Stanford MS&E 236 / CS 225: Lecture 13
Algorithm configuration

Ellen Vitercik[*]
`vitercik@stanford.edu`

May 23, 2024

Today, we will discuss *automated algorithm configuration*, a broadly applicable way of using ML to optimize the parameters of any parameterized algorithm, such as an integer programming (IP) solver. Integer programming solvers like CPLEX and Gurobi each come with over one hundred tunable parameters. In automated algorithm configuration, our goal is to use a data-driven approach to find optimized, application-specific parameter settings.

This lecture will overview a seminal approach to automated algorithm configuration by Hutter et al. [3]. This will give us a glimpse into some of the historical origins of this field, though there are related papers from several years earlier as well [e.g., 1, 2, 6, 7].

1 Setup

Our goal will be to optimize the parameters of an arbitrary algorithm with k tunable parameters. The i^{th} parameter setting is from a set Θ_i. We will assume, for now, that $|\Theta_i|$ is finite (for example, by discretizing continuous parameters). The entire parameter space is then $\Theta \subseteq \Theta_1 \times \cdots \times \Theta_k$. Hutter et al. [3] report that for IP solvers, $|\Theta|$ can be as large as 10^{37} (for context, the number of stars in the universe is 10^{24})!

Our goal is to find algorithm parameter settings that lead to particularly strong algorithmic performance (e.g., low runtime) on problems from the particular application domain at hand. Thus, we need a way of modeling an application domain. We will use Π to denote a set of problem instances that the parameterized algorithm may take as input. For example, $\pi \in \Pi$ may be an integer program. Moreover, we will assume that there is an application-specific distribution $\mathcal{D}$ over Π, which may or may not be known. For example, $\mathcal{D}$ might be a distribution over the particular routing IPs that a Bay Area shipping company has to solve on a day-to-day basis. Alternatively, if Π is a known benchmark set, then $\mathcal{D}$ might be the uniform distribution over this set. We use $r_{\boldsymbol{\theta}}(\pi)$ to denote the runtime of the algorithm with parameters $\boldsymbol{\theta} \in \Theta$ on instance π.

2 Problem statement

Formally, our goal is to find a parameter setting $\boldsymbol{\theta}$ with low expected runtime $\mathbb{E}_{\pi\sim\mathcal{D}}[r_{\boldsymbol{\theta}}(\pi)]$. For example, if $\mathcal{D}$ represents the distribution of routing IPs that a shipping company has to

[*]These notes are course material and have not undergone formal peer review. Please feel free to send me any typos or comments.

solve on a day-to-day basis, then $\mathbb{E}_{\pi\sim\mathcal{D}}[r_{\boldsymbol{\theta}}(\pi)]$ is what we expect the runtime of our solver to be on IPs that we'll encounter in the future. Alternatively, if $\mathcal{D}$ is the uniform distribution over a benchmark set Π, then

$$\mathop{\mathbb{E}}_{\pi\sim\mathcal{D}}[r_{\boldsymbol{\theta}}(\pi)] = \frac{1}{|\Pi|}\sum_{\pi\in\Pi} r_{\boldsymbol{\theta}}(\pi).$$

There are several key challenges we face in achieving this goal. First, $\mathcal{D}$ may be unknown. Therefore, our approach will be to sample a "training set" $S = \{\pi_1, \dots, \pi_N\} \sim \mathcal{D}^N$ and find a parameter setting $\boldsymbol{\theta}$ with low empirical runtime

$$\hat{r}_{\boldsymbol{\theta}}(S) = \frac{1}{N}\sum_{i=1}^{N} r_{\boldsymbol{\theta}}(\pi_i).$$

You can think of this training set as a set of historical instances that have been encountered in the past (for example, all of the routing integer programs that the shipping company has had to solve over the past year). Ideally, this parameter setting will also have low expected runtime (if we are not overfitting), which is a proxy for future runtime when the algorithm is fielded in the wild. However, finding a parameter setting with low empirical runtime is easier said than done: we do not know the analytical form of $r_{\boldsymbol{\theta}}$ and it is typically a gnarly, discontinuous function of $\boldsymbol{\theta}$.

3 ParamILS (Parameter Iterated Local Search)

ParamILS is an algorithm proposed by Hutter et al. [3] for parameter optimization that is based on local search. It begins with an initial, default configuration $\boldsymbol{\theta}_{\text{ils}}$. It then performs local search in the configuration space. In particular, it changes the setting of one parameter at a time, thereby searching within the neighborhood $N(\boldsymbol{\theta})$ of the current parameter setting $\boldsymbol{\theta}$. Specifically, $N(\boldsymbol{\theta})$ denotes the set of configurations that differ from $\boldsymbol{\theta}$ in exactly one of its k coordinates. It keeps the changes that result in an empirical runtime improvement. After finding a local minimum in this fashion, it randomly changes some parameter settings to escape. Algorithm 1 is the main ParamILS routine, and Algorithm 2 is the local search

Algorithm 1 Vanilla ParamILS

```
Input: Initial configuration $\boldsymbol{\theta}_{\text{ils}}$
1: repeat
2:     $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_{\text{ils}}$
3:     for $s$ rounds do                                      ▷ Do random exploration
4:         Set $\boldsymbol{\theta}$ to be a uniformly random configuration in $N(\boldsymbol{\theta})$
5:     $\boldsymbol{\theta} \leftarrow$ IterativeFirstImprovement($\boldsymbol{\theta}$)
6:     if $\hat{r}_{\boldsymbol{\theta}_{\text{ils}}}(S) > \hat{r}_{\boldsymbol{\theta}}(S)$ on a set $S$ sampled from $\mathcal{D}$ then
7:         $\boldsymbol{\theta}_{\text{ils}} \leftarrow \boldsymbol{\theta}$
8:     With some small probability, restart: set $\boldsymbol{\theta}$ to be a uniformly random configuration in $\Theta$
Output: $\boldsymbol{\theta}$
```

subroutine that it depends on.

Algorithm 2 IterativeFirstImprovement

```
Input: Initial configuration $\boldsymbol{\theta}$
1: $\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}$
2: repeat
3:     for each $\boldsymbol{\theta}' \in N(\boldsymbol{\theta}^*)$ (in a random order) do
4:         if $\hat{r}_{\boldsymbol{\theta}^*}(S) > \hat{r}_{\boldsymbol{\theta}'}(S)$ on a set $S$ sampled from $\mathcal{D}$ then
5:             $\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}'$; break        ▷ $\boldsymbol{\theta}'$ has lower empirical runtime than $\boldsymbol{\theta}^*$
6: until $\boldsymbol{\theta}^* = \boldsymbol{\theta}'$
Output: $\boldsymbol{\theta}^*$                                  ▷ Found a local minimum
```

3.1 Adaptive capping

Hutter et al. [3] improve upon the basic ParamILS algorithm (Algorithm 1) in a number of ways, one of which is adaptively capping configuration evaluations. Solving integer programs is time-consuming, so checking if $\hat{r}_{\boldsymbol{\theta}'}(S) > \hat{r}_{\boldsymbol{\theta}}(S)$ in Step 4 of Algorithm 2 can have extremely high runtime. Hutter et al.'s adaptive capping strategy stems from the observation that if we have already evaluated $\hat{r}_{\boldsymbol{\theta}}(S)$ and, while evaluating $\hat{r}_{\boldsymbol{\theta}'}(S)$, we discover that $\boldsymbol{\theta}'$ is clearly worse than $\boldsymbol{\theta}$, then we might be able to give up evaluating $\hat{r}_{\boldsymbol{\theta}'}(S)$ early.

As an illustrative example, suppose we evaluate $\boldsymbol{\theta}$ and $\boldsymbol{\theta}'$ on one hundred instances $S = \{\pi_1, \dots, \pi_{100}\} \sim \mathcal{D}^{100}$. Suppose that the algorithm parameterized by $\boldsymbol{\theta}$ takes ten seconds to solve all one hundred instances:

$$\hat{r}_{\boldsymbol{\theta}}(S) = \frac{1}{100}\sum_{i=1}^{100} r_{\boldsymbol{\theta}}(\pi_i) = \frac{1}{100} \cdot 10 = 0.1.$$

Meanwhile, suppose that the algorithm parameterized by $\boldsymbol{\theta}'$ takes at least 11 seconds to solve the first instance. We can deduce that

$$\hat{r}_{\boldsymbol{\theta}'}(S) = \frac{1}{100}\sum_{i=1}^{100} r_{\boldsymbol{\theta}'}(\pi_i) \geq \frac{1}{100} \cdot (11 + \underbrace{0 + \cdots + 0}_{99 \text{ zeros}}) = 0.11.$$

Therefore, we can stop evaluating $\boldsymbol{\theta}'$ eleven seconds into the first run, but still be sure that $\hat{r}_{\boldsymbol{\theta}'}(S) > \hat{r}_{\boldsymbol{\theta}}(S)$. Hutter et al. [3] and follow-up research [4, 5, 8, 9] develop more advanced approaches to adaptive capping, building on this intuition.

References

[1] Carla Gomes and Bart Selman. Algorithm portfolios. *Artificial Intelligence*, 126:43–62, 2001.

[2] Eric Horvitz, Yongshao Ruan, Carla Gomez, Henry Kautz, Bart Selman, and Max Chickering. A Bayesian approach to tackling hard computational problems. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, 2001.

[3] Frank Hutter, Holger Hoos, Kevin Leyton-Brown, and Thomas Stützle. ParamILS: An automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36(1):267–306, 2009. ISSN 1076-9757.

[4] Robert Kleinberg, Kevin Leyton-Brown, and Brendan Lucier. Efficiency through procrastination: Approximately optimal algorithm configuration with runtime guarantees. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2017.

[5] Robert Kleinberg, Kevin Leyton-Brown, Brendan Lucier, and Devon Graham. Procrastinating with confidence: Near-optimal, anytime, adaptive algorithm configuration. *Conference on Neural Information Processing Systems (NeurIPS)*, 2019.

[6] Michail G Lagoudakis and Michael L Littman. Learning to select branching rules in the dpll procedure for satisfiability. *Electronic Notes in Discrete Mathematics*, 9:344–359, 2001.

[7] L Lobjois and M Lemaître. Branch and bound algorithm selection by performance prediction. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 353–358, San Jose, CA, 1998.

[8] Gellért Weisz, András György, and Csaba Szepesvári. LeapsAndBounds: A method for approximately optimal algorithm configuration. In *International Conference on Machine Learning (ICML)*, 2018.

[9] Gellért Weisz, Andrés György, and Csaba Szepesvári. CapsAndRuns: An improved method for approximately optimal algorithm configuration. *International Conference on Machine Learning (ICML)*, 2019.