AGI engineering

Subscribe

NEXT VOLUME OF THE AGI SAGA

TRANSITION TO DETAILS

Mykola Rabchevskiy
6 hr ago

This, the thirty in a row, chapter completes the first part of the cycle, in which the general principles of building an AGI system, architecture, specificity of components, and, in the end, the choice of a prototype capable of demonstrating the workability of the approach were discussed. The next part will describe the ***algorithms*** for implementing the ***components*** suitable for creating the described ***prototype*** in more detail.

At the same time, the nature of publications is changing. So far, all chapters have been available to free edition subscribers; from the next chapter, most of the texts will be available only to subscribers of the paid version. Subscriber funds will help improve our team's supply of thought fuel: coffee and beer [1].

Finally, the last of the general aspects.

It is easy to see that search has been repeatedly mentioned as an element of the algorithms of various AGI components:

- finding unknown objects in the environment
- finding what deserves attention
- finding causal relationships
- finding repeating sequences of events
- finding the best course of action from among the possible

A common feature of the search in all these cases is the need to choose from such many objects/options, which is either too large to have time to test them before the time allotted for this is exhausted, or the number of options is completely infinite.

The impossibility of an exhaustive search means that the result is ***radically dependent on how the search is organized***. Although each case has its own specifics, there is an aspect common to all: if we use a specific regular order of enumeration and each time you start a deliberately incomplete enumeration in the same order, then some options will be tested more often than others, and some can be completely permanent be among the untested.

A simple and effective way to solve the problem is to use a random order of enumeration, ensuring the equiprobability of testing and eliminating the systematic omission of some options.

Random order has another advantageous property. Parallelization is a natural way to speed up the search: the more processors, working simultaneously, are looking for the desired option, the more options will be tested in the time allotted for the examination. But for this, of course, it is required to organize the work in such a way that parallel processes do not duplicate each other, checking the same options. An obvious way is to use a manager module to distribute tasks among the instances of the variant evaluator. Naturally, such a module also consumes computing resources. Random enumeration of variants by each tester module eliminates the use of a manager: if the total number of variants that can be tested by one evaluator in the allotted time is much less than the total number of possible variants, then the number of overlapped tests will be tiny, and this can be neglected.

---

1 *“Nighthawks” by Edward Hopper, 1942*