Using `Siplib.jl`

Yongkyu Cho

POSTECH
E-mail: jyg1124@postech.ac.kr

June 22, 2018

1 Prerequisites

We assume that you are in Linux environment. To use `Siplib.jl`, you need to perform the following steps:

1. Download `Julia` $\geq$ 0.6.2 and set up.
2. Install `Julia` packages: `Distributions.jl`, `StructJuMP.jl`, `PyPlot.jl` by executing
   - `Pkg.add("Distributions")`
   - `Pkg.add("StructJuMP")`
   - `Pkg.add("PyPlot")`

   Then, execute `Pkg.update()` to make them up-to-date.
3. Download and place the `Siplib.jl` package to any directory (say `dir`) in your computer
4. Open a terminal and change working directory to `dir/Siplib/src/`:
   - `cd dir/Siplib/src`
5. Run `Julia` in that directory
6. excute `include("Siplib.jl")`
7. excute `using Siplib`

Then, you are all set to use `Siplib.jl`. To make it sure, execute the following line to generate DCAP_2_2_2_10 instance:

```
julia> generateSMPS(:DCAP, [2,2,2,10])
```

If it works well, you will see the three files in `dir/Siplib/instance`:

- DCAP_2_2_2_10.cor
- DCAP_2_2_2_10.tim
- DCAP_2_2_2_10.sto

2 Basic usage

2.1 Generating SMPS instance

Use `generateSMPS(problem, params_arr)` with proper values in Table 1 to generate SMPS instances, for example:

```
julia> generateSMPS(:DCAP, [2,2,2,10])
```

The default directory in which the files are stored is `dir/Siplib/instance`. You can change the directory by specifying explicit path:

```
julia> generateSMPS(:DCAP, [2,2,2,10], "another/directory")
```

`generateSMPS(problem, params_arr)` has four more optional keyword arguments: `seed`, `splice`, `genericnames`, `lprelax`. For details, please see the manual.

Table 1: Acceptable values for `problem` and `params_arr` arguments pairs

problem	params_arr	Remark
:DCAP	[R, T, N, $\mathcal{S}$]	All parameters are integer.
:MPTSPs	[D, N, $\mathcal{S}$]	String D $\in$ {"D0", "D1", "D2", "D3"}. All other parameters are integer.
:SIZES	[$\mathcal{S}$]	Integer $\mathcal{S}$.
:SMKP	[I, $\mathcal{S}$]	All parameters are integer.
:SSLP	[I, J, $\mathcal{S}$]	All parameters are integer.
:SUC	[D, $\mathcal{S}$]	String D $\in$ {"FallWD", "FallWE", "WinterWD", "WinterWE", "SpringWD", "SpringWE", "SummerWD", "SummerWE"}. Integer $\mathcal{S} \leq 1000$.

Note: $\mathcal{S}$ is always the number of scenarios.

2.2 Plotting sparsity

Use `generateSparsityPlots(problem, params_arr)` to generate the sparsity plots in constraint matrix. Excute the following lines:

```
julia> generateSparsityPlots(:DCAP, [2,2,2,10])
```

The default directory in which the plots are stored is `dir/Siplib/plot`. You can change the directory by specifying explicit path:

```
julia> generateSparsityPlots(:DCAP, [2,2,2,10], "another/directory")
```

2.3 Reporting size and sparsity

Use `getSize(problem, params_arr)` and `getSparsity(problem, params_arr)` to get the size and sparsity information. Excute the following lines to construct objects that contain the information:

```
julia> size = getSize(:DCAP, [2,2,2,10])
julia> sparsity = getSparsity(:DCAP, [2,2,2,10])
```