Strategies to Fast Evaluation of Tree Networks

Raed Basbous

Eastern Mediterranean University, Faculty of Arts and Sciences,
Department Mathematics,
Famagusta, North Cyprus via Mersin 10, Turkey, and
Al-Quds Open University, Department of Administrative Affairs,
Jerusalem, PO Box 58100, Palestine
E-mail: rbasbous@qou.edu

Benedek Nagy

Eastern Mediterranean University, Faculty of Arts and Sciences,
Department Mathematics,
Famagusta, North Cyprus via Mersin 10, Turkey, and
University of Debrecen, Faculty of Informatics, Department of Computer Science,
4010 Debrecen, PO Box 12, Hungary
E-mail: nbenedek.inf@gmail.com

Abstract: Special tree graphs could model Cognitive Infocommunication Networks. The various modalities of the network are represented by various types of vertices, e.g., additions, multiplications. Tree graphs/networks are widely used in several other theoretical and practical fields. Expression trees are well-known tools to visualize the syntactic structure of the expressions. They are helpful also in evaluations, e.g., decision trees are widely used. Games and game theory form an important field in Artificial Intelligence and it has several connections to Optimization, Business and Economy. Game trees are used to represent games.

In this paper, certain types of tree networks are considered using various operations at their inner vertices, e.g., multiplication, (constrained) addition and the usual minimum and maximum (related to conjunction and disjunction of Boolean algebra). Evaluation techniques are presented, as well as, various pruning algorithms (related to short circuit evaluation in the Boolean case) that can quicken the evaluation in most cases. Based on the commutativity of the used operations, the evaluations can be more effective (faster) by reordering the branches of the tree. The presented techniques are useful to optimize (minimize) the size of the tree networks in various cases without affecting the final result/decision of the network.

Keywords: expression trees; game trees; short circuit evaluation; pruning of trees; decision making; reordered trees

1 Introduction

Cognitive infocommunications (CogInfoCom) form an interdisciplinary field that works with various modalities [1]. These multimodal sensors usually form various networks. In this paper, we deal with special networks that form tree graphs. The multimodality is modelled by various types of nodes, let us say, operators in the tree. In our model the leaves may represent some measured data, and via the network a value (a decision, an evaluation of the situation, etc.) must be determined/computed. The edges of the tree specify the communication channels in which some data or (sub)result can be transferred from one node to another. Our model is also connected to the efficient evaluation process of various expression-trees that may represent CogInfoCom Networks.

Evaluation of various types of expressions is essential in every field connected to Mathematics, Computer Science and Engineering. Logical expressions can be evaluated in a fast way based on the following facts: if a member of a conjunctive formula is false, then the whole formula is false; if a member of a disjunctive formula is true, then the whole formula is true [10]. These facts are used in several programming languages to have a fast evaluation of logical formulae, e.g., in conditions [4]. This technique is called short circuit evaluation. There is a very similar idea that is used in game trees in Artificial Intelligence (AI). At the most investigated zero-sum two-player games, the minimax algorithm gives the best strategies for the players and it also answers the question, who has a winning strategy [8]. To compute the minimax algorithm every leaf of the tree is computed and the whole tree is evaluated. However, in most of the cases, it can be done with a much less effort, using alpha and beta pruning techniques [5, 8].

We assume that (AI) agents are starting from the leaves of the tree and they are communicating until the result is given at the root. In our CogInfoCom Network model the agents are various types of cognitive sensors. The sensors at leaves measure the environment/receive input data. The sensors/nodes communicate to each other in a bottom-up direction. When an inner cognitive sensor receives the data from its children nodes, it computes its task (based on its modality, i.e., type of operation) and sends its result to its parent node. The root node generates the final result/decision of the whole CogInfoCom Network. By using pruning techniques, there, usually a (much) less number of agents is enough to obtain the same results. We can reduce the communication cost of the network and possibly we do not need to use all the measured data, some subset of these data could be enough to infer the exact result of the network/expression tree. In this way fast evaluation techniques are important not only by saving time and space in a one-processor system when an expression is evaluated, but they also help to optimize the parallel resources in (bounded) parallel systems, e.g., CogInfoCom networks. There is also an enhancement of the previously mentioned pruning techniques: by reordering the branches of the tree network in such a way that the evaluation starts with the shortest branch.

Our algorithms can also be applied in artificial decision making systems that are closely related to systems used in cognitive-informatics and info-communications [9, 11]. Related human decisions and cognitive systems were also presented in [3].

In this paper, we consider special formula/expression trees (as models of CogInfoCom Networks) that can be considered a type of extension of the usual game trees/logical expression trees. Special extensions, as a mixture of decision and game-trees were already discussed in [6]. Here, by a further step, such extensions of game-trees are investigated in which some operations may not be directly connected to games, but with usual (mathematical or logical) expressions to give more modalities to our systems. We use values 0 and ± 1 at the leaves of the trees. In some cases, it is not necessary to know the value of every descendant to evaluate a node of the tree, these cases lead to various pruning techniques that are presented here, in this way simplifying the (evaluation of the) network.

In the next section we recall some analogous techniques (mostly from [6, 8]) and we also fix our notations. We also show that reordering the branches of the tree (restructuring a Boolean network based on two modalities represented by conjunction/minimum and disjunction/maximum) may lead to an even faster evaluation. In Section 3 we present the results about pruning the tree networks considered here. In Section 4, it is presented that reordering of the branches of the tree may lead to a faster exact evaluation of the network. Finally, a short concluding section closes the paper.

2 Preliminaries

In this section we recall some related concepts, specially decision trees, game trees, minimax algorithm and alpha-beta pruning. We also present short circuit evaluation techniques for Boolean logic and, further, in Section 4 we show, as one of our new results, its more advanced version, based on the reordering of the branches of the expression tree.

In this paper, we use the term inner node for every node (including the root node) that is not a leaf.

2.1 Decision Trees

In this subsection we recall decision trees (see, e.g. in [6]); they can also be used to evaluate games against the “Nature”. Let a person be given who has some decision points, and some random events with known probabilities, also let us consider a tree with decision nodes and chance nodes. At decision nodes the person chooses a successor node. At chance nodes a random event happens: the successor node is chosen randomly with the probabilities known in advance. The leaves of the tree represent the payoff values. The aim of the player is to maximize the payoff value and for this purpose she/he chooses the branch that leads to the highest expected value at every decision node.

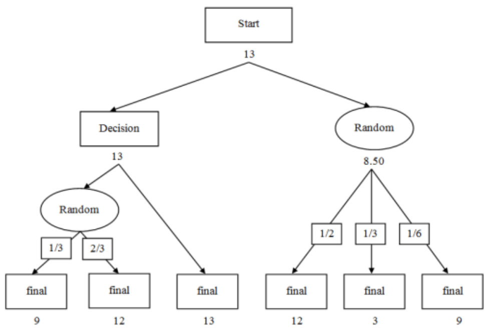

Figure 1
A decision tree

Assume that P(N) denotes the probability of the event N. Figure 1, shows an example of a decision tree. It includes decision nodes, where the person chooses among the successor, i.e. children nodes (represented by rectangles). At nodes represented by ellipses random events will determine the successor node (the probabilities are written on the edges). A numeric value, the expected outcome, is assigned to every node of the tree; it is shown under the rectangle/ellipse. These values are computed by Algorithm 1 (it is from [6]).

Algorithm 1 (Decision)

```
1.   function Dec(N)
2.   begin
3.   if N is leaf then
4.     return the value of this leaf
5.   else
6.    let $N_1, N_2, \ldots, N_m$ be the successors of N
7.    if N is a decision node then
8.      return max {Dec($N_1$), Dec($N_2$), …, Dec($N_m$)}
9.    if N is a chance node then
10.     return P($N_1$)Dec($N_1$)+ … + P($N_m$)Dec($N_m$)
11.  end Dec
```

The technique presented in Algorithm 1 is called expectimax [6], since at decision points it computes the maximum of the expected values of the successor nodes, while at chance nodes the expected values of the successor nodes are computed by finding the sum of the assigned probabilities multiplied by the value of the given successors.

In this paper we deal only with problems having a bounded set of possible outcomes $\{0, \pm 1\}$. This restriction/simplification on the input/measured data allows us to develop some efficient algorithms.

2.2 Game Theory

An important and widely investigated field of game theory is about deterministic, strategic, two-player, zero-sum, finite games with perfect information. An instance of a game begins with the first player's choice from a set of specified alternatives, called moves. After a move, a new state of the game is obtained; the other player is to make the next move from the alternatives available to that player. In same state of the game there is no move possible, the instance of the game has been finished. In such states each player receives a payoff, such that their sum is zero, and therefore it is enough to know the payoff of the first player. In some typical zero-sum games, the value +1 assigned to the player in case of win, -1 in case of lose, and 0 in case of draw.

Game trees can be used to represent games. The nodes represent the states of the game, while the arcs represent the available moves. In the game tree there are two kinds of nodes representing the decision situations of the two players. At the root node the first player has decision. The leaves represent terminal states with their payoff values (for the first player).

In every two player, zero sum, deterministic game with perfect information there exists a perfect strategy for each player that guarantees the at least result in every instance of the game. The most fundamental result of game theory is the minimax theorem and the minimax algorithm. The theorem says: If a minimax of one player corresponds to a maximin of the other player, then that outcome is the optimal for both players.

2.2.1 Minimax Algorithm

Minimax theorem is a fundamental result of game theory. Players adopt strategies which maximize their gains, while minimizing their losses. Therefore, the solution is the optimal for each player that she/he can do for him/herself in the face of opposition of the other player. These optimal strategies and the optimal payoff can be determined by the minimax algorithm. It uses simple recursive functions to compute the minimax values of each successor state [8].

Algorithm 2 represents the way that minimax algorithm works (it is recalled from [6, 8]), see also, e.g., Figure 2 for examples.

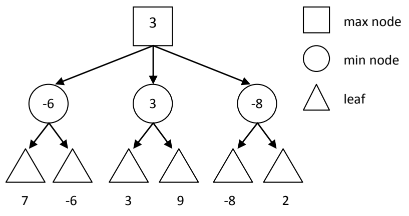

Figure 2
A minimax tree

Algorithm 2 (MIN and MAX)

```
1.   function MAXValue(N)
2.   begin
3.   if N is leaf then
4.       return the value of this leaf
5.   else
6.      let v = -∞
7.      for every successor $N_i$ of N do
8.          let v=  max{v,  MINValue($N_i$)}
9.   return v
10.  end MAXValue

1.   function MINValue(N)
2.   begin
3.   if N is a leaf then
4.      return the value of this leaf
5.   else
6.      let v = +∞
7.      for every successor $N_i$ of N do
8.         let v=  min{v,  MAXValue($N_i$)}
9.   return v
10.  end MINValue
```

2.2.2 Alpha-Beta Pruning

The minimax algorithm evaluates every possible instance of the game, and thus to compute the value of the game, i.e. its optimal payoff, takes usually exponential time on the length of the instances of the game. To overcome on this issue special cut techniques can be used. The alpha-beta pruning helps find the optimal values without looking at every node of the game tree. While using minimax, some situations may arise when searching of a particular branch can safely be terminated. So, while doing search, these techniques figure out those nodes that do not require to be expanded.

The way that this algorithm works can be described as below.

- Max-player cuts off search when she/he knows that Min-player can force a clear bad (for the first player, i.e. for Max-player) outcome.
- Min-player cuts off search when she/he knows that Max-player can force a clear good (for Max-player) outcome.
- Applying alpha-pruning (beta-pruning) means the search of a branch is stopped because a better opportunity for Max-player (Min-player) is already known elsewhere. Applying both of them is called alpha-beta pruning technique.

Figure 3 shows an example of beta-pruning, when β becomes smaller than or equal to α, we can stop expanding the children of N.

These algorithms, shown in Algorithm 3, are recalled from [6, 8].

Algorithm 3 (MIN and MAX PRUNING)

```
1.   function ALPHAPrune(N, $\alpha$, $\beta$)
2.   begin
3.   if N is leaf then
4.       return the value of this leaf
5.   else
6.       let v = $-\infty$
7.       for every successor $N_i$ of N do
8.           let v=  max{v,  BETAPrune($N_i$, $\alpha$, $\beta$)}
9.           if $v \geq \beta$ then return v
10.          let $\alpha = \max\{\alpha, v\}$
11.  return v
12.  end ALPHAPrune

1.   function BETAPrune(N, $\alpha$, $\beta$)
2.   begin
3.   if N is a leaf then
4.       return the value of this leaf
5.   else
6.       let v = $+\infty$
7.       for every successor $N_i$ of N do
8.           let v=  min{v,  ALPHAPrune($N_i$, $\alpha$, $\beta$)}
9.           if $v \leq \alpha$ then return v
10.          let $\beta = \max\{\beta, v\}$
11.  return v
12.  end BETAPrune
```

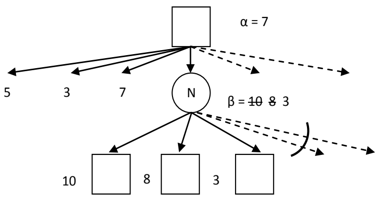

Figure 3
A beta-pruning example

2.3 Short Circuit Evaluation

Short circuit evaluation of a logical expression is widely used in programming languages for optimization. The short circuit evaluation technique is a type of time saving and, in some cases, it is also used for safety reasons [7]. For example, if it is known that all three conditions/variables must be true in order to proceed, it is not necessary to check the second and third conditions if the first one is already known to be false. In some programming languages the symbols && and || are used for the logical operations AND and OR, respectively. These operations are working in short circuit evaluation.

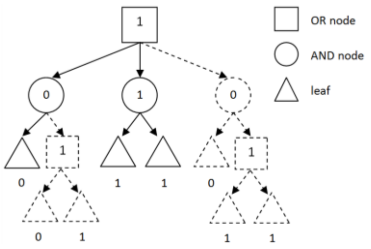

Figure 4
An example of applying the short circuit evaluation on a tree. Only three inner nodes and three leaves (total 6 nodes) out of six inner nodes and eight leaves (total 14 nodes) are explored and evaluated to get the final result in the root.

2.3.1 The AND Version

Let A, B and C be three Boolean expressions. Let them be in a conditional statement using && as follows.

```
if ( A && B && C )          {   statement; }
```

The only way this expression can be true is if A and B and C are all true. If A is false, we don't even need to consider B and C because we already know that the entire expression is false, etc. There is no need for further evaluation, as soon as it can be determined, through short circuit evaluation that the expression is false.

2.3.2 The OR Version

Similar to the previously described AND operation, the OR operation can also be terminated early in some cases. Let us consider the following statement.

```
if (A || B || C)            {   statement; }
```

In the case of OR, if it is known that the first condition is true, then there is no need to check the value of second and third conditions. If A is false, but B is true, then it is not needed to evaluate C to know the truth value of the condition.

2.3.3 The Mixed Version

Let us consider a more complex Boolean formula having only conjunctions and disjunctions. In this way, these special Boolean formulae can model tree networks having two types of operations/modalities. Figure 4, shows an example of a complex tree and also of applying the short circuit evaluation on the tree. As shown in this example, when evaluating the AND nodes, if a zero value (0) returned back from one of the children nodes, then we cut the next connected nodes and leaves and there is no need to evaluate them since their value will not affect the final result. The same technique is applied in evaluating the OR nodes, the idea here is to cut-off when the value one (1) returned back from a connected node or leaf. The final result of evaluating that node will be one (1) regardless the value of the remaining connected children nodes. See also Algorithm 4 below which describes how the idea of the short circuit evaluation is used. We allow not only binary conjunctions and disjunctions and thus, we may assume that they are alternating by levels.

3 Algorithms for Trees with Several Modalities

Now, various algorithms are proposed to quicken the evaluation of special trees in which there are specific operations are used. We are dealing with trees with a bounded set of payoff values: -1, 0, 1. Apart from the usual min and max operations (that can be seen as AND and OR by restricting the values to the Boolean set, i.e. to {0,1}) we use the operations multiplication (that can also be seen as AND on the Boolean set) and sum (that usually can be seen as binary addition, i.e. OR on the Boolean set), as well. We keep only the sign of the sum to have the result inside the domain {-1, 0, +1}. The proposed modified minimax, sum and product pruning, minimax with sum, and minimax with product algorithms are described below in detail.

Algorithm 4 (OR and AND PRUNING)

```
1.  function ORPrune (N)
2.  begin
3.  if N is leaf then
4.      return the value of this leaf
5.  else
6.    let v = 0
7.    for every successor N_i of N do
8.       while v ≠ 1  do
9.           let v= v + ANDPrune(N_i)
10.      return v
11. end ORPrune
1.  function ANDPrune (N)
2.  begin
3.  if N is a leaf then
4.      return the value of this leaf
5.  else
6.     let v = 1
7.     for every successor N_i of N do
8.         while v ≠ 0  do
9.             let v = v · ORPrune(N_i)
10. return v
11. end ANDPrune
```

3.1 Modified Alpha-Beta Pruning Algorithm

We start with an obvious modification of Algorithm 3. Since the set of payoff values is bounded, we can modify the alpha-beta pruning algorithm to do a cut when the maximum (+1) or the minimum (-1) value of the set is already found in ALPHAPrune and BETAPrune, respectively, as shown in Algorithm 5 below. Line 8 of Algorithm 3 is modified in both ALPHAPrune and BETAPrune functions to test if the values +1 and -1 (the possible maximum and minimum, respectively) have been found in the evaluated node, thus there is no need to visit the remaining successors for that node: a cut can be done.

Figure 5 shows an example for the usage of the modified pruning algorithm.

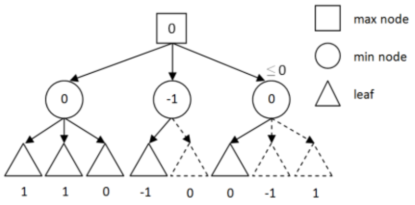

Figure 5
A modified minimax alpha-beta pruning. Only five leaves out of eight are explored to have an exact evaluation of the tree.

Algorithm 5 (MIN and MAX PRUNING)

```
function MAXPrune(N, $\alpha$, $\beta$)
begin
if N is leaf then
    return the value of this leaf
else
  let v = $-\infty$
  for every successor $N_i$ of N do
     while v < +1  do
        let v=  max{v,  MINPrune($N_i$, $\alpha$, $\beta$)}
        if v$\geq \beta$ then return v
        let $\alpha$ = max{$\alpha$,v}
return v
end MAXPrune

function MINPrune(N, $\alpha$, $\beta$)
begin
if N is a leaf then
    return the value of this leaf
else
  let v = $+\infty$
  for every successor $N_i$ of N do
      while v > -1  do
          let v=  min{v,  MAXPrune($N_i$, $\alpha$, $\beta$)}
          if v $\leq \alpha$ then return v
          let $\beta$ = max{$\beta$,v}
return v
end MINPrune
```

3.2 Sum and Product Pruning Algorithm

Now let us consider a new type of expression in which product operations follow the additions and vice versa, e.g., expressions of the form abc+de+fghi and also more complex expressions with these two operations appearing in the expression tree. Remember that both the possible values of the variables (leaves) and of the expressions (other nodes) are restricted to the set {-1, 0, +1} and thus both the Sum and Product functions can have these three output values.

Ideas similar to the short circuit evaluation can be used to cut during the evaluations of sum and product (multiplication) functions as they are described in Algorithm 6, see also, e.g. Figure 6 for examples.

Algorithm 6 (MUL and SUM PRUNING)

```
1.    function SUMPrune (N)
2.    begin
3.    if N is leaf then
4.        return the value of this leaf
5.    else
6.      let v = 0
7.      for every successor $N_i$ of N do
8.         while |v|≤ (the number of remaining nodes to visit do)
9.            let v= v + MULPrune($N_i$)
10.        if v > 0 then
11.              return 1
12.        if v< 0 then
13.              return -1
14.        else
15.              return 0
16.   end SUMPrune
```

```
1.    function MULPrune (N)
2.    begin
3.    if N is a leaf then
4.        return the value of this leaf
5.    else
6.      let v = 1
7.      for every successor $N_i$ of N do
8.          while $v \neq 0$  do
9.             let v = v * SUMPrune($N_i$)
10.   return v
11.   end MULPrune
```

The possible cuts that are applied here are the following: For the SUM function, if the absolute value of the actual value is greater than the remaining successors to visit, then pruning is applied since there is no need to evaluate the remaining nodes: the result is already known: the return value is 1 if the sum value greater than zero, and -1 if the sum value is less than zero. This can be done using the while loop in line 8 in the function SUMPrune.

At the product operator a zero-cut is done, since 0 is the zero element of the multiplication, as it is written in the function MULPrune.

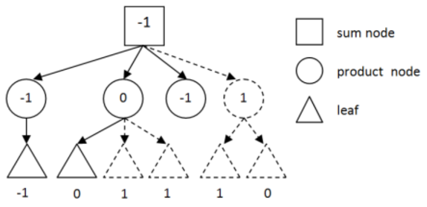

Figure 6
A sum and product pruning example. Only four inner nodes and two leaves (total 6 nodes) out of five inner nodes and six leaves (total 11 nodes) are explored and evaluated to get the final result in the root.

3.3 Minimax-Product Pruning Algorithm

In minimax-product case, the evaluation of the tree can be quickened by applying the zero-cut at product layer when a node with zero value is returned from one of the connected successors. In addition to this, similarly to the usual alpha-beta pruning, pruning can be applied in max and min layers. Example for these kinds of pruning are shown in Figure 7: a cut applied to the second node in min layer when a zero value (β) returned from the product layer, which is less or equal to the maximum value (α) that is found in the first node.

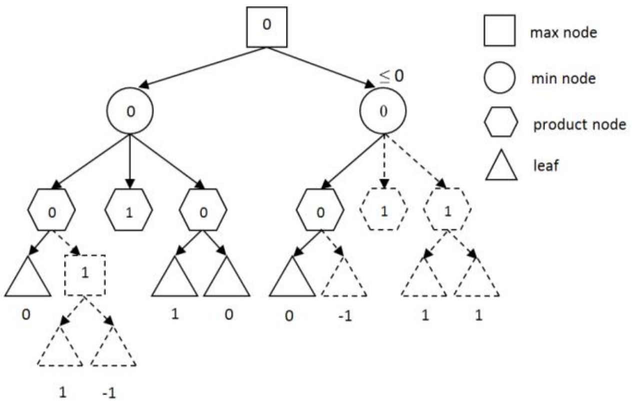

Figure 7

A minimax and product pruning example. Only seven inner nodes and three leaves (total 10) out of ten inner nodes and nine leaves (total 19) are explored and evaluated to get the final result in the root.

The proposed process for pruning is shown in Algorithm 7. The same MULPrune function that proposed previously in Algorithm 6 is used here with MAXPrune and MINPrune functions. (They can easily be modified having other order of layers in the expression tree.)

Algorithm 7 (MUL, MIN and MAX PRUNING)

```
1.   function MAXPrune(N, $\alpha$, $\beta$)
2.   begin
3.   if N is leaf then
4.       return the value of this leaf
5.   else
6.      let v = $-\infty$
7.      for every successor $N_i$ of N do
8.          while v <+1  do
9.             let v=  max{v,  MINPrune($N_i$, $\alpha$, $\beta$)}
10.         let $\alpha = \max\{\alpha, v\}$
11.  return v
12.  end MAXPrune
```

```
function MINPrune(N, $\alpha$, $\beta$)
begin
if N is a leaf then
    return the value of this leaf
else
   let v = $+\infty$
   for every successor $N_i$ of N do
      while v > -1  do
          let v=  min{v,  MULPrune($N_i$, $\alpha$, $\beta$)}
      if v $\leq \alpha$ then return v
      let $\beta$ = max{$\beta$,v}
   return v
end MINPrune
```

```
function MULPrune(N, $\alpha$, $\beta$)
begin
if N is a leaf then
   return the value of this leaf
else
   let v = 1
   for every successor $N_i$ of N do
       while v $\neq$ 0 do
          let v = v * MAXPRUNE($N_i$, $\alpha$, $\beta$)
   return v
end MULPrune
```

3.4 Minimax-Sum Pruning Algorithm

In expression trees with three layers (max, min and sum), we use the idea of sum short circuit evaluation initiated in Subsection 3.2.

As mentioned before, the sum function has three output values (-1, 0, +1). When the absolute value of the sum of the visited successors is more than the number of remaining successors, then the cut can be done and if the sum is positive, then +1 is returned; if the sum is negative, then -1 is returned.

Furthermore, similar alpha and beta pruning for min and max functions can be done as in the previous algorithms to cut and quicken the evaluation of the tree (see Algorithm 8). In Figure 8, in the minimum layer a pruning is applied when the minimum value (-1) has been found, and after that in maximum layer again, when the maximum value (+1) has been found. The sum short circuit is applied too; since the absolute value of the sum function (+2) is greater than the remaining nodes to evaluate (we have one node remain to evaluate, which is less than 2).

Algorithm 8 (ADD, MIN and MAX PRUNING)

```
1.    function MAXPrune(N, α, β)
2.    begin
3.    if N is leaf then
4.        return the value of this leaf
5.    else
6.       let v = -∞
7.       for every successor N_i of N do
8.           while v < +1  do
9.               let v=  max{v,  MINPrune(N_i, α, β)}
10.          let α = max{α,v}
11.   return v
12.   end MAXPrune

1.    function MINPrune(N, α, β)
2.    begin
3.    if N is a leaf then
4.        return the value of this leaf
5.    else
6.        let v = +∞
7.        for every successor N_i of N do
8.            while v > -1  do
9.                let v=  min{v,  ADDPrune(N_i, α, β)}
10.           if v ≤ α then return v
11.           let β = max{β,v}
12.   return v
13.   end MINPrune

1.    function ADDPrune(N, α, β)
2.    begin
3.    if N is a leaf then
4.       return the value of this leaf
5.    else
6.      let v = 0
7.      for every successor N_i of N do
8.         while |v|≤  (the number of remaining nodes to visit do)
9.             let v= v + MAXPRUNE(N_i, α, β)
10.        if v > 0 then
11.              return 1
12.        if v< 0 then
13.              return -1
14.        else
15.              return 0
16.   end ADDPrune
```

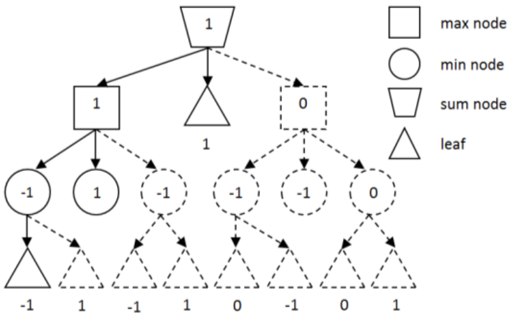

Figure 8
A minimax and sum pruning example. Only four inner nodes and four leaves (total 8) out of nine inner nodes and nine leaves (total 18) are explored and evaluated to get the final result in the root.

4 Reordering the Branches of the Trees

Notice that each of the used operations are commutative, and thus, the result of the operand does not depend on the order of the children branches. Therefore, in addition to the above proposed algorithms to quicken the evaluation of the tree networks with the presented logical and mathematical operations, a reordering technique can be applied on these kinds of trees before starting the evaluation.

The intuitive idea behind the reordering is that shorter branches can be computed faster. Therefore, the aim is to move the node that is the root of a subtree having least depth to the left side to be evaluated first. The process of reordering must be started from the lowest layers and goes up to the root. Then, after the reordering is done, the evaluation process can be started where the previously mentioned pruning techniques can be applied to quicken the evaluation.

4.1 Reordering the Branches of Boolean Expressions

In this subsection we show how the evaluation of a Boolean expression/CogInfoCom tree-network with two modalities can be done in a more efficient way. Figure 9, shows how the evaluation of the tree in Figure 4, is done after reordering its branches. After applying the reordering process, only two inner nodes (operators) out of six evaluated to get the final result in the root. While, only two of the leaves are used instead of the total eight during the evaluation process using the operators/modalities (AND and OR). The process of reordering the tree branches described in Algorithm 9. The same idea can be applied if we have different types of operators/modalities (e.g., SUM).

Algorithm 9 (AND, and OR Tree Reordering)

```
1.  function ORReorder(N)
2.  begin
3.  if N is leaf then
4.       move the leaf to the left  side
5.  else
6.      for every successor $N_i$ of N do
7.          ANDReorder($N_i$)
8.          Count the number of connected leaves and nodes
9.   move the node $N_i$ with less leaves and nodes to the left side.
10. end ORReorder

1.  function ANDReorder(N)
2.  begin
3.  if N is leaf then
4.       move the leaf to the left  side
5.  else
6.      for every successor $N_i$ of N do
7.          ORReorder($N_i$)
8.          Count the number of connected leaves and nodes
9.      move the node $N_i$ with less leaves and nodes to the left side.
10. end ANDReorder
```

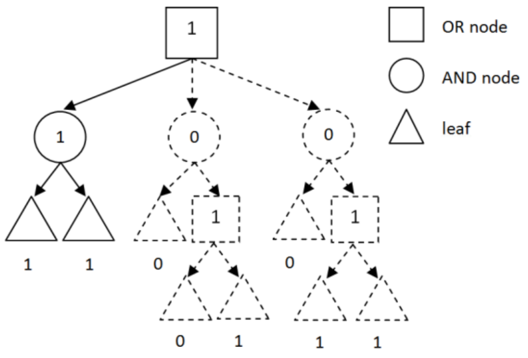

Figure 9
An example of reordering and pruning algorithm. Only 2 inner nodes and 2 leaves out of 4 inner nodes and 6 leaves were evaluated after the reordering and pruning process.

4.2 Reordering and Pruning Complex Trees

In this subsection we use the reordering technique for tree networks with more than two modalities and show that it can be used very efficiently in these cases as well.

In the first example of this subsection an expression tree is shown (see Figure 10 below). This tree includes four different operators (SUM, MULTIPLICATION, MAX, and MIN) in such a way that in the same level only the same operator is used. Since, in this case, the order of the operators are fixed (as at game trees), the evaluation functions call each other in a predefined order. To evaluate the tree of that example, without applying the reordering technique and pruning algorithms mentioned and detailed in this paper, 23 inner nodes and 26 leaves must be explored and evaluated. Figure 11 shows the same tree evaluated after applying the proposed pruning algorithms without reordering the branches. The same result of evaluation returned back to the root, but this was by exploring and evaluating only 19 inner nodes and 19 leaves (total 38) out of 23 inner nodes and 26 leaves (total 49).

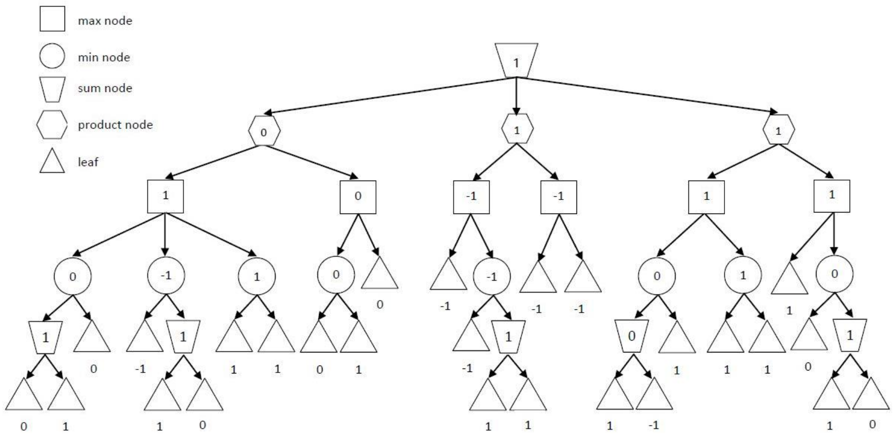

Figure 10
An example of an expression tree with SUM, MULTIPLICATION, MAX, and MIN operators. The tree contains 23 inner nodes and 26 leaves (total 49 nodes).

Figure 12 shows the same tree evaluated after reordering the branches and then applying the proposed pruning algorithms. The same result of evaluation returned back to the root, but this was by exploring and evaluating only 12 inner nodes and 7 leaves (total 19) out of 23 inner nodes and 26 leaves (total 49). This result shows how fast the evaluation process can be after applying the proposed technique to evaluate an expression trees that include various modalities (logical and mathematical operators, here).

In our other example, shown in Figure 13, the operators have no fixed order in the expression, which maybe much closer to real word applications in some cases. Also, leaves can be found in various levels (i.e., various depths) of the tree. To evaluate the tree without applying the reordering technique and pruning algorithms mentioned above, 11 inner nodes and 13 leaves must be explored and evaluated (total 24). Using various pruning strategies the number of inner nodes that must be evaluated is 9, while the number of leaves that must be explored is 7 (total 16 nodes, see Figure 14). The proposed reordering technique together with the pruning algorithms evaluates the expression tree of this example by exploring and evaluating only 5 inner nodes and 4 leaves (total 9 nodes are visited for the evaluation) as it can be seen in Figure 15.

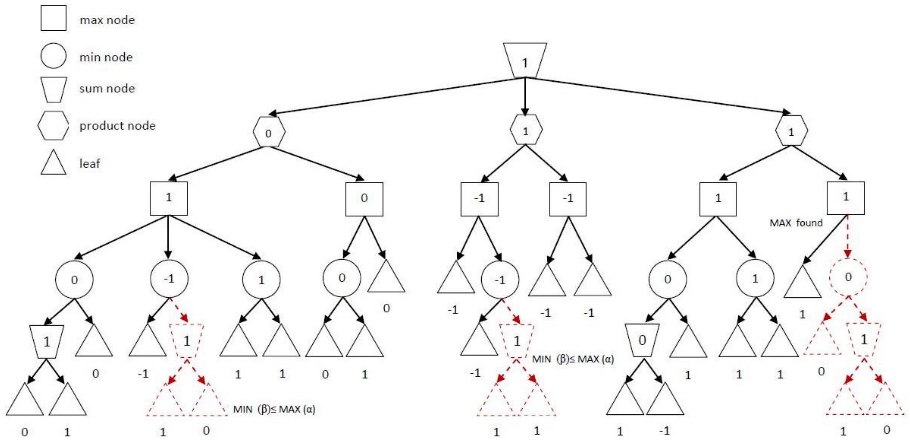

Figure 11
The expression tree of the example of Figure 10 is evaluated by pruning techniques. After applying the pruning algorithms without reordering the branches, 19 inner nodes and 19 leaves remained (total 38) out of 23 inner nodes and 26 leaves (total 49 nodes).

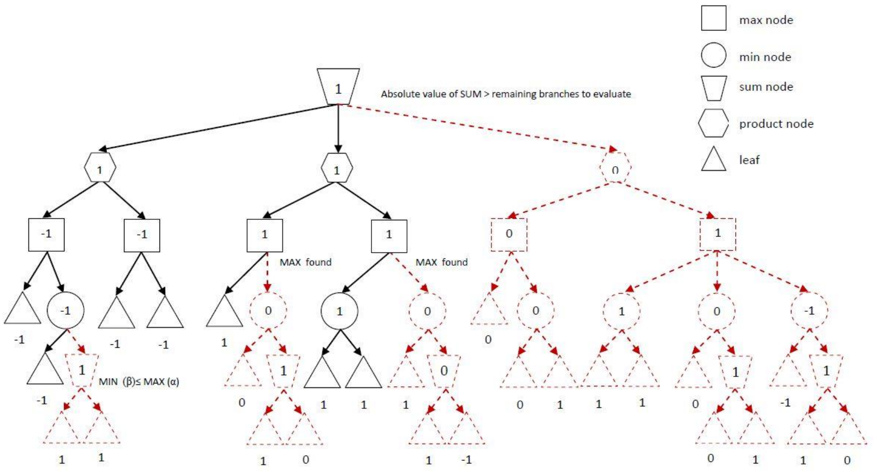

Figure 12
The expression tree of Figures 10 and 11 is evaluated by reordering and pruning. After applying the reordering and then the pruning algorithms, only 9 inner nodes and 7 leaves remained (total 16) out of 23 inner nodes and 26 leaves (total 49 nodes).

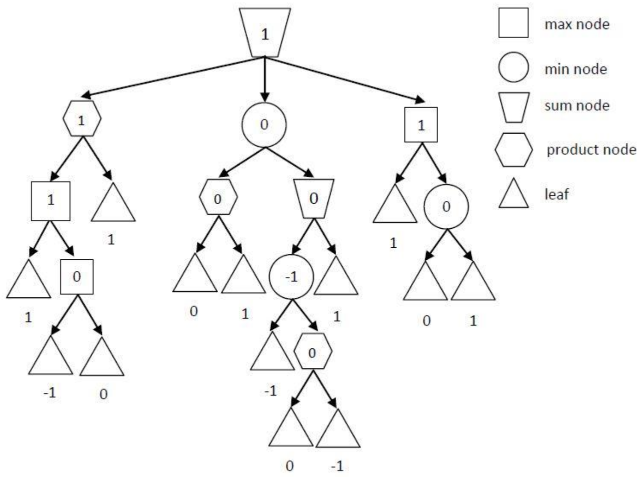

Figure 13
An example of an expression tree with SUM, MULTIPLICATION, MAX, and MIN operators in various order. The tree contains 11 inner nodes and 13 leaves (total 24 nodes).

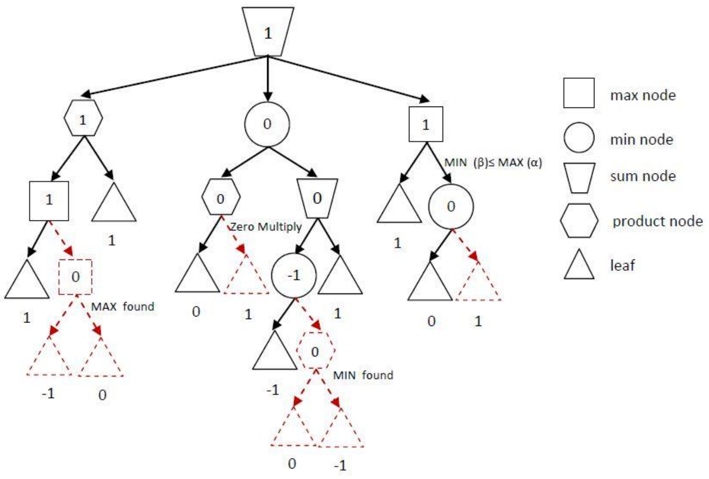

Figure 14
The example of Figure 13 is evaluated applying the pruning algorithms without reordering the branches: 9 inner nodes and 7 leaves evaluated and explored (total 16) out of 11 inner nodes and 13 leaves (total 24).

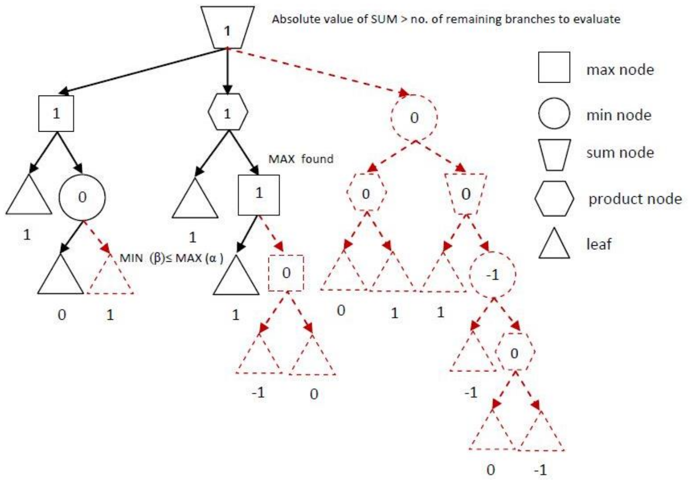

Figure 15

The expression tree of Figure 13 is evaluated by applying both the reordering and then the pruning algorithms: only 5 inner nodes and 4 leaves (total 9 nodes) are needed for the evaluation.

Conclusions

We have considered simple network models, in which there are only finite communication channels. In our tree networks children nodes send their data/results to their parents and thus, the whole process is finite and can be modelled by evaluation techniques. There are well-known techniques to quicken the evaluation of Boolean expressions and similar techniques are used at game trees in alpha-beta pruning algorithms. In this paper, we have shown other pruning strategies: techniques for expressions where sum, product, minimum and maximum modeling various modalities in CogInfoCom networks. Similar optimization techniques could help in fast evaluations of various expressions (other types of trees and networks). Reordering the branches, by evaluating first, the shorter ones, may also help a lot to save time and energy, especially, when the evaluation costs of the leaves (or other nodes) consumes a large amount of energy or other limited source.

Considering a kind of parallel approach in which various agents can work together on the evaluation, these types of techniques can easily reduce the number of required agents and also to reduce their communication costs. Therefore, the presented algorithms can reduce algorithmic costs (time, space, number of agents/processors) in decision making systems and in various extensions of two-player zero-sum games. Artificial cognitive systems, CogInfoCom systems and other systems based on artificial/computational intelligence can be used in a more efficient way.

Acknowledgement

Some parts of the results of this paper were already presented in the conference CogInfoCom 2014, [2].

References

[1] Péter Baranyi, Ádám Csapó: Cognitive Infocommunications, CogInfoCom, Proceedings of 11th CINTI: IEEE International Symposium on Computational Intelligence and Informatics, Budapest, Hungary, pp. 141-146, 2010

[2] Raed Basbous, Benedek Nagy: Generalized Game Trees and their Evaluation, Proceedings of CogInfoCom 2014: 5th IEEE International Conference on Cognitive Infocommunications, Vietri sul Mare, Italy, pp. 55-60, 2014

[3] Peter Földesi, János Botzheim: Computational Method for Corrective Mechanism of Cognitive Decision-Making Biases, Proceedings of CogInfoCom 2012: IEEE 3rd International Conference on Cognitive Infocommunications, Kosice, Slovakia, pp. 211-215, 2012

[4] Brian Kernighan, Dennis Ritchie: The C Programming Language. Prentice Hall, Englewood Cliffs, NJ, 1988

[5] Donald Knuth, Ronald Moore: An Analysis of Alpha-Beta Pruning, Artificial Intelligence, Vol. 6, pp. 293-326, 1975

[6] Ervin Melkó, Benedek Nagy: Optimal Strategy in Games with Chance Nodes, Acta Cybernetica, Vol. 18, pp. 171-192, 2007

[7] Benedek Nagy: Many-valued Logics and the Logic of the C Programming Language, Proc. of ITI 2005: 27th International Conference on Information Technology Interfaces (IEEE) Cavtat, Croatia, pp. 657-662, 2005

[8] Stuart Russell, Peter Norvig: Artificial Intelligence, a Modern Approach. Prentice-Hall, 2003

[9] David Schum, Gheorghe Tecuci, Dorin Marcu, Mihai Boicu: Toward Cognitive Assistants for Complex Decision Making under Uncertainty, Intelligent Decision Technologies, Vol. 8, pp. 231-250, 2014

[10] Joseph Shoenfield: Mathematical Logic, A K Peters, 2001

[11] Yingxu Wang, Guenther Ruhe: The Cognitive Process of Decision Making, International Journal of Cognitive Informatics and Natural Intelligence (IJCINI) Vol. 1, pp. 73-85, 2007