Intelligent Backtracking for Automated Deduction in FOL

Stanisław Matwin
Department of Computer Science, University of Ottawa, Ontario, Canada

Tomasz Pietrzykowski
School of Computer Science, Acadia University, Nova Scotia, Canada

Abstract

An "intelligent" backtracking algorithm for depth-first search of the solution space generated during linear resolution in fol has been designed. It inspects only a small portion of the total solution space, which consists of special graphs representing the deductive structure of the proof. These graphs are generalization of AND/OR trees. Our (partially) complete search algorithm has natural potential for parallel implementation. However, it may generate redundant refutations; it seems that this is the effect of the prevailing design objective, which in our case was completeness of the method.
A preliminary estimate of the efficiency of the algorithm has been carried out. It indicates exponential speed-up over the worst case of linear backtracking.
An implementation (3000 lines of PASCAL code, under CMS) has now been completed. That allows us to experiment with the algorithm and investigate certain open questions.

1. Introduction

Many researchers working in Artificial Intelligence and its applications agree that an efficient backtracking mechanism will drastically expand applicability of Logic Programming ([Warren et al 77], [Pereira and Porto 80], [Nau 82], [Stallman and Sussman 77]). One such algorithm has been designed by M. Bruynooghe [78] and L.M. Pereira [79], [80]. This paper presents an alternative and different approach. Our method is based on a graph-based, depth-first proof procedure [Cox and Pietrzykowski 81]. The basic notion, on which this algorithm is based, is the plan: a directed graph, representing deductive structure of the proof. The plan is a natural generalization of AND/OR trees. The unifications, generated during the proof, are kept in a separate graph structure, called the graph of constraints. In this way, even if backtracking along a particular path of the plan does not lead to a solution and this path will have to be re-generated, there is no need to re-generate the unifications obtained along that path.
Furthermore, our method is applicable to general first order logic, without being restricted to Horn clauses. Also, as it will be demonstrated later, intelligent plan-based deduction has natural potential for a parallel implementation.
Finally, it has to be emphasized that the prevailing design criteria of our algorithm was completeness of search of the search-space. This has been achieved, and the proof of (partial) completeness has been obtained [Matwin and Pietrzykowski 83]. However, a price which had to be paid is redundancy (i.e. the same solution may be obtained more than once). Bruynooghe-Pereira method does not suffer from this deficiency, but then it is not certain that their solution is a complete one.

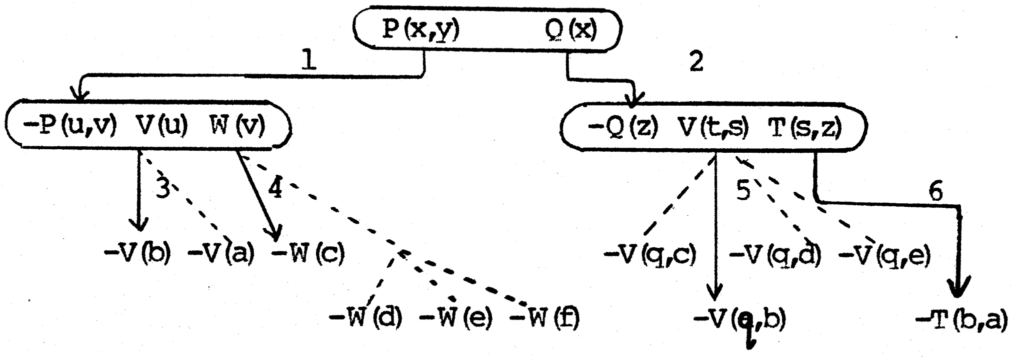
MATO21

. Operation of the Intelligent Backtracking System.

Before introducing the algorithm and a more complete example, let us illustrate he difference between "exhaustive" and "intelligent" backtracking using a very imple case. Assume that the following set of clauses is given:

```
   P(x) Q(x).                     -Q(z) S(t,s) T(s,z).
 -P(u,v) V(u) W(v).              -S(q,b).     -S(q,c)
 -V(b).     -V(a).               -S(q,d).     -S(q,e)
 -W(c).     -W(d).                -T(b,a).
 -W(e).     -W(f).
```

learly, with the left-to-right "reduction" (although "expansion" seems to be more dequate term) policy, the following plan, which in this case is just and AND/OR ree, is obtained:



he continuous lines represent the AND arcs, the dotted are the OR arcs. bviously, in this tree there is a clash between constant b, generated by arc 3, and onstant a, generated by arc 6. One look at the plan convinces us that arc 3 is e culprit, and that a reduction following its alternative remedies the problem. owever, exhaustive backtracking will perform 33 reductions [3*2 + 3*((4*2) + 1) = 3] before generating the solution. The reason for that is the fact that all the lternatives between the arcs 3 and 6 involved in the conflict are tried by xhaustive bactracking. Our method is different: it only tries 6, 3 and the lternatives lying above them. In this case, one reduction replacing 3 with its lternative will do the job. In a reasonably balanced AND/OR tree, the number of lternative deductions obtainable in between two modes is of the order exponential rt the height of the tree. Therefore, a method which operates only above the lashes will be exponentially faster than the worst-case behaviour of exhaustive acktracking discussed here.

We shall now proceed with a more thorough discussion of our method, beginning ith the underlying notions and concepts.
The basic structure, involved in the algorithm is the plan. By a plan we nderstand a directed graph, nodes of which represent variants of clauses. One of e nodes, referred to as TOP, represents the clause to be proven. Arcs of the an connect pairs of literals, belonging to individual nodes. Each two literals, efining ann arc, are unifiable and of opposite sign. There are two types of arcs: UB arcs and RED arcs (as proven in [Cox and Pietrzykowski 81], those two rules rovide a complete set). Informally speaking, SUB arcs point "downwards" in the an, while RED arcs point "upwards". Each node, except the TOP, is entered by xactly one SUB arc (and, possibly, by zero or more RED arcs). The literal within

a node, pointed to by a SUB arc, is the key of this node. Each other literal of this node is called a goal. A goal is called closed if there is an arc, originating in this goal, otherwise the goal is an open one.
With each goal of the plan we associate a set of arcs, called the set of potentials. They are the arcs which could have been generated instead of the one actually created. Let us notice that, if the plan is a tree, then the initial value of all potentials represents all the OR arcs. In any case, this initial value is static information.
As mentioned before, the information gathered as result of unification is kept separately, in a special data structure called the graph of constraints. This graph reflects the history of unifications which have taken place in the proof during its progress. A node of the graph of constraints, called a constraint, represents the information about the bindings which have ben imposed on a variable during the history of proof. Therefore, presence of two different constants in a constraint is an indication of a clash. This clash is then mapped on the plan. Each minimal set of plan arcs such that its removal annihilates the clash is referred to as a conflict. The conflict set is the set of all such conflicts for a given plan. In some situations, even though the conflict set is empty, we want to create an artificial conflict set, in order to assure completeness. Artificial conflict set contains all the arcs entering unit clauses, and all the reduction arcs.

Finally, our method introduces two other notions, motivated by memory management problems. The algorithm uses a repository of plans, accompanied by their graphs of constraints and conflict sets. This repository, called the store resides on disk, and plans are fetched from it and added to it. There is always one plan being operated on: it is called the table plan (or simply the table).

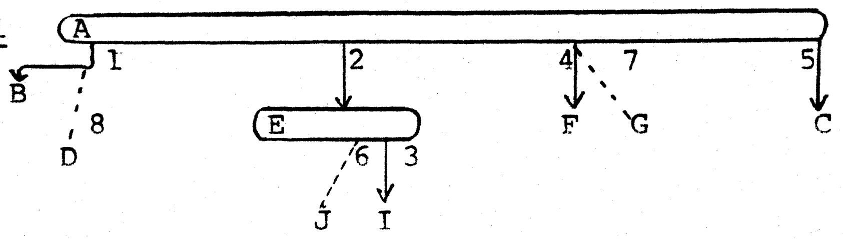
With this background, we can now follow the operation of our algorithm on the following set of clauses:

U?

A. P(x) Q(y) R(x,y) ~~V~~(a,x) = TOP
B. -P(a)
/U? C. -~~V~~(t,y)
D. -P(c)
E. -Q(w) V(v,w)
F. -R(z,z)
G. -R(u,v) S(u)
H. -S(a)
I. -V(b,b)
J. -V(c,c)

Initially, the store is empty. Clause A is chosen as the TOP and a single-node plan consisting of A is generated. Since it is not closed, it will be further developed until either a closed plan is obtained or a non-empty conflict set is generated. In our case, we get the following plan:

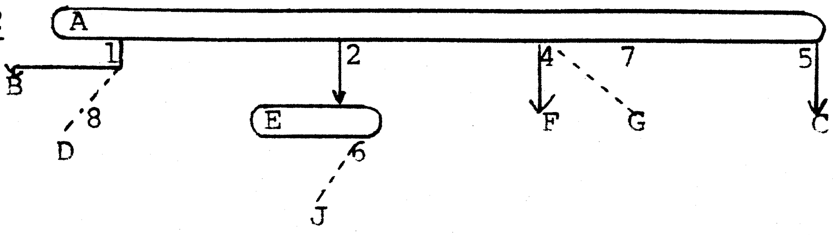
plan P1



[illegible]

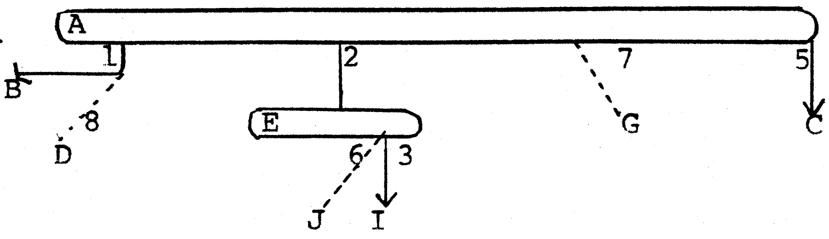
s conflict set is (3, 1∧5; 4). Suppose that 3 is chosen for removal; the open lan P2 is obtained and placed in the store:

lan P2



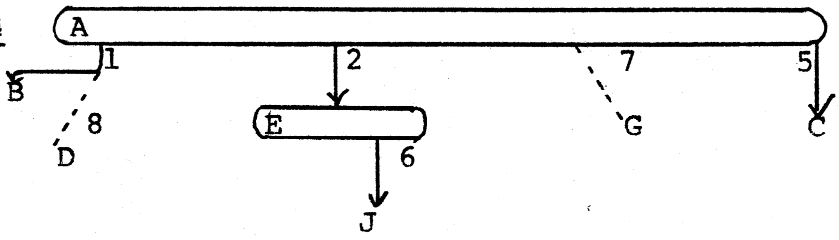
he conflict on the table is now (1∧5, 4). If 1∧5 is chosen, prunning annihilates e plan, as 5 has no potential and A is the TOP. With the choice of 4, open plan 3 is obtained and placed in store:

lan P3



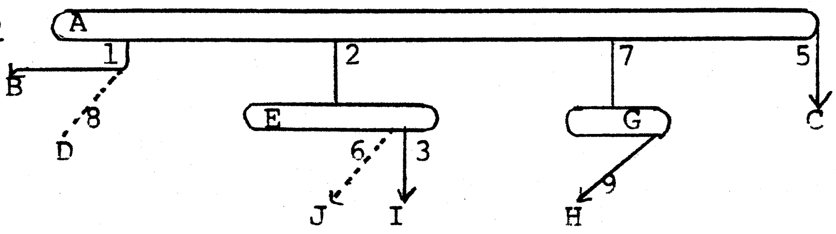
ince there are no more conflicts on the table, one of the store plans (suppose it ; P2) is placed on the table. Potential 6 is realized as an arc, which leads to a onflict set (1∧5, 6, 4) on the table. Since choice of either 6 or 1∧5 leads owhere, suppose that 4 is chosen and P4 is sent to store.

lan P4



s the conflict set is again empty, one of the store plans P4 and P3 is placed on e table. Assume P3 is chosen; the only open goal is closed with its potential 7 nd a solution is obtained:

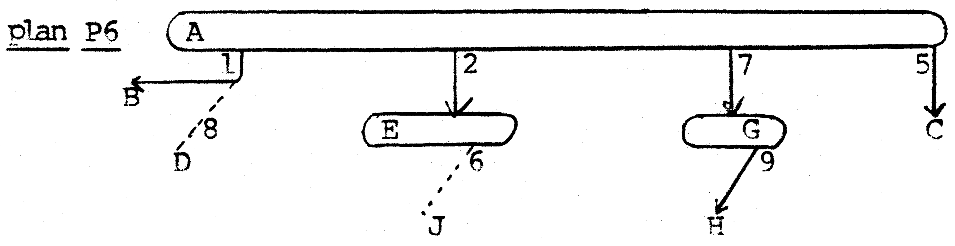
lan P5



rtificial conflict set of P5 is (1, 3, 5, 9). Removal of 5 and 9 leads nowhere as ere are no potentials between these arcs and the top. Replacement of 1 by 8 is lso unproductive (the reader will easily see why). This leaves us with 3 as a easonable candidate for removal, which results in plan P6.

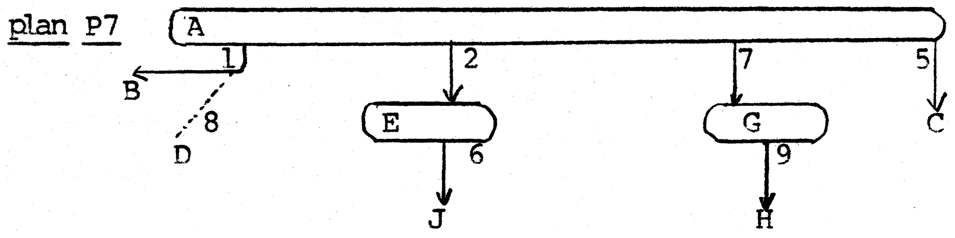
HAYLOIN-4

plan P6



The only plans in store are now P4 and P6. With the choice of P6, potential J is realized and we get solution P7:

plan P7



Since all attempts of obtaining new plans from its artificial conflict set fail, the only remaining store plan, P4, is placed on the table. Its potential 7 and then arc 9 are realized, which gives a redundant solution identical to P7. The store is now empty and the algorithm terminates.

We have proven elsewhere [Matwin, Pietrzykowski 83] that when our algorithm terminates, it generates all the existing proofs (partial completeness).

3. Further Inhancements of the Algorithm

There are at least three directions of further research, leading to potentially interesting enhancements of the algorithm.

1. Different strategies for nondeterminism. A number of nondeterministic choices is involved in the algorithm. Two types of such choices were mentioned in the brief description in the preceding section: choice of the plan from the store to be placed on the table, and choice of a conflict from the set of conflicts. It is not clear, at this stage, what are the right criteria for these choices. This is particularly important when the objective is to find a proof, rather than all the possible proofs. It seems that in this case the right strategy may bring about significant increase in efficiency.

2. Applicability of the algorithm in the domain of expert systems. The researchers in expert systems point out that a method of limiting the search space is of great importance for implementation of practical systems [Nau 83]. The early work of [Stallman and Sussman 77] bears a good deal of resemblance to our method, although their approach is less general. System ARS, reported in [Stallman and Sussman 77] implements a method of dependency directed backtracking, tailored to the particular environment of algebraic relationships encountered in the analysis of electric circuits. Therefore it seems that a method like ours may be productive, particularly in case of expert systems using fol or its derivatives [Skuce 83] to represent knowledge bases.

3. Distributed implementation. Since no ordering of conflicts in within the

Matw.

conflict set is assumed, an interesting parallel implementation seems possible. It will involve a number of processors, each of which would remove a conflict, carry out the necessary pruming (if any) and develop the plan. The result of development is placed in store, ready to be picked up by another processor. The whole system stops when the store becomes empty. Such a parallel, distributed implementation seems to be feasible. Let us notice that the similar approach to Bruynooghe-Pereira method would not work, since their algorithm specifically orders the conflicts, which in turn allows them to avoid the redundancy problem.

eferences

3ruynooghe 78] Bruynooghe, M., Intelligent Backtracking for an Interpreter of Horn Clause Logic Programs, Procs. of Colloquim on Mathematical Logic in Programming, Salgotarjan, Hungary, 1978.

3ruynooghe and Perreira 81] Bruynooghe, M., Pereira, L.M., revision of Top-Down Logical Reasoning Through Intelligent Backtracking, res. Report of KUL and CIUNL, 1981.

:ox and Pietrzykowski 81] Cox, P., Pietrzykowski, T., Deduction Plans: A Basis for Intelligent Backtracking, IEEE PAMI, Jan. 1981.

1atwin and Pietrzykowski 82] Matwin, S., Pietrzykowski, T., Exponential Improvement of Exhaustive Backtracking: Data Structure and Implementation, Procs. of CADE-6, 1982.

1atwin and Pietrzykowski 83] Matwin, S., Pietrzykowski, T., Intelligent Backtracking in Plan-Based Deduction, Submitted to IEEE PAMI.

Jau 83] Nau, D.S., Expert Computer Systems, IEEE Computer, Feb. 1983.

'ereira 79] Pereira, L.M., Backtracking Intelligently in AND/OR Trees, Research Report, CIUNL 1979.

ereira and Porto 80] Pereira, L.M., Porto, A., Selective Backtracking for Logic Programs, Procs. of CADE-5, 1980.

ietrzykowski and Matwin 82] Pietrzykowski, T., Matwin, S., Exponential Improvement of Exhaustive Backtracking: A Strategy for Plan-Based Deduction, Proc. of CADE-6, 1982.

kuce, 83] Skuce, D., KNOWLOG, Submitted to IEEE Computer.

tallman, R.M. and Sussman 77] Stallman, R.M., Sussman, G.J., Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-aided Circuit Analysis, Artificial Intelligence, 1977.

'arren et al 1977] Warren, D.H.D., Pereira, L.M. Pereira, F.,. PROLOG - The language and its implementation compared to LISP, ACM SIGPLAN, Aug. 1977.

[illegible]