Using the MONITOR in PISA

Lothar Thiele [*]

1 Objective

The purpose of this document is to provide a documentation for the MONITOR program in the PISA framework [1, 2]. It can be used for those who want to (a) apply the existing MONITOR program together with existing VARIATOR and SELECTOR programs, or to (b) add new SELECTOR and/or VARIATOR programs that are compatible with it. In order to properly understand the document, knowledge about the basic specification and background of PISA as described in [1] is necessary.

The purpose of the MONITOR is

- to observe the communication between a VARIATOR and SELECTOR program in order to store information about the generated populations in text files,
- to automatically invoke several optimization runs as an input to a subsequent statistical analysis, and
- to control the VARIATOR and SELECTOR programs.

In particular, the monitor initiates a number of R runs of a specific selector-variator pair with different seeds of the variator. Each run consists of a number of G generations, each one generating a population. The monitor writes a set of output files where each one is related to exactly one generation. The output file for generation x contains information about the populations after x generations for all runs.

Following the concept described in [1], the MONITOR is an independent program and the communication to the other entities (VARIATOR and SELECTOR) is done via a shared file system. The MONITOR writes the collected information into a set of human-readable files that can be used by other programs for further processing.

[*]Department Information Technology and Electrical Engineering, Computer Engineering and Networks Lab, ETH Zürich, Switzerland, thiele@tik.ee.ethz.ch

2 Embedding

2.1 File System

This section describes, how the MONITOR, SELECTOR and VARIATOR programs preferably are embedded into a file system and the naming conventions for the associated communication files. Each of the programs takes a set of arguments which determine the location of the various files it needs. In order to come to a concise description, we will use a set of abbreviations, see Table 1.

abbr.	description	program
CV	filename base of communication files	VARIATOR
PV	filename base of parameter file	VARIATOR
CS	filename base of communication files	SELECTOR
PS	filename base of parameter file	SELECTOR
OM	filename base of output files	MONITOR
PM	filename base of parameter file	MONITOR

Table 1: Naming conventions for filename bases.

By convention, the SELECTOR and VARIATOR programs read the local parameter files that contain specific information (`PS` and `PV`) and they communicate with each other through files with names `Carc`, `Ccfg`, `Cini`, `Csel`, `Csta` and `Cvar`, see Table 2.

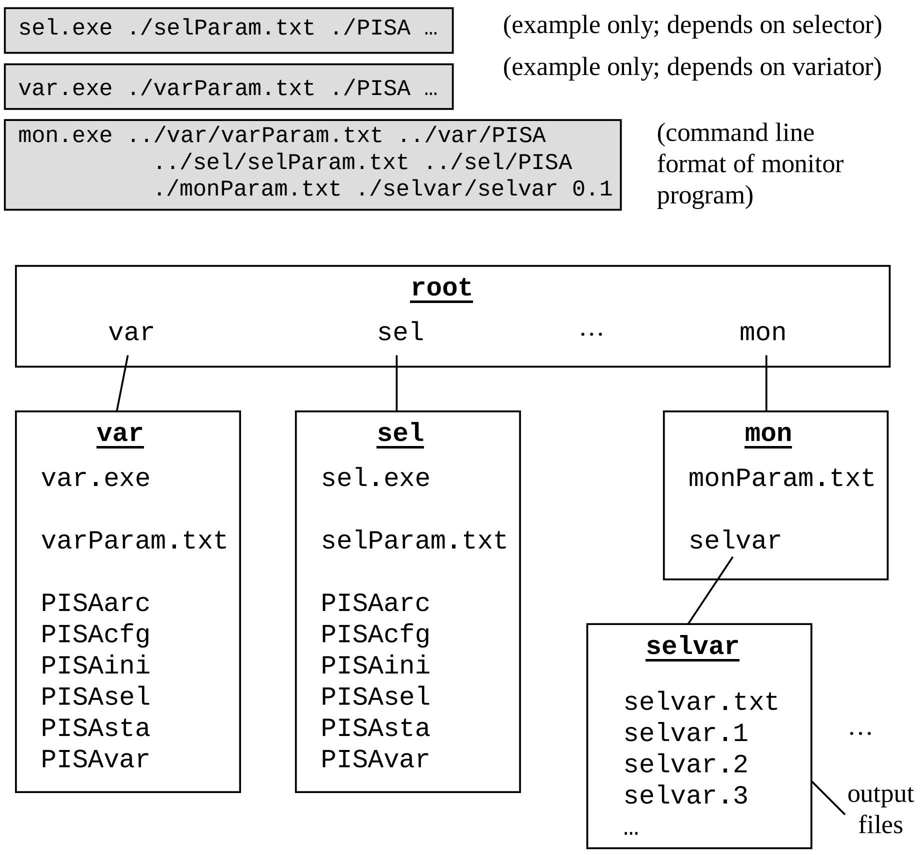
abbr.	description	written by
Carc	archive of SELECTOR	SELECTOR
Ccfg	configuration of SELECTOR and VARIATOR	given by user
Cini	initial population	VARIATOR
Csel	selected parent individuals	SELECTOR
Csta	state variable	SELECTOR and VARIATOR
Cvar	generated offsprings	VARIATOR

Table 2: Naming conventions for communication files.

If no MONITOR program is used, the filename base used by the SELECTOR and VARIATOR point to the same name base, namely `C`; in other words, we have `CV`=`CS`=`C`. If a monitor is used, this direct communication is replaced by a transfer between files `CVxxx` and `CSxxx` through the MONITOR, where `xxx` may denote `arc`, `cfg`, `ini`, `sel`, `sta` or `var`. An example of a preferred embedding of the programs in a file system is shown in Fig. 1. Here, each program is residing in its own directory. If we have the SELECTOR program `sel.exe`, the VARIATOR program `var.exe` and the MONITOR program `mon.exe`, they may be called `sel`, `var` and `mon`, for example. The command line arguments for the SELECTOR and VARIATOR programs are examples only, the concrete form depends on the particular implementation. Nevertheless, they all have as arguments `PV`, `CV` or `PS` and `CS`, respectively. The MONITOR program command line has the form

```
mon.exe PV CV PS CS PM OM poll
```

where poll denotes the polling interval of the monitor. Note that there may be many different VARIATOR and SELECTOR programs in general. They be located in parallel directories. By assigning appropriate filename bases to the output of the monitor (`OM`), for each new pair, the sets of output files can be written to separate directories.



```
sel.exe ./selParam.txt ./PISA …
var.exe ./varParam.txt ./PISA …
mon.exe ../var/varParam.txt ../var/PISA
          ../sel/selParam.txt ../sel/PISA
          ./monParam.txt ./selvar/selvar 0.1
```

Figure 1: Example of an embedding into a file system.

The output files `OM.txt` and `OM.x` where `x` is the generation number (`selvar.txt`, `selvar.1`, `selvar.2`, ... in the example) contain the result of the set of runs that has been controlled and observed by the monitor, see Table 3. The monitor initiates a number of runs of the selector-variator pair with different seeds of the variator. Each run consists of a number of generations. An output file `OM.x` now contains information about the populations that have been generated in generation x in all runs. The exact format of `FNMB.x` and the kind of information that can be stored will be described later in more detail.

abbr.	description
OM.txt	text file that contains information about the run, e.g. the date of the run and the parameter files used
OM.x	x can be any non-negative integer; the file contains information about the populations that have been generated in the generation x of all runs

Table 3: Output files of the MONITOR program.

2.2 Communication and States

This chapter describes the communication mechanism between the MONITOR, the SELECTOR and the VARIATOR. This information is relevant for those who want to design new monitors, selectors or variators. Knowledge of [1] is required. The algorithms 2, 1 and 3 in pseudo code contain simplified templates for the programs of the SELECTOR, VARIATOR and MONITOR in order to clarify the basic communication mechanisms.

The MONITOR according to Alg. 3 controls the operation of the other two programs and stores the archives generated in the intermediate generations into the files `OM.x`, see line 13. The function **ResetAll()** in line 8 updates the seed of the variator that needs to be stored in the parameter file `PV`.

The VARIATOR and SELECTOR need to satisfy some properties in order to be compatible with the above communication scheme. These properties are mentioned already in [1], but not as mandatory requirements:

- The VARIATOR and SELECTOR need to implement a reset functionality that cleans up all internal state. It is triggered by the state variables 8 and 10, respectively. After finishing the reset, the programs need to change the state to 9 and 11, respectively.

- It is recommended, that the Variator and Selector also implement the termination functionality triggered by states 4 and 6, respectively.
- The parameter file `PV` of the Variator needs to contain information about the seed that is used. The seed is changed by the Monitor for each new run. Therefore, `PV` needs to contain a line starting with the key word `seed` followed by a space and an integer, e.g. `seed 11` .

Algorithm 1 Pseudocode of Variator

```
PROCESS var()
 1: CVstat ⇐ 0;
 2: loop
 3:   if CVstat = 0 then
 4:     readCommonParameters(CVcfg); readParameters(PV);
 5:     writeInitialPopulation(CVini); CVstat ⇐ 1;
 6:   if CVstat = 2 then
 7:     readArchive(CVarc); readSample(CVsel); writeOffspring(CVvar);
 8:     CVstat ⇐ 3;
 9:   if CVstat = 8 then
10:     resetVariator(); CVstat ⇐ 9;
11:   if CVstat = 4 then
12:     CVstat ⇐ 5; return();
13:   wait(poll);
```

Algorithm 2 Pseudocode of Selector

```
PROCESS sel()
 1: loop
 2:   if CSstat = 1 then
 3:     readCommonParameters(CScfg); readParameters(PS);
 4:     readInitialPopulation(CSini); CSstat ⇐ 2;
 5:   if CSstat = 3 then
 6:     readOffspring(CSvar); writeArchive(CSarc); writeSample(CSsel);
 7:     CSstat ⇐ 2;
 8:   if CSstat = 10 then
 9:     resetSelector(); CSstat ⇐ 11;
10:   if CSstat = 6 then
11:     CSstat ⇐ 7; return();
12:   wait(poll);
```

Algorithm 3 Pseudocode of MONITOR

```
PROCESS mon()
writeContext(OM.txt); ResetAll();
for all runs $y = 1...R$ do
    for all generations $x = 1...G$ do
        loop
            if CVstat = 3 then
                copyFromTo(CVvar,CSvar); CSstat $\Leftarrow$ 3; break();
            wait(poll);
        loop
            if CSstat = 2 then
                copyFromTo(CSarc,CVarc); copyFromTo(CSsel,CVsel);
                CVstat $\Leftarrow$ 2; break();
            wait(poll);
        appendArchiveFromTo(CSarc,OM.x);
    ResetAll();
CVstat $\Leftarrow$ 4; CSstat $\Leftarrow$ 6; return();
FUNCTION ResetAll()
CVstat $\Leftarrow$ 8; CSstat $\Leftarrow$ 10;
loop
    if CVstat $\neq$ 9 then
        CVstat $\Leftarrow$ 8;
    if CSstat $\neq$ 11 then
        CSstat $\Leftarrow$ 10;
    if (CVstat = 9 $\wedge$ CSstat = 11) then
        updateSeed(PV); CVstat $\Leftarrow$ 0; break();
    wait(poll);
loop
    if CVstat = 1 then
        copyFromTo(CVini,CSini); CSstat $\Leftarrow$ 1; break();
    wait(poll);
loop
    if CSstat = 2 then
        copyFromTo(CSarc,CVarc); copyFromTo(CSsel,CVsel);
        CVstat $\Leftarrow$ 2; break();
    wait(poll);
return();
```

3 The Monitor

In this section, the functionality of the MONITOR is described in more detail. In particular, the MONITOR initiates a number of R runs of a specific selector-variator pair with different seeds of the variator. Each run consists of a number G of generations, each one generating a population. The MONITOR writes a set of output files `OM.x` where each one is related to exactly one generation $0 \leq x \leq G$. It contains information about the populations that have been determined in the particular generation in all runs. Details of this process can be determined using the parameter file `PM`.

3.1 Parameters

An example of a parameter file `PM` that contains the control information for the MONITOR is given in Table 4. The ordering of the entries must not be changed.

```
seed 13
numberOfRuns 50
numberOfGenerations 100
outputType online
outputSet 0
debug 0
```

Table 4: Example of a MONITOR parameter file `PM`.

seed The seeds of the variator are random numbers based on the integer seed of the MONITOR. Therefore, each new set of runs should yield identical results if the same seed of the monitor is chosen.

numberOfRuns This integer determines the number of runs R that are executed.

numberOfGenerations This integer denotes the number of generations G within each run. The generation number x with $0 \leq x \leq G$ is appended to the names of the output files. $x = 0$ denotes the first archive that is written by the selector, usually the initial population.

outputType The output type can be `all`, `online` or `offline`.

outputType = `all`: The archive of the selector of a generation x is directly appended to the output file `OM.x`, i.e. the objective vectors of all individuals are written.

outputType = `online`: Only the objective vectors of the Pareto set of the archive in generation x are written to `OM.x`.

outputType = `offline`: The objective vectors of the Pareto set of the union of the current and all previous generations are written to `OM.x`.

```
start commandLine
../var/varP.txt ../var/PISA ../sel/selP.txt ......
end commandLine

start monParameter
...... (contents of file PM)
end monParameter

start varCommonParameter
...... (contents of file CVcfg)
end varCommonParameter

start varParameter
...... (contents of file PV)
end varParameter

start selCommonParameter
...... (contents of file CScfg)
end selCommonParameter

start selParameter
...... (contents of file PS)
end selParameter
```

Table 5: Example of a MONITOR information file `OM.txt`.

outputSet The integer *outputSet* determines whether the archive of a generation x is written to an output file `OM.x` or not. If generations x are not written, the output file is not created.

outputSet $= 0$: Only the last generation $x = G$ is written.

$1 \leq$ *outputSet* $\leq G$: If x is a non-negative integer multiple of *outputSet*, then the corresponding archive is written to `OM.x`. For example, if *ouputSet* $= 3$, $G = 9$, then generations $x \in \{0, 3, 6, 9\}$ are written. Note, that in this case, the initial archive with $x = 0$ is written always.

debug The value can be 0 (no output is generated on stdout) or 1 (information about the state of the monitor is continuously written to stdout).

3.2 Output Files

The Monitor creates the file `OM.txt` that contains information about the run and files `OM.x` for some $0 \leq x \leq G$ containing collected archives. An example of an information file is give in Table 5. It contains the command line argument of the call of the Monitor program as well as all parameter files. The sections are separated by the key words `start`, `end` with arguments `commandLine`, `monParameter`, `varCommonParameter`, `varParameter`, `selCommonParameter` and `selParameter`.

The output files `OM.x` contain the objective vectors of individuals in the archives of the selector at generations x. The sets of the different runs are separated by blank lines. Each line contains floating point numbers corresponding to the objective vector of the individual.

The total number of digits after the decimal point is 9. For the exponent value *exp* applies $-37 \leq exp \leq 37$ and the number of exponent digits is smaller than 3. In the programming language C, the numbers could be generated using `printf("%.9e", ...)`. The BNF reads as follows:

```
Float := ('+'?  | '-'?)  FloatN
FloatN := (Digit '.'  Digit+ Exp?)
Exp := ('e+' | 'e-') Digit+
Digit := '0-9'
```

```
-1.324350000e+2 -4.723235445e-015
2.434243690e-004 -2.000000000e+03
-1.324350000e+2 2.434243690e-004
-2.000000000e+03 -1.324350000e+2

2.434243690e-004 4.723535445e-015
-2.000000000e+03 -4.723253445e-015
-4.732535445e-015 2.434243690e-004
-2.000000000e+03 -1.324350000e+2
-1.324350000e+2 -4.723235445e-015

2.434243690e-004 -1.324350000e+2
-4.723535445e-015 2.434243690e-004
2.434243690e-004 -2.000000000e+03
```

Table 6: Example of a Monitor output file `OM.x` for a two dimensional objective and 3 runs.

References

[1] Stefan Bleuler, Marco Laumanns, Lothar Thiele, and Eckart Zitzler. PISA – A Platform and Programming Language Independent Interface for Search Algorithms. In Carlos M. Fonseca, Peter J. Fleming, Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele, editors, *Evolutionary Multi-Criterion Optimization (EMO 2003)*, volume 2632/2003 of *LNCS*, pages 494 – 508. Springer-Verlag Heidelberg, 2003.

[2] Stefan Bleuler, Marco Laumanns, Lothar Thiele, and Eckart Zitzler. The PISA Homepage. http://www.tik.ee.ethz.ch/pisa/, 2003.