The .pcs parameter configuration space format

Frank Hutter
Department of Computer Science
Freiburg University
fh@informatik.uni-freiburg.de

Steve Ramage
Department of Computer Science
University of British Columbia
seramage@cs.ubc.ca

April 27, 2013

This document specifies the .pcs parameter configuration space format, a simple, human-readable file format for the description of an algorithm's configurable parameters, their possible values, as well as any parameter dependencies. The .pcs format requires each line to contain one of the following 3 clauses, or only whitespace/comments.

- **Parameter Declaration Clauses** specify the names of parameters, their domains, and default values.
- **Conditional Parameter Clauses** specify when a parameter is active/inactive.
- **Forbidden Parameter Clauses** specify when a combination of parameter settings is illegal.

Comments are allowed throughout the file; they begin with a #, and run to the end of a line.

1 Parameter Declaration Clauses

The .pcs format supports two types of parameters: categorical and numeric.

1.1 Categorical parameters

Categorical parameters take one of a finite set of values. Each line specifying a categorical parameter should be of the form:

```
<parameter_name> {<value 1>, ..., <value N>} [<default value>]
```

where '`<default value>`' has to be one of the set of possible values.

Example 1:

```
decision-heuristic {1,2,3} [1]
```

This means that the parameter '`decision-heuristic`' can be given one of three possible values, with the default assignment being '`1`'.

Example 2:

```
@1:loops {common,distinct,shared,no}[no]
```

In this example, the somewhat cryptic parameter name '`@1:loops`' is perfectly legal; the only forbidden characters in parameter names are spaces, commas, quotes, and parentheses. Categorical parameter values are also strings with the same restrictions; in particular, there is no restriction for categorical parameter values to be numbers.

Example 3:

```
DS {TinyDataStructure, FastDataStructure}[TinyDataStructure]
```

As this example shows, the parameter values can even be Java class names (to be used, e.g., via reflection).

Example 4:

```
random-variable-frequency {0, 0.05, 0.1, 0.2} [0.05]
```

Finally, as this example shows, numerical parameters can trivially be treated as categorical ones by simply discretizing their domain (selecting a subset of reasonable values).

1.2 Numerical parameters

Numerical parameters (both real and integer) are specified as follows:

```
<parameter_name> [<min value>, <max value>] [<default value>] [i] [l]
```

The trailing '`i`' and/or trailing '`l`' are optional. The '`i`' means the parameter is an integer parameter, and the '`l`' means that the parameter domain should be log-transformed for optimization (see Examples 3 and 4 below).

Example 1:

```
sp-rand-var-dec-scaling [0.3, 1.1] [1]
```

Parameter `sp-rand-var-dec-scaling` is real-valued with a default value of 1, and we can choose values for it from the (closed) interval `[0.3, 1.1]`. Note that there may be other parameter values outside this interval that are in principle legal values for the parameter (e.g., your solver might accept any positive floating point value for the parameter). What you specify here is the range that automated configuration procedures should search (i.e., a range you expect a priori to contain good values); of course, every value in the specified range must be legal. There is a tradeoff in choosing the best range size; see Section 4 for some tips on defining a 'good' parameter space.

Example 2:

```
mult-factor [2, 15] [5]i
```

Parameter `mult-factor` is integer-valued, takes any integer value between `2` and `15` (inclusive), and has a default value of 5. Technically, one could also specify this as a categorical parameter with possible values {`2,3,4,5,6,7,8,9,10,11,12,13,14,15`}. However, categorical parameters are not ordered, and using an integer parameter allows the configuration procedure to make use of the natural order relation (this is useful since, a priori, we expect close-by values to yield similar performance).

Example 3:

```
DLSc [0.00001, 0.1] [0.01]l
```

Parameter `DLSc` is real-valued with a default value of `0.01`, and we can choose values for it from the (closed) interval `[0.00001, 0.1]`. The trailing '`l`' denotes that this parameter naturally varies on a log scale. If we were to discretize the parameter, a natural choice would be {`0.00001, 0.0001, 0.001, 0.01, 0.1`}. That means, a priori the distance between parameter values `0.001` and `0.01` is identical to that between `0.01` and `0.1` (after a $\log_{10}$ transformation, `0.001`, `0.01`, and `0.1` become -3, -2, and -1, respectively). We express this natural variation on a log scale by the '`l`' flag. See Section 4 for further tips on transformations.

Example 4:

```
first-restart [10, 1000] [100]il
```

Parameter `first-restart` is integer-valued with a default value of `100`, and we can choose values for it from the (closed) interval `[10, 1000]`. It also varies naturally on a logarithmic scale. For example, due to this logarithmic scale, after the transformation drawing a uniform random value of `first-restart` will yield a number below `100` half the time.

Restrictions

- Numerical integer parameters must have their lower and upper bounds specified as integers, and the default must also be an integer.
- The bounds for parameters with a log scale must be strictly positive.

2 Conditional Parameter Clause

Depending on the instantiation of some 'higher-level' parameters, certain 'lower-level' parameters may not be active. For example, the subparameters of a heuristic are not important (i.e., active) if the heuristic is not selected. All parameters are considered to be active by default, and conditional parameter clauses express under which conditions a parameter is not active. The syntax for conditional parameter clauses is as follows:

```
<child name> | <parent name> in {<parent val1>, ..., <parent valK>}
```

This can be read as "The child parameter `<child name>` is only active if the parent parameter `<parent name>` takes one of the K specified values." Parameters that are not listed as a child parameter in any conditional parameter clause are always active. A parameter can also be listed as a child in multiple conditional parameter clauses, and it is only active if the conditions of each such clause are met.

Example:

```
sort-algo{quick,insertion,merge,heap,stooge,bogo} [bogo]
quick-revert-to-insertion{1,2,4,8,16,32,64} [16]
quick-revert-to-insertion|sort-algo in {quick}
```

In this example, `quick-revert-to-insertion` is conditional on the `sort-algo` parameter being set to `quick`, and will be ignored otherwise.

3 Forbidden Parameter Clauses

Forbidden Parameters are combinations of parameter values which are invalid (e.g., a certain data structure may be incompatible with a lazy heuristic that does not update the data structure, resulting in incorrect algorithm behaviour). Configuration methods should never try to run an algorithm with a forbidden parameter configuration. The syntax for forbidden parameter combinations is as follows:

```
{<parameter name 1>=<value 1>, ..., <parameter name N>=<value N>}
```

Example:

```
DSF {DataStructure1, DataStructure2, DataStructure3}[DataStructure1]
PreProc {NoPreProc, SimplePreproc, ComplexPreproc}[ComplexPreproc]
{DSF=DataStructure2, PreProc=ComplexPreproc}
{DSF=DataStructure3, PreProc=SimplePreproc}
{DSF=DataStructure3, PreProc=ComplexPreproc}
```

In this example, there are different data structures and different simplifications. `DataStructure2` is incompatible with `ComplexPreproc`, and `DataStructure2` is incompatible with both `SimplePreproc` and `ComplexPreproc`. Note that the default parameter setting is not allowed to contain a forbidden combination of parameter values.

4 Interplay With Algorithm Wrapper

This file specifies the parameters of an algorithm, but it does not specify how to pass these parameters to your algorithm. The specifics of that are defined by the algorithm wrapper, which receives an instantiation of parameters, calls your algorithm with that instantiation (typically through a command line call) and returns the performance result.

In typical cases, there is a direct 1-to-1 mapping between the parameters in the .pcs file and the parameters the corresponding algorithm accepts. In those cases, the wrapper simply hands the parameters over to the algorithm. However, the wrapper can also implement transformations of parameters, scale parameters by instance size, etc.