Two fundamental problems connected with AI

Abstract

This paper is about two fundamental problems in the field of computer science. Solving these two problems is important because it has to do with the creation of Artificial Intelligence. In fact, these two problems are not very famous because they have not many applications outside the field of Artificial Intelligence.

In this paper we will give a solution neither of the first nor of the second problem. Our goal will be to formulate these two problems and to give some ideas for their solution.

Introduction

Since year 2000 we have a definition of AI [1,2,3] and since 2005 we have a program which satisfies this definition [4, 5]. Actually, these two facts are not very popular, first because the definition of AI is no accepted from almost no one except its author and second because the program which satisfies the definition of AI is useless from the practical point of view due to the combinatorial explosion.

From theoretical point of view we divide the programs in two types. The first are the non-terminating programs which will work infinitely long and the second are the terminating programs which will stop after a finite number of steps. On the other hand, from practical point of view, we divide the programs in ones that work in real time and ones which cannot work in real time. So, the fact that one program is a terminating one is useless for practical purposes if this program will work practically for infinitely long time.

That is why the program which is described in [4, 5] has no use for practical purposes and no one recognises it as AI because it does not satisfy the major requirement which is to work in real time. Even the program from [4, 5] is represented only as an algorithm. It is not written as a program because it is useless to write a program which will terminate after the end of the universe.

Therefore, if we want to make a program which will be recognised as AI we have to correct the program from [4, 5] and make it work in real time. Here we have to deal with the problem of the combinatorial explosion. Even in this case the term "combinatorial explosion" is not very proper because we use this term for the cases when a programmer writes a program which should work in real time but, actually, is not working. Also, we usually assume that when we have a combinatorial explosion a faster computer can eventually help us solve the problem. In this case the situation is different. We have an algorithm which is not designed to work in real time. There is not any attempt to make the algorithm faster. The main priority has been to make the description short and clear without taking into consideration the efficiency because it is obvious that this algorithm has only theoretical value and that it will never work as a real program.

Example with the perfect compression program

So, our task is to make a real program from one algorithm which is not designed to work in real time. Actually, the algorithm in [4, 5] describes the perfect AI but we need a working AI, which does not need to be perfect.

We have a similar problem with the perfect compression algorithm and real compression programs. Let us define the perfect compression algorithm in order to see how little the connection between it and the real compression programs is.

Here perfect compression algorithm is called the algorithm which enumerates all programs and returns the first one (i.e. the shortest one) which generates the string which has to be compressed.

There are two things to note here. First, we have to mention that this algorithm is a non-terminating one due to the undecidability of the halting problem. In order to make it a terminating one we have to add a requirement for efficiency of the program which we search for. We can say: "the first one which generates the string for no more than ***N*** steps" but we do not want to include an additional parameter ***N*** in the definition. That is why we will say: "the program which generates the string and which has the minimal sum between its length and the number of steps which it makes while generating the string". With such correction we will obtain a compression algorithm which is a terminating one from the theoretical point of view. (Anyway, this algorithm is non-terminating in practice and therefore it is useless.)

On second place, this algorithm generates the perfect self-extracting compression file but if we assume that we have a decompression program then a shorter data file may exist, which will return our string if we input this data in the decompression program. This means that here we are talking only of self-extracting compressions.

So, we have the perfect compression algorithm. We do not say the perfect compression program because no one wrote this algorithm as a program because this is useless work. The description of this algorithm can be obtained directly from the definition of Kolmogov's complexity [10]. This means that we can say that Kolmogov is the author of the first compression algorithm but maybe this is not correct because this algorithm cannot work in real time. Today we have many programs which make compression (including self-extractable compression). These programs are not perfect but they can work in real time. Actually, these programs are much more complicated than the perfect compression algorithm and you cannot construct them directly from the perfect compressor because they are based on totally different principles.

The situation is similar with the perfect AI and the real AI. We have the perfect AI but we cannot extract a real AI which will be able to work in real time directly from it. This comes to show how difficult our task to make a real AI is.

Dividing the problem in two parts

In order to construct a real AI we will divide its work in two parts. The first part is to find a good model of the world and the second part is to choose the best action on the basis of the selected model.

Actually, in the perfect AI these two parts are not separated. We will remind that the perfect AI from [4, 5] works by trying all possible strategies in all possible models and chooses the best strategy with the biggest average result (the average result is calculated on the basis of all possible models). So, the perfect AI solves these two tasks jointly, without separating them. Nevertheless, the separation of this two problems is natural and we will make it.

If we have real time solution of both these problems then we will have a real AI. Unfortunately, both these problems lead us to a combinatorial explosion. These two problems are not very famous because they do not have many applications outside the field of the Artificial Intelligence.

We will start with the second problem which is more famous and better studied.

Finding of the correct action on the basis of a given model

We have an algorithm for solving of this problem. The name of this algorithm is Min-Max and we use it with great success in Chess playing programs. Nevertheless, this algorithm is not proper in all cases because sometimes it gives a combinatorial explosion. Actually, it gives combinatorial explosion even with chess but in this game we can go around the combinatorial explosion by limiting the depth in which we examine the tree of the game. This is possible with the game of chess because we can make good evaluation of the position on the basis of things like the number of pieces on the board and on the "territory" which these pieces cover. Therefore, in some cases this problem is solvable in real time but not in all cases.

A famous example is the problem how to make a program which can play the Go game well enough to beat a professional player. A price of one million dollars was offered for working out this problem [11]. Unfortunately, the prize was not taken because the problem is too complex. The Go game looks like the chess but in it you cannot apply the Min-Max algorithm directly because you do not have a good evaluation function for the positions. The problem is that we have too many possible moves and mostly because in the

Go game after many moves nothing essential happens (nothing which can be easily detected by a simple evaluation function).

As we said at the beginning, we will not give a solution to this problem. This is not because the One Million Dollar Prize has already expired but because we do not know how to solve this problem. Anyway, we will give some ideas. The main idea is to define intermediate goals and large steps. Actually, intermediate goal is used in the chess playing programs where this goal is to increase the value of the position. Unfortunately, this intermediate goal is given by the programmer but for AI this goal should be generated automatically because AI cannot depend on a programmer to say what is right to be done in each case.

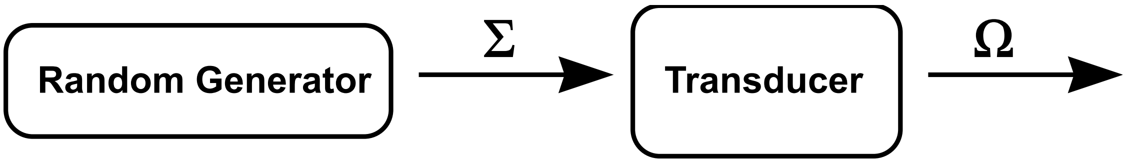
What is to think in large steps. This means to plan a chain of intermediate goals which leads to the main goal. Here we will say "goal" for events which we evaluate as good ones. One event can be evaluated as a good one by apriory or because it is part of a chain which leads to an event which has already been evaluated as a good one. So, thinking by large steps will be planing chains of events. For such planing we can use the Min-Max algorithm but here the problem is how to define events automatically and how to automatically find the way for transition from one event to another. For example, with the game of chess you have events "taking of enemy piece" and "winning the game". There is a connection between these two events and this connection is built in the chess playing programs by their creators. So, the chess playing program tries to take enemy pieces in order to win the game. The problem is how to make a program which defines events automatically and automatically evaluates these events as good or bad. Also, AI has to be able to automatically find connections between these events in order to plan a chain of events.

Actually, all these thoughts lead us to the fact that in order to solve the second problem we need a solution of the first one because in order to think in large steps we need an automatic detection of events and this is part of the problem of finding a good model of the world. From this point on we will talk only about the solution of the first problem.

Formalisation of the first problem

Here when we say a good model we mean an adequate one. So, this means a model which will give a correct predictions for the future.

The first step in solving a problem is to formalise it. Let us examine the following formal problem. Let us have a two finite alphabets Σ and Ω. Let us have a random generator which generates letters from the alphabet Σ and a transducer which inputs a letter from Σ and outputs a letter from Ω. The goal is to built a model of this transducer which will give us the possibility to guess its next output if we know the input letter and if we know the entire history (i.e. if we know the row a_0, b_0, a_1, b_1, ... , a_{n-1}, b_{n-1}, a_n where a_i are the letters from the random device which are inputted in the transducer and b_i are the letters which are outputted.) So, the question is what will be the output on the step n if we know all data from step 0 to step n-1 and the input on the step n. In other words, what will be b_n if we know a_0, b_0, a_1, b_1, ... , a_{n-1}, b_{n-1}, a_n.



What is the connection between this formalisation and the definition of AI [1,2,3]? Here we have a random generator and transducer, which interact. In the definition of AI the transducer corresponds to the concept of World. Here we try to make a model of the transducer but in [1,2,3] AI tries to understand the World. This means that here the random generator corresponds to the AI from [1,2,3]. Where is the difference? AI reads the output of the World (of the transducer) but the random generator does not have any input. AI is able to carry out some experiments in order to understand the World but the random generator does not make any intentional experiments. Anyway, if the observer waits long enough the random generator will make all experiments (accidentally).

Note: In [1,2,3] the alphabets Σ and Ω are Ω and Σ and the letters ***a*** and ***b*** are ***d*** and ***v***. This can cause confusion in understanding the connection between [1,2,3] and this paper.

How to find a good model of the world (transducer)

So, we have a formalisation and now our problem is formal. As we said this problem is not famous because it has not many applications outside the field of AI. It is even difficult to find an example for a practical problem which leads to this theoretic formalisation. The only such example which we have in mind is the following. Let us have a program protected against illegal use by a hardware key device. If we want to break this protection we have to understand how this hardware key works and try to recreate it. Really, the practical problem allows us to open the key and see how it is designed but here we assume that we have no such possibility and that we have to observe the key as a black box and to study only its input and output.

As you see, there are not many applications of this problem. Maybe this is the reason that nobody offers a price for its solution but nevertheless, here we will discus this problem.

So, is this problem solvable. In the general case the answer is no because if we do not make any suggestions about the transducer then we will not be able to say anything about its next output. For example, if it outputs one and the same letter one hundred times in a row irregardless of the input then we can predict that on the next step it will workout the same letter. This prediction looks natural but it lies on the conjecture that the simpler explanation is more probable than the complicated one. Without this conjecture we cannot make any prediction because it is possible that in this case we have a transducer which outputs one hundred times one and the same letter and on the next step it outputs another letter.

We said that we will look for the simplest model of the transducer. Also, we have to bear in mind that we need a solution which works in real time.

Another question is whether our transducer is deterministic device or not. It will be much easier if we assume that the transducer is a deterministic device but if we restrict our search only in the set of deterministic models then the chance to find a proper model in a concrete situation is very small.

Next question. How many internal states our transducer has. It is reasonable to suggest that the number of internal states is finite (i.e. that it is finite automata). Anyway, the more general case is to suggest countably many internal states. It is no use suggesting an uncountable number for the internal states because only a countable subset of them will be obtainable in the deterministic case. In the non-deterministic case there is some use in suggesting an uncountable number of internal states but if we restrict our observation to the set of calculable functions then again there is no use suggesting an uncountable number for the internal states of the transducer.

The last question. Is our transducer a calculable function or not. Definitely yes. We are looking for a practical solution so it has to be a calculable function and even it has to be an easy calculable function (i.e. calculable for small number of steps without problems like combinatorial explosion). Beside that, every non-calculable function can be approximated with a calculable one (of course, until the concrete moment n but not until the infinity).

One theoretical solution

Here we will give the next useless theoretical solution which cannot work in real time. The reason that we give this solution is to show that such one exists. This is important because we cannot give a solution which can work in real time. Instead of that at the end of this paper we will give some ideas about the creation of real time solution.

Here is our theoretical solution. First for the deterministic case:

It will enumerate all programs and will return the first one (i.e. the shortest one) which generates the row b_0, ..., b_{n-1} if the input is a_0, ... , a_{n-1}. Here we have a problem with undecidability of the halting problem again. So, we will take not the shortest one but this which has minimal sum between its length and the maximum number of program steps which it needs to generate any of the outputs (i.e. any of b_0, ..., b_{n-1}). So, this algorithm will give us a short and quick program which makes a very good prediction of b_n. The only problem is that we will have to wait this algorithm to finish almost forever.

For the non-deterministic case we have to complicate our algorithm a little bit.

First we will complicate our programs (which we use as models) by adding one subroutine **random()** which will return zero or one with possibility 1/2. With this subroutine we cannot generate even the possibility 1/3 but by using subroutine **random()** we can approximate any possibility (nevertheless is it rational or irrational number).

Now, when we deal with non-deterministic models we cannot say simply yes or no to the question does this model generate our sequence or not. Instead of that, we can calculate the possibility for our sequence to be generated. Of course, here we will have the problem with the non-terminating models again and in order to keep things calculable we will add one constant ***Max*** and we will calculate the possibility of the model to generate our sequence for no more than ***Max*** steps per output.

What is the prediction of one non-deterministic model for b_n. First we do not know what is the internal state of the model when it inputs a_n because there may be more than one possible way for this model to generate b_0, ..., b_{n-1}. Even if we know the internal state we cannot say which letter will be worked out as b_n because our model is non-deterministic. Nevertheless, we can calculate for concrete model the possibility for every letter to be worked out.

Every model will give us some prediction but we have to choose which one to trust and which prediction to accept as the better one. This question will not be discussed in this paper.

Some ideas about the practical solution

First, in order to make real time solution we will restrict the observation to the set of models with a finite number of states (finite automata). Of course, this restriction is essential because some of the worlds (transducers) cannot be described with a finite models. Anyway, in many cases the finite models are sufficient or at least they can give a good approximation of the World. (You can find in [7] the idea that we can raise the finite models with first order axioms in order to make models for more complex worlds.)

Second, we have to mention that we will look for a set of good models instead of a single model. The chance to find a single model which describes the world is small. It is more probably to find many different simple models which describe different features of the world. Also, in this way our system will be more consistent because in its life (work time) it will change some of the selected models instead of changing the only model which can make its behaviour totally different.

Now, let us start with the case of deterministic models. Such model looks like a deterministic finite automata (with finite number of states, starting state, arcs labelled with the letters from Σ, etc.) but here we will have only one type of states (no final states) and we will have a second label on every arc which will be a letter from Ω.

If we have such a model with a reasonable number of states we can easily find it by a backtracing algorithm similar to the one from [8]. Anyway, the existence of such model is very suspicious because if we have deterministic model then we will be able 100% correctly to predict the future. This will mean that the world is very simple, which is not the interesting case.

Let us look for a non-deterministic model of the world. Actually, as we said, we will look for a set of many non-deterministic models.

We will divide the non-deterministic models in two groups - partially deterministic and totally non-deterministic. Examples of these two types of non-deterministic models are found in [6, 7].

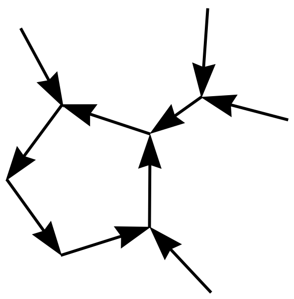
Partially deterministic models will look like the deterministic models but with the difference that they will have a second label on the arcs, which is a set of letters from Ω instead of one letter. Actually, this will be a set of 2-tuples from letter and possibility because every letter from Ω will have its own possibility to be worked out in the case when the model is in the respective state and the input letter from Σ is that which is the first label of the arc.

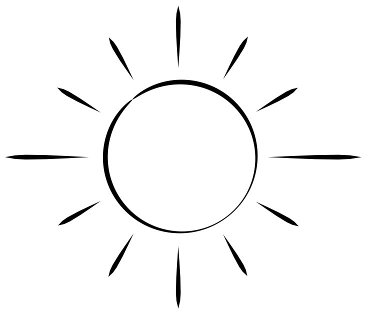
The good side of partially deterministic models is that their current state is determined. From every deterministic automata on alphabet Σ we can make partially deterministic model by defining the possibilities through statistics on the basis of life experience (a_0, b_0, ... , a_{n-1}, b_{n-1}). In fact, statistics will not give us the possibility but the number of times certain letter is worked out in certain situation. In this case (1, 1) is different from (30, 30) because in both cases we have 50% possibility but in the second case this is more certain.

So, we have so many partially deterministic models and the question is which of them are better. This is a very difficult question and we will not discus it here. We will say only that if one model gives in some cases (i.e. arcs) prediction which is useful (for example 100% possibility for certain letter) and reliable (i.e. this link is used many times) then this model is useful.

How to find a good partially deterministic model. First we need a definition which strictly says which model is better. The second problem is that we have to search for this model in huge set of possible candidates.

The idea which we will give in this paper is to observe the behaviour of a single letter. We will call this method the "sunshine" method for finding finite automata. This idea is based on the fact that if we observe only the arcs which have a certain letter as a first label these arcs make one or more figures which we will call "Suns". The Sun is a cycle with paths which flow in it. This figure looks like the picture of the sun which children use to draw.





The idea is that we will be able to relatively easy detect the dependency in the figure Sun and after constructing several suns to construct the finite automate from these suns. In order to catch dependencies for one letter we will need to observe long sequences of this letter. We may wait long until the random generator generates such sequence (especially if the alphabet Σ is big which is the general case). That is why we will use elimination of letters and construction of compound letters. Elimination of letters is when we assume that some letters do not change the state of the model. Compound letters are sets of letters which we accept as one letter. In [6, 7] we have an example of partially deterministic model where we use letters "left" and "right". There we assume that all other letters do not change the state of the model (i.e. these letters are eliminated). In the next model in [6, 7] we have the letter "victory or loss" which is compound. Actually, this compound letter is not from Σ but from Ω. Really, in the deterministic model there is no sense to include the output of the transducer as information our model depends on but in a non-deterministic case this information is essential and it is reasonable to use it in our model.

Bibliography

[1] Dobrev D. D. AI - What is this. In: PC Magazine - Bulgaria, November'2000, pp.12-13 (in Bulgarian, also in [9] in English).

[2] Dobrev D. D. AI - How does it cope in an arbitrary world. In: PC Magazine - Bulgaria, February'2001, pp.12-13 (in Bulgarian, also in [9] in English).

[3] Dobrev D. D. A Definition of Artificial Intelligence. In: Mathematica Balkanica, New Series, Vol. 19, 2005, Fasc. 1-2, pp.67-74.

[4] Dobrev D. D. Formal Definition of Artificial Intelligence. In: International Journal "Information Theories & Applications", vol.12, Number 3, 2005, pp.277-285.

[5] Dobrev D. D. Formal Definition of AI and an Algorithm which Satisfies this Definition. In: Proceedings of XII-th International Conference KDS 2006, June, 2006 Varna, Bulgaria, pp.230-237.

[6] Dobrev D. D. Testing AI in One Artificial World. In: Proceedings of XI-th International Conference KDS 2005, June, 2005 Varna, Bulgaria, pp.461-464.

[7] Dobrev D. D. AI in Arbitrary World. In: Proceedings of 5th Panhellenic Logic Symposium, July 2005, University of Athens, Athens, Greece, pp. 62-67.

[8] Dobrev D. D. First and oldest application, http://www.dobrev.com/AI/first.html

[9] Dobrev D. D. AI Project, http://www.dobrev.com/AI/

[10] Kolmogorov A. N. and Uspensky V. A. Algorithms and randomness. - SIAM J. Theory of Probability and Its Applications, vol. 32 (1987), pp.389-412.

[11] The Million Dollar Prize http://www.reiss.demon.co.uk/webgo/million.htm