Parallel between definition of chess playing program and definition of AI

Dimiter Dobrev
dobrev@2-box.com

24 of May, 2006
this paper is a part from AI - Project
illustration - Konstantin Lakov

In this report we will explain some earlier papers [1, 2] which are about definition of Artificial Intelligence and about perfect AI. The definition of AI is intuitive in [1] and formal in [2]. The perfect AI is a program that satisfies the definition for AI but which is absolutely useless due to the combinatory explosion.

Most people do not understand these papers because they never saw AI and that is why for them the notion of AI is too abstract. In this report we will make parallel between definition of chess playing program and definition of AI. Of course, the definition of chess playing program is useless because people already know what this is. Anyway, we will give you this definition because its construction follows closely the construction of the definition of AI. Also the results are almost the same with the only difference that we can optimise the perfect chess playing program in order to obtain a real chess playing program, but for the moment we cannot optimise the perfect AI in order to obtain a real AI.

In this report we will not speak about AI. The only matter which we will observe will be about chess playing programs. If you understand the construction and the results about chess playing programs then you can read the papers [1, 2] and to see similar results about AI.

What is a chess playing program?

For us a chess playing program (CPP) will be a step device which inputs the move of the opponent and outputs a correct move as an answer on every step. If this

was the only requirement, then the random player would be chess playing program, but this is not true because the random player plays too bad. That is why we will want from the chess playing program to play no worse than a human being.

Here you can ask the question: "Who is the human being?" The answer is that this is not so important because the difference between the average chess player and the world chess champion is not that big. This is because the program which plays better than the average chess player is almost the same as the program which plays better than the world chess champion. The main difference is in the number of moves which are calculated in order the next move to be chosen. Of course, the second program needs more time or faster computer than the first one.

The definition which includes the world human is not formal but it can be formalised.

To make the definition formal first we have to formalise the opponent. We will suppose that the opponent of our chess playing program is the random player. This is the opponent which every time plays a random move. Of course, you can have many different random players which have different possibility for choosing their random moves, but we will assume that our random player chose its random move with equal possibility. That means that if this player has ***N*** possible correct moves, then the possibility for each of them to be chosen is ***1/N***.

The random player is very weak opponent, but, anyway, it is a dangerous partner because sometimes it makes genius moves. Really, the possibility to make a genius move is pretty small. The good side of the random player is that it is simple and well defined. Another positive feature of the random player is that it has no memory. This means that it cannot learn and that the next game will not depend on the previous.

We will make some additional assumptions. We will assume that the rules of chess are fixed and that these rules do not allow infinite game (for example, if 20 moves no pawn is moved and no figure is taken then the game is draw). We will assume that our CPP plays with whites in every game.

We may assume that our CPP is a deterministic program (i.e. that it plays only one strategy). If we assume that CPP is nondeterministic program and that it plays many different strategies, then we have to know what is the possibility of each possible strategy. In the first case the average success of CPP will be the average success of its strategy. In the second case its average success will be the average from the average successes of all strategies (the sum of ***AverageSuccesses(i).Possibility(i)*** where ***i*** runs true all strategies of CPP). Notice, that the number of all strategies is finite due to the fact that the tree of the game is finite, which is so because the rules of chess do not allow infinite games.

Now we have to ask the question about the goal of the game. In the case of AI this question was difficult and we had to introduce the notion of meaning of life. Here we

have no such problem because the goal is obvious and it is to win the game. Speaking more formally, let us give to CPP one for victory, zero for lost and half for draw game. The goal will be to make the biggest average success.

The perfect CPP

After this formalisation we can say when one CPP is better than another and even something more, we can show the perfect chess playing program (or the perfect chess playing strategy, which is the same if the program is deterministic). The perfect CPP will calculate the tree of all possible moves before making any move, and that is why this program will be practically useless due to the combinatory explosion. Anyway, it is interesting from the theoretical point of view to have such a program.

The perfect CPP will evaluate all positions in the tree of the game by **Max-Sum** algorithm (maybe in this case it is better to call this algorithm **Max-Average** instead of **Max-Sum**). This algorithm is theoretically possible because the tree of the game is finite. The **Max-Sum** algorithm evaluates first the leaves of the tree and gives one for these leaves whose position is victory for CPP, zero respectively for victory for the opponent and half for draw positions. After this, **Max-Sum** algorithm evaluates the rest of the vertexes calculating maximum from the evaluation of the successors when the move is made by CPP and calculating average when the move is made by the opponent. Of course, after this evaluation **Max-Sum** algorithm plays by selecting this move which leads to the vertex with maximal value. (Of course, if vertexes with maximal value are more than one the **Max-Sum** algorithm chooses one of them. Because of this choice we can have more than one perfect strategy and also we can have nondeterministic perfect CPP which combines several perfect strategies.)

The perfect CPP is perfect but useless due to the combinatory explosion. It is easy to see that this program will make the best average success against the random player. Anyway, the perfect CPP is a little bit strange. If it is in a winning position, then it will win, which is OK, but if the perfect CPP is in draw position, then it can lose the game. Actually, perfect CPP can move from draw position to losing position, which is strange, but this is because the perfect CPP is perfect against the random player. If the opponent was different then the perfect CPP would be different.

Possible variants of the definition

Now we have the perfect CPP and the question is how to define what is a CPP. We can say that this is any program which is as good as the perfect CPP. At least the

perfect CPP covers this requirement. This is not a good idea for definition because we need a program which can play in real time. Also, we do not need a player so perfect as the perfect CPP.

That is why we will define CPP in the following way:

First variant of the definition: CPP will be the program which makes average success which is on smaller distance than ε from the perfect.

If this ε is chosen smaller enough, then the chess playing program will play no worse than a human.

Here we have one problem. We do not know how big is the perfect result. If the starting position in chess is winning for the whites, then the perfect result is one (or $1 - \delta$ where δ is zero). If the starting position is not winning for the whites, then the perfect result is $1 - \delta$, where δ is some very small number bigger than zero. We have an algorithm for calculating the value of δ, but we cannot calculate it because this algorithm will not end in reasonable time (i.e. before the end of the universe). The conclusion is that the previous definition is not convenient because it includes two constants whose value cannot be calculated (δ cannot be calculated due to the combinatory explosion and ε cannot be calculated at all because it depends on the human beings, which means that this constant has no exact value). Even if we choose a value for ε we will still not know the value of the $\delta+\varepsilon$. This is the reason to define a CPP in a different way.

Second variant of the definition: CPP will be the program which makes average success which is bigger than $1 - \alpha$.

Here α is some small positive number. It must be equal to $\delta+\varepsilon$ in order to be the new variant of the definition the same as the previous variant. In any case α must be greater than δ because in the opposite case CPP will not exist.

The second variant of the definition gives us the possibility to check it for a concrete program by the tools of the statistics. There are two problems. First, we cannot say how big should be α for the CPP to exist and in order to play no worse than a human being. Of course, such value exists but we cannot say which it is. The second problem is that the value of α is very small, which makes it extremely difficult to calculate it by the methods of the statistics.

In order to eliminate the second problem we will change the opponent. We need a stronger player as opponent in order to have bigger expectation for $\delta+\varepsilon$. Let us try with human being as an opponent.

Third variant of the definition: CPP will be the program which makes average success against human being bigger than 1/2.

The third variant of the definition is not formal, but this variant is usually used as a definition of CPP. The good side of this definition is that it can be checked by the methods of the statistics. For example, if we make ten games between one program and a human being and if the average success of the program is 60% (0.6), then we can say with big possibility that this program satisfies the definition for CPP. Really

ten games are too few and for bigger certainty we may make 100 or even 1000 games. Here our expectation for $\delta+\varepsilon$ is 1/2. Really the game is not symmetric (opponent plays always with the blacks) and that is why we are not sure that human against human will make 1/2 average success. Also, you have to notice that this variant of the definition is not formal and that the results will depend on the humans we use for opponents.

New fixed opponent

Anyway, we need a formal definition of CPP and that is the reason why we will use a well-defined opponent. Such opponent is the standard chess playing program which calculates ***N*** moves in the tree of the game and by **Min-Max** algorithm selects the best move. The problem is that this opponent is deterministic. We prefer the opponent to be nondeterministic because if both CPP and the opponent are deterministic, then the result of the game between them is also deterministic (actually, there is only one possible game between them). In this case we cannot use the methods of the statistics. Also, it is not a good idea to use deterministic opponent for the definition because all deterministic programs which have average success one will be CPP but maybe some of them accidentally made victory in this only game, which does not prove that these programs are CPP.

For opponent we will use nondeterministic variant of the standard chess playing program. Let our opponent depend on one integer ***N*** and two functions - ***F*** and ***P***. Let function ***F*** evaluate positions and on every position ***F*** return an integer. Let function ***P*** give the possibility for one move to be chosen. This means that the input of ***P*** is a list of integers which represent the values of the positions which can be obtained on the next move and the output is a list of the possibilities each of these positions to be obtained. The integer ***N*** and the functions ***F*** and ***P*** should be concrete and fixed because they will be parameters in our definition. If we change one of these parameters, then the definition will also be changed.

How will this fixed opponent work. It will evaluate by the function ***F*** all positions which are obtainable after ***N*** moves from the current position. After this, by the **Min-Max** algorithm, it will calculate the values of the positions which are obtainable on the next move. After that, by ***P***, it will calculate the possibility of each of these positions to be obtained and on the end will randomly choose one of the moves, but in such a way that the possibilities of the next positions to be the same as the ones

given by ***P***.

This fixed opponent is nondeterministic (except the case when ***P*** gives ***1*** to one of the numbers and zero to the rest of them).

After the change of the opponent we also have change in the perfect CPP. It will be constructed almost in the same way, but instead of average value it will calculate the sum of ***Value(i).Possibility(i)*** where ***i*** runs true all possible moves of the opponent. Here ***Possibility(i)*** will be given by the function ***P***. (Of course, before applying the function ***P*** we have to use function ***F*** and **Min-Max** for ***N*** moves in order to calculate the values of all positions which are obtainable on this move.)

If the number ***N*** is big enough and if the functions ***F*** and ***P*** are reasonable, then our fixed opponent is a good chess player. This means that our expectation for δ and ε will be high. Let us assume that ε is 10%. This means that we suppose that a human being will make average success against the fixed opponent 10% worse than the perfect CPP. Let us assume that the average success of the perfect CPP is 80% (i.e. that δ is 20%). If the starting position is winning for the whites, then δ is zero but our expectation is that the starting position is not wining and not losing. This leads us to the following:

Final variant of the definition: CPP will be the program which makes average success against the fixed opponent which is bigger than 70%.

As we mentioned, this definition depends on the fixed opponent and from the number α, which here is fixed to 30%. If we fix also the number ***N*** and functions ***F*** and ***P***, then we will have one concrete definition. We can easily check whether one program satisfies the requirement of this definition by the methods of statistics. Really, we cannot check this for sure, but with a great percent of possibility (which percent grows with the number of the test games).

This definition is made with one assumption:

Assumption 1: Here we assume that the average success of the perfect CPP is about 80%. If this conjecture is true, then there exists a program which satisfies the definition (at least the perfect CPP does). If the average success of the perfect CPP is smaller than 70%, then there is no such a program (of course, in such case we can change this parameter and make it smaller than 70%).

Conclusions

The first concussion is that CPP exists and that the perfect CPP satisfies the demands of the definition. Really, this is true if assumption 1 is true, but if we take the fist variant of the definition then this is true without assumption 1.

The second conclusion is that the perfect CPP is CPP but it is useless due to the combinatory explosion. In order to obtain a real CPP we have to optimise the perfect CPP and make a program which may be a worse player than the perfect but which can play in real time. That is an easy task in the case of chess, but not so easy in the case of AI.

References

[1] Dobrev D. A Definition of Artificial Intelligence. In: Mathematica Balkanica, New Series, Vol. 19, 2005, Fasc. 1-2, pp.67-74. (The same in popular form)

[2] Dobrev D. Formal Definition of AI. In: IJ ITA, Volume 12, Number 3, p.277.