1 CAPO - Pseudo Algorithm

Algorithm 1 CAPO: Cost-Aware Prompt Optimization

Require: Dataset $\mathcal{D} = \{(X_i, y_i)\}_{i=1}^{n}$, Meta-LLM $\Phi(x)$, Downstream LLM $\phi(x)$, Initial instructions $\Lambda = \{\lambda_1, \ldots, \lambda_p\}$, Population size p, Block size b, Number of iterations n, Number of crossovers per iteration c

for $\lambda \in \Lambda$ **do**
    $\text{num\_shots} \sim \text{U}(\text{lower}, \text{upper})$
    $\xi \leftarrow \text{sample}(X, \text{num\_shots})$ ▷ Sample few shot examples
    $\theta \leftarrow \phi(\lambda || \xi)$ ▷ Generate few shots with reasoning
    $\pi \leftarrow (\lambda, \theta)$
    $\Pi \leftarrow \Pi.\text{append}(\pi)$
end for
Divide dataset $\mathcal{D}$ into blocks $\mathcal{B} = \{B_1, ..., B_k\}$ where $|B_i| = b$
for $i = 1$ to n **do**
    $\Pi_{\text{off}} \leftarrow \text{cross\_over}(\Pi, c)$
    $\Pi_{\text{off}} \leftarrow \text{mutate}(\Pi_{\text{off}})$
    $\Pi \leftarrow \text{do\_racing}(\Pi \cup \Pi_{\text{off}}, k = p)$
end for
$\text{best\_prompt} \leftarrow \text{do\_racing}(\Pi, k = 1)$
return best_prompt

Is reasoning for few-shots always necessary? Maybe this is not important for simple tasks and costs a lot of tokens. (Idea: some few shot examples can be included without reasoning)

Algorithm 2 cross_over

Require: Population Π, Meta-LLM $\Phi(x)$, Cross-Over-Meta-Prompt λ_{C}, Number of crossovers c

$\Pi_{\text{off}} \leftarrow []$

for $j = 1$ to c **do**

  $P_1 \leftarrow \text{sample}(\Pi, 1)$ ▷ $P_1 = (\lambda_1, \theta_1)$

  $P_2 \leftarrow \text{sample}(\Pi, 1)$ ▷ $P_2 = (\lambda_2, \theta_2)$

  $\lambda_{\text{off}} \leftarrow \Phi(\lambda_{\text{C}} || \lambda_1 || \lambda_2)$ ▷ Let Meta-LLM cross over the prompts

  $\theta_{\text{off},j} \leftarrow \text{sample}(\theta_1 \cup \theta_2, \left\lfloor \frac{|\theta_1| + |\theta_2|}{2} \right\rfloor)$ ▷ Sample from all few-shot examples

  $\pi_{\text{off},j} \leftarrow (\lambda_{\text{off},j}, \theta_{\text{off},j})$

  $\Pi_{\text{off}} \leftarrow \Pi_{\text{off}}.\text{append}(\pi_{\text{off},j})$

end for

return Π_{off}

Currently we do a random parent selection. In EvoPrompt they do a roulette wheel selection based on the fitness scores. This would require us to find a way of already having scores here.

We have to clarify where the new few-shot examples are coming from.
Extra Split:
+ no data leakage (fair, comparable assessment of prompts)
- constrained pool of few-shot examples (how to get this pool?) - potentially smaller dev Split

From Train Split:
- data leakage (prompts that contain eval data point already as few-shot examples which are already confirmed as correct have advantages)
+ we can use the full train set

Algorithm 3 mutate

Require: Population of offsprings Π_{off}, Meta-LLM $\Phi(x)$, Mutation-Meta-Prompt λ_{M}, Dataset samples X

for $\pi_{\text{off}} \in \Pi_{\text{off}}$ **do**

    $\lambda_{\text{off}} \leftarrow \Phi(\lambda_{\text{M}} || \lambda_{\text{off}})$

    $\text{num\_shots} \sim \text{U}(\text{lower}, \text{upper})$

    $\text{num\_new\_shots} \sim \text{U}(\text{lower}, \text{num\_shots})$

    $\xi \leftarrow \text{sample}(X, \text{num\_new\_shots})$ ▷ Sample new few shot examples

    $\theta_{\text{new}} \leftarrow \phi(\lambda_{\text{off}} || \xi)$ ▷ Generate few shots with reasoning

    $\theta_{\text{old}} \leftarrow \text{sample}(\theta_{\text{off}}, \text{num\_shots} - \text{num\_new\_shots})$

    $\theta \leftarrow \theta_{\text{old}} \cup \theta_{\text{new}}$

    $\theta \leftarrow \text{shuffle}(\theta)$

    $\pi_{\text{off}} \leftarrow (\lambda_{\text{off}}, \theta)$

end for

return Π_{off}

Algorithm 4 do_racing

Require: Prompts Π, Top-k k, blocks $\mathcal{B}$, Downstream LLM $\phi(x)$, Max number of evaluated blocks max_n_blocks_eval

$i \leftarrow 0$
scores $\leftarrow [0] * \text{len}(\Pi)$
shuffle($\mathcal{B}$) ▷ Whether to shuffle is left as a HP
while len(Π) $> k \ \wedge \ i <$ max_n_blocks_eval **do**
    $i \leftarrow i + 1$
    scores $\leftarrow \frac{1}{i}\left(\text{evaluate}(\Pi, B_i) + (i-1) * \text{scores}\right)$ ▷ Already evaluated blocks are cached
    $\Pi \leftarrow \text{racing\_elimination}(\Pi, \text{scores}, \alpha, k)$
end while
if len(Π) $> k$ **then**
    $\Pi \leftarrow$ top_k(Π) ▷ Make sure to return only k prompts
end if
return Π

Algorithm 5 racing_elimination

Require: Survivors Π, scores S, confidence level α, top-k k

$c_\alpha \leftarrow \text{getCriticalValue}(\alpha)$
for $\pi_i \in \Pi$ **do**
    n_sig_better $\leftarrow \sum_{j \neq i} \mathbf{1}_{[\text{getTestStatistic}(s_j, s_i) > c_\alpha]}$
    if n_sig_better $\geq k$ **then**
        $\Pi \leftarrow \Pi \setminus \{\pi_i\}$ ▷ Eliminate π_i
    end if
end for
return Π

Potential test statistics:

- Hoeffding races: $\varepsilon = \sqrt{\frac{B^2 \log(2/\delta)}{2n}}$
- z-score: $z = (s_i - s_j)/\sqrt{n}$
- t-score: $t = (s_i - s_j)/\sqrt{\frac{2s_i(1-s_i)}{n}}$
- Mann-Whitney-U-test
- Friedmann test (via ranks per block)

Further Ideas:
Multi-objective optimization (predictive performance and cost(e.g. prompt length)) - how to consider both objectives in the test statistic? -> only then we are "cost aware" (also reduces cost of optimization because of shorter prompt, fewer few-shot examples, ...)
Multi-fidelity can be e.g. done by first using a smaller model to determine a promising population and then continue with a larger model.