AI1 proposal

Yura Perov

1 Main idea

This note drafts a way to perform automatic exhaustive search (i.e. automatic programming or program synthesis) for AI-kind computer programs.

The main idea is to advance the field of automatic programming as follows[1]:

1. Collect, make public and maintain an extensive collection of benchmarks for the automatic induction of valuable AI agents.
2. Perform search not only over the procedures that solve tasks from the benchmark, but also perform the search over the procedures that generates the former procedures.
3. Prepare and maintain the set of predefined compound procedures (see details in Section 8).
4. Run this optimisation continuously, using as many resources as possible, reasonably of course.
5. Run this project publicly, so that other people can contribute to the set of benchmarks and to the predefined compound procedures. As well as other people can contribute to the optimisation by running optimisation on their machines and clusters.

Otherwise, this note is just based on what already exists in the field of automatic programming and program synthesis, including evolutionary (genetic) programming.

It would be nice to run this project in the similar way as the SETI is run[2].

2 Main definitions

1. **The set of benchmarks** is an extensive collection of AI-related tasks.

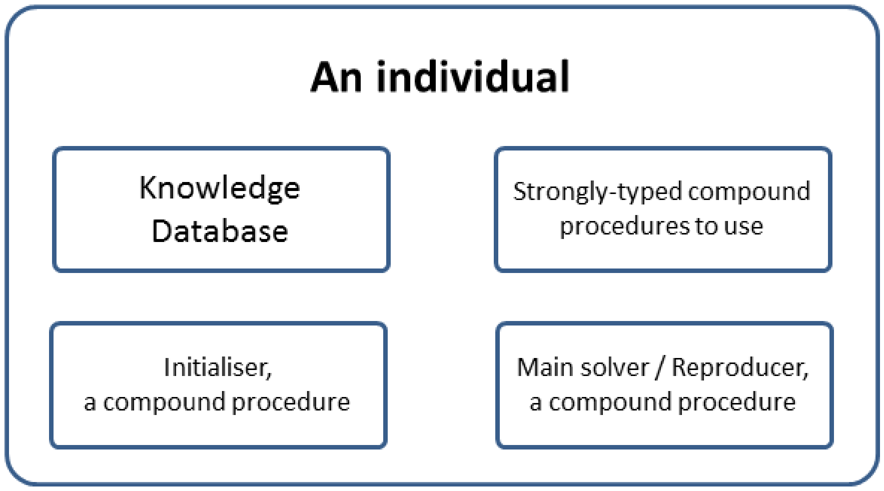
[1]I am not an expert in automatic programming. There is a very good chance that these ideas were already proposed and even successfully attempted.

[2]There was at least one similar project, but I did not find any recent information on it.

Each task has an input and a desired output of a particular type. For example, the input is an arbitrary image 2D matrix of real bounded values, and the output is a language word type. Also, each task is provided with a likelihood or noise function that measures the difference between the received and desired output. This set of benchmarks has a multi-level hierarchy. In each sub-category, the task is ordered by the level of its complexity.

2. **A worker invidual** is program code and supplementary data. A worker individual is a candidate for an AI.
3. **A reproducer individual** is program code and supplemntary data. A reproducer individual is an individual that is responsible to generate new individuals based on several existing ones.
4. **A primitive type** might be integer, string, fixed-sized matrix, unfixed-size matrix, etc.
5. **A compound procedure** has formal arguments and transparent body of program code, which is executed when the procedure is called with values of those arguments.

3 An individual

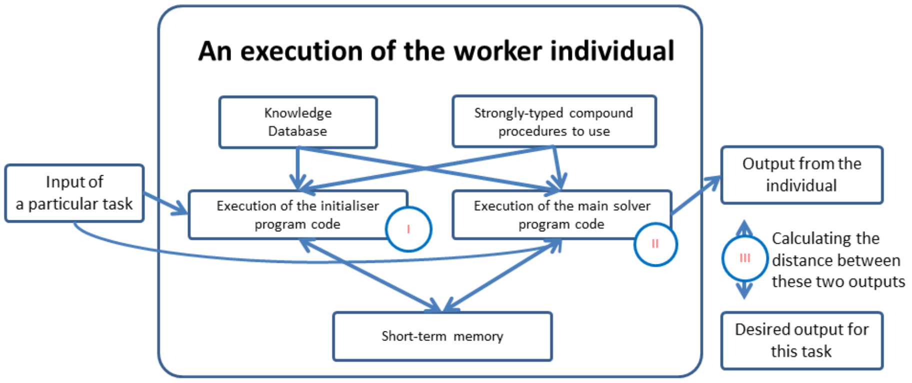

An individual
Knowledge Database
Strongly-typed compound procedures to use
Initialiser, a compound procedure
Main solver / Reproducer, a compound procedure

An individual consists of following elements:

1. A relational database of facts, which this individual knows.
2. Code of compound procedures, which this individual inherites and might use.

3. Code if a procedure for the individual initialisation.
4. Code of (a) a main solver procedure or (b) a reproduction procedure.

4 An execution of a worker individual

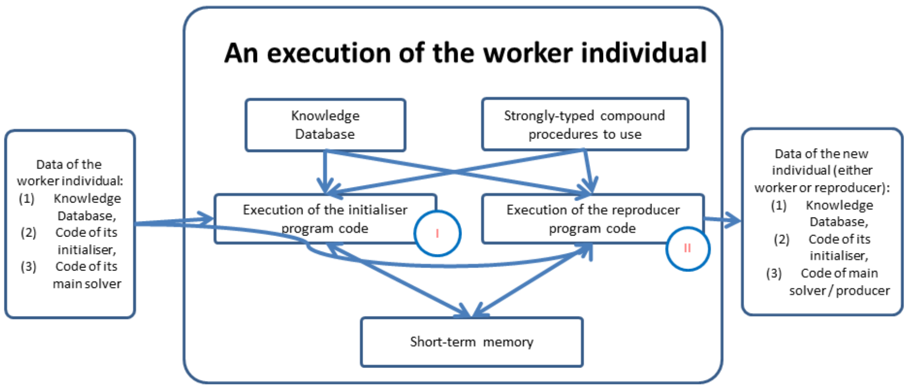

An execution of the worker individual
Knowledge Database
Strongly-typed compound procedures to use
Input of a particular task
Execution of the initialiser program code
I
Execution of the main solver program code
II
Output from the individual
III
Calculating the distance between these two outputs
Short-term memory
Desired output for this task

Given an individual and given a particular task from the set of benchmarks, we execute this individual to get its output for the input of this task. We do this as follows:

1. We run the procedure of the individual initialisation. To this procedure, we provide the following input arguments: the type of the task, the type signatures of the input and output tasks, and might be even the value of the input for this task. The desired purpose of this initialisation procedure is to prepare the individual to solve the task. This initialisation procedure has access to the tempory short-term memory of the individual, to which the main solver procedure will also have the access.
2. We run the main solver procedure of the individual. We provide the value of the input, of this task, to this main solver procedure. The procedure should return the output.
3. We measure how well this individual have performed this task, with the help of the likelihood function. We calculate the metric distance between the desired output and the individual output.

The evaluation of the initialisation procedure is stopped if its takes longer than T_1 seconds. The same for the main solver procedure: it is stopped after T_2. Some tasks might have adjusted T_1 and T_2. The generation of a child is bounded by T_3 seconds.

5 An execution of a reproducer individual


An execution of the worker individual
Knowledge Database
Strongly-typed compound procedures to use
Data of the worker individual: (1) Knowledge Database, (2) Code of its initialiser, (3) Code of its main solver
Execution of the initialiser program code
I
Execution of the reproducer program code
II
Data of the new individual (either worker or reproducer): (1) Knowledge Database, (2) Code of its initialiser, (3) Code of main solver / producer
Short-term memory

This execution is similar to the execution of a worker individual. Just instead of executing the main solver procedure, we execute the reproducer procedure given code of another worker.

6 The performance evaluation of the individual

The worker individuals are appraised immediately after performing several random tasks from the set of benchmarks.

The reproducer individuals are appraised based on how well their children or (grant-)... grant-children are appraised.

7 Generations of individuals

Each generation has some number of individuals. The higher the appraisement of the worker individual, the more likely that it will be selected for the child reproduction. The same for the reproducer individuals.

Rules here might just be based on evolutionary (genetic) algorithms' rules of the generation formation.

8 Predefined compound procedures

Individuals at first generations are supplied with predefined compound procedures.

For the sake of the worker individuals, those compound procedures include the code of simple MCMC samplers, code to use neural networks, etc.

For the sake of the reprocuder individuals, those compound procedure include compound procedures from the field of generic programming, as well as the code that train and creates simple neural networks.

9 Language choice

The choice of the language is important. Important requirements:

1. A fast language.
2. A multi-platform language so that individuals might be run and evaluated on different clusters with varying hardware and software architecture.
3. A language, where the representation of compound procedures is also a data structure (i.e. a homoiconic language). This is crucial for the reproducer individuals. This is also important so that compound procedures, in individuals, can perform MCMC sampling on probabilistic programs from the knowledge database.

10 Strong types

Almost everything is strongly typed so that the process of generating a new individual and the process of selecting appropriate compound procedures for solving the task is simplified.