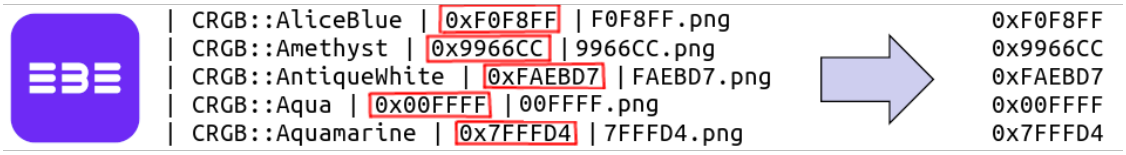
BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2022
http://excel.fit.vutbr.cz

Ebe – A tool for automated file editing using genetic programming

Marek Sedláček*



Abstract
File editing is a big part of today's work for many people, but not everyone has programming skills or deep knowledge of editing tools to make their editing efficient and quick. This is exactly what Ebe is trying to solve. Ebe takes snippets of file edits done by the user and using genetic programming it finds the correct algorithm to transform the whole file or even multiple files into desired output. Ebe is currently in early version 0.3, but despite that Ebe already achieves some notable results and already contains some additional features for more skilled users to get the most out of it. Ebe is not only for non-programmers, since it can find some edits on its own within seconds, it is a great alternative to handwriting a script for such edits. Even though machine learning is the current hot topic, Ebe uses the approach of evolution – genetic programming – to find the solution, which makes Ebe quite a unique tool and this approach brings in some advantages such as low computational requirements and no need for internet communication with a cloud.

Keywords: Compiler — Interpreter — Genetic programming

Supplementary Material: Downloadable Code

*xsedla1b@vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

Almost everyone who does their work on computer has to work with files and edit them in some ways. It might be as simple as deleting a few lines or as complicated as deleting specific values and moving data or even whole columns around. For programmers, when it comes to those more difficult edits, this might usually mean writing a script to do these edits, but what about the others? Physicist that wants to transform a data set, linguist that want to remove unwanted entries and just overall anyone. Is editing by hand really the best solution? Or forcing these people to learn to program just to do a few edits like this?

Doing small edits is a perfect job for a file editor and hands-on approach, but when editing a file with thousands of lines, this is no longer a viable option and some other tool needs to be employed. In many cases such "tool" is a program in some programming language, but writing this code requires programming skills and, depending on the skill, notable time to write it and tests it. So a tool that claims to do this, should not only "get the job done", but also should not take a long time to do so and require deep knowledge from the user, otherwise working with the tool will take longer than the actual work for which the file is edited.

Nowadays there are many methods and tools for automated specialized file editing and file transformation. File types and their formats vary a lot and thus only a one tool cannot handle all these types, but rather specializes on a certain file type and format. But there

still exist tools, which can handle almost all possible formats, but those tools then require the user to put in a lot of effort to make them correctly work for all the possible inputs and cases. Such example can be the AWK programming language, which was designed in the late 1980s by Alfred Aho, Peter Weinberger and Brian Kernighan for the sole purpose of file editing [1]. Since AWK is used till this day and is a built-in tool for many operating systems [2], this only suggests that the need has not gone away, but in fact may have even gone up, with the rise of high computing and large storage capabilities for computers.

Another popular editing approach is the use of general purpose programming language, such as Python 3, Perl or even Bash. This approach allows to work with almost any file format, but requires additional work to adjust to it and deep enough knowledge of the language and the file that is being edited.

Ebe is trying to tackle all of these problems, it requires the user to only know how to run it and offers a quite fast editing (compilation and interpretation) times compare to writing and running scripts by hand or even doing all the editing by hand.

Currently Ebe is not a perfect do-it-all tool, but already offers a quite powerful ability to do many edits in a short time with the option that it will find these edits for the user. And unlike many other tools, offers the option to edits multiple similar files at once. It also uses data types and thus in combination with user defined expressions allows for changing values in the whole file based on an expression. And if someone does not trust the evolution, they can always write the editing algorithm themselves and then interpret it using Ebe.

2. Genetic programming for code generation

Genetic programming (*GP*) is technique popularized by John Koza in the 1990s and it is a process of optimization [3]. When speaking about code generation with genetic programming, the whole process somewhat imitates evolution in the nature. It starts with randomly initialized populations of candidate phenotypes, which in this case would be simple programs. Just as in the nature, these candidates are crossed with each other, mutated, scored and then the best ones are selected into future populations based on the scoring [4]. Scoring of the program and the strength of GP compare to pure randomness can be seen in figure 1.

Scoring is a big part of genetic programming and there always needs to be a scoring function, called the "fitness function". This function guides the evolution,

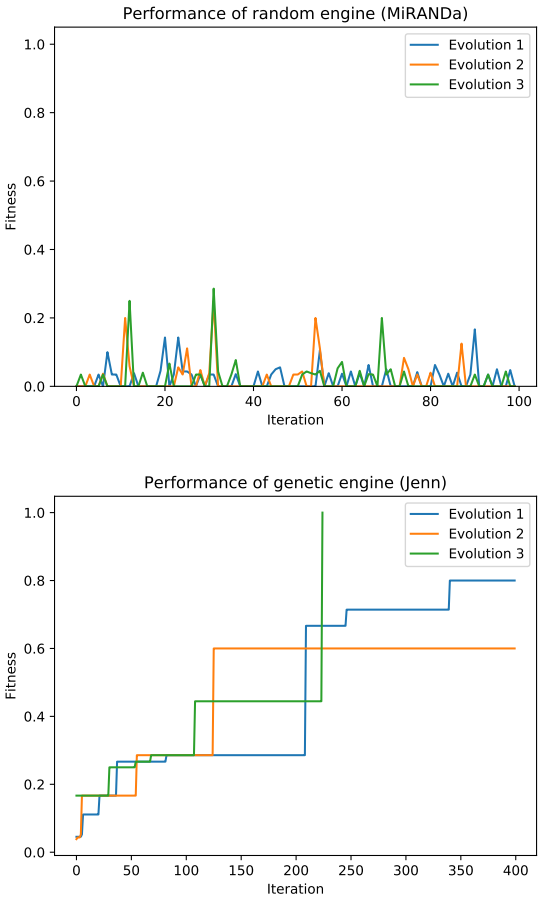


Figure 1. Comparison of 3 iterations of program evaluation using a pure random – MiRANDa engine – program generation (*top graph*) and genetic programming – Jenn engine – approach (*bottom graph*). The GP approach steadily improves the quality of the program, whereas random approach relies only on brute force approach.

by favoring phenotypes with better fitness score [3]. Without a fitness function, there is no way to guide the evolution and thus this limits genetic programming to a limited set of problems, where it can be employed.

The need for fitness function, somewhat limits the use for code generation, since often the implementation of fitness function would result in no longer needing the GP, since the fitness function would be the solution itself. An example of this would be when one was to evolve a sorting algorithm using genetic programming. To score a solution, the output of the generated code needs to be compared to some ground truth, which in this case would be already sorted input and thus the fitness function could be used for this job instead of the generated program.

But this limitation does not mean that genetic programming has no use in this field. On top of generating code, where fitness function can be provided (such as

algorithms for file editing), it can help optimize functions (its time or even space complexity), by evolving parts of the code into ways a programmer would not think of [5] or it can also help automatically fix bugs [6]. In other similar fields GP can be used for tasks such as generating computer art [7] [1], compose music [8] and many other uses.

3. Ebe – Edit by Example

Ebe is a program (compiler and interpreter) for editing files just from given examples.

All the user has to do is create a snipped of file before and after the desired edits (input and output example), give this to Ebe and it will try to find a fitting algorithm (program in *Ebel* language), which would do the desired transformations from input example to the output example (example of Ebe's usage can be seen in figure 2).

Since Ebe is aimed also at people with little to no programming knowledge, Ebe's philosophy is to not cause exceptions and errors as long as it is not necessary. Meaning that, when an incorrect instruction or input is encountered, rather than exiting the execution with error, only a warning is printed and the instruction or input is ignored. Ebe is very verbal about this and will notify about any problems, but this philosophy is quite handy when interpreting multiple files, which might slightly differ (a division by 0 might be encountered or different data type at some position), but one Ebel will work on all of them.

Ebe consists of multiple independent modules (see figure 3), where some of them can even be used independently. One of the main modules is *Ebec* (Ebe compiler), which does all the necessary file parsing and guides the evolution, which happens in the *Eben* (Ebe engine). Once the compiler and engine evolve sufficiently fitting program, then this can be outputted or loaded into the *Ebei* (Ebe interpreter), which can then edit any number of files using this algorithm.

The time for Ebe to generate a suitable program is non-deterministic and depends on lots of external factors. Some very simple programs can be evolved within tenth of a second and some difficult ones might take a few minutes to be evolved. This might be a significant time to some people, but compare to alternative approach with editing by hand or writing an editing script, this might be faster and simpler approach, since it can run in the background, while other work is done.

[1] Abstract evolution – a program for generating abstract computer art using genetic programming – https://github.com/mark-sed/abstract-evolution.

```
[mark-sed@ebe]$ ls
example.in  example.out  temps.data
[mark-sed@ebe]$ cat temps.data
Prague: 42
Paris: 50
Venice: 60
Munich: 38
[mark-sed@ebe]$ cat example.in
Prague: 42
[mark-sed@ebe]$ cat example.out
Prague: {! (($ - 32) * 5) / 9 !}
[mark-sed@ebe]$ ebe -in example.in -out ex
Perfectly fitting program found.

Best compiled program has 100% precision
[mark-sed@ebe]$ ebe -i f2c.ebel temps.data
Prague: 5
Paris: 10
Venice: 15
Munich: 3
```

Figure 2. Example of Ebe's user defined expressions to change values in Fahrenheit to Celsius. This example uses split compilation and interpretation, but it is possible to do both in one Ebe invocation.

4. Ebe's implementation

One of Ebe's strengths is that it generates Ebel instructions, which in most cases care only about the word's position rather than its type or even value. This means that the example provided can only have the same structure as the actual file that needs to be edited. Meaning that one can generate a general editing script by only knowing the structure of the actual input file (see listing 1 and 2). This gives the option to provide someone an editing script without requesting the actual data that needs to be edited, which could otherwise be a problem for security or other reasons. This approach can also be useful in cases, where the output is ambiguous to the input because of the values.

Listing 1. Example input file containing actual values.

```
Brno 42 5.0e-8 + True
```

Listing 2. Example input file, which is structurally and type-wise equivalent to an input file in listing 1 and therefore can be used in place of 1 for Ebe.

```
A 0 0.0 + B
```

On the other hand Ebel language contains constructs, which allow to match on string value or word's type. These more advanced scripts can for example be generated using Bee language and bee-hs compiler[2]. This compiler uses a higher level abstraction language, which it then compiles into Ebel and offers this way an alternative scripting language for file editing for programmers.

[2] Bee-hs – a compiler for Bee language, which is compiled into Ebel – https://github.com/mark-sed/bee-hs.

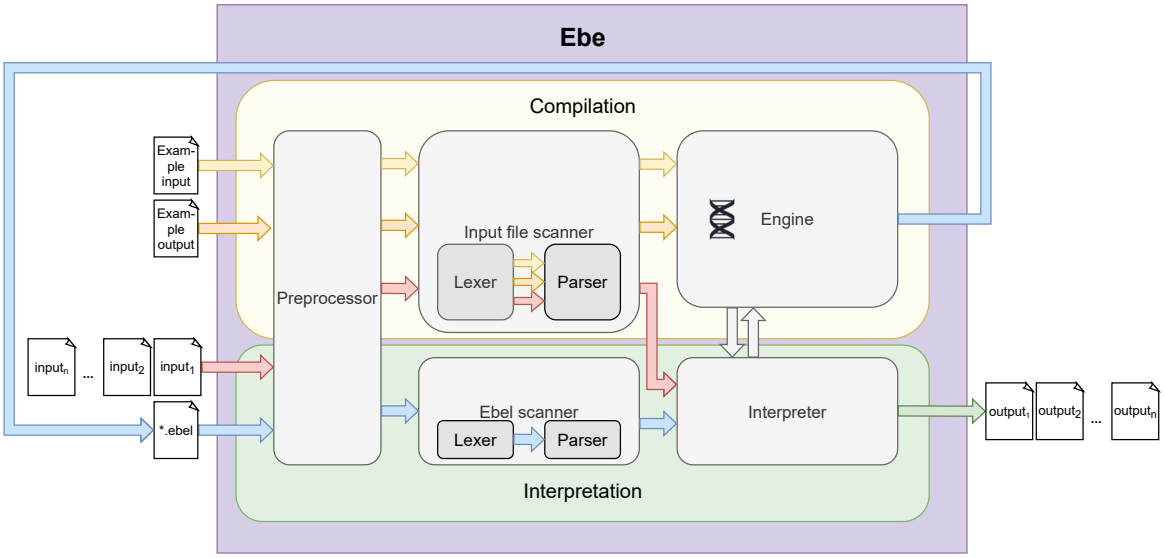


Figure 3. Diagram of Ebe's workflow.

The genetic algorithm used in Ebe varies based on selected engine. Ebe currently offers 3 engines – the experimental engine *MiRANDa*, which uses only random walk, engine *Taylor* and engine *Jenn*. Taylor's approach is quite similar to the approach used in Cartesian genetic programming, where only mutations are used [9] and the starting phenotypes contain only `NOP` instructions ("NOP tail"). Jenn is the default engine and uses genetic programming process quite similar to the one described by J. Koza [10], where staring population are random programs of random size and are during evolution based on set probabilities crossed over, mutated (where mutation only changes one instruction for another random one) and then selected into new populations. Fitness is calculated using one of the string comparison methods (listed bellow), which is applied to the example input interpreted using current phenotype and example output.

Ebe is written in C++ and does not use any external libraries (for general use, otherwise GoogleTest) to allow for easier portability and compilation. Ebe is a free and open-source project and is designed to be modular and extensible by others. It uses Flex and Bison for lexer and parser generation and thus allows developers to easily implement new parser and lexers for new file formats. The same can be done with engines, where engines are what powers and does the whole process of code generation using genetic programming (although it is possible to use different approach). On top of this Ebe's evolutionary process can be even controlled at compile time with multiple command line options to set attributes such as the fitness function (Levenshtein distance, Jaro distance, Jaro-Winkler distance or "one-to-one" character comparison), population size, number of generations, number of evolutions or even a timeout based on compilation time or minimum required output precision. These options are for more skilled users and more often meant for experimenting since Ebe contains heuristics, which decide all these attributes for the user based on the input.

5. Ebel – Ebe language

Ebel is an imperative, case insensitive, programming language designed for file editing and to work well with genetic programming.

Ebel is not really meant to be written, but can be and contains some syntactic sugar to make writing and editing it more user-friendly. Ebel resembles a bytecode and was designed in this way to allow for quick parsing and execution in the interpreter. It is interpreted over a file, where the Ebel code can be thought of as a pipeline of instructions through which the file objects (lexemes) go and get modified by.

Ebel is composed of multiple sections called passes. A pass defines in which way the input file is read. Each pass is then composed of instructions which take as an input objects its parent pass parses (word or line).

Listing 3. Example of Ebel code.

```
PASS Words
  NOP
  DEL
  DEL
  PASS number Expression
    SUB $1, $0, 32
    MUL $2, $1, 5
    DIV $0, $2, 9
    RETURN NOP
  PASS derived Expression
    RETURN DEL
PASS Lines
  SWAP 1
  LOOP
```

The Ebel code in listing 3 will do the following:

1. **PASS Words** - file will be interpreted word by word and for each line:
    1.1. **NOP** - 1st object will left as is,
    1.2. **DEL** - 2nd object will be deleted,
    1.3. **DEL** - 3rd object will be deleted,
    1.4. **PASS number Expression** - 4th object, if it is a number will be:
        1.4.1. **SUB** `$1,$0,32` - subtract 32 from its value,
        1.4.2. **MUL** `$2,$1,5` - multiply the new result by 5,
        1.4.3. **DIV** `$0,$2,9` - divide the result by 9 and save it as the new value for the object,
        1.4.4. **RETURN NOP** - end expression and do not modify the new result.
    1.5. **PASS derived Expression** - if 4th object was not a number, then use the following without regarding its type:
        1.5.1. **RETURN DEL** - delete the object.
2. **PASS Lines** - file will be interpreted line by line and for each line:
    2.1. **SWAP** `1` - swap current line with the following one,
    2.2. **LOOP** - repeat until all lines were processed.

As can be seen in listing 3, Ebel contains the means to carry out computations over a single word (numbers, floats and even strings) in the edited file, but because of the problems with finding correct expressions, where for symbolic regression a large data set would be needed [11] and even then it could be ambiguous, this task is left to the user in the form of, already mentioned, user-defined expressions. If such expression is defined Ebe treats it as always correct, but still can evolve the other parts of the file to find a correct Ebel program.

6. Real world Ebe use examples

Ebe has not been long enough in a public version, but there are already some real world cases, where it has proven to be useful.

6.1 Extracting hexadecimal color values from markdown table

A documentation for WS2811 LED library [3] contained a table of predefined colors and their hexadecimal values, which needed to be extracted (see first page teaser image for reference). This is a very easy edit for Ebe and it took only 100 ms to compile the correct Ebel and interpret it.

It required the first line of the table to be put into the input example file and then edit this by hand in the output example file, but the overall time is insignificant to the time it would take to edit this file by hand or writing a script for this edit.

6.2 Adjusting feature indexes in biological data set

In this case a desired section of human genome was extracted from the whole genome in GTF format. But to display this correctly in a genome browser program, it needed for the start and end feature index (column 4 and 5) to be moved to a different position. In other words the value 153350000 was needed to be subtracted from all values in the column 4 and 5.

This problem is a perfect case for the use of user defined expressions in Ebe. Here are the example input file and example output file (which contains at the position 4 and 5 the index subtraction):

Listing 4. Input example file for ebe (with line breaks for readability).

```
chr1 hg38_knownGene exon 153357854
153357881 0.000000 + . gene_id
"ENST00000368738.4"; transcript_id
"ENST00000368738.4";
```

Listing 5. Output example file for ebe (with line breaks for readability).

```
chr1 hg38_knownGene exon {!$-153350000!}
{!$-153350000!} 0.000000 + . gene_id
"ENST00000368738.4"; transcript_id
"ENST00000368738.4";
```

In the output example file (listing 5) the user defined expression are between the {! and !} control sequences and define the expression to be done over the value at the expression's position. Since this is the only edit needed, Ebe compiles and interprets this within milliseconds. The generated Ebel file can then be used for any other GTF file (since GTF format is standardized and does not change structure [12]) and handles even large 50 MB file (123 705 lines) within 13 seconds time (on Intel® Core™ i5-4690 CPU).

7. Conclusions

Ebe displays a promise as an editing tool for not only non-programmers. It requires minimum knowledge of the tool, finds the algorithm for the user and even allows to edit multiple similar files at once.

[3] https://github.com/FastLED/FastLED/wiki/Pixel-reference

Although Ebe does not yet contain all the wanted features and optimizations, it still allows for some advanced edits, which it can find in a short time and most importantly it can do large simple edits, which might be lot of times needed more than the complex ones. With the addition of user defined expressions Ebe can also tackle more complex problems.

Ebe also greatly showcases the possibilities and power of genetic programming for code generation and as an alternative for deep learning or neural network in some cases.

As for the future of Ebe – it is still in development as a free and open source tool for anyone to try it out, with stable releases being published. The goal is to get Ebe to the point, where it can do more complicated edits faster than using other file editing approaches or at least in a similar time frame without any additional help from the user. Graphical interface is also considered for future releases to make Ebe more usable to non-programmers.

Since Ebe is open source, anyone is highly encouraged to try it out, play around with it, change parts of it or even integrate Ebe or parts of it into their own project. It also allows for easy parser extensibility, so new file formats can be easily added into Ebe.

Acknowledgements

I would like to thank my supervisor prof. Ing. Lukáš Sekanina, Ph.D. for his help and Bc. Klára Ungrová for the design of Ebe's logo.

References

[1] Alfred V. Aho, Brian W. Kernighan, and Peter J. Weinberger. *The AWK Programming Language*. Addison-Wesley Longman Publishing Co., Inc., USA, 1987.

[2] Canonical. Ubuntu manpage: Awk - pattern scanning and processing language. http://manpages.ubuntu.com/manpages/trusty/man1/awk.1posix.html.

[3] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

[4] Wildor Ferrel and Luis Alfaro. Genetic programming-based code generation for arduino. *International Journal of Advanced Computer Science and Applications*, 11(11), 2020.

[5] Keith D. Cooper, Philip J. Schielke, and Devika Subramanian. Optimizing for reduced code space using genetic algorithms. In *Proceedings of the ACM SIGPLAN 1999 Workshop on Languages, Compilers, and Tools for Embedded Systems*, LCTES '99, page 1–9, New York, NY, USA, 1999. Association for Computing Machinery.

[6] Stephanie Forrest, ThanhVu Nguyen, Westley Weimer, and Claire Le Goues. A genetic programming approach to automated software repair. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, GECCO '09, page 947–954, New York, NY, USA, 2009. Association for Computing Machinery.

[7] Yuchen Wang and Rong Xie. Pixel-based approach for generating original and imitating evolutionary art. *Electronics*, 9(8), 2020.

[8] Dragan Matić. A genetic algorithm for composing music. *Yugoslav Journal of Operations Research*, 20, 01 2010.

[9] Julian Miller. *Cartesian Genetic Programming*, volume 43. 06 2003.

[10] John R. Koza. Genetic programming - on the programming of computers by means of natural selection. In *Complex adaptive systems*, 1993.

[11] Hitoshi Iba, Ji Feng, and Hossein Izadi Rad. Gp-rvm: Genetic programing-based symbolic regression using relevance vector machine. In *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 255–262, 2018.

[12] GFF/GTF File Format. Definition and supported options. http://www.ensembl.org/info/website/upload/gff.html.