STUDIA UNIV. "BABEŞ–BOLYAI", INFORMATICA, Volume **XLIV**, Number 2, 1999

AN EVOLUTIONARY ALGORITHM FOR THEOREM PROVING IN PROPOSITIONAL LOGIC

D. DUMITRESCU AND M. OLTEAN

Abstract. Standard theorem proving algorithms normally have exponential complexity. This drawback could be eliminated within the Evolutionary computation framework. In this paper an evolutionary theorem proving method is proposed. The method is designed for propositional logic but may be extended to first order logic.

The proposed evolutionary approach represents a new paradigm of automated theorem proving. Method complexity is polynomial. Reducing time complexity with respect to non-evolutionary methods is an important feature of the proposed approach.

A population of theorems is evoluated using two search operators: recombination and mutation. Recombination operator implements Modus Ponens inference rule. Mutation corresponds to the substitution operation.

1. Introduction

It is well known (see [3, 4, 5]) that the problem of deciding a given formula in propositional logic is (or not) a theorem in an NP problem. As we do not know if there exists a polynomial algorithm for solving NP problems they are considered as hard problems.

Many algorithms for automated theorem proving have been proposed. Robinsons resolution method ([7]) is one of the most powerful methods. However, resolution has the great disadvantage of being NP complete. This means that on a standard (sequential) computer the algorithms run a time that is bound by an exponential function of the input sequence length.

Evolutionary Algorithms (EAs) (see [1, 2]) are useful tools for solving complex optimization and search problems. EAs can also be successfully used to solve NP problems.

In this paper we propose an evolutionary algorithm, called ETP, for automated theorem proving within the propositional logic.

A model of propositional calculus consisting from three axioms and two inference rules is considered. The inference rules of the model are Modus Ponens (MP) and substitution. Our goal is to determine if a given well-formed formula R is (or it is not) a theorem. To solve this problem the ETP algorithm will be used.

A substitution operation by which every variable from an axiom is replaced by a well-formed formula is used to obtain theorems from axioms. The replacing formulae in a substitution are generally different. In what follows by a *substitution* we designate the formulae used for replacement in an axiom or in a theorem.

2. Evolutionary model for theorem proving

Within a generation t a population $S(t)$ of substitutions is used to obtain theorems from the system axioms. These substitutions may remain constant during the search process or may be evoluated by an evolutionary procedure. In the first case we have:

$$S(t) = S, t = 1, 2, ...,$$

where t is the generation index.

Each substitution population $S(t)$ generates a theorem population $T(t)$.

Theorems from $T(t)$ will be modified using variation operators. In our model considered variation operators are recombination and substitution. Recombination operator corresponds to the application of Modus Ponens rule and substitution plays the role of mutation operator.

Using MP from two parent theorems an offspring is obtained. As usual parent recombination is guided by a fitness function.

For a given theorem T all the possible mating candidates are established. A candidate partner TP of T has to have the form

$$TP = T \rightarrow A,$$

where A is a well formed formula.

Let us note that the Modus Ponens rule can be applied only for pairs of theorems having a particular form.

Tournament selection, or other selection procedures may be used to find the mating partner of T. Let T' be the tournament winner. From T and T' an offspring is obtained applying MP inference rule. Parent T' is considered to be dominant.

Within survival dominant parent is compared with its offspring. The winner will enter the new generation.

3. Evolutionary algorithm

Let R be a formula representing a target (or tentative) theorem. Our aim is to decide if R is a theorem or it is not.

Evolutionary theorem proving (ETP) algorithm starts with an arbitrary population of substitutions. At the first step we randomly make substitutions in the axioms. A *substitution* operator realizes this operation. Performing substitution operation an initial population of theorems is obtained.

In the next phase the theorem population is evolved. New theorems are obtained using MP inference rule. The obtained theorems are modified via a mutation operator (i.e. by performing substitutions in theorems).

As MP rule implies two theorems, we can assimilate MP inference rule with recombination operator. Theorems obtained by recombination are subsequently modified by the effect of substitution operator.

After a number of generations the algorithm stops with the answer "Yes" or "No". Yes means that that the target formula R exists in the last theorem population. Therefore R is a theorem. No means either that the formula is not a theorem, or our algorithm fails to determine it. The algorithm may fail to prove either due to a bad chosen population of substitutions, or due to an incomplete exploration of solution space by search operators.

If the algorithm result is Yes then the deduction chain representing the proof of theorem R may be established.

When the output of the procedure is No then we may restart the algorithm trying to prove the theorem T, where

$$T = \sim R.$$

If the new output is Yes then definitively R is not a theorem. Otherwise we can not make a decision about the truth value of the assertion R is a theorem.

4. Individual representation

Each well-formed formula of propositional logic can be represented as a tree. The non-terminal nodes contain logical connectives and the terminal ones contain the propositional variables. Each node has maximum two descendants. If the node contains the connective "$\rightarrow$" then the node has two descendents. If the node contains the negation connective "$\sim$" then it has to have just one descendent. If a node contains a propositional variable, then it is a leaf and it has no descendents at all.

In our approach axioms, substitutions and theorems will be represented as trees. Therefore the search space for our problem is the set of trees describing well-formed formulae. This representation is similar to that used in Genetic Programming (see [6]).

To limit the dimension of the search space we have to reduce the tree depth and the number of propositional variables used in substitutions. However, the number of the propositional variables in a substitution must be higher than the number of propositional variables in the target theorem R.

Denote by $h(R)$ the target theorem height.

As by recombination the height of the trees decrease, the height of a substitution tree must be no less than $h(R) - 1$. Therefore we have

$$h(s) + 2 \geq h(R),$$

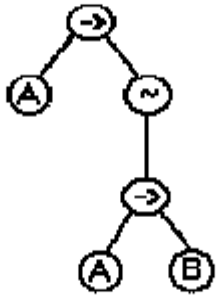

Figure 1 Tree for formula $(A \rightarrow\sim (B \rightarrow A))$

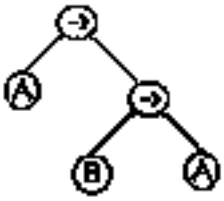

Figure 2 Tree for axiom A1 $A \rightarrow (B \rightarrow A)$

for each substitution s. (We added two because the axioms have the height one or two.)

Example. Let us consider the formula $F = (A \rightarrow\sim (B \rightarrow A))$. The tree corresponding to formula F is depicted in Figure 1.

Let us denote by A the system axioms. We may interpret each axiom as a potential solution of the problem (an individual in the search space).

The system axioms are:

A1:: $A \rightarrow (B \rightarrow A)$.
A2:: $(A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))$.
A3:: $(A \rightarrow B) \rightarrow (\sim B \rightarrow\sim A)$.

The three system axioms are represented by the trees as depicted in Figures 2-4.

We may assume the set A contains three particular candidate solutions (or individuals) corresponding to the three axioms from propositional logic. We may consider A as a static solution population.

Moreover two different evolving populations S and T will be considered where:

- S is a *substitution population.* It contains some randomly generated formulae.

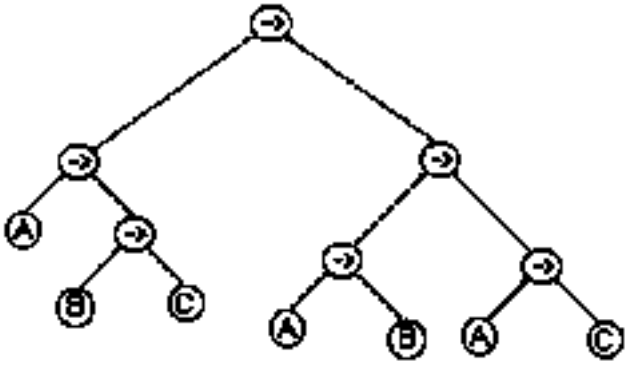

Figure 3 Tree for axiom A2 $(A \to (B \to C)) \to ((A \to B) \to (A \to C)))$

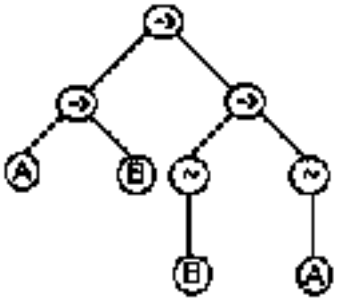

Figure 4 Tree for axiom A3 $(A \to B) \to (\sim B \to \sim A)$

- T is a *theorem population.* Each individual in this population is a theorem. Each population member will be obtained by a substitution or by a crossover operation performed on another two theorems.

Remark. By an axiom, substitution or theorem we will generally mean the corresponding trees representation.

5. Selection and Search Operators

We use two search (variation) genetic operators: MP recombination and substitution. Let us observe that the substitution operator may be interpreted as a special, problem-dependent type of mutation.

5.1. **Selection.** Each member of theorem population is selected for recombination. Each individual x in the theorem population is considered as a *recessive* parent. The corresponding *dominant* parent is chosen from the possible candidates using a tournament selection procedure.

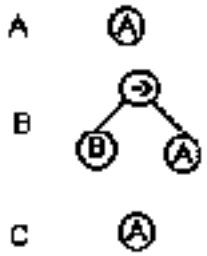

Figure 5 Substitution S1

5.2. **MP Recombination Operator.** MP recombination operator implements the Modus Ponens rule. We recall that Modus Ponens may be expressed as

$$A, A \to B \vdash B$$

Let us consider two theorems A and $A- > B$ represented as trees. Performing crossover (which corresponds to Modus Ponens rule) the offspring B is obtained. Of course B it is also represented as a tree.

From a biological point of view we may consider the first parent (theorem A) as the recessive parent. This interpretation is justified because no part of the first parent is present in the offspring. The second parent (namely $A \to B$) is the dominant one. The offspring is a part of its dominant parent.

5.3. **Substitution Operator.** Substitution (or mutation) operator implements the substitution inference rule. Substitution combines an individual from the substitution population with an individual from the axiom set or from the theorem population. The obtained offspring is a theorem and it may be added to the theorem population.

5.4. **Survival.** Several survival mechanisms may be considered (see [1]). Some survival strategies are generational and some are steady-state. In this paper we consider a steady-state survival mechanism. Each offspring is compared with its dominant parent. The best from the parent and offspring will become a member of the new population.

6. Example

Let us consider a substitution S_1 where propositional variables are replaced as follows:

- A: is replaced by the formula A.
- B: is replaced by the formula $B \to A$.
- C: is replaced by A.

Figure 6 Substitution S2

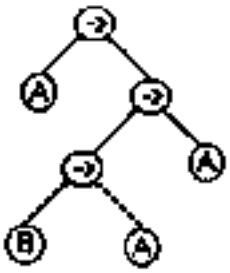

Figure 7 Tree of theorem T1

The trees describing substitution S_1 are represented in Figure 5.

Consider also a substitution S_2 specified as follows:

- A: is replaced by the formula A.
- B: is replaced by the formula B
- C: is replaced by A.

The trees describing substitution S_2 are depicted in Figure 6.

Let us consider the target theorem R is

$$R : A \to A.$$

Our aim is to prove the target theorem R using the substitution S_1 and S_2 only. In this respect S_1 and S_2 will be applied to the axioms.

By using substitution S_1 in axiom A we obtain the theorem T_1. The tree representing this theorem is depicted in Figure 7.

By using S_1 and A_2 we obtain a theorem T_2. The corresponding tree is depicted in Figure 8.

By using substitution S_2 in axiom A_1 we obtain a theorem T_3. The corresponding tree is depicted in Figure 9.

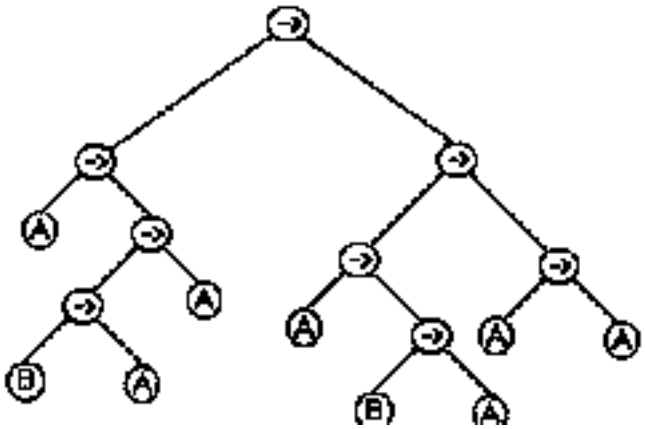

Figure 8 Tree of theorem T2

Figure 9 Tree of theorem T3

MP recombination of theorems T_2 and T_1 produces the offspring theorem T_4. The process is depicted in Figure 10.

By MP recombination of theorems T_3 and T_4 we obtain the target theorem R. The corresponding recombination process is depicted in Figure 11.

Therefore we have obtained a complete proof of theorem R.

7. Fitness Function

From the previous example we may observe that the target theorem R can be proved if and only if in the theorem population arises an individual T such that R may be reached from the root of the tree T by following the right sub-trees of T only.

Fitness of a theorem T may be defined by using the tree representing this theorem.

We may define the *handle* $H(T)$ of a theorem T by the distance from the root node (of the tree representing T) to the right sub-tree representing the target theorem R. Our aim is to minimize the theorem handle.

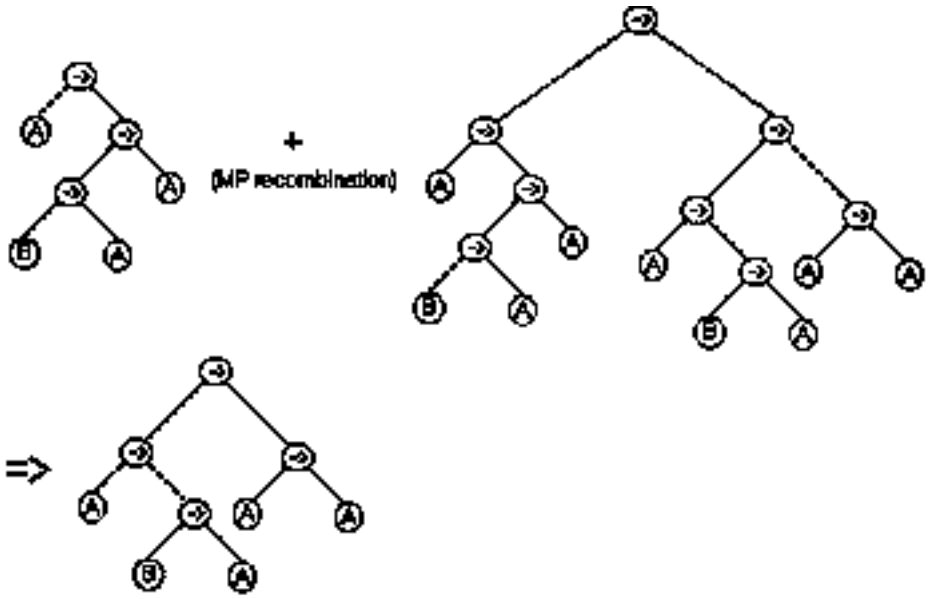

Figure 10 MP recombination of theorems T1 (recesive) and T2 (dominant)

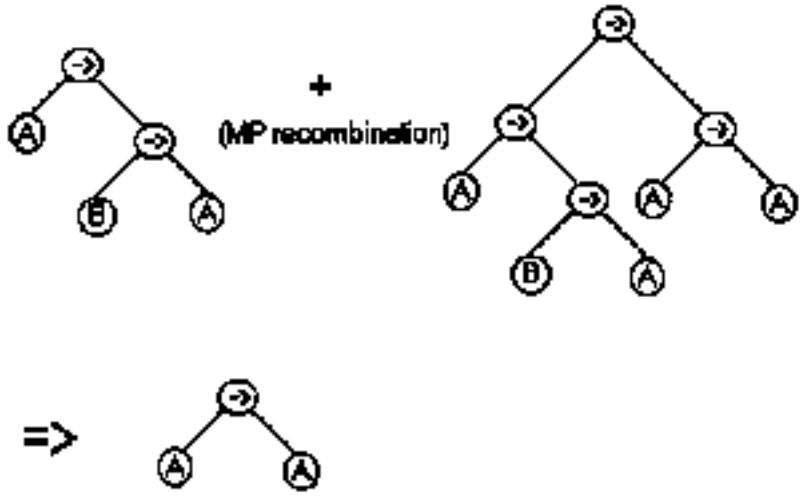

Figure 11 MP recombination of theorems T3 (recesive) and T4 (dominant)

Fitness $f(T)$ of a theorem T may be defined as

$$f(T) = \frac{1}{H(T)+1}.$$

The fitness is to be maximized.

The fitness of a theorem that does not contains the target theorem R as a right (sub)sub-tree is considered to be zero.

Examples. For the theorem T_1 represented in Figure 10 has the handle

$$H(T_1) = 2.$$

The fitness of T_1 is

$$f(T_1) = 1/3.$$

For the theorem T_4 (the offspring depicted in Figure 10) has the handle

$$H(T_4) = 1.$$

The fitness of T_4 is

$$f(T_4) = 1/2.$$

For the target theorem R has the handle

$$H(R) = 0.$$

The fitness of $R$is

$$f(R) = 1.$$

For the theorems T_2 (Figure 8) and T_3 (Figure 9) has the handle

$$H(T_2) = H(T_3) = \infty.$$

The fitness of T_2 and T_3 is

$$f(T_2) = f(T_3) = 0.$$

8. ETP procedure

In this section an evolutionary theorem proving method based on previous considerations rule is presented. The method uses a function ETP_F that evolves a population of candidate theorems. If the function output is Yes i.e. R is a theorem then the ETP algorithm stops. Otherwise we try to prove the formula $\sim R$ is a theorem. For this respect function ETP_F is used again.

Evolutionary theorem proving (ETP) algorithm may be outlined as below:

ETP ALGORITHM

```
begin
    if ETP_F(R) = Yes    {R is a theorem}
    then print deduction chain;
    else if ETP_F(~ R) = Yes   {R is not a theorem}
        then print R is not a theorem
        else   {we can not say if R is or is not a theorem}
            print we can not make a decision about R
        endif
    endif
end
```

Function ETP_F is outlined as follows:

```
function ETP_F(R theorem);    {returns Yes if R is theorem otherwise return No}
begin
    Initialization:
    generate randomly a substitution population (S);
    generate a theorem population T(t) using substitutions from S;
    {apply substitutions from S}
    Evolving theorems:
    t = 0;
    while t <= MaxGen do
        for each individual c in T(t)
            find b - the best mate for c;
            z = crossover(b, c);
            if fitness(z) > fitness(c)
            then discard c from the theorem population T(t);
                add z to the theorem population T(t)
            endif
        endfor
        Mutate chromosomes from the current theorem population T(t);
        t = t + 1;
    endwhile
end
```

9. Concluding remarks and further research

A new evolutionary approach of automated theorem proving is proposed. The method is designed for propositional logic. This algorithm illustrates a new philosophy of theorem proving: According to our knowledge a similar approach does not exist in the literature.

We consider that using the proposed method some well-known drawbacks of classical method may be eliminated.

Numerical experiments have shown the effectiveness of the proposed algorithm. The method uses a heuristic fitness function.

The proposed approach will be extended for the first order logic.

References

[1] Bck, T., Fogel, D.B., Michalewicz, Z. (Eds.), (1997), Handbook of Evolutionary Computation, Institute of Physics Publishing, Bristol, and Oxford University Press, New York.

[2] Dumitrescu, D., Lazzerini, B., Jain, L.C., Dumitrescu, A., (2000) Evolutionary Computation, CRC Press, Boca Raton, FL.

[3] Fitting, M., (1990), First-Order Logic and Automated Theorem Proving, Springer-Verlag, New- York.

[4] Gallier, J.H., (1986), Logic for Computer Science, Foundation of Automatic Theorem Proving, Harper and Row.

[5] Garey, M.R., Johnson, D.S., (1978), Computers and Intractability: A Guide to NP- completeness, W.H. Freeman and Company, New York.

[6] Koza, J.R., (1992), Genetic Programming, MIT Press, Cambridge, MA.

[7] Robinson, J.A., (1965), A Machine-Oriented Logic Based on the Resolution Principle, Journal of ACM, vol 12, pp 23-41.

Acknowledgments

The authors thank the anonymous reviewer for pertinent observations.

"Babeş-Bolyai" University of Cluj-Napoca, Department of Computer Science
E-mail address: {ddumitr,moltean}@cs.ubbcluj.ro