Quick start guide for ParamILS

Frank Hutter
Department of Computer Science
University of British Columbia
Vancouver, BC V6T 1Z4, Canada. hutter@cs.ubc.ca

May 22, 2007

1 Introduction

ParamILS is a tool for parameter optimization. It works for any parameterized algorithm whose parameters can be discretized. ParamILS searches through the space of possible parameter configurations, evaluating configurations by running the algorithm to be optimized on a set of benchmark instances.

Users provide

- a parametric algorithm $\mathcal{A}$ (executable to be called from the command line),
- all parameters and their possible values (parameters need to be configurable from the command line), and
- a set of benchmark problems, $\mathcal{S}$.

Users can also choose from a multitude of optimization objectives, reaching from minimizing average runtime to maximizing median approximation qualities.

ParamILS then executes algorithm $\mathcal{A}$ with different combinations of parameters on instances sampled from $\mathcal{S}$, searching for the configuration that yields overall best performance across the benchmark problems. For details, see *Frank Hutter, Holger Hoos, and Thomas Stützle. Automatic Algorithm Configuration based on Local Search. In Proceedings of the Twenty-Second Conference on Artificial Intelligence (AAAI-07).* If you use ParamILS in your research, please cite this article.

2 Download and Configuration

Download the file ParamILS.tar.bz2 from the ParamILS website, move it to a new folder and extract (tar jxvf paramils.tar.bz2).

The version of ParamILS being distributed is a simple collection of Ruby scripts (Ruby is a scripting language similar to Perl, but object-oriented and *a lot* easier to read – Ruby can be freely downloaded for all platforms from `http://www.ruby-lang.org/en/`, and there is also extensive documentation available). The ParamILS scripts are research code, not very clean and also containing some dead code.

In order to help Ruby find the ParamILS scripts, you will need to edit the environment variable RUBYLIB. For example, if you extracted ParamILS to the folder /tools/paramils and you are running a csh shell, add the following line to ./csh_init/cshrc:

```
setenv RUBYLIB $HOME/tools/paramils
```

3 Running a simple example

The zip file contains two sample applications, in which the local search algorithm SAPS is optimized for a quasigroup with holes instance, and for a small number of morphed graph colouring instances.

Go to the directory `example_saps` and type

```
ruby ../param_ils_1_0_run.rb -algo "ruby saps_wrapper.rb" -numRun 0
-instance_seed_file QWH-single-instance/instance_seed_file_0.txt -db
0
```

This starts FocusedILS, one version of ParamILS, optimizing SAPS performance for a single QWH instance. (You probably want to cancel this optimization after a while by pressing CTRL-C.) The parameters in the above command have the following meaning:

algo An executable of the algorithm to optimize (here a ruby script wrapping the SAPS algorithm). The input/output specification for this executable is given in Section 6.2.

instance_seed_file The name of the file containing seeds and instances to use for the optimization. (In the terminology of our ParamILS paper, these specify which samples of the cost distribution should be used – e.g., if $N = 100$ the seeds and instances in the first 100 lines of this file will be used. The same seeds and instances are used for evaluating each parameter configuration.) The syntax of this file is defined in Section 5.1.

numRun enables multiple independent executions of the same optimization; `numRun` is used to compute the internal seed used by ParamILS (ParamILS is itself a randomized algorithm). In combination with `instance_seed_file`, using different values for `numRun` enables parallel optimizations of the same algorithm with different seeds and different instances sampled from the same benchmark set.

db For my own research, I use a MySQL database to store algorithm results, such that I can quickly repeat previous runs of ParamILS. However, for

most end users, this would be way too much overhead, so I do not support this component. Thus, any settings other than `-db 0` will result in an error.

ParamILS writes output about its progress to stdout and also writes to two files in the directory of the `instance_seed_file`, in the above case directory `QWH-single-instance/`. These two files have quite long filenames, containing information about the particular call of ParamILS. The file with `-log` in its name logs ParamILS behaviour, whereas the file with `-traj` only keeps track of its trajectory, giving the best configuration found at each time, the so-called *incumbent* configuration. Note that for FocusedILS, the quality of the incumbent solution does not improve monotonically because the number of runs it is based on varies. Solution quality starts at the worst possible value, `1000000000000000000`. It then typically quickly improves to a very low value because in the beginning each evaluation is only based on a single algorithm execution (leading to initial over-tuning); as more runs become available the performance estimates get more realistic. Detailed information about the incumbent configurations found is written to the log file (the trajectory file is merely a summary file to track progress).

In order to get reasonable performance estimates, ParamILS performs multiple runs for each configuration. These runs can differ in the input instance and the algorithm seed. In this first example, there is only a single instance, so the algorithm runs only differ in their random seeds. Conversely, for deterministic algorithms, the runs only differ in their input instances (the seed for deterministic algorithms should always be fixed to -1).

The file `QWH-single-instance/instance_seed_file_0.txt` contains the information which seeds and instances (only one in this example) should be used for the optimization. Different input instances and seeds will lead to different results, or the same final result being found faster or slower. If multiple CPUs are available for the optimization, one can, e.g., execute

```
ruby ../param_ils_1_0_run.rb -algo "ruby saps_wrapper.rb" -numRun 1
-instance_seed_file QWH-single-instance/instance_seed_file_1.txt -db
0
```

on another machine to perform a second, independent optimization – one execution may find good parameter settings much faster than the other, even if the two will tend to find the same one in the limit of an infinite number of runs. Different output files are used for different values of `numRun`.

4 Configurable parameters

There are a number of configurable parameters the user can set:

tunerTimeout The time after which the optimization is terminated.

maxEvals The number of algorithm executions after which the optimization is terminated.

run_obj A scalar quantifying how "good" a single algorithm execution is, such as its required runtime. Implemented examples for this include `runtime`, `runlength`, `approx` (approximation quality, i.e., 1-(optimal quality divided by found quality)), `speedup` (speedup over a reference runtime for this instance – note that for this option the reference needs to be defined in the `instance_seed_file` as covered in Section 5.1). Additional objectives for single algorithm executions can be defined by modifying function `single_run_objective` in file `algo_specifics.rb`.

overall_obj While `run_obj` defines the objective function for a single algorithm run, `overall_obj` defines how those single objectives are combined to reach a single scalar value to compare two parameter configurations. Implemented examples include `mean`, `median`, `q90` (the 90% quantile), `adj_mean` (a version of the mean accounting for unsuccesful runs: total runtime divided by number of succesful runs), `mean1000` (another version of the mean accounting for unsuccessful runs: (total runtime of succesful runs + 1000× runtime of unsuccesful runs) divided by number of runs – this effetively maximizes the number of successful runs, breaking ties by the runtime of successful runs; it is the criterion I use in most of my experiments), and `geomean` (geometric mean, primarily used in combination with `run_obj=speedup`. The empirical statistic of the cost distribution (across multiple instances and seeds) to be minimized, such as the mean (of the single run_objectives).

approach Use `basic` for BasicILS and `focused` for FocusedILS.

N For BasicILS, N is the number of runs to perform to evaluate each parameter configuration. For FocusedILS, it is the *maximal* number of runs to perform to evaluate a parameter configuration.

cutoff_time The time after which a single algorithm execution will be terminated unsuccesfully. This is an important parameter: if choosen too high, lots of time will be wasted with unsuccessful runs. If chosen too low the optimization is biased to perform well on easy instances only.

cutoff_length The runlength after which a single algorithm execution will be terminated unsuccesfully. This length can, e.g. be defined in flips for an SLS algorithm or decisions for a tree search.

R ParamILS internal parameter: The number of random configurations evaluated at the start of ParamILS.

pertubation_strength_basic ParamILS internal parameter: The pertubation strength of ParamILS.

pertubation_strength_scaling ParamILS internal parameter: If set to 1, the pertubation strength is the maximum of the basic pertubation strength and the basic pertubation strength multiplied by the number of parameters divided by 10.

p_restart ParamILS internal parameter: The probability of restarting from a random configuration at the end of an iteration of ParamILS.

maxbonus ParamILS internal parameter: The maximal number of bonus runs in FocusedILS.

Each of these parametesr is set as `-param_name param_value`. For example, `ruby ../param_ils_1_0_run.rb -algo "ruby saps_wrapper.rb" -numRun 0 -instance_seed_file QWH-single-instance/instance_seed_file_0.txt -db 0 -approach basic -N 100 -run_obj runlength -overall_obj mean` operates on the same instances as above, but applies BasicILS(100) to find the parameter configuration with the lowest mean runlength.

5 Multiple instances and the instance_seed_file

Let's assume we want to optimize SAPS for satisfiable morphed graph colouring instances based on small world graphs. 10 such instances are available in directory `example_saps/SWGCP-satisfiable-instances`. We will use 5 instances for training and 5 for test of the learned parameter settings (note that 5 instances for training is much too low in practice, I just didn't want to include hundreds of instances in the distribution).

5.1 The instance_seed_file

The `instance_seed_file` defines which samples of the cost distribution to use for the optimization. One sample is defined in each line of the file, in the form `seed filename_of_instance optimal_solution_quality`. Only some objective functions use `optimal_solution_quality`, but it has to be defined has to be defined, at least to a dummy numerical value; for SAT, just use 0 (the optimal solution has 0 clauses unsatisfied). Each line can also contain a fourth value, namely a reference runtime (or runlength, or whatever) for the instance. This is very useful if the objective is to beat a competing algorithm, or a previous version of the same algorithm (In my opinion, this objective is used way too much in computer science research, but since the demand is there I provide the option). The single run objective `speedup` is currently the only objective function using this reference value, but as mentioned above, you are welcome to implemente additional objective functions.

5.2 Creating instance_seed_files

In order to ease the generation of `instance_seed_files`, I provide a simple script, `create-instance-seeds-files.rb`. Given a file listing names of instances (complete absolute or relative path), this performs a simple stratified sampling of instances, i.e. it never picks an instance twice before all the instances have been picked at least once. (Stratified sampling is a lot like standard uniform sampling but leads to lower variance.) If you don't want stratified sampling,

call it with `-stratified 0`; if you have a deterministic algorithm, call it with `-deterministic 1` and it will use a constant seed -1.

Let's build 10 training `instance_seed_files` for parallel optimization on multiple machines by running the following command from directory `example_saps`:
`ruby ../create-instance-seeds-files.rb -instanceFile SWGCP-5-train-instances.txt -numRepetitions 10`.
This creates a new subdirectory `SWGCP-5-train-instances-strat` containing 10 instance lists. Let's also create one `instance_seed_file` for testing:
`ruby ../create-instance-seeds-files.rb -instanceFile SWGCP-5-test-instances.txt -numRepetitions 1`.

5.3 A complete optimization run

Now let's do a short complete optimization run for SAPS on SWGCP. Call
```
ruby ../param_ils_1_0_run.rb -algo "ruby saps_wrapper.rb" -numRun 0
-instance_seed_file SWGCP-5-train-instances-strat/instance_seed_file_0.txt
-test-instance_seed_file SWGCP-5-test-instances-strat/instance_seed_file_0.txt
-db 0 -overall_obj median -cutoff_time 1 -tunerTimeout 100
```

This will run an optimization run for 100 seconds, cutting off each single algorithm run if unsuccesful in the first second (this is *very* aggressive since I want this example to run quickly ;-). At the end of the 100 seconds, validation runs are performed on the test set, completing the execution of ParamILS.

6 Running ParamILS for your own code

In order to employ ParamILS to optimize your own code, you need to provide instance lists in the same format as in the above example, provide a file listing your algorithm's parameters in a predefined format, and match the required input/output format. These two latter points are covered in this section.

6.1 Algorithm parameter file

I recommend you create a separate subdirectory for each algorithm you want to optimize. The parameters of your algorithm need to be defined in a file called `params.txt`.

This file consists of three parts: basic parameters, specification of conditional parameters, and forbidden parameter combinations, where each of the the latter two can be empty. Examples for such files can be found in directory `example_params/`.

In the first part, each line lists one parameter, in curly parentheses the possible values considered, and in square parentheses the default value.

In the second part, conditional parameters that are only active when some higher-level parameters take on certain values are specified as follows: condi-

tional_param — higher_level_param in values for higher_level_param that allow conditional_param to be active, separated by commas.

In the third part, forbidden combinations of parameters may be listed. These forbidden combinations are listed one per line, in curly parentheses in the form: {`param1=value1,param2=value2,...`}.

6.2 Algorithm executable / wrapper

The algorithm executable must comply with the following input/output criteria. It is called as:
`algo_exectuable instance_name run_obj cutoff_time cutoff_length seed params`
and outputs (possibly amongst others) a line
`Result for ParamILS: solved, runtime, runlength, best_sol, seed`
containing information about the algorithm execution. `solved` can be either SAT, UNSAT, or TIMEOUT; `best_sol` is the best solution found (for SAT, the lowest number of unsatisfied clauses), and the other fields should be self-explanatory. It is important to output a value for each of these fields, even if they don't make sense for your algorithm (just output e.g. -1 in that case). If you don't want to change your algorithm output, you can write a simple wrapper around it; I did this for the SAPS example above. In fact, a wrapper could reuse most parts of that SAPS wrapper. If you want to write a simple ruby wrapper around your algorithm executable, have a look at `saps_wrapper.rb` in directory `example_saps`.