ParamILS: Iterated Local Search in Parameter Configuration Space

Jiří Kubalík Department of Cybernetics, CTU Prague

Substantial part of this material is based on the paper
Frank Hutter, Holger H. Hoos, Kevin Leyton-Brown, and Thomas Stützle: ParamILS: An Automatic Algorithm Configuration Framework,

Journal of Artificial Intelligence Research (JAIR),

volume 36, pp. 267-306, October 2009.

See http://www.cs.ubc.ca/ hutter/papers/Hutter09PhD.pdf



http://cw.felk.cvut.cz/doku.php/courses/a0m33eoa/start

Algorithm Configuration (Parameter Tuning) Problem

The behaviour of the algorithms can vary significantly between multiple runs on different instances or when randomized algorithms are run repeatedly with fixed parameters on a single problem instance.

Therefore, the cost of a candidate solution θ is defined as

$$c(\theta) = m(O_{\theta})$$

the statistical population parameter m of the cost distribution $O_{\theta}(\mathcal{A}, \theta, \mathcal{D})$, over instances drawn from distribution of instances, \mathcal{D} , and multiple independent runs.

An optimal solution, θ^* , minimizes $c(\theta)$:

$$\theta^* \in \arg\min_{\theta \in \Theta} c(\theta).$$

For example, we might aim to minimize mean runtime or median solution cost.

The $O_{\theta}(\mathcal{A}, \theta, \mathcal{D})$ is typically unknown, so we can only acquire approximations of their statistics, $c(\theta)$, based on a limited number of samples (i.e. the cost of single executions of $\mathcal{A}(\theta)$) – let's denote an approximation of $c(\theta)$ based on N samples by $\widehat{c}_N(\theta)$.

- For deterministic algorithms, the algorithm \mathcal{A} is run on $N \leq M$ instances (M is the size of the finite training set of instances).
- For randomized, algorithms, we can run multiple runs with different seeds if M < N.

ParamILS: Iterated Local Search in Parameter Configuration Space

Employs **Iterated Local Search** that builds a chain of local optima by iterating through a main loop consisting of:

- 1. a solution perturbation to escape from local optima,
- 2. a subsidiary local search procedure and
- 3. an acceptance criterion to decide whether to keep or reject a newly obtained candidate solution.

$ParamILS(\theta_0, r, p_{restart}, s)$

- 1. uses a combination of default and random settings for initialization,
 - θ_0 is the initial parameter configuration, and
 - r is the number of randomly chosen configurations for initialization,
- 2. uses a one-exchange neighborhood (one parameter is modified in each search step),
- 3. employs iterative first improvement as a subsidiary local search procedure,
- 4. uses a fixed number, s, of random moves for **perturbation**,
- 5. always accepts better or equally-good parameter configurations,
- 6. **re-initializes** the search at random with probability $p_{restart}$.

ParamILS(N): Algorithm

```
Input: Initial configuration \theta_0 \in \Theta, algorithm parameters r, p_{restart}, and s.
    Output: Best parameter configuration \theta found.
         i = 1, ..., r do

\theta \leftarrow \text{random } \theta \in \Theta;

if \text{better}(\theta, \theta_0) then \theta_0 \leftarrow \theta;
 1 for i = 1, ..., r do
                                                                        Initialization
 4 \theta_{ils} \leftarrow IterativeFirstImprovement(\theta_0); First iteration of the ILS procedure
 5 while not TerminationCriterion() do
          \theta \leftarrow \theta_{ils};
          // ===== Perturbation
          for i = 1, ..., s do \theta \leftarrow \text{random } \theta' \in Nbh(\theta);
 7
                                                                                        Main body
         // ===== Basic local search
                                                                                        of the ILS procedure
          \theta \leftarrow IterativeFirstImprovement(\theta);
          // ===== AcceptanceCriterion
          if better(\theta, \theta_{ils}) then \theta_{ils} \leftarrow \theta;
          with probability p_{restart} do \theta_{ils} \leftarrow \text{random } \theta \in \Theta;
11 return overall best \theta_{inc} found;
12 Procedure IterativeFirstImprovement (\theta)
13 repeat
14
          foreach \theta'' \in Nbh(\theta') in randomized order do
                if better(\theta'', \theta') then \theta \leftarrow \theta''; break;
16
                                                                                  How to define
                                                                                   better(\theta'', \theta')?
17 until \theta' = \theta:
18 return \theta;
```

BasicILS(N): Procedure $better_N(\theta_1, \theta_2)$

Basic variant, BasicILS(N), uses procedure $better_N(\theta_1, \theta_2)$ that compares two cost approximations $\widehat{c}_N(\theta_1)$ and $\widehat{c}_N(\theta_2)$ based on exactly N samples from the respective cost distributions $O_{\theta}(\mathcal{A}, \theta_1, \mathcal{D})$ and $O_{\theta}(\mathcal{A}, \theta_2, \mathcal{D})$ – the same N instances are used for all configurations θ_i .

Procedure $better_N(\theta_1, \theta_2)$ simply compares estimates $\widehat{c}_N(\theta_1)$ and $\widehat{c}_N(\theta_2)$ based on the same N instances using the same random seeds.

• It updates the best-so-far solution, θ_{inc} .

```
Input : Parameter configuration \theta_1, parameter configuration \theta_2
```

Output : True if θ_1 does better than or equal to θ_2 on the first N instances; false otherwise

Side Effect: Adds runs to the global caches of performed algorithm runs \mathbf{R}_{θ_1} and \mathbf{R}_{θ_2} ; potentially

updates the incumbent θ_{inc}

```
1 \hat{c}_N(\boldsymbol{\theta}_2) \leftarrow objective(\boldsymbol{\theta}_2, N)
```

- 2 $\hat{c}_N(\boldsymbol{\theta}_1) \leftarrow objective(\boldsymbol{\theta}_1, N)$
- 3 return $\hat{c}_N(\boldsymbol{\theta}_1) \leq \hat{c}_N(\boldsymbol{\theta}_2)$

FocusedILS

The question is how to choose the optimal number of training instances, N?

- ullet Using too small N leads to good training performance, but poor generalization to previously unseen test benchmarks.
- On the other hand, we cannot evaluate every parameter configuration on an enormous training set if we did, search progress would be unreasonably slow.

FocusedILS is a variant of ParamILS that adaptively varies the number of training samples considered from one parameter configuration to another in order to focus samples on promising configurations.

• $N(\theta)$ denotes the number of runs available to estimate the cost statistic $c(\theta)$.

FocusedILS

The question is how to choose the optimal number of training instances, N?

- ullet Using too small N leads to good training performance, but poor generalization to previously unseen test benchmarks.
- On the other hand, we cannot evaluate every parameter configuration on an enormous training set if we did, search progress would be unreasonably slow.

FocusedILS is a variant of ParamlLS that adaptively varies the number of training samples considered from one parameter configuration to another in order to focus samples on promising configurations.

• $N(\theta)$ denotes the number of runs available to estimate the cost statistic $c(\theta)$.

The question is how to compare two parameter configurations θ_1 and θ_2 for which $N(\theta_1) \leq N(\theta_2)$?

• What if we computed the empirical statistics based on the available number of runs for each configuration?

Can lead to systematic bias if, for example, the first instances are easier than the average ones.

FocusedILS: Procedure $better_{Foc}(\theta_1, \theta_2)$

Domination: Configuration θ_1 dominates θ_2 when at least as many runs have been conducted on θ_1 as on θ_2 , and the performance of $\mathcal{A}(\theta)$ on the first $N(\theta_2)$ runs is at least as good as that of $\mathcal{A}(\theta_2)$ on all of its runs.

$$\theta_1$$
 dominates θ_2 if and only if $N(\theta_1) \geq N(\theta_2)$ and $\widehat{c}_{N(\theta_2)}(\theta_1) \leq \widehat{c}_{N(\theta_2)}(\theta_2)$.

FocusedILS – procedure $better_{Foc}(\theta_1, \theta_2)$ implements a comparison strategy based on the domination

- 1. first it acquires one additional run for the configuration i having smaller $N(\theta_i)$, or one run for both configurations if $N(\theta_1) = N(\theta_2)$;
- 2. then, it continues performing runs in this way until one configuration dominates the other. It returns true if θ_1 dominates θ_2 , and false otherwise.

It keeps track of the total number, B, of configurations evaluated since the last improving step.

- Whenever $better_{Foc}(\theta_1, \theta_2)$ returns true, B extra (bonus) runs are performed for θ_1 and B is reset to 0.
- This way it is ensured that many runs are performed with good configurations \Longrightarrow the error made in every comparison of two configurations θ_1 and θ_2 decreases on expectation.

FocusedILS: Procedure $better_{Foc}(\theta_1, \theta_2)$

```
: Parameter configuration \theta_1, parameter configuration \theta_2
     Input
                       : True if \theta_1 dominates \theta_2, false otherwise
     Output
     Side Effect: Adds runs to the global caches of performed algorithm runs \mathbf{R}_{\theta_1} and \mathbf{R}_{\theta_2}; updates the
                          global counter B of bonus runs, and potentially the incumbent \theta_{inc}
 1 B \leftarrow B + 1
 2 if N(\theta_1) \leq N(\theta_2) then
 3 | \theta_{min} \leftarrow \theta_1; \theta_{max} \leftarrow \theta_2
4 | if N(\theta_1) = N(\theta_2) then B \leftarrow B + 1
 5 else \theta_{min} \leftarrow \theta_2; \theta_{max} \leftarrow \theta_1
 6 repeat
 7 i \leftarrow N(\boldsymbol{\theta_{min}}) + 1
 8 \hat{c}_i(\boldsymbol{\theta}_{max}) \leftarrow objective(\boldsymbol{\theta}_{max}, i) \text{ // If } N(\boldsymbol{\theta}_{min}) = N(\boldsymbol{\theta}_{max}), \text{ adds a new run to } \mathbf{R}_{\boldsymbol{\theta}_{max}}.
 9 \hat{c}_i(\boldsymbol{\theta}_{min}) \leftarrow objective(\boldsymbol{\theta}_{min}, i) // Adds a new run to \mathbf{R}_{\boldsymbol{\theta}_{min}}.
10 until dominates(\theta_1, \theta_2) or dominates(\theta_2, \theta_1)
11 if dominates(\theta_1, \theta_2) then
           // ===== Perform B bonus runs.
            \hat{c}_{N(\theta_1)+B}(\theta_1) \leftarrow objective(\theta_1, N(\theta_1)+B) // Adds B new runs to \mathbf{R}_{\theta_1}.
            B \leftarrow 0
13
           return true
15 else return false
16 Procedure dominates(\theta_1, \theta_2)
17 if N(\theta_1) < N(\theta_2) then return false
18 return objective (\theta_1, N(\theta_2)) \leq objective(\theta_2, N(\theta_2))
```

Adaptive Capping of Algorithm Runs

Often, the computational resources are wasted with evaluating a parameter configuration that is much worse than other, previously-seen configurations.

Ex.: Let's assume a case where parameter configuration θ_1 takes a total of 10 seconds to solve N=100 instances (i.e. it has a mean runtime of 0.1 seconds per instance), and another parameter configuration θ_2 takes 100 seconds to solve the first of these instances.

Clearly, when comparing the mean runtimes of θ_1 and θ_2 based on this set of instances, it is not necessary to run θ_2 on remaining 99 instances. Instead, we can terminate the first run of θ_2 after $10 + \epsilon$ seconds, which is a lower bound on θ_2 's mean runtime of $0.1 + \epsilon/100$. This lower bound exceeds the mean runtime of θ_1 , so we can already be sure that θ_2 cannot do better than θ_1 .

Question is how to determine the cutoff time for each run of the target algorithm, A, in an automated way?

Adaptive Capping of Algorithm Runs

Often, the computational resources are wasted with evaluating a parameter configuration that is much worse than other, previously-seen configurations.

Ex.: Let's assume a case where parameter configuration θ_1 takes a total of 10 seconds to solve N=100 instances (i.e. it has a mean runtime of 0.1 seconds per instance), and another parameter configuration θ_2 takes 100 seconds to solve the first of these instances.

Clearly, when comparing the mean runtimes of θ_1 and θ_2 based on this set of instances, it is not necessary to run θ_2 on remaining 99 instances. Instead, we can terminate the first run of θ_2 after $10 + \epsilon$ seconds, which is a lower bound on θ_2 's mean runtime of $0.1 + \epsilon/100$. This lower bound exceeds the mean runtime of θ_1 , so we can already be sure that θ_2 cannot do better than θ_1 .

Question is how to determine the cutoff time for each run of the target algorithm, A, in an automated way?

Adaptive capping is based on the idea of avoiding unnecessary runs of the algorithm \mathcal{A} by developing bounds on the performance measure to be optimized.

- Trajectory-preserving capping provably does not change BasicILS's search trajectory, but can lead to large computational savings.
- Aggressive capping potentially yielding even better performance.

BasicILS: Trajectory-Preserving Capping

Trajectory-Preserving Capping (for the case of minimizing the mean of non-negative cost function) – implements bounded evaluation of a parameter configuration θ , procedure $objective(\theta, N, bound)$, based on N runs so that

- it sequentially performs runs for θ and after each run computes a lower bound on $\widehat{c}_N(\theta)$ based on $i \leq N$ runs performed so far (i.e. the lower bound $\widehat{c}_N(\theta)$ is calculated as the sum of runtimes of each of the i runs, divided by N);
- once the lower bound exceeds the bound passed as an argument, the remaining runs for θ_2 are skipped and a large constant, worstPossibleObjective, is returned.

Procedure $better_N(\theta_1,\theta_2)$ is modified as follows

- 1 $bound \leftarrow \infty$
- 2 $\widehat{c}_N(\theta_2) \leftarrow objective(\theta_2, N, bound)$
- 3 bound $\leftarrow \widehat{c}_N(\theta_2)$
- 4 $\widehat{c}_N(\theta_1) \leftarrow objective(\theta_1, N, bound)$
- 5 return $\widehat{c}_N(\theta_1) \leq \widehat{c}_N(\theta_2)$

Trajectory-preserving capping typically requires **much less runtime** than the standard BasicILS.

Trajectory-Preserving Capping: Summary

The *trajectory-preserving capping* computes the upper bound on the cumulative runtime **from** the best configuration encountered in the current ILS iteration.

New ILS iteration starts with a **perturbation of the best-so-far configuration** θ .

Frequently, the new parameter configuration θ is of poor quality, thus **the capping criterion does not apply as quickly as it could** if the comparison was performed against the overall *incumbent*.

ParamILS(N): Algorithm

```
Input: Initial configuration \theta_0 \in \Theta, algorithm parameters r, p_{restart}, and s.
    Output: Best parameter configuration \theta found.
         i = 1, ..., r do

\theta \leftarrow \text{random } \theta \in \Theta;

if \text{better}(\theta, \theta_0) then \theta_0 \leftarrow \theta;
 1 for i = 1, ..., r do
                                                                        Initialization
 4 \theta_{ils} \leftarrow IterativeFirstImprovement(\theta_0); First iteration of the ILS procedure
 5 while not TerminationCriterion() do
          \theta \leftarrow \theta_{ils};
          // ===== Perturbation
        for i = 1, ..., s do \theta \leftarrow \text{random } \theta' \in Nbh(\theta);
 7
                                                                                        Main body
         // ===== Basic local search
                                                                                        of the ILS procedure
          \theta \leftarrow IterativeFirstImprovement(\theta);
          // ===== AcceptanceCriterion
          if better(\theta, \theta_{ils}) then \theta_{ils} \leftarrow \theta;
          with probability p_{restart} do \theta_{ils} \leftarrow \text{random } \theta \in \Theta;
11 return overall best \theta_{inc} found;
12 Procedure IterativeFirstImprovement (\theta)
13 repeat
          \theta' \leftarrow \theta:
14
          foreach \theta'' \in Nbh(\theta') in randomized order do
                if better(\theta'', \theta') then \theta \leftarrow \theta''; break;
16
                                                                                   How to define
                                                                                    better(\theta'', \theta')?
17 until \theta' = \theta:
18 return \theta;
```

ParamILS

BasicILS: Aggressive Capping

Aggressive Capping – **bounds the evaluation of any configuration** by the performance of the overall best-so-far (*incumbent*) configuration, θ_{inc} , multiplied by a factor **bound multiplier**, bm.

- ullet Different values of bm may imply different search trajectories:
 - $-bm = \infty$ reduces to trajectory-preserving but **no savings strategy**.
 - -bm=1 is a **very aggressive strategy**, since once we know the evaluated configuration θ is worse than the θ_{inc} , its evaluation is terminated.

Recommended setting is bm = 2.

• When two configurations θ_1 and θ_2 are compared and the evaluations of both are terminated preemptively, the configuration having solved more instances within the allowed time is considered the better one.

Ties are broken to favor moving to a new parameter configuration.

FocusedILS: Adaptive Capping

Focused/LS varies the number of runs used to compare two parameter configurations.

The number of runs can differ from one comparison to the next.

How can the adaptive capping be used here?

FocusedILS: Adaptive Capping

Focused/LS varies the number of runs used to compare two parameter configurations.

The number of runs can differ from one comparison to the next.

How can the adaptive capping be used here?

lacktriangle By using **separate bounds** for every number of runs, N.

.

ParamILS: Final Remarks

Suitable for configuration problems with many parameters and huge configuration spaces.

Successful applications:

- **SPEAR**, a complete SAT solver
 - 26 parameters,
 - $-8.34\cdot 10^{17}$ possible configurations,
 - FocusedILS produced configurations that solved test problems about 100 times faster than previous state-of-the-art solvers.
- CPLEX, a prominent solver for mixed integer programming problems with carefully chosen default parameter settings
 - 76 parameters,
 - $-1.9 \cdot 10^{47}$ possible configurations,
 - FocusedILS obtained substantial improvements in terms of both the time required to find optimal solutions (speedup factors from 1.98 to 52.3) and minimizing the optimality gap with factors from 1.26 to 8.65.

ParamILS

Recommended Material

Frank Hutter, Holger H. Hoos, Kevin Leyton-Brown, and Thomas Stützle: ParamILS: An Automatic Algorithm Configuration Framework. In *Journal of Artificial Intelligence Research* (JAIR), volume 36, pp. 267-306, October 2009.

Other papers and SW available at http://www.cs.ubc.ca/labs/beta/Projects/ParamILS/

ParamILS