PolySCIP user guide

Sebastian Schenker

Contents

1 General information 1

2 Installation 1

3 Usage 1

4 File format 3

5 User-friendly .mop file generation 4

1 General information

PolySCIP is a solver for multi-criteria integer programming problems and multi-criteria linear programming problems. In other words, it aims at solving optimization problems of the form:

$$\begin{aligned} \min \,/\, \max \; & (c_1^\top x, \ldots, c_k^\top x) \\ \text{s.t. } & Ax \leq b, \\ & x \in \mathbb{Z}^n \vee \mathbb{Q}^n, \end{aligned}$$

where $k \geq 2,\ A \in \mathbb{Q}^{m \times n},\ b \in \mathbb{Q}^m$.

The name PolySCIP is composed of Poly (from the Greek πολύς meaning "many") and SCIP. PolySCIP is part of SCIP and its source code resides in the 'applications' directory.

2 Installation

See the INSTALL file in the PolySCIP directory or the section 'Installation' at http://polyscip.zib.de.

3 Usage

The problem file (in `MOP` format) is the only mandatory command line argument:

```
polyscip path_to_problem_file/problem_file.mop
```

runs PolySCIP on the given problem file. To switch off the SCIP solver output you can execute

```
polyscip -p scipmip.set path_to_problem_file/problem_file.mop
```

where `scipmip.set` is an existing SCIP parameter file (in the PolySCIP folder) containing the line `display/verblevel=0`.

To get a complete list of available PolySCIP command line arguments execute

```
polyscip -h
```

-h, --help Display usage information and exit

-p | --params <`param_file.set`> Specify a file consisting of SCIP parameter settings

- PolySCIP comes with the parameter settings file *scipmip.set*
- A list of all available SCIP parameters is available at http://scip.zib.de
- To switch, e.g., the verbosity level of the internal SCIP solution process to 1, write `display/verblevel=1` in the *scipmip.set* file and run *polyscip* with `-p scipmip.set`

-W | --writeSolsPath <path> Path where the solution file should be written to if **-w** was set

-e | --Epsilon <double> Specify epsilon used in computation of unsupported points; the default value is 1e-3

-d | --Delta <double> Specify delta used in computation of feasible boxes; the default value is 0.01

-r | –round <5|10|15> Round the weighted objective coefficient used in the function 'setWeightedObjective' at the 'r'-th decimal position; this might be helpful in case of numberical troubles with unbounded rays

-t | --timeLimit <seconds> Set a time limit in seconds on the overall SCIP computation time

-o | --noOutcomes Switch off the output of computed outcomes

-s | --noSolutions Switch off the output of computed solutions

-w | --writeResults Write results to a file; default path is ./

-v | --verbose Switch on verbose PolySCIP output

-x | --extremal Compute only supported non-dominated extreme point results

--version Display PolySCIP version

4 File format

The PolySCIP file format (with suffix `.mop`) is based on the widely used `MPS` file format (see [1], [2]). `MPS` is column-oriented and all model components (variables, rows, etc.) receive a name. An objective in `MPS` is indicated by an `N` followed by the name in the `ROWS` section. Similarly, in the `MOP` format the objectives are indicated by `N` followed by the name in the `ROWS` section. In general, `MPS` might not be as human readable as other formats. However, one of the main reasons to base the file format of PolySCIP on it is its easy extension towards several objectives and its wide availability in most of the linear and integer programming software packages such that available `MPS` parsers could easily be adjusted to parse an `.mop` file as well. Furthermore, no user is expected to write `.mop` files by hand, but to use a modelling language that does the job. See Section 5 for a description of how to use the freely available Zimpl and the Python script `mult_zimpl_to_mop.py` (comes with PolySCIP) to generate `.mop` files.

The following simple equation-based bi-criteria integer problem

$$
\begin{array}{lll}
\text{maximize} & \text{Obj1:} & 3x_1 + 2x_2 - 4x_3 \\
& \text{Obj2:} & x_1 + x_2 + 2x_3 \\
\text{subject to} & & \\
& \text{Eqn:} & x_1 + x_2 + x_3 \quad = 2 \\
& \text{Lower:} & x_1 + 0.4x_2 \quad \leq 1.5 \\
& & x_1,\ x_2,\ x_3 \quad \geq 0 \\
& & x_1,\ x_2,\ x_3 \quad \in \mathbb{Z}
\end{array}
\tag{1}
$$

is written in MOP format as follows:

```
NAME          BICRIT
OBJSENSE
 MAX
ROWS
 N  Obj1
 N  Obj2
 E  Eqn
 L  Lower
COLUMNS
    x#1       Lower             1
    x#1       Eqn               1
    x#1       Obj2              1
    x#1       Obj1              3
    x#2       Lower           0.4
    x#2       Eqn               1
    x#2       Obj2              1
    x#2       Obj1              2
```

```
    x#3       Eqn                1
    x#3       Obj2               2
    x#3       Obj1              -4
RHS
    RHS       Eqn                2
    RHS       Lower            1.5
BOUNDS
 LI BOUND     x#1                0
 LI BOUND     x#2                0
 LI BOUND     x#3                0
ENDATA
```

5 User-friendly `.mop` file generation

Zimpl is a freely available modelling language (also part of the SCIP Optimization Suite) to translate a mathematical model of a problem into a mathematical program in `.mps` (or `.lp`) file format. Together with the `mult_zimpl_to_mop.py` script (located in the 'mult_zimpl' directory of PolySCIP) it can/should be used to generate your `.mop` files. For a more detailed description of Zimpl, see the Zimpl user guide. In this section we will only describe how to make use of it, but not all options how to write different models. Zimpl does generally not support several objectives; this is where `mult_zimpl_to_mop.py` comes into play. It takes an 'extended' Zimpl file containing several objectives, internally rewrites all but the first objectives into constraints, calls Zimpl on the rewritten file and changes the file generated by Zimpl containing 'artificial' constraint indicators back to objective indicators which yields an `.mop` file.

- Zimpl comes with the SCIP Optimization Suite
  - Please see the INSTALL file of the SCIP Optimization Suite (you basically just need the GMP library in order to build).
- `mult_zimpl_to_mop.py` is a Python3 script and comes with PolySCIP; it is located in the 'mult_zimpl' directory
  - Execute `python3 mult_zimpl_to_mop.py your_model.zpl` to run it on the file `your_model.zpl` containing your multi-criteria model
  - The following command line arguments are available

    -h, --help Show the help message and exit

    -o <basename> Basename used for the output file; the default is the basename of the input file

    -p <path> Directory where the generated `.mop` file should be saved

    --path_to_zimpl <path> Directory where your *zimpl* binary can be found

E.g., if the Zimpl executable is not installed globally but in `/home/user/bin`, and, furthermore, you would like to save the generated `.mop` file under `/tmp`, then execute `python3 mult_zimpl_to_mop.py -p /tmp/ -path_to_zimpl /home/user/bin model.zpl`

Please note (in the following examples) that the direction of optimization, i.e., `minimize` or `maximize`, is declared only once followed by the first objective. All other objectives follow without a direction specification implying that all objectives are assumed to be either minimized or maximized.

Example 5.1. The bi-criteria maximization problem (1) can be modelled as follows:

```
set I := {1..3};
param c1[I] := <1> 3, <2> 2, <3> -4;     #coefficients of the first objective
param c2[I] := <3> 2 default 1;          #coefficients of the second objective
param low[I] := <1> 1, <2> 0.4, <3> 0;   #coefficients of the lower constraint
var x[I] integer >= 0;

maximize Obj1: sum <i> in I: c1[i]*x[i];
Obj2: sum <i> in I: c2[i]*x[i];

subto Eqn: sum <i> in I: x[i] == 2;
subto Lower: sum <i> in I: low[i]*x[i] <= 1.5;
```

Saving the file, e.g., as test.zpl and running `mult_zimpl_to_mop.py` on it would generate a file named `test.mop` which can be solved with PolySCIP.

Example 5.2. Assume you want to model a tri-criteria assignment problem and your data is stored in a file named `my_data.txt` in the following format:

```
3
5
6, 1, 20, 2, 3,
2, 6, 9, 10, 18,
1, 6, 20, 5, 9,
6, 8, 6, 9, 6,
7, 10, 10, 6, 2
,
17, 20, 8, 8, 20,
10, 13, 1, 10, 15,
4, 11, 1, 13, 1,
19, 13, 7, 18, 17,
15, 3, 5, 1, 11
,
10, 7, 1, 19, 12,
2, 15, 12, 10, 3,
11, 20, 16, 12, 9,
```

10, 15, 20, 11, 7,
1, 9, 20, 7, 6

The first line specifies the number of objectives, the second line specifies number of variables and the following three 5×5 matrices contain the objective value coefficients. The tri-criteria assignment problem could then be modeled as follows:

```
param prob_file := "my_data.txt";
param no_objs := read prob_file as "1n" use 1;
param no_vars := read prob_file as "1n" use 1 skip 1;

set I := {1..no_vars};
set T := {1..no_objs*no_vars*no_vars};
param coeffs[T] := read prob_file as "n+" match "[0-9]+" skip 2;
param offset := no_vars*no_vars;
param Obj1[<i,j> in I*I] := coeffs[(i-1)*no_vars + j];
param Obj2[<i,j> in I*I] := coeffs[(i-1)*no_vars + j + offset];
param Obj3[<i,j> in I*I] := coeffs[(i-1)*no_vars + j + 2*offset];

var x[I*I] binary;

minimize Obj1: sum <i,j> in I*I: Obj1[i,j]*x[i,j];
Obj2: sum <i,j> in I*I: Obj2[i,j]*x[i,j];
Obj3: sum <i,j> in I*I: Obj3[i,j]*x[i,j];

subto row: forall <i> in I do
      sum <j> in I: x[i,j] == 1;
subto col: forall <i> in I do
      sum <j> in I: x[j,i] == 1;
```

Again, saving the file, e.g., as testCube.zpl and running `mult_zimpl_to_mop.py` on it would generate a file named `testCube.mop` which can be solved with PolySCIP.

Please see the Zimpl user guide for more modelling details.

References

[1] MPS format (short), https://en.wikipedia.org/wiki/MPS_%28format%29

[2] MPS format (detailed), http://lpsolve.sourceforge.net/5.5/mps-format.htm

[3] ZIMPL webpage, http://zimpl.scip.de