Yugoslav Journal of Operations Research
16 (2006), Number 2, 211-226

COMPARISON OF THE EFFICIENCY OF TWO ALGORITHMS WHICH SOLVE THE SHORTEST PATH PROBLEM WITH AN EMOTIONAL AGENT

Silvana PETRUSEVA
Mathematics Department, Faculty of Civil Engineering,
"St. Cyril and Methodius" University, Skopje, Macedonia
silvanap@unet.com.mk

Received: October 2003 / Accepted: June 2006

Abstract: This paper discusses the comparison of the efficiency of two algorithms, by estimation of their complexity. For solving the problem, the Neural Network Crossbar Adaptive Array (NN-CAA) is used as the agent architecture, implementing a model of an emotion. The problem discussed is how to find the shortest path in an environment with *n* states. The domains concerned are environments with *n* states, one of which is the starting state, one is the goal state, and some states are undesirable and they should be avoided.

It is obtained that finding *one* path (one solution) is efficient, i.e. in polynomial time by both algorithms. One of the algorithms is faster than the other only in the multiplicative constant, and it shows a step forward toward the optimality of the learning process. However, finding the *optimal solution* (*the shortest path*) by both algorithms is in exponential time which is asserted by two theorems.

It might be concluded that the concept of subgoal is one step forward toward the optimality of the process of the agent learning. Yet, it should be explored further on, in order to obtain an efficient, polynomial algorithm.

Keywords: Emotional agent, complexity, polynomial time, exponential time, adjacency matrix, shortest path.

1. INTRODUCTION

We shall recall some notions of the theory of complexity which will be used in this paper.

The complexity of the algorithm is the cost of the computation measured in running *time* or *memory*, or some other relevant unit. The complexity of the spending time is presented as a function from the input data which describe the problem.

In a typical case of a computational problem some input data are given, and a function from them should be computed. The rates of growth of different functions are defined by symbols in order to compare the speeds with which different algorithms do the same job. Some of these symbols, which are used here, are defined in the following way: [8], [11]

Let $f(x)$ and $g(x)$ are functions from x.

Definition 1. *We say that* $f(x)=O(g(x))$, $x \to \infty$ *if* $\exists C, x_0$ *so that* $|f(x)| \le Cg(x)$, $\forall x > x_0$ *, which means that f grows like g or slower.*

Definition 2. *We say that* $f(x) = \Omega(g(x))$ *if holds the opposite*: $g(x) = O(f(x))$ *when* $x \to \infty$ *and* $\exists \varepsilon > 0$, *and* x_0, *so that for* $x > x_0$ $|f(x)| > \varepsilon\, g(x)$.

Definition 3. *We say that* $f(x)= \Theta(g(x))$ *if there are constants* $c_1 > 0, c_2 > 0$ *,* x_0 *, such that for* $\forall x > x_0$ *it is true that* $c_1g(x) < f(x) < c_2g(x)$. *We might say then that f and g are of the same rate of growth, only the multiplicative constants are uncertain.*

This definition is equivalent to the definition:

$f(x)= \Theta(g(x))$ means that $f(x) = O(g(x))$ and $f(x) = \Omega(g(x))$ [8].

The *classes of complexity* are sets of languages which present an important decision problems. The property which these languages share is that all of them can be decided in some specific boundary of any aspect of their performances (time, space, or other).

The classes of complexity are determined by a few parameters: the model of computing, which can be *deterministic* or *nondeterministic,* and the resources we would like to restrict, like *time, space* or other. For example, the class P is the set of languages decided in *polynomial time* and the class EXP is the set of languages decided in *exponential time* [2]. If the problem is solved in polynomial time, it means that it is solved efficiently, but solving the problem in exponential time means that maybe this problem can not be solved in an efficient way.

2. DESCRIPTION OF THE ALGORITHMS "AT GOAL GO BACK" AND "AT SUBGOAL GO BACK"

The algorithms "at goal go back" and "at subgoal go back" solve the problem of finding the shortest path from the starting state to the goal state in an environment with n states. These algorithms were proposed for the first time in 1995 [3].The domains concerned are environments with n states, one of which is the starting state, one is the goal state, and some states are undesirable and they should be avoided. It is assumed that a path exists from every state to the goal state and from the starting state to every state, so there is a path from the starting state to the goal state. If the starting state can be every other state, i.e. if the problem is finding the shortest path from whatever other state to the goal state, then the assumption is that the graph is *strongly connected*, i.e. every state can be reached from every other state. The agent approach is used for solving this problem [3],[12]. The agent architecture used here is neural network, i.e. Neural Network-Crossbar Adaptive Array (NN - CAA) [3]. (Fig. 1)

The method which CAA uses is the backward chaining method. It has 2 phases: 1) search for a goal state, and 2) when a goal state is found define the previous state as a subgoal state. The search is performed using some searching strategy, in this case random walk. When executing a random walk the goal state is found, then a subgoal state is defined (with both algorithms), and with algorithm "at subgoal go back" this subgoal becomes a new goal state. The process of moving from the starting state to the goal state is a single run (iteration, trial) trough the graph. The next run starts again from the starting state, and will end in the goal state. In each run a new subgoal state is defined. The process finishes when the starting state becomes a subgoal state. That completes a solution finding process.

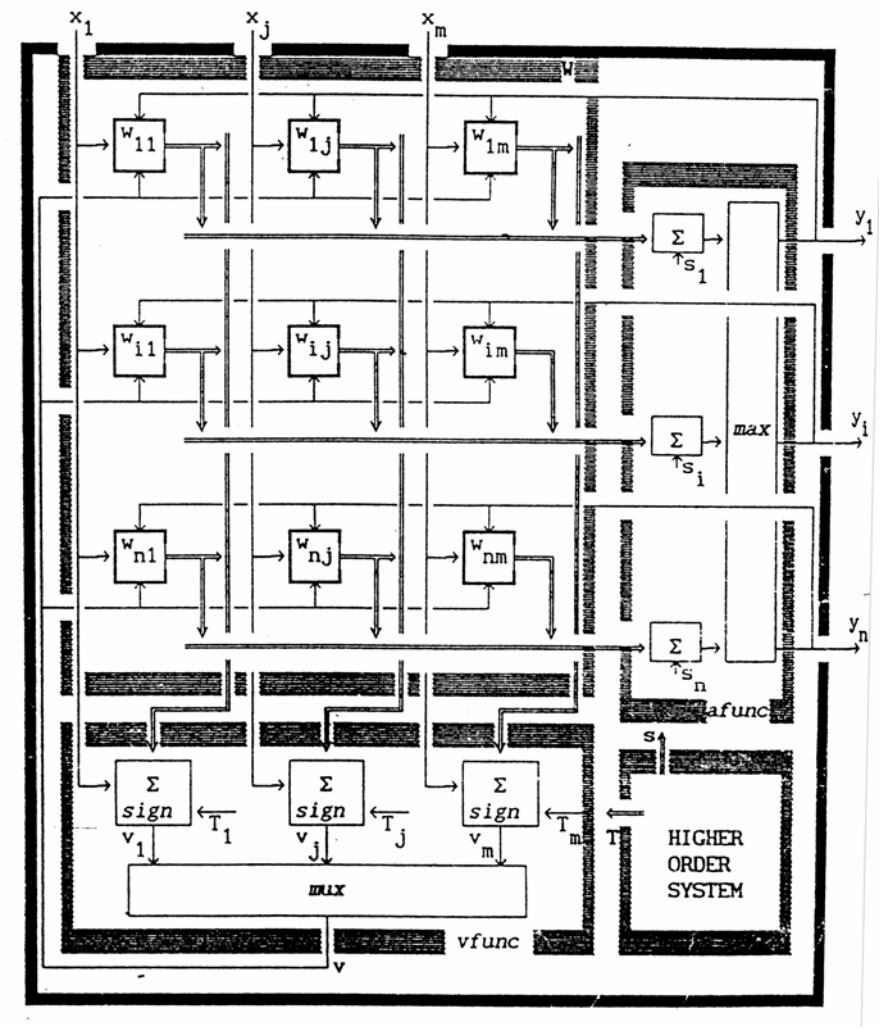

Figure 1: The CAA architecture

The CAA has a random walk searching mechanism implemented as a random number generator with uniform distribution. Since there is a path to the goal state by assumption, then there is a probability that the goal will be found. As the number of steps in a trial approaches infinity, the probability of finding a goal state approaches unity [3].

The time complexity of online search strongly depends upon the size and the structure of the state space, and upon a priori knowledge encoded in the agent's initial parameter values. When a priori knowledge is not available, search is unbiased and can be exponential in time for some state spaces. Whitehead [10] has shown that for some important classes of state spaces reaching the goal state for the first time, moving randomly, can require number of action executions that is exponential in the size of the state space. Because of this, the state spaces which are concerned here (described above) are with additional assumption that the number of transitions between 2 states from the starting state to the goal state, in every iteration is linear function of n (the number of states).

The agent starts from the starting state and should achieve the goal state, avoiding the undesirable states. From each state the agent can undertake one of maximum m actions, which can lead to another state or to a certain obstacle. The agent moves through the environment randomly, and after a few iterations it learns a policy to move directly from the initial state to the goal state, avoiding the undesirable states, i.e. it learns one path. After that it learns the optimal solution, the shortest path [3], [9].The criterion of optimality is defined as minimum path length. By *path* we mean a sequence of arcs of the form (j_1, j_2), (j_2, j_3), (j_3, j_4),..,(j_{k-1}, j_k). By *length of a path* we mean the sum of the lengths of its arcs. The *shortest path* is the path with minimal number of arcs.

The framework for considering the CAA agent environment relation is the two – environment framework. The environments assumed here are: 1) the genetic environment by which the agent receives hereditary information and 2) behavioral environment, or some kind of reality, where the agent expresses its presence and behavior Fig.2.

This framework assumes that they are performing some kind of mutual optimization process which reflects itself on the agent. There is an optimisation loop including the agent and the behavioral environment, and also an optimisation loop including the agent and genetic environment. The behavioral environment optimisation loop is actually the agent's learning loop: this process optimises the knowledge in the agents read/write memory. The genetic environment optimisation loop is a loop which optimises the read/only memory of the agent. That memory represents its primary intentions drives underlying behavior.

The task of the genetic algorithms (GA) research is to produce a read only memory in the genetic environment, produce an agent with that memory and test the performance of the agent in the behavioral environment. If the performance is below a certain level, a probability exists that the agent (organism) will be removed and another one will be generated. The objective of the optimisation process is to produce organisms which will express a high level of performance in the behavioral environment. The main interest is to construct an agent which will receive the genetic information and use it as a bias for its learning (optimisation) behavior in the behavioral environment. The genetic information received from the genetic environment is denoted as Darwinian genome. Additional assumption of this framework is that the agent can also export a genome. The exported genome will contain information acquired from the behavioral environment [3].

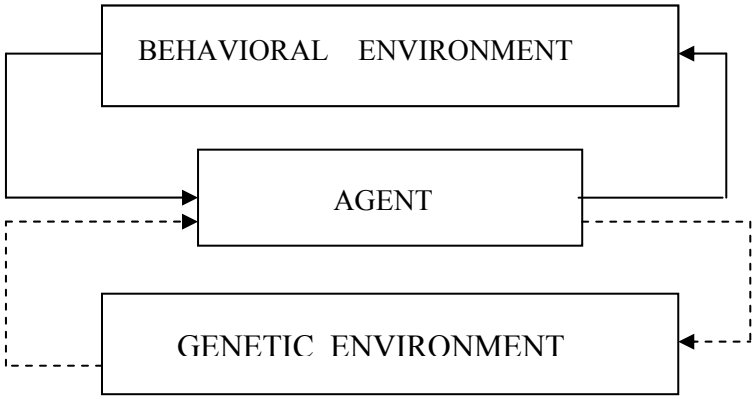

Figure 2: The two environment framework

The initial knowledge is memorized in the matrix W_{mxn}, Fig.1. The elements of the matrix W, w_{ij} (i =1,...,m; j =1,...,n) give information for states and are used for computing the actions, and they are called SAE - components (state - action - evolution). Each component w_{ij} represents the emotional value toward the performing action i in a state j. From the emotional values of performing actions, CAA computes an emotional value of being in a state. The elements of each column (j =1,...,n) give information for the states. The initial values of the elements of the column are all 0 if the state is neutral, with values -1 if the state is undesirable, and with values 1 if the state is a goal. (Here, on Fig. 2 number of rows is n, number of columns is m).

The learning method for the agent is defined by 3 functions whose values are computed in the current and the following state. They are: 1) *the function for computing an action* in the current state, 2) *the function for estimation of the internal state,* computed in the consequence state, and 3) *the function for updating the memory*, computed in the current state.

It means that when the agent is in a state j, it chooses an action with:
1) the function for computing an action in the current state, and here it is of neural type:

$$i = y_j = \arg\max_{a \in A(j)} \{ w_{aj} + s_a \}$$

A (j) is the set of actions in state j, s_a is the action modulation variable from the higher order system, which presents searching strategy. The simplest searching strategy is random walk, implemented as: ***s* = montecarlo[-0.5**, **0.5**] where **montecarlo**[interval] is a random function which gives values uniformly distributed in the defined interval. With this function, the agent selects the actions randomly. Having that, NN- CAA will perform a random walk until the SAE components receive values which will dominate the behavior.

2) the functions v_k, k=1, 2,...n for computing the internal, emotional value of being in a state in NN-CAA are computed in a "neural" fashion:

$$v_k = \mathrm{sgn}\left(\sum_{a=1}^{m} w_{ak} + T_k\right)$$

T_k is a neural threshold function (or warning function) whose values are:

$$T_k = \begin{cases} 0 & \text{if} \quad \eta_k = m \\ \eta_k & \text{if} \quad p_k \le \eta_k < m \\ 0 & \text{if} \quad \eta_k < p_k \end{cases}$$

where p_k is the number of positive outcomes, η_k is the number of negative outcomes that should appear in the current state. The threshold function T plays a role of a modulator of a caution with which CAA will evaluate the states which are on the way.

3) the learning rule in NN-CAA is defined by: $w_{aj} = w_{aj} + v_k$

SAE components in the previous state are being updated with this rule, using the desirability of the current state.

In such a way, using crossbar computation over the crossbar elements w_{aj}, CAA performs its crossbar emotion learning procedure which has 4 steps:

1) state j: perform an action depending on SAE components; obtain k
2) state k: compute state value using SAE components
3) state j: increment active SAE value using the k-th state value
4) $j = k$; go to 1

The experiment needs several iterations.

When the goal state is reached, the previous state is defined as a *subgoal.* The goal is considered a consequence of the previous state, from which the goal is reached. A subgoal is a state which has positive value for some of its elements w_{ij}. With the algorithm "at goal go back" the agent moves randomly until it reaches the subgoal (found in one of the previous iterations), and from that state it moves directly to the goal state, from where a new iteration starts (because all states after that subgoal are also subgoals and have positive values for some of its SAE component, so, from that subgoal state the agent moves directly to the goal state). With the algorithm" at subgoal go back" the agent doesn't go to the end goal state, but it starts a new iteration when it reaches a subgoal. The process of learning finishes when the initial state becomes a subgoal - with both algorithms. It means that at that moment the agent learnt *one path* - it learnt a policy how to move directly from the initial state to the goal state, avoiding the undesirable states.

The algorithms guarantee finding one solution, i.e. a path from the starting state to the goal. For solving the problem of the shortest path, another memory variable should be introduced, which will memorize the length of the shortest path, found in one *reproduction period.* The period in which the agent finds one path is called reproductive period, and in general, in one reproductive period the agent can not find the shortest path. After finding one path, the agent starts a new reproductive period when it learns a new path, independent of the previous solution. The variable *shortest path* is transmitted to the following agent generation in a genetic way. In this way the genetic environment is an optimisation environment which enables memorisation of the shortest path only, in a

series of reproductive periods, i.e. the agent always exports the solution (the path) if it is better (shorter) than the previous one. This optimisation period will end in finding the shortest path with probability 1. Since the solutions are continuously generated in each learning epoch, and since they are generated randomly and independently from the previous solution, then, as time approaches infinity, the process will generate possible solutions with probability 1. Among all possible solutions the best solution, the shortest path is contained with probability 1. Since CAA will recognize and store the shortest path length, it will produce the optimal path with probability 1 [3].

The CAA *At-subgoal-go-back* algorithm for finding the shortest path in a stochastic environment is given in the next frame:

```
CAA AT-SUBGOAL-GO-BACK ALGORITHM:

repeat
  forget the previously learnt path
  define starting state
  repeat
        from the starting state
        find a goal state moving randomly
        produce a subgoal state using CAA learning method
        mark the produced subgoal state as a goal state
  until starting state becomes a goal state
  export the solution if better than the previous one
forever
```

The main difference between this "At subgoal go back" and the original (1981) "At goal go back" algorithm is that in the original one new iteration starts always when a goal state is reached. Here the new iteration starts when a subgoal state is reached.

3. ESTIMATION OF THE COMPLEXITY OF THE ALGORITHMS

The initial knowledge for the agent is only the matrix W of the SAE components, and the environment is given by the matrix Y which gives the connection between the states in the graph; $y[i,j] = k$ means that when the agent is in the state j and chooses the action i, the consequence is the state k ($i = 1,...,m$; $j = 1,...,n$).

The domains which are concerned here are described above: with n states, some of which are undesirable; in each state the agent can undertake one of maximum m actions which can lead to another state or to some obstacle. Between some states there may be return actions. The number of transitions between 2 states, in every iteration, is linear function of n. The agent moves in the environment randomly, and after a few iterations it learns a policy to move directly from the initial state to the goal state, avoiding the undesirable states, i.e. it learns one path. After that it learns the optimal solution, the shortest path.

The complexity of the algorithms is estimated for the domain shown in Fig.3.

The starting state is the state 6, the goal state is the state 10, and the undesirable states are: 5, 13 and 20.

The complexity of the procedures which are common for both algorithms will be estimated first, and the complexity for the main program for each of the algorithms will be estimated after that.

Common procedures for both algorithms are:

(1) The procedure **compX.** This procedure is used for computing the function for choosing an action in every state *j*.

```
procedure compX(j: integer);
 begin
   for i=1 to m do
    begin
     x[i] = w[i,j] + random - 0.5
    end
  end.
```

The complexity for this procedure can be estimated by $\Theta(m)$

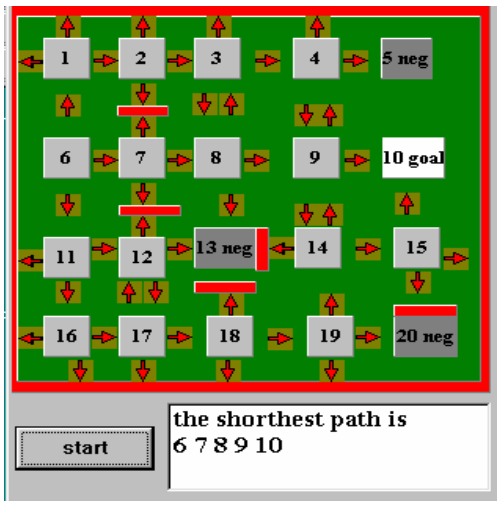

Figure 3: The domain for which the complexity of the algorithms is estimated

(2) The procedure **maximum** finds the index of the maximal element of $x(i)$ ($i = 1,\ldots,m$)

```
procedure maximum;
  begin
    max=1;
    for i=2 to m do
     begin
      if x(i)>x(max) then
      max=i
      end;
  end.
```

The complexity for this procedure can also be estimated with $\Theta(m)$.

(3) The procedure **compT** computes the values of the threshold function T which is defined in sec. 2

```
procedure compT (k:integer);
  begin
    neg=0; pos=0;
    for i=1 to m do
     begin
      if w[i,k]<0 then neg = neg + 1;
      if w[i,k]>0 then pos = pos + 1
     end;
      if neg = m then T=0;
      if neg<pos then T=0;
      if (neg≥pos) and (neg < m) then T = neg
   end.
```

The spending time for this procedure is also Θ (m).

(4) The procedure **compV** computes the function for estimating the emotional value for the state *k*.

```
procedure compV (k: integer);
 begin
  v[k]= T;
  for i=1 to m do
   begin
    v[k]=v[k]+w[i,k];
   end;
  if v[k]>0 then v[k] = 1;
  if v[k]<0 then v[k] = -1
 end.
```

The number of operations of this procedure can be also estimated by $\Theta(m)$.

(5) The procedure **solution** finds one solution - the path which the agent has learnt in one reproductive period and through which it moves directly to the goal, avoiding the undesirable states.

```
procedure solution;
  begin
    c = init;
    mark[end]='goal*';
    pat[k]=' ';
30   pat[k]=pat[k]+'c';
     compX(c);
     maximum;
      d=y[max,c];
     compT(d);
     compV(d);
     w[max,c]= w[max,c]+v[d];
     if mark[d] ≠ 'goal*' then
       begin
        c=d; goto 30
       end;
     pat[k]=pat[k]+'d'
    end.
```

This procedure can have at most $(am + b)(n - p - 1)$ operations, because there are $am + b$ operations for passing from one to another state, and the maximal length of the path can

be $n - p - 1$, (p is the number of undesirable states).The complexity can be estimated by $\Theta(n)$ (if we set $am + b$ = const.)

(6) The procedure **length** memorises the smallest length of the path which the agent has learnt, and decides whether to go to another reproductive period.

```
procedure length;
 begin
   if leng<l then l= leng;
   if s<Smax then
    begin
     {new reproductive period}
     s=s+1;
      goto 1
    end;
   Lmin=l; goto 200
  end.
```

This procedure has at most 5 operations.

3.1. The complexity of one reproductive period with the algorithm "at goal go back"

The algorithm "at goal go back":

```
  begin
    s = 1; l = n-p-1; {s - counts the reproductive periods;l-
    variable which memorizes the lengths of the paths, the initial
    value is the longest path}
1   W[i,j]=Winit [i,j](initial values );
3   a = init; {initial state}
7   compX(a);
    maximum;
    b = y[max,a];
    if b=0 then       {if b is obstacle }
      begin
        w[max,a] = -1;
        goto 7
      end;
    else
      begin
       compT(b);
       compV(b);
       w[max,a] = w[max,a] + v[b]
      end;
    if a = init then
      begin
       if[max,a]>0 then
       begin
        solution;
         length;
       end;
      end;
     if mark[b]='goal' then goto 3
     else
      begin
       if mark[b]='neg' then goto7
```

```
        else
         begin
          a=b; goto7
         end;
        end;
200 end.
```

The maximal number of operations for one reproductive period with this algorithm can be $(cm+d)(n+e-1)(n-p-1) + (am+b)(n-p-1)$, where $(cm+d)(n+e-1)(n-p-1)$ is the maximal number of operations from all iterations while the solution is found: it is the maximal number of operations of one iteration: $(cm+d)(n+e-1)$ $(cm+d)$ - the maximal number of operations for the transition from one to another state, $(n+e-1)$ is the length of the longest path by the random walk, for the domains which are described above (Fig. 3); e - the maximal number of repetitions of two states if there are return actions between them (the total number from all such places), $(n-p-1)$ is the maximal number of iterations, p - the number of undesirable states. The number of iterations is at most $n-p-1$, (for learning the longest path) because in every iteration, one state has a positive value of the function v for the estimation of the emotional state of the agent and after maximum $n-p-1$ iterations the initial state will have positive value for some $w[i, \text{init}]$, which means that agent has learnt one path through which it can move deterministically. $(am+b)(n-p-1)$ is the maximal number of operations in the procedure **solution.** So, the maximal number of operations can be:

$$\begin{aligned} f(n) &= (cm+d)(n+e-1)(n-p-1) + (am+b)(n-p-1) \\ &= (n-p-1)[am+b+(cm+d)(n+e-1)] \end{aligned} \tag{1}$$

If we set $am+b=$ Const1, $cm+d=$Const2, then the complexity can be estimated by $\Theta(n^2)$.

3.2 The complexity of one reproductive period with the algorithm "at subgoal go back"

The algorithm "at subgoal go back":

```
  begin
     s = 1; l = n-p-1; {s - counts the
        reproductive periods;l - variable which
        memorizes the lengths of the paths, the
        initial value is the longest path}

 1         W[i,j]=Winit [i,j](initial values );
 3    a = init; {initial state}
 7    compX(a);
      maximum;
      b = y[max,a];
      if b=0 then {if b is obstacle}
        begin
         w[max,a] = -1;
         goto 7
        end;
       else
        begin
         compT(b);
         compV(b);
         w[max,a] = w[max,a] + v[b]
```

```
      end;
    if mark[b]='goal' then
     begin
      mark[a] = 'goal';
      if a = init then
       begin
        solution;
        length;
       end;
      else
       goto3 {new iteration}
     end;
     else
      begin
        if mark[b]='neg' then goto7
         else
          begin
           a=b; goto7
          end;
      end;
200 end.
```

The maximal number of operations in one reproductive period is:

$$g(n) = (cm+d)[(n+e-1)+(n+e-2)+(n+e-3)+\cdots +(n+e-(n-p-1))]+(am+b)(n-p-1) \quad (2)$$

where $cm+d$ is the maximal number of operations for passing from one to another state, and there are at most $n+e-1$ transitions in the first iteration, maximum $n+e-2$ in the second iteration, and so on..., where e is the maximal number of repetitions of two states, if they have return actions (the total number from all such places). There are at most $n-p-1$ (p is the number of undesirable states) iterations, because in every iteration one state becomes a goal, it means that after maximum $n-p-1$ iterations the initial state will become a goal and the reproductive period in which the agent has learnt one path through which it moves directly, avoiding the undesirable states, will end. In fact, if we compare this algorithm with the algorithm "at goal go back", we can conclude that this algorithm is distinguished only in the length of the iterations. The procedure **solution** has at most $(am+b)(n-p-1)$ operations, and if we put in order the first term in (2):

$$\begin{aligned}
&(cm+d)[(n+e-1)+(n+e-2)+(n+e-3)+\cdots+(n+e-(n-p-1))]\\
&=(cm+d)[(n+e-1)+(n+e-2)+(n+e-3)+\cdots\\
&+(e+p+1)+(e+p)+(e+p-1)+(e+p-2)+\cdots\\
&+1-[(e+p)+(e+p-1)+(e+p-2)+\cdots+1]]\\
&=(cm+d)\left[\frac{(n+e-1)(n+e)}{2}-\frac{(e+p)(e+p+1)}{2}\right].
\end{aligned}$$

So, for (2) we have:

$$g(n) = (cm+d)\left[\frac{(n+e-1)(n+e)}{2}-\frac{(e+p)(e+p+1)}{2}\right]+(am+b)(n-p-1) \quad (2^*)$$

If we put $cm+d$ = Const1, $am+b$ = Const2, Const1, Const2 are some constants, we can estimate the complexity of the algorithm by $\Theta(n^2)$.

If we compare the maximal number of operations for one reproductive period of the algorithm "at goal go back" (1) with the maximal number of operations for one reproductive period of the algorithm "at subgoal go back" (2*), we can conclude that the algorithm "at subgoal go back" is faster only in a constant, i.e. for one reproductive period these algorithms have the same speed of convergence, they differ only in the multiplicative constants.(If we divide $f(n)$ and $g(n)$ we will obtain that "at subgoal go back" is 2 times faster than "at goal go back"). We can write:

$f(n)= \Theta(g(n))$.

4. ESTIMATION OF THE NUMBER OF REPRODUCTIVE PERIODS

The *adjacency matrix* [1] is used for estimation of the number of reproductive periods.

The adjacency matrix A_{nxn} is used for representing directed graph and it consists of n (number of nodes of the graph) rows and columns. Its elements are 1 or 0 and show the connection of the nodes in the graph. a_{ij}=1 if the node i is connected with the node j with one directed arc, and a_{ij}=0 if the node i is not connected to the node j. It is known that the element c_{ij} of the matrix A^k shows the number of paths with length k (k=1,2,..., n–1) from the node i to the node j in the graph [1]. This assertion will be proved and it will be used for the estimation of the number of paths from one node to another, depending on n - the number of nodes (and m - number of arcs from a node) in the graph.

Theorem 1: *If A is the adjacency matrix for the graph G with n nodes, then the element c_{ij} of the matrix A^k shows the number of paths with length k from the node i to the node j.*
Proof: (By mathematical induction): If A_{nxn} is the adjacency matrix of the graph G with n nodes, then the i-th row shows the connection of the i-th node with the other nodes, and from j-th column we can read which nodes have directed arcs entering j. In fact, from the elements of the matrix $A=A^1$ we can read the number of paths with length 1 between the nodes, because a_{ij}=1 if there is a path with length 1 (directed arc) from the node i to j.

1) For k=2: In $A^2=AA$, the elements c_{ij} are computed from multiplication of the i-th row with the j-th column: $c_{ij} = a_{i1}a_{1j} + a_{i2}a_{2j} + ... + a_{in}a_{nj}$. The product $a_{il}a_{lj}$ (l=1,...,n) is nonzero if both multiplicands are nonzero, i.e. if $a_{il} = a_{lj} = 1$, which means that there is a path with length 1 from the node i to the node l, and there is a path with length 1 from the node l to the node j, and so there is a path with length 2 from i to j. Every nonzero product in this sum is a path with length 2 from i to j through l (l=1,...,n), and the sum c_{ij} is the sum of all paths with length 2 from i to j.

2) Let us assume that the assertion holds for $k = s$, i.e. let the element b_{ij} of $A^s = [b_{ij}]$ is equal to the number of all paths with length s from the node i to the node j.

3) Let $k = s+1$.

The element c_{ij} of $A^{s+1}=A^sA$ is obtained with multiplication of the elements b_{il} from the i-th row of A^s with the elements a_{lj} (l=1,...,n) from the j-th column of A:

$c_{ij} = b_{i1}a_{1j} + b_{i2}a_{2j} + ... + b_{in}a_{nj}$. Every nonzero product $b_{il}a_{lj}$, (l=1,...,n) is the number of paths with length s+1 from the node i to the node j through the node l, because b_{il} is the number of paths with length s from i to l (by the inductive assumption) and a_{lj}=1 shows that there is an arc with length 1 from l to j; it means that there are b_{il} paths with length s+1 from i to j. The sum c_{ij} is the total number of paths with length s+1 from the node i to the node j. In fact, c_{ij} from A^{s+1} *is the number of paths with length s from the node i to those nodes l which enter the node j with arcs with length 1 - and that is the total number of paths with length s+1 from i to j.*

Since the agent learns one path in every reproductive period, it is necessary to express the total number of paths from the initial state to the goal state as a function of the number of states n (and the number of actions m in every state).

The following theorem gives that dependence.

The following theorem gives the estimation of the lower and upper boundary for number of paths with length s from one node to another one, in a graph with n nodes.

Theorem 2: *The number of paths with length s from the node i to the node j,* b_s^{ij} *, can be estimated* $cm\,k_{\min}^{s-2} \le b_s^{ij} \le mk_{\max}^{s-2}$*, for* $s \ge 2$*, where c is some constant, m is number of arcs which go out of i,* k_{min} *and* k_{max} *is the minimum and maximum number of arcs which can enter certain state in the graph. (ij are indices, s-2 is power)*

[Note: for s=1 b_s^{ij} **=1,** and then we have: $cm \quad k_{\min}^{1-2} \le 1 \le mk_{\max}^{1-2}$, i.e. $\frac{cm}{k_{\min}} \le 1 \le \frac{m}{k_{\max}}$. The left inequality holds for every m and $k_{\min}$, because c can be chosen arbitrary, but the right inequality holds for $m \ge k_{\max}$.]

Proof: First, we can prove $b_s^{ij} \le mk_{\max}^{s-2}$, by induction:

1) By the theorem 1 the number of paths with length 2 from the node i to the node j is the number of paths from i with length 1 to those nodes which enter j with arcs with length 1, and those are at most m paths. It means that the number of paths with length 2, $b_2^{ij} \le m = k_{\max}^{2-2}m$.

Since this estimation doesn't depend on j, it holds for all paths with length 2 from i. So,

$$b_3^{ij} = \sum_{q\ enters\ j} b_2^{iq} = b_2^{iq_1} + b_2^{iq_2} + ... + b_2^{iq_k} \le kmk_{\max}^{2-2} \le k_{\max}mk_{\max}^{2-2} = mk_{\max}^{3-2}.$$

k is the number of arcs which enter j, so the maximum number of addends in the sum is k. This estimation holds for all paths with length 3 from i.

2) Let

$$b_s^{ij} \le mk_{\max}^{s-2}.$$

3) Then:

$$b_{s+1}^{ij} = \sum_{q\ enters\ j} b_s^{iq} = b_s^{iq_1} + b_s^{iq_2} + ... + b_s^{iq_k} \le kmk_{\max}^{s-2} \le k_{\max}mk_{\max}^{s-2} = mk_{\max}^{s-1}$$

Now, we can prove $b_s^{ij} \geq cmk_{\min}^{s-2}$, by induction, also:

1) If $b_2^{ij} \neq 0,$ then we can estimate $b_2^{ij} \geq cm = cmk_{\min}^{2-2}$, c is constant. Since this estimation doesn't depend on j, it holds for all paths with length 2 from i.

If $b_3^{ij} \neq 0,$ then

$$b_3^{ij} = \sum_{q\ enters\ j} b_2^{iq} = b_2^{iq_1} + b_2^{iq_2} + ... + b_2^{iq_k} \geq k_{\min} cmk_{\min}^{2-2} = cmk_{\min}^{3-2}.$$

$k_{\min} \leq k$ **,** k is the number of arcs which enter j.

2) Let

$$b_s^{ij} \geq cmk_{\min}^{s-2}.$$

3) Then:

$$b_{s+1}^{ij} = \sum_{q\ enters\ j} b_s^{iq} = b_s^{iq_1} + b_s^{iq_2} + ... + b_s^{iq_k} \geq k_{\min} cmk_{\min}^{s-2} = cmk_{\min}^{s-1}$$

The number of all paths B from the initial state i to the goal state g is sum of all paths from i to g with length d, d+1, d+2,…., n–1–p, where d is the length of the shortest path, and n–1–p is the length of the longest path:

$$B = b_d^{ig} + b_{d+1}^{ig} + b_{d+2}^{ig} + ... + b_{n-1-p}^{ig} \geq$$

$$\geq ck_{\min}^{d-2} m + ck_{\min}^{d-1} m + ... + ck_{\min}^{n-p-1-2} m =$$

$$= ck_{\min}^{d-2} m(1 + k_{\min} + k_{\min}^2 + k_{\min}^3 + ... + k_{\min}^{n-p-1-2-d+2}) =$$

$$= ck_{\min}^{d-2} m(1 + k_{\min} + k_{\min}^2 + ... + k_{\min}^{n-p-d-1}) = ck_{\min}^{d-2} m\left(\frac{1 - k_{\min}^{n-p-d}}{1 - k_{\min}}\right) =$$

$$= cm\left(\frac{k_{\min}^{d-2} - k_{\min}^{n-p-2}}{1 - k_{\min}}\right) = cm\left(\frac{k_{\min}^{n-p-2} - k_{\min}^{d-2}}{k_{\min} - 1}\right)$$

B can be estimated by:

$$\Omega\left[m\left(\frac{k_{\min}^{n-p-2} - k_{\min}^{d-2}}{k_{\min} - 1}\right)\right].$$

S - number of reproductive periods is greater or equal to B, so S can also be estimated by:

$$S = \Omega\left[m\left(\frac{k_{\min}^{n-p-2} - k_{\min}^{d-2}}{k_{\min} - 1}\right)\right],$$

i.e. S is exponential function from n - the number of nodes.

5. CONCLUSION

Two algorithms: "at goal go back" and "at subgoal go back" which solve the shortest path problem, are described in this paper. The problem is solved with the agent architecture NN-CAA which implements the model of an emotion. To establish how much "at subgoal go back" is faster than the original "at goal go back", an estimation of their complexity has been made. The domains which are concerned are with n states, some of the states are undesirable, in each state the agent can undertake one of maximum m actions, which can lead to another state or to some obstacle. Between some states there may be return actions. The number of transitions between 2 states in every iteration is linear function of n.

It is obtained that the two algorithms find *one solution* (one path) efficiently, i.e. in polynomial time. The algorithm "at subgoal go back" is faster in a constant than the original one "at goal go back", because only the iterations are shorter in "at subgoal go back", but the function v which evaluates the emotion in every state is the same for both algorithms, so they have the same speed of convergence: $f(n) = \Theta(g(n))$ (they differ only in a constant, for one reproductive period). But finding *the optimal solution, the shortest path*, with the two algorithms is in exponential time. It is proved that the number of paths from one node to another one in a graph depends exponentially on the number of the states n in the graph.

It might be concluded that the concept of subgoal is one step forward toward the optimality of the process of the agent learning, but still, it should be explored further on, to obtain efficient, polynomial algorithm; the changes can be made in the function v which is decisive in the process of learning.

REFERENCES

[1] Bang-Jensen, J., and Gutin, G., *Digraphs, Theory, Algorithms and Applications*, Springer-Verlag London Limited, 2001.

[2] Blondel, V., and Tsitsiklis J., "A survey of computational complexity results in systems and control"*, Elsevier IFAC Publication "Automatica"* 36, Oxford, 2000.

[3] Bozinovski, S., "Consequence driven systems, teaching, learning and self-learning agents", *Gocmar Press, Amherst 1995.*

[4] Goldsmith, J., and Mundhenk, M., "Complexity issues in Markov decision processes"*, Proceedings of IEEE Conference on Computational Complexity*, 1998, 272-280.

[5] Mansour, Y., and Singh, S., "On the complexity of policy iteration", *Proceedings of the 15-th Conference on Uncertainty in Artifical Intelligence,* Stockholm, Sweeden, 1999.

[6] Mundhenk, M., Goldsmith, J., and Allender, E., "The complexity of the policy existence problem for finite horizon partially-observable Markov decision processes"*, Proceedings of the 25-th Mathematical Foundations of Computer Sciences,* Springer Verlag, 1997, 129-138.

[7] Orponen, P., "Computational complexity of neural networks: a survey*", Nordic Journal of Computing,* 2 (1), Helsinki, Finland, 1995, 77-93.

[8] Papadimitriou, H.C., *Computational Complexity*, Addison -Wesley Publishing Company, San Diego, California, 1994

[9] Petruseva, S., and Bozinovski, S., "Consequence programming: the algorithm 'at subgoal go back' ", *Mathematical Bulletin*, book 24 (L), 2000.

[10] Whitehead, S.D., "Reinforcement learning for the adaptive control of perception and action*"*, PhD thesis, University of Rochester, 1992.

[11] Wilf, H.S., *Algorithms and Complexity*, Prentice-Hall, New Jersey, 1986.

[12] Wooldrige, M., "The computational complexity of agent design problems", *Fourth International Conference on Multi-Agent Systems (ICMS)*, IEEE Press 2000.