. A predicate node I said to be joined to a clause with multiarcs if the clause contains more than one literal with predicate symbol of the predicate node. For example, the node P in Fig.2 is joined to clause 1,2 and 3 (clause 4 is joined with the predicate node P with an ordinary arc) by the multiarcs.

1. $P(x,y) \rightarrow P(f(x),y)$ $(or \quad \neg P(x,y) \vee P(f(x),y))$
2. $P(u, f(x)) \,\&\, P(v, g(w)) \rightarrow$ $(or \quad \neg P(u,f(x)) \vee \neg P(v,g(w)))$
3. $\rightarrow P(g(x),y) \vee P(a,z)$ $(or \; P(g(x),y) \vee P(a,z))$
4. $\rightarrow P(a,x)$ $(or \quad P(a,x))$

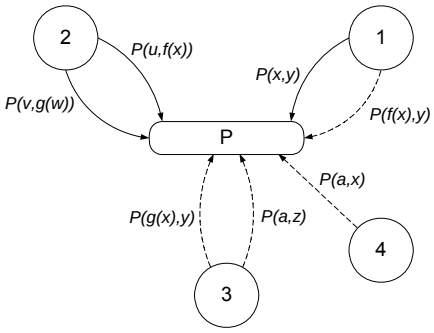

Fig. 2

In this method clauses are expressed in implicative form.

Here 1-4 are the nodes corresponding to the clauses of the logical model. The oval vertex represents the predicate symbol P. Continuous arcs are weighted with conditional literals of clauses, whereas dotted ars are weighted with inference literals of the clause.

An operation similar to colouring of clause graphs is introduced. Clause condition is «network is coloured» by color C1 (the corresponding arcs in figures are shown by a continuous line) and the condition for inference is colored by color C2 (a dotted line in figures). If clause graphs are represented in colored form, then it is easy to

search for information and new assertions can be inferred easily and thus the effectiveness of the inference system is enhanced.

The general inference scheme for an empty clause via transformation of clause graphs consists of the following:

1. if the network contains predicate nodes to which the node elimination operator can be applied, then such nodes are removed by this elimination operator;
2. if there are nodes with multiarcs, then the splitting operator is applied to generate nodes without multiarcs. Then the node elimination operator is applied, etc. until a contradiction is obtained in network.

If the node elimination operator consists of a set of usual operations of resolution of specially chosen pairs of clauses, then the splitting operator contains a distinct feature of this algorithm, i.e., a feature that has no analog in other logical inference mechanisms.

The predicate node elimination operator is applied to nodes having no multiarcs. It resolves in all possible ways those clauses that contain contrary pairs of literals. Resolution is implemented by the predicate of the chosen predicate node. Upon completion of all resolutions, parent clauses and the predicate node are removed from the network and the new clauses found via resolution are included in the network. Figures 3a and 3b show a clause graph before and after the application of the elimination operator to the vertex M for the following disjunct set:

$$1.\ Q \,\&\, M \rightarrow$$
$$2.\ M \,\&\, H \rightarrow Q$$
$$3.\ F \,\&\, M \rightarrow H$$
$$4.\ \rightarrow M$$
$$5.\ \rightarrow F$$

The node M and clauses (1)-(4) were removed and the clauses

$$6.\ Q \rightarrow . \quad (1,4)$$
$$7.\ F \rightarrow . H \quad (3,4)$$
$$8\ H \rightarrow Q \quad (2,4)$$

were added to the network.

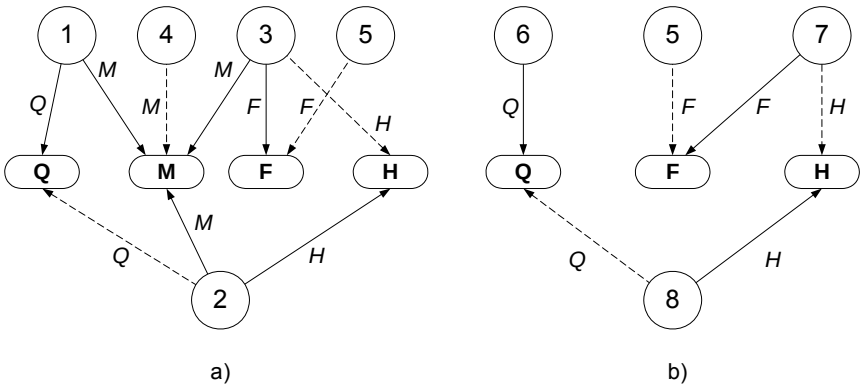

Fig. 3

The predicate splitting operator is introduced in order to remove multiarcs from nodes and thereby create conditions for the application of the elimination operator. The splitting operator generates copies of clauses and a few new vertices having the same predicate symbol as the split vertex, but with new indexes 1,2,3, … . The clause copies and new nodes are interrelated with each other. First, the elimination operator removes those nodes that have higher indexes.

Figure 4 shows the action of a splitting operator for the case of three multiarcs starting from clauses 1,2 and 3 (Fig. 2).

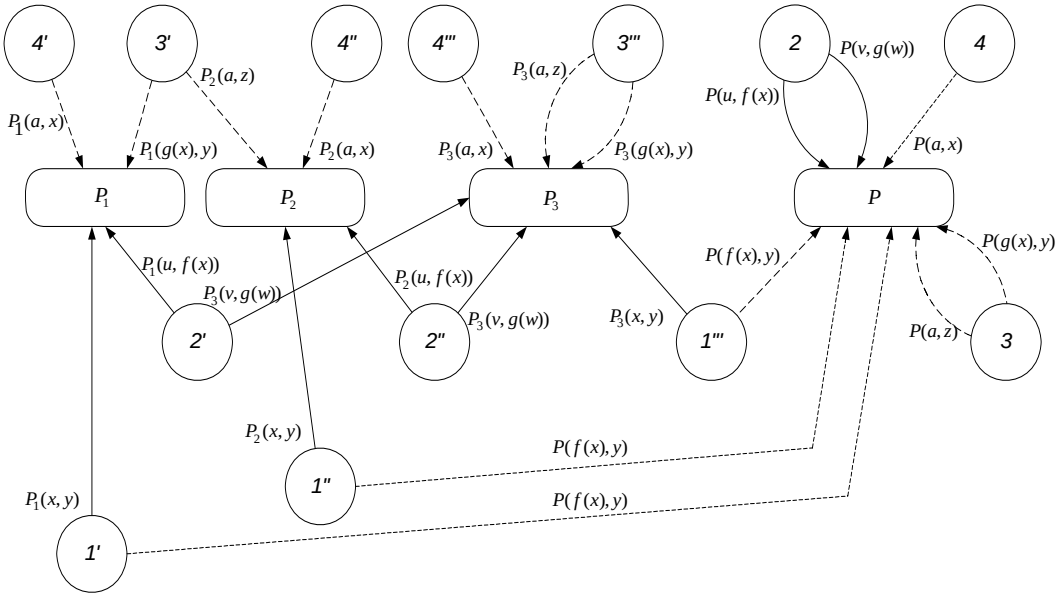

Fig.4

Splitting the three multiarcs ("petals") of the node P by this operator 1we obtain four nodes P_1, P_2, P_3 and P (their amount is one more than the amount of «petals»), four copies of clause 4, three copies of the «split» clause 1, two copies of the «split» clause 2 with a duplicate at vertex P, and a «split» clause 3 with a duplicate at nodes P and P_3. Splitting occurs in the following order: first the multiarc (1 – P), then the multiarc (2 – P), and finally the multiarc (3 – P) are split. The priority elimination strategies for removing nodes having no multiarcs and nodes with minimal number of arcs are reasonable for forming a sequence of processed nodes in a clause graph. They yield a sequence of eliminations and generate a lesser number of intermediate clauses than the variant in which these strategies are not used. Both these operations are correct i.e. unsatisfiable clauses remain unsatisfiable after the application of these operators[1]. A parallel algorithm for deductive inference on colored clause graphs is described in [7,8].

5. Efficiency

As the test problem, we have researched the "steamroller" problem formulated by L. Schubert in 1978 to test automatic proof systems [1]. This problem requires the generation of an exponential number of intermediate clauses and unifications during the inference process. Below we describe the results obtained by our and other algorithms of deductive inference for the «steamroller» problem.

procedures have also shown their efficiency for deductive inference on connection graphs.

CG is the inference strategy with a connection graph. SOS is the inference on a connection graph with a goal statement as a support set. TR is the inference within Theory Links - the extension of the standard resolution method. UR is inference with Unit Resolution - a modification of the resolution method. LUR is the inference by Linked Unit Resolution - a modification of the UR resolution method.

According to Figs. 5 and 6, the best results are obtained by the McCune/LUR procedures. Parallel inference

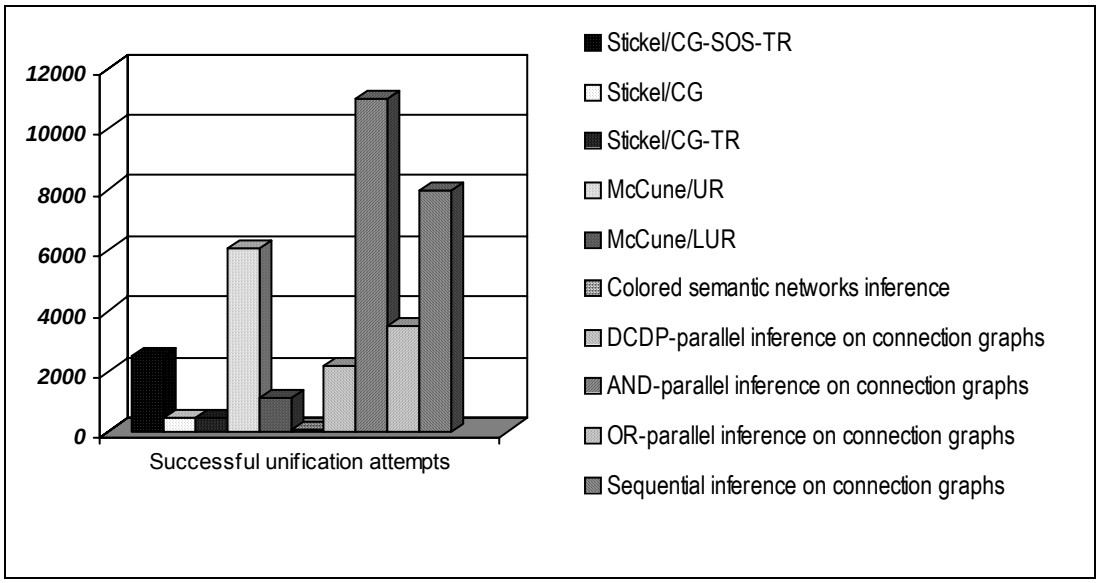

Fig. 5

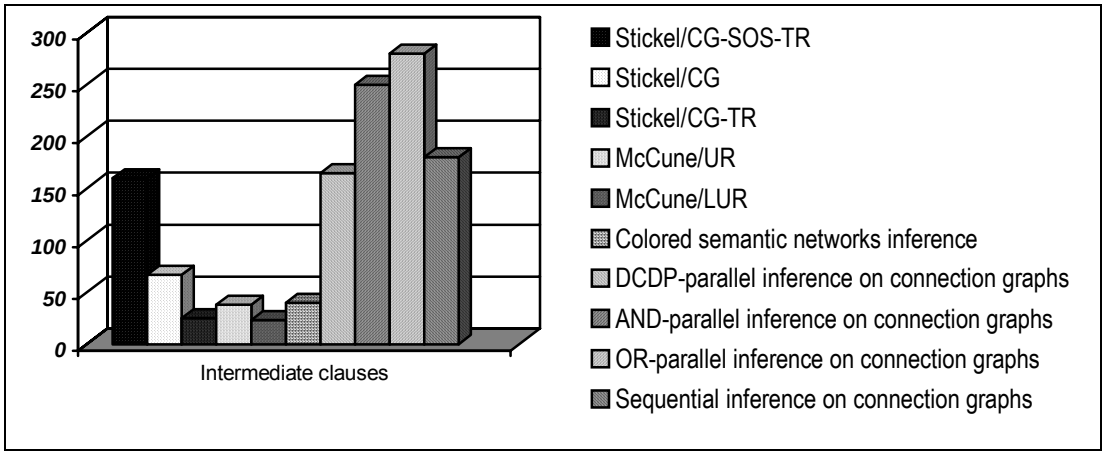

Fig. 6.

6. Parallel Unification in Connection Graph Inference

Due to a high amount of the unification tasks in deductive inference, the efficiency of the implementation of the unification procedure is one of the main factors in designing deductive inference procedures. There is a variety of effective unification procedures, but all of them have their own disadvantages. The main disadvantage is the use of complex, tightly coupled data structures, which are very difficult to parallelize.

We develop rather simple data structure for the term representation based on the notion of path strings. Terms (and clauses) are stored in tables, which are connected with links. Some tables and parts of the tables can be processed in parallel (with having links in mind). As we have no possibility to describe the algorithm and the data structure for term representation in detail, we just briefly outline the main principles used in this approach.

The term is stored as the sequence of strings, where every string presents one symbol in the term. Also such information as the type of a symbol (a variable or a constant symbol), the index number of an argument in a function symbol and the depth of a symbol (i.e. the number of function symbols which this symbol belongs to) is stored. The terms that can be unified are connected with the links of two types.

The first type corresponds to the terms (and strings) of different depth. The second type corresponds to the terms of the same depth. Let us illustrate these notions by the simple example.

Consider the unification task $\{t_1=t_2\}$, where t_1= *f(a,g(h(x)))* and t_2=*f(x,g(y))*. The representation of the term *f(a,g(h(x)))* is *f1.a.1.constant*(1) and *f2.g1.h1.x.3.variable* (2).

The representation of the term *f(x,g(y))* is *f1.x.1.variable*(3) and *f2.g1.x.2.variable*(4) (the index number is shown in brackets). The links are created between the strings (1) and (3) and between the strings (2) and (4). The first link has the second type and the second link has the first type.

The task of link establishing is one of the main tasks in the proposed approach. Two techniques can be used. The simplest one is the special sorting procedure on tables containing strings from the unification task. The main drawback of this approach is the deficit of efficiency caused by the nature of the sorting problem (the complexity of the sorting problem is *nlog(n)*). The second approach is based on automata representation of strings and is similar to discrimination trees. Using automata increases the efficiency of the procedure, but requires higher memory consumption.

The main idea lying in the parallelization of the unification procedure is the use of dependencies graph, where strings connected with arcs cannot be proceeded in parallel. The complexity of the task of determining maximum sets for parallel proceeding is equal to the complexity of the graph coloring task, though heuristics can be used to find non-maximum but satisfactory sets. This unification procedure has been combined with the parallel inference procedure on connection graphs. The structure for term representation and unification are thoroughly described in [9].

7. Conclusions

The Kowalski connection graph and the sequential inference procedure on connection graphs are investigated. The structure of the OR-, AND- and DCDP-parallel inference methods is described. Methods of modifying parallel inference procedures are analyzed and the main principles underlying the design of heuristic link estimations are stated. The heuristic function for link selection is designed. A procedure of inference on colored clause graphs is also described. The developed deductive inference procedures are compared with other procedures on the "steamroller" problem. The procedures of parallel inference on connection graphs and clause graphs are the effective ones for the deductive inference. They can be applied in expert and decision making systems.

Bibliography

[1] M.F. Stickel, Schubert's Steamroller Problem: Formulations and Solutions // *J. Autom. Reasoning,* 1986, vol. 2, pp. 89-100.

[2] R. Kowalski, A Proof Procedure using Connection Graphs // J. ACM, 1975, 22(4), pp. 595-599.

[3] V.N. Vagin and N.O. Salapina, A System of Parallel Inference with the Use of a Connection Graph // *Journal of Computer and System Sciences International,* Vol. 37, No. 5, 1998, pp. 699-708.

[4] G. Hornung, A. Knapp and U. Knapp, A Parallel Connection Graph Proof Procedure // *German Workshop on Artificial Intelligence. Lecture Notes in Computer Science,* Berlin: Springer-Verlag, 1981. pp. 160-167.

[5] R. Loganantharaj, Theoretical and Implemetational Aspects of Parallel Link Resolution in Connection Graphs, *Ph.D. Thesis,* Dept. of Computer Science, Colorado State Univ., 1985.

[6] Vagin V.N. Deduktsiya i obobshcenie v sistemakh prinyatia reshenii (Deduction and Generalization in Decision Making Systems), Moscow: Nauka, 1988.

[7] Vagin V.N., Parallel Inference on Logical Networks, IFIP/WG 12.3 International Workshop on Automated Reasoning, Beijing, China, 1992, pp. 305-310.

[8] A.I. Averin, V.N. Vagin and M.K. Khamidulov, The Investigation of Algorithms of Parallel Inference by the Steamroller Problem // *Journal of Computer and System Sciences International,* Vol. 39, No. 5, 2000, pp. 758-766.

[9] A.I. Averin, V.N. Vagin, Using Parallelism in Deductive Inference, // Journal of Computer and System Sciences International. -2004. - vol. 45, №4. – pp.603-615.

[10] Vagin V.N., Golovina E.Y., Zagoryanzkaya A.A., Fomina M.V.. «Dostoverniy i pravdopodobnyi vivod v intellektualnykh systemakh». (Relaible and Plausible Inference in Intellectual Systems) – Moscow: Fizmatlit. - 2004.

Authors' Information

Andrey Averin – Moscow Power Engineering Institute, Krasnokasarmennaya str., 14, Moscow, Russia; e-mail: averin@rbcmail.ru

Vadim Vagin – Moscow Power Engineering Institute, Krasnokasarmennaya str., 14, Moscow, Russia; e-mail: vagin@apmsun.mpei.ac.ru