Anthropomorphic Artificial Intelligence

Claes Strannegård*

Abstract.

Human cognitive resources, such as the working memory, are subject to limitations. In the present note, a method is described for exploiting such limitations in the context of artificial intelligence. In order to obtain a notion of psychological difficulty for some sampled problems of a given problem class, psychological experiments are performed. A model is then constructed that divides the set of solutions associated with the problem class into layers that reflect the degrees of psychological difficulty. Any search algorithm can then be confined to these layers in an iterative deepening fashion. The purpose is to speed up algorithms locally on the set of problems that human beings tend to solve efficiently.

Introduction

Several definitions of the term artificial intelligence (AI) have been suggested since the term was coined in 1955 [MMRS55]. Some definitions make explicit reference to human beings and use e.g. the terms human intelligence, human thinking, or human rationality. Others refer to natural intelligence in a sense that includes a wide range of natural organisms and systems. Still other definitions are purely mathematical in nature and consider rational behavior in a formalized set-

* I would like to thank the following persons for valuable input: Thierry Coquand, Niklas Engsner, John Hughes, Per Lindström, Lance Rips, Kristofer Sundén Ringnér, and Johan Tavelin.

ting, where there are agents, environments, actions, goals and utility functions.

One tradition in AI uses methods that are inspired by biology. Examples include methods like artificial neural networks, genetic programs, cellular automata, and ant algorithms [CDM92]. Another tradition uses methods that are inspired by human problem solving. Examples of algorithms in this tradition are the general problem solver [NS61] and the logic theorist [NSH56]. In contrast to some the biologically inspired methods, these methods tend to be transparent in the sense that people can understand how the final programs work.

To obtain knowledge about human problem solving on a given problem class, two classical methods of psychology are available: introspection and experimentation. By doing introspection one may hope to identify some useful concepts, operators, methods, and heuristics. These components can then be put together into a computer model. By doing experimentation, the psychological realism of the computer model can then be evaluated. After a number of improvements of the model, one may eventually be able to validate it statistically.

When looking for inspiration in human raw-models, it seems reasonable to focus on the strengths of human problem solving. For instance, this was the idea underlying the development of expert systems in the 1970s [FBL71]. Thus one would exploit some powerful knowledge operators and clever heuristics that the human raw-models would use. After some years, it became increasingly clear that the time and space complexities of many expert systems imposed severe limitations on their usefulness in practice. Part of the blame for that could be put on the heuristic functions, which failed to confine the search to a sufficiently small portion of the search space.

In this perspective one may look for ways of confining the search space further. We are going to introduce psychologically relevant complexity measures on solutions and use those complexity measures to approximate the comprehensible part of the search space.

A vast amount of complexity measures exists in the literature of mathematics, computer science, logic, and linguistics. Examples include versions of Kolmogorov complexity, circuit complexity, proof complexity, and formula complexity.

To approximate the comprehensible part of the search space, we are first going to make experiments and measure comprehensibility by means of certain performance measures that are used in experimental psychology. Examples of such performance measures are the average time it takes to answer a problem correctly, the proportion of the subjects who answer the problem correctly, or the average level of difficulty of the problem, as estimated by the subjects of the experiment.

In cognitive psychology, a great number of models have been proposed for various aspects of human cognition. For instance, there are specialized models for human reasoning in the mental models tradition [Joh83] and the mental logic tradition [BO98]. There are also several models of human cognition that are presented in the form of computer programs. Examples of this include general cognitive models like ACT–R [AL98] and specialized models like the system PSYCOP [Rip96] for first-order reasoning.

In the following it is suggested that computer models of human cognition may be used, not only to shed light on human thinking, but also to develop algorithms with anthropomorphic behavior. A number of potential software applications based on these algorithms are then discussed.

Idea

In view of the limited practical usefulness of certain AI-algorithms, one may look for ways of confining the search space. To do this, one may again turn to human problem solving for inspiration, but this time considering not only the strengths, but also the weaknesses of human problem solving. After all, in many problem classes, people outperform the state-of-the-art programs, so by mimicking human raw-models even more closely, one may hope to push the limits further. Aiming to mimic also the weaknesses of human problem solvers a priori means aiming to match

the performance of the human problem solvers, but not exceed it. For problem classes where human beings tend to perform better than computers, however, that level of ambition is certainly high enough.

From cognitive psychology we know that people generally have severe limitations in their cognitive capacity. In particular, the working memory is typically only big enough to fit a 7-digit telephone number [Mil56]. These limitations contribute to the difficulty of understanding large numbers, complicated proofs, irregular patterns, etc. Therefore, in the context of problem solving, such limitations typically imply that a large portion of the search space will consist of solution-candidates that are incomprehensible to most people. This suggests that search should be specifically directed towards the comprehensible part of the search space. Thus one may aim for algorithms that perform well on the same problems as the human raw-models. In the following, this approach, which will be referred to as anthropomorphic AI, will be outlined and some of its potential applications discussed.

Method

Production systems

Production systems have several advantages in the present context. First, they are conceptually simple. Second, they are general in that they are Turing complete and subsume formalisms like proof systems, grammars, and term rewriting systems. Third, they have been used in AI to describe algorithms [McD82]. Fourth, they have been used in cognitive modeling [LNR87].

Production systems occur in many variations, for instance:

- the rules of the systems may operate on sets, multi-sets, or sequences,
- the right-hand sides of the rules may be restricted or not restricted to one element,
- when several elements are allowed on the right-hand sides of the rules, the reading may be either conjunctive or disjunctive,

- the left-hand side of the rules may be consumed or not consumed when the rules are used.

In the following a *production system* will denote a countable set of objects O together with a countable set of production rules R. The rules operate on multisets of objects, the reading of the right-hand sides of the rules is conjunctive and the left-hand sides are consumed when the rules are applied.

Example 1. The following is a production system:

Objects: {a,b,c,d,e},

Rules: {a} → {b,b}; {c,d} → {a}; {c} → {e}.

A *problem class* is given by a production system (O,R) and all pairs (A,B), where A and B are finite multisets of elements of O. Such a pair (A,B) is called a *problem*. A *solution* to a problem (A,B) is a finite sequence of subsets of O that begins with A, ends with B and takes steps according to the rules of R. Problems with solutions are called *solvable*.

Example 2. Consider the problem class defined by the production system of Example 1 and the problem ({c,c,d}, {e,b,b}).

Here is one solution to this problem:

{c,c,d} → {c,a} → {e,a} → {e,b,b}.

Now it is time to introduce our human raw-model h, who is a real person or a group of people with or without equipment like pencil and paper. We assume that h is better at solving problems in (O,R) than the state-of-the-art computer programs. For instance, we could imagine that (O,R) is an encoding of the game of go and that h is a go master. We shall try to use the cognitive limitations of h to our advantage.

We have assumed that h is good at solving problems in (O,R). For all we know, h may solve these problems working in a different production system, e.g. a version of (O,R) augmented with some derived rules, or some completely different system in which (O,R) can somehow be interpreted.

Following an old idea in cognitive science, we are going to use a production system (O',R') for modeling h. (O',R') should be as psychologically realistic as possible with respect to h working on (O,R). More precisely, as many as possible of the concepts, derived rules and tricks that h actually uses in the problem solving processes should have formal counterparts in (O',R'). Since h is good at solving problems in (O,R), there must be some connection between (O,R) and (O',R'). More precisely, the former must be "embeddable" in the latter. Therefore we may often think of the former as a subsystem of the latter or even go further and identify the two systems.

Experiments

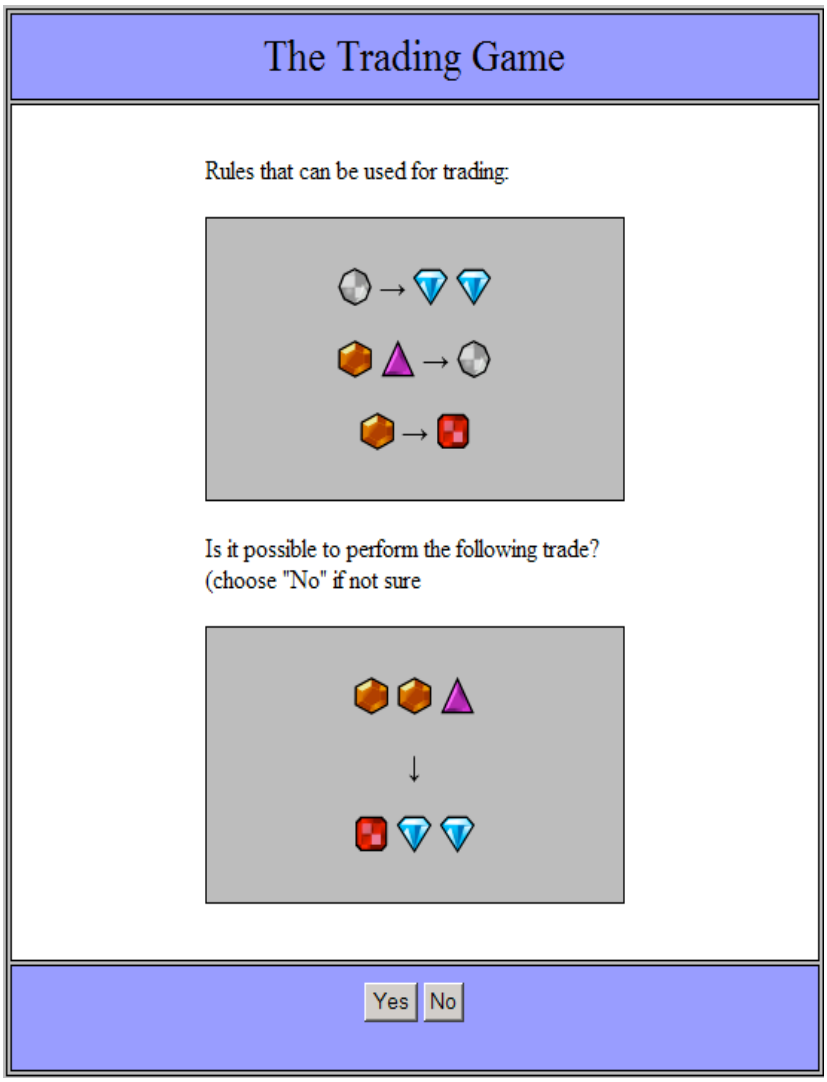
Now that the problem class (O,R) and the human raw-model h have been fixed, experiments can be made in order to approximate the set of solutions of (O,R) that are comprehensible to h. Below we exemplify how such experiments can be performed.

Start by selecting a *test*, i.e. a finite set of problems $\{p_0, p_1,\ldots,p_n\}$ from (O,R). The test should be a mix containing easy and hard, solvable as well as unsolvable problems. Then the set-up of the experiment must be determined. For instance, for the problem class in the above examples, the set-up can be as follows. Use a computer and a graphical user interface that is specially developed for the purpose. Represent the production systems graphically as rules for trading e.g. jewels of different sorts and then ask whether or not a given trade is possible according to the rules. The only answer alternatives given are YES and NO.

This is illustrated in the figure.

A timeout is used and if no answer is registered before timeout, the answer will be set to NO.

Then the test is given to h and the answers and the response times are recorded on the computer.



STRATIFICATION

Next the outcome of the experiment will be used to approximate the comprehensible part of the search space. This part will in turn be divided into levels of comprehensibility. To define these levels, we shall be using complexity measures.

A *complexity measure* is a function from solutions to natural numbers.

Example 3. Here are some examples of complexity measures, with values computed for the solution {c,c,d}→{c,a}→{e,a}→{e,b,b} of Example 2:

- Length: 4
- Maximum working memory: max(3,2,2,3) = 3
- Accumulated working memory: 3+2+2+3 = 10
- Accumulated working memory additions: 1+1+2 = 4
- Total number of applications of a rule with two premises: 1+0+0 = 1.

Complexity measures of interest in the present connection should reflect critical and scarce psychological resources. Before we go on with some more definitions, let us fix a set of complexity measures $\{c_0,c_1,\ldots,c_m\}$.

A *block* is a finite set of solutions B such that for all b∈B and s∈S, if $c_i(s) \leq c_i(b)$ for each i≤m, then s∈B.

Let B be a set of solutions. Then prob(B) denotes the set of problems that have solutions in B.

Let P(t) be those problems of the experiment for which h correctly answered YES within t seconds.

Let q(t,B) = |P(t)⊠prob(B)| / |P(t)|. Here |X| denotes the cardinality of X.

The function q(t,B) can be regarded as a quality measure that indicates how well B performs with respect to h. Note that by increasing the size of B, one can make q(t,B) grow and eventually reach 1.

Now we are in a position to define the layers of comprehensible solutions. First fix a sequence of time-points of interest, e.g. 10, 20 and 30 seconds. Then define a corresponding sequence of blocks B_{10}, B_{20}, B_{30}, so that e.g. $q(t,B_t) \geq .95$ for t = 10, 20, and 30. This should be done in such a way that the sizes of the blocks are minimized. These sets constitute the layers of the comprehensible part of the search space.

Note that the predictive power of these blocks with respect to the performance of h can be evaluated statistically, e.g. by conducting more experiments. The exact shape of the blocks can also be adjusted until they reach a satisfactory level of predictive power. Also, to reduce the size of the layers, one may want to resort to other shapes than blocks.

Search

Now let A be any search algorithm for the problem class (O,R). Let B_{10}, B_{20}, B_{30} be the sequence of blocks mentioned above. Then we can direct the search of A to the comprehensible part of the search space in an iterative deepening fashion as follows: first run A restricted to B_{10}, then A restricted to B_{20}, and finally A restricted to B_{30}. When running A restricted to some B_t, the search tree will be pruned as soon as an object outside of B_t is generated. The purpose of modifying A in this way is to make the algorithm fast on problems of (O,R) where h is fast. Note that we are not trying to obtain a global speed-up for all problems of (O,R), but only a local speed-up on those problems where the human raw-model tends to be fast.

Potential applications

Below are some examples of potential software applications of models of human cognition.

Applications where output should be comprehensible

Analyzing software. Programs and program specifications should be as comprehensible as possible. This reduces the risk of introducing bugs, facilitates communication and keeps maintenance costs down. Already with rather rough mathematical models, one may be able to scan programs and specifications automatically and find parts that are hard to understand and would benefit from being rewritten. Program comprehensibility is discussed in [Wie91].

Generating code. With a model for comprehensible program code, one may cause automatic code generators to output more comprehensible

code. Also one may steer the evolutionary processes in genetic programming towards more comprehensible and thus more trustworthy code.

Generating explanations. An engineer may want to understand why a certain property holds for a system and may use an automatic theorem prover to generate a proof to this effect. That proof clearly has to be comprehensible to the engineer to serve its purpose.

Synthesizing natural language. Natural language systems that communicate with people, e.g. dialogue systems, clearly should produce comprehensible language. To ensure this one may incorporate precisely defined models of the comprehensible sentences. Natural language comprehensibility is discussed in [TB01].

Analyzing natural language. When analyzing natural language texts, there is often a large number of possible analyses, arising e.g. from ambiguities and underspecified references. With a computer model of the comprehensible sentences, and the assumption that the text is actually comprehensible, one may cut down on the number of possibilities and thus facilitate the linguistic analysis.

Applications where output should be incomprehensible

Generating passwords. In general, passwords should not be too short or too regular. A model for comprehensible sequences may be helpful to ensure this or to help crack passwords that have perhaps been chosen for the ease of remembering them.

Playing games. Programs that are intended for play against humans in certain strategic or probabilistic games may potentially exploit systematic errors in the human probabilistic judgment or the incapability of the human player to make certain calculations.

Applications where comprehensible solutions should exist

Verifying engineering constructions. An engineer who makes some construction according to a given requirement specification should be able to produce two things in the end for the job to be deemed successful: (i) the construction and (ii) arguments that are comprehensible to his or her col-

leagues to the effect that the construction meets each one of the requirements. In certain situations this means that comprehensible proofs will exist in the corresponding formalized setting. In that case the construction can be verified by a theorem prover that searches among comprehensible proofs only.

Checking mathematical proofs. It is far from uncommon that theorems published in mathematical journals are false as they stand [Lam95]. This justifies the existence of the movement that seeks to prove mathematical theorems rigorously using interactive proof-systems like AUTOMATH [deB80]. Potentially, those parts of the original proofs that translate into comprehensible formal proofs can be handled by automatic theorem provers that search among comprehensible proofs only.

Discussion

The usefulness in practice of the anthropomorphic AI approach is yet to be shown. From the viewpoint of potential applications, it would be interesting if it could be applied to first-order theorem proving. For instance, it could be that an efficient decision algorithm for the "comprehensible" fragment of first-order logic is within reach, even though full first-order logic is not decidable. In general, by restricting attention to the comprehensible solutions, one may hope to develop efficient local algorithms in situations where no (efficient) global algorithms are known. Thus, in practical applications one may hope to circumvent certain theoretical obstacles by focusing on suitable subclasses of problems. Such theoretical obstacles include the NP-completeness of the SAT-problem and the incompleteness of elementary arithmetic.

Claes Strannegård
Intelligent Systems Design
IT-university of Gothenburg
claes.strannegard@ituniv.se

References

[AL98] J. Anderson, C. Lebiere. *The atomic components of thought*, Mahwah, NJ: Erlbaum, 1998.

[BO98] M. Braine, D. O'Brien. *Mental logic*, L. Erlbaum Associates, 1998.

[CDM92] A. Colorni, M. Dorigo, V. Maniezzo. *Distributed Optimization by Ant Colonies*. Proceedings of the First European Conference on Artificial Life, Paris, F.Varela and P.Bourgine (editors), Elsevier, pp. 134-142, 1992.

[deB80] N. de Bruijn. *A survey of the project AUTOMATH*. To H. B. Curry: essays on combinatory logic, lambda calculus and formalism, pp. 579-606, Academic Press, London-New York, 1980.

[FBL71] E. Feigenbaum, B. Buchanan, J. Lederberg. *On generality and problem solving: A case study using the DENDRAL program*. In B. Melzer and D. Michie (editors), Machine intelligence 4, pp. 165-190, 1971.

[Joh83] P. Johnson-Laird. *Mental models*, Harvard University Press, 1983.

[LNR87] J. Laird, A. Newell, P. Rosenbloom. *Soar: An architecture for general intelligence*, Artifcial Intelligence, 33(3), pp. 1-64, 1987.

[Lam95] L. Lamport. *How to write a proof*, American Mathematical Monthly, Volume 102, Number 7, pp. 600-608, 1995.

[MMRS55] J. McCarthy, M. Minsky, N. Rochester, C. Shannon. *Proposal for the Dartmouth summer research project on artificial intelligence*, Technical Report, Dartmouth College, 1955.

[McD82] J. McDermott. *R1: A Rule-based Configurer of Computer Systems*, Artificial Intelligence, Vol. 19, North-Holland, pp. 39-88, 1982.

[Mil56] G. Miller. *The magical number seven, plus or minus two: Some limits on our capacity for processing information*, Psychological Review 63, pp. 81-97, 1956.

[NS61] A. Newell, H. Simon. *GPS, a program that simulates human thought*. In H. Billing (editor), Lernende Automaten, pp. 109-124. R. Oldenbourg, München, 1961.

[NSH56] A. Newell, J. Shaw, H. Simon, *The logic theory machine*, IRE Trans. Information Theory, IT-2, No, 3, pp61-79, 1956.

[Rip96] L. Rips. *The Psychology of Proof*, Bradford, 1996.

[TB01] D. Townsend, T. Bever. *Sentence Comprehension: The Integration of Habits and Rules*, Bradford Books, 2001.

[Wie91] S. Wiedenbeck. *The initial stages of program comprehension,* International Journal of Man-Machine Studies, Vol. 35, No. 4, pp. 517-540, 1991.