Principles of Possible Core Structures of Artificial Intelligences

Holger Burbach
Lüdenscheid, Germany
holger@burbach.eu

April 2022

***Abstract*—Since construction of an artificial intelligence is an explosive topic maybe first the concepts of its construction should be discussed before starting to implement them. This is a contribution to this discussion. The solution of the problem how to achieve representational completeness in a universal way by nowpoint graphs suggests to find universal engines to process these graphs. The engines represent areas of a neural network brain. The intelligent solution is meant to be more than the sum of its unintelligent parts. The evolution of the human brain seems to be still working on a good setup of these engines. The trick is to set up an engine collection which produces meaningful results and then get it trained correctly.**

***Index Terms*—artificial intelligence construction, natural language understanding, representational completeness, representation problem, nowpoint graph, chat bot**

I. Introduction

This article is not based on software working as desired. It is inspired by current human brain research. It describes a concept for the construction of an artificial intelligence to avoid the problems arising by first secretly building an artificial intelligence. The first rversion of the software described in this article will not be (that) intelligent, but it can serve as a proof of concept.

The key idea is a universal engine which can process a nowpoint graph [1]. The solution is not one all purposes fitting universal engine version but a collection of engines each processing one computable unintelligent aspect of an intelligent brain. These aspects can be emotions, mind, will, input, output, self recognition of output and endless more. Between the engines lists of references to nodes of the nowpoint graph are passed. Each engine can add further nodes to the graph and pass lists of node references to connected engines depending on the aspect it processes.

Since addition of nodes has to be complemented by the deletion of nodes a cleanup unit is necessary to delete the least important nodes when the graph becomes too big or when too much rubbish has been produced.

Since the whole brain has to be managed in some way a brain management unit for at least boot and shutdown is necessary.

Since the engines work in parallel the passing of nodes between them has to be synchronized by a clockwork unit. The length of the processing of one single unit of work has to be optimized carefully to avoid engines waiting too long for other engines to complete the current unit of work.

In the following sections several elements of the ideas mentioned so far in this text will be discussed. First the data structure (nowpoint graph) is described. Then types of nowpoint graph engines are focused on. After that the overall setup of a collection of engines is discussed.

As illustrating example the engine setup for a program you can chat with in natural language is used, commonly called a chat bot. This example shows an additional approach to natural language understanding and processing [2] [3] [4].

II. The Nowpoint Graph Data Structure

As described and motivated in [1] a graph is used as data structure. Most nodes in this graph are of one single type. One node can represent anything. Just regard it as a thought. It can optionally be attributed by a byte array of arbitrary length. This byte array can be any meaningful piece of data. Derived from [1] possible operations on these nodes are:

- Create: A node is created and optionally attributed with a byte array. An example for this can be an input word read from the terminal. Depending on the functionality of the creating engine in normal cases the new node is further connected to other nodes by new directed edges, for example to create a chain of nodes attributed with words from an input sentence from the terminal.
- Delete: A node together with its incoming and outgoing edges is removed from the graph.
- Read: The byte arrays of a set of nodes are read for further ordinary processing by a computer like output on a device.
- Concretisation: This operation produces a set of new nodes with directed edges pointing from the source node to the new nodes. Depending on the functionality of the engine doing this concretisation the byte array of the source node may optionally be used to create new byte arrays for attribution of the new nodes. An example would be the simple operation of assigning one byte of the array of the source node to one node each.
- Abstraction: For a set of nodes one new node is created with directed edges pointing to each of the nodes of the set. Depending on the functionality of the engine doing

this abstraction optional bytes from the byte arrays of the nodes from the set may optionally be used to produce a new byte array for attribution of the new node.

The above list of operations is only a suggested subset of the unlimited set of possible operations. Additional single management nodes may be necessary to enable the engines doing their work or setting up working sets of nodes at boot time. Be aware, that the byte arrays described above are always optional. A thought (node) can exist without bytes attached to it.

III. A Single Nowpoint Graph Engine

One single nowpoint graph engine consists of the following elements:

- Unique identification of the engine type, for example by a name character string and a Java class.
- Multiple input lists of nodes from other connected engines, for example of the type of a single list in Java.
- Multiple output lists of nodes to other connected engines, one list to one engine in a step of work from the clockwork unit
- Management of input and output subscribers of node lists by an API
- Proper communication with clockwork unit, brain management unit, cleanup unit and connected engines by an API
- Processing of all input node lists with one output node list to each connected engine: This element makes up the difference between engine types. A subset of unlimited possible ways of processing input nodes is described in the next section.

Be aware that the overall connection of engines to a network of engines is fixed.

IV. Types of Nowpoint Graph Engines

There are four differences between different types of engines:

1) Simple identification of the engine type
2) Way of processing input note lists and producing the output node lists
3) Way the engine is connected to other engines by the management unit at boot time
4) Own central management node, new nodes created by this engine can be connected to by incoming directed edges from the management node

Types of engines can be derived from the different areas of a human brain identified so far [5]. For the description here we concentrate on a simplified subset of possible types adapted to our application of a chat bot. Operations on the nowpoint graph and input and output node lists of our example engines during one processing step synchronized by the clockwork unit are as follows. All engines are named to simplify references during description.

- Engines for which the node processing does not depend on the type of input engines:
  - Text input: A sentence from the input terminal is transformed into a sequence of nodes connected with directed edges. Each node in the sequence is attributed with the character string of one word. An abstraction node is created for all the nodes of the sentence. The list of nodes created is used as output list. This engine is named *Text Input Engine*.
  - Emotions input: The engine issues an emotions node as output list depending on information from outside the chat bot like pitch of spoken words used for text input or direct input emotion choice by the user. Details have to be fine tuned here. This engine is named *Emotions Input Engine*.
  - File input: This engine reads a file, assigns it to a node as byte array and outputs this node to connected engines. This engine is named *File Input Engine*.
  - Text output: Each input list of nodes is printed in the sequence of input to the output terminal. The attributed byte array of the nodes are interpreted as character strings. Multiple input lists during one step of work from the clockwork unit are used for output in random order. This engine is named *Text Output Engine*.
  - Current time input: For each input list of nodes an abstraction node is created and attributed with the current time in milliseconds. The created nodes are used as output list in random order. If there is no input node list a single node is created and used as output list. This engine is named *Current Time Engine*.
  - Current text input user recognition: For each input list of nodes an abstraction node is created and attributed with the text string representing the name of the current user. The created nodes are used as output list in random order. If there is no input node list a single node is created and used as output list. This engine is named *User Recognition Engine*
  - Self recognition of text output: For each list of input nodes an abstraction node is created. This abstraction node is connected to the one single main management node of this engine by an incoming directed edge. The created nodes are used as output list in random order. This engine is named *Output Recognition Engine*.
  - Current nowpoint graph size: For each input list of nodes an abstraction node is created and attributed with the current graph size at this processing time as an integer number. The created nodes are used as output list in random order. If there is no input node list a single node is created and used as output list. This engine is named *Graph Size Engine*.
  - User is inputting: During this processing step a single output node is created if the user is currently inputting a sentence. This engine is named *Inputting Engine*.
  - Speech recognition: This engine maintains category

abstraction nodes connected to the management node of this engine based on the concepts of appearance and neighbourhood. For each word node in the input node list an appearance abstraction node gets an outgoing edge to this node. For the numbers 1 to 7 each row of neighbouring word nodes with this count get a new neighbouring abstraction node with outgoing edges to the appearance abstraction nodes of the word nodes in the input node list. These neighbouring abstraction nodes get incoming edges from the management node of this engine. The neighbouring abstraction nodes are not duplicated for a given combination. Instead additional appearances of a given combination reduce the chance of deletion of this abstraction node by the cleanup unit managed by the reference counter. For the 1st level neighbouring abstraction nodes of a given input node list which have not just been newly generated but have been found to be already existing the generation of 2nd level neighbouring abstraction nodes is done. This process is repeated until only one single node is the result. This node represents the language of the input node list sentence. This engine is named *Speech Recognition Engine*.
  - Speech production: This engine has access to the management node of the *Speech Recognition Engine*. It reverses the process described for speech recognition. The engine is named *Speech Production Engine*.

- Engines with input list processing depending on the engine the list is output by:
  - Emotions: This engine links simple value nodes (emotions) to input nodes and outputs the emotion nodes. The value (emotion) used depends on input nodes from the *Current Time Engine*, the *Graph Size Engine*, the *Inputting Engine*, the *User Recognition Engine*, the *Mind Engine*, the *Will Engine*, the *Consciousness Engine* and the *Emotions Input Engine*. The engine is named *Emotions Engine*.
  - Mind: This engine interprets input node lists as predicate logic expressions and performs unification over this expression as proposed by Robinson [6] [7] . It regards nodes with byte attribution as constants and nodes without byte attribution as variables. Search space are all nodes of the graph with byte attribution. The engine takes input node lists from *Consciousness Engine* and the *Emotions Engine* and produces output node lists of the nodes it is currently working on for the *Emotions Engine*, the *Consciousness Engine* and the *Speech Production Engine*. The engine is named *Mind Engine*.
  - Will: This engine is responsible of the basic behaviour of a chat bot: takes input lists from the *Speech Production Engine* and the *Consciousness Engine*, remembers them for some time and after some time sends the input list from the the *Speech Production Engine* with the most positive emotions to the *Text Output Engine* dropping the memory of the other input node lists since the last output. This engine is named *Will Engine*.
  - Consciousness: This engine takes input node lists from a lot of engines and produces output node lists for the *Will Engine*, the *Mind Engine* and the *Emotions Engine*. In this engine global goals of the engine collection can be set up, for example affection or aversion for a specific subject or behaviour. This engine is named *Consciousness Engine*.

V. Meaningful Input and Output of the Chat Bot

Until the time this text is written there is no engine collection producing anything else but rubbish. The proper collaboration of the *Emotions Engine*, the *Mind Engine*, the *Will Engine* and the *Consciousness Engine* is critical for meaningful intelligent behaviour. The author is convinced that it makes sense to invest time in developing this collaboration. Even when the engine setup is finally useful a lot of training will be necessary to make this AI adult. The author is convinced that this can be done. The result would be intelligent behaviour of a collection of unintelligent parts. The author is convinced that intelligence is an emergent property.

VI. Conclusion

Of course it is problematic to describe a concept without having solved all detail problems and finished the corresponding software. However, it is the author's opinion that the ideas described in this article show the right direction to think in to construct one possible version of an artificial intelligence. Obviously the parts of the chat bot described in this article are unintelligent. The main goal is to produce intelligence out of unintelligent parts. It is important to be aware that there will never be the one and only solution to the problem of setting up a proper engine collection and thus constructing an artificial intelligence. The next step is a proof of concept of the ideas in this article.

References

[1] H. Burbach, "On representational completeness," 2020, https://burbach.eu/nowpoint5.pdf.

[2] J. Allen, *Natural Language Understanding*. The Benjamin/Cummings Publishing Company Inc., 1987.

[3] H. Helbig, *Knowledge Representation and the Semantics of Natural Language*. Springer Verlag, 2006.

[4] P. M. Nugues, *An Introduction to Language Processing with Perl and Prolog*. Springer Verlag, 2006.

[5] S. Standring, Ed., *Gray's Anatomy: The Anatomical Basis of Clinical Practice (40th ed.)*. Churchill Livingstone, 2008.

[6] J. A. Robinson, "A machine-oriented logic based on the resolution principle," *Journal of the ACM*, vol. 12, pp. 23–41, 1965.

[7] ——, "Computational logic: The unification computation," *Machine Intelligence*, vol. 6, pp. 63–72, 1971.