# ProBO: Versatile Bayesian Optimization Using Any Probabilistic Programming Language

W. Neiswanger et al. 2019

Paper review by Sean Parker

# Structure

- Background / Motivation
- Overview of ProBO
- Key contributions
- Experiments & Evaluation
- Review

# Bayesian Optimization (BO)

- Aim: Optimize the function $f(\mathrm{x})$
- Restricted to sampling the function at points x
- *Surrogate model* used to approximate objective function
- Uses *acquisition function* to sample areas of interest
    - MPI, EI, UCB, TS

# Probabilistic Programming Languages (PPLs)

- Often built upon existing languages
  - PyMC3/PyMC4 (Python)
  - Edward (Tensorflow)
  - Pyro (PyTorch)
- Each PPL uses a different inference strategy + posterior representations
  - MCMC, SMC
  - VI, EI

# Probabilistic Programming Languages (PPLs)

- Domain-specific languages
- Inference on probabilistic models
- Assumptions encoded over variables of the model
- Output: Probability Distribution

# PPL: Example

## Coin toss

- Calculate the bias of a coin:
    - Bernoulli distribution with latent variable $\theta$
        - $P(x_i = 1 \mid \theta) = \theta$ and $P(x_i = 0 \mid \theta) = 1 - \theta$
        - Infer $\theta$ based on previous results of coin toss - $P(\theta \mid x_1, x_2, \ldots, x_N)$

# Motivation

- Models built in PPL is optimised using BO techniques in that PPL

- BOPP - BO in specific PPL to estimate latent variables
- BOAT - Custom framework, uses exact inference & expected improvement

# Key contributions

- General abstraction for PPL programs
- ProBO system implementation*
- Evaluation of ProBO using BO models, implemented in various PPLs

# Probabilistic Programs Abstraction

- Three core PPL operations:
  - `inf(D)` - returns `post` (PPL dependent)
  - `post(s)` - returns a sample from the posterior distribution
  - `gen(x, z, s)` - returns sample from generative distribution

# ProBO Algorithm

**Algorithm 1** ProBO($\mathcal{D}_0$, $\texttt{inf}$, $\texttt{gen}$)

**for** $n = 1, \ldots, N$ **do**
  $\texttt{post} \leftarrow \texttt{inf}(\mathcal{D}_{n-1})$ ▷ Run inference algorithm to compute $\texttt{post}$
  $x_n \leftarrow \text{argmin}_{x \in \mathcal{X}}\, a(x, \texttt{post}, \texttt{gen})$ ▷ Optimize acquisition using $\texttt{post}$ and $\texttt{gen}$
  $y_n \sim s(x_n)$ ▷ Observe system at $x_n$
  $\mathcal{D}_n \leftarrow \mathcal{D}_{n-1} \cup (x_n, y_n)$ ▷ Add new observations to dataset
Return $\mathcal{D}_N$.

- Goal: Return $x^* = \text{argmin}_{x \in \mathcal{X}}\, \mathbb{E}_{y \sim s(x)}\left[f(y)\right]$
- Algorithm:
  - Invoke the PPLs inference procedure via `inf()`
  - Get new $x$ by optimising acquisition function
  - Observe system at $x$
  - Add new observation to dataset

# ProBO - Computation Cost

- `inf()` cost dependent on PPLs inference algorithm
  - e.g. MCMC algorithms - $O(n)$ per iteration
- `inf()` only executed **once per query**
- Acquisition optimisation executed 100s times per query
  - `post()` & `gen()` cheaply implemented - $O(1)$

# Acquisition function optimisation

- `post()` & `gen()` not analytically differentiable
- Authors explored zeroth-order optimisation of $a_{MF}$
  - `post()` & `gen()` called $M_f$ times
  - Any zeroth-order optimisation algorithm can be used

**Algorithm 6** $a_{\mathrm{MF}}\,(x, \texttt{post}, \texttt{gen})$

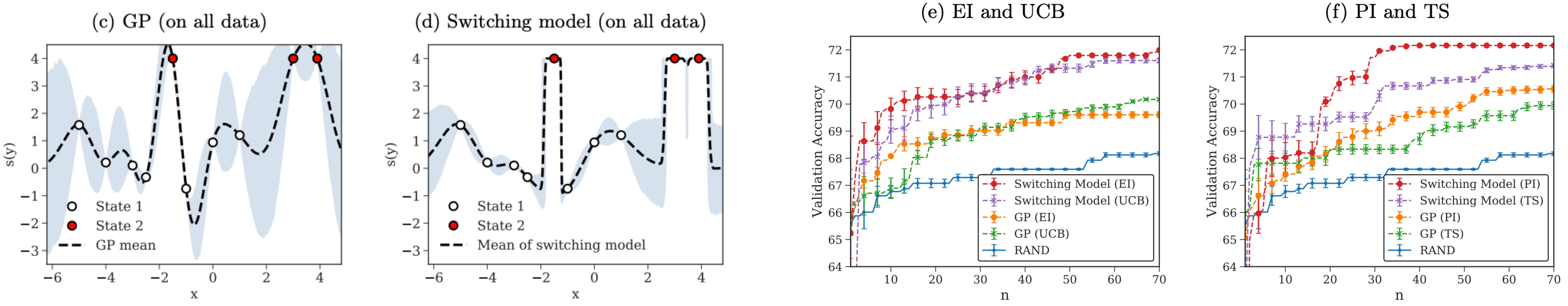
$a_{\min} \leftarrow$ Min value of $a$ seen so far
$\ell = -\infty,\ f = 1$
**while** $\ell \leq a_{\min}$ **do**
    $\ell \leftarrow$ LCB-bootstrap $(\texttt{post}, \texttt{gen}, M_f)$
    $f \leftarrow f + 1$
Return $a(x, \texttt{post}, \texttt{gen})$ using $M = M_f$

**Algorithm 7** LCB-bootstrap$(\texttt{post}, \texttt{gen}, M_f)$

$y_{1:M_f} \leftarrow$ Call **post** and **gen** $M_f$ times
**for** $j = 1, \ldots, B$ **do**
    $\tilde{y}_{1:M_f} \leftarrow \mathrm{Resample}(y_{1:M_f})$
    $a_j \leftarrow \lambda(\tilde{y}_{1:M_f})$ ▷ See text for details
Return LCB $(a_{1:B})$

# Evaluation

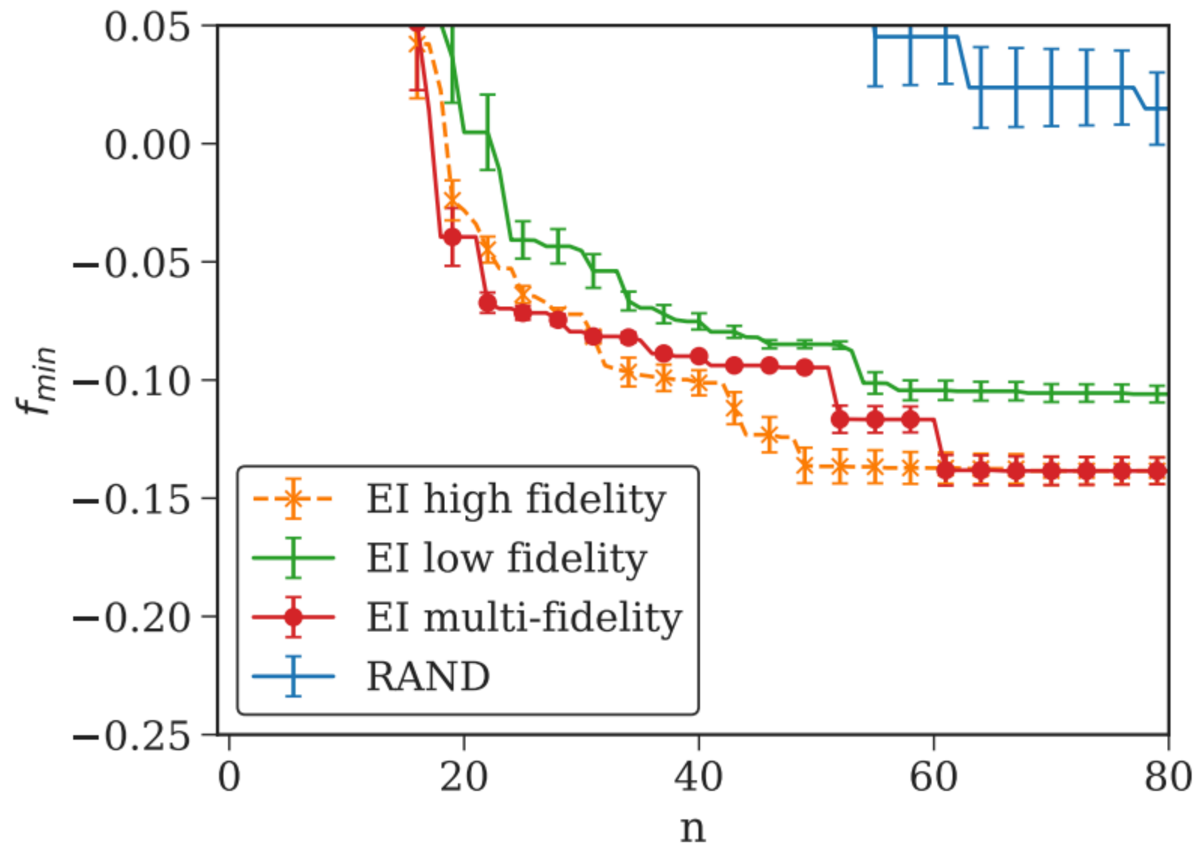
- Optimisation of MLP hyperparameters
  - “Switching model” is ProBO using a dynamic value of $M_f$



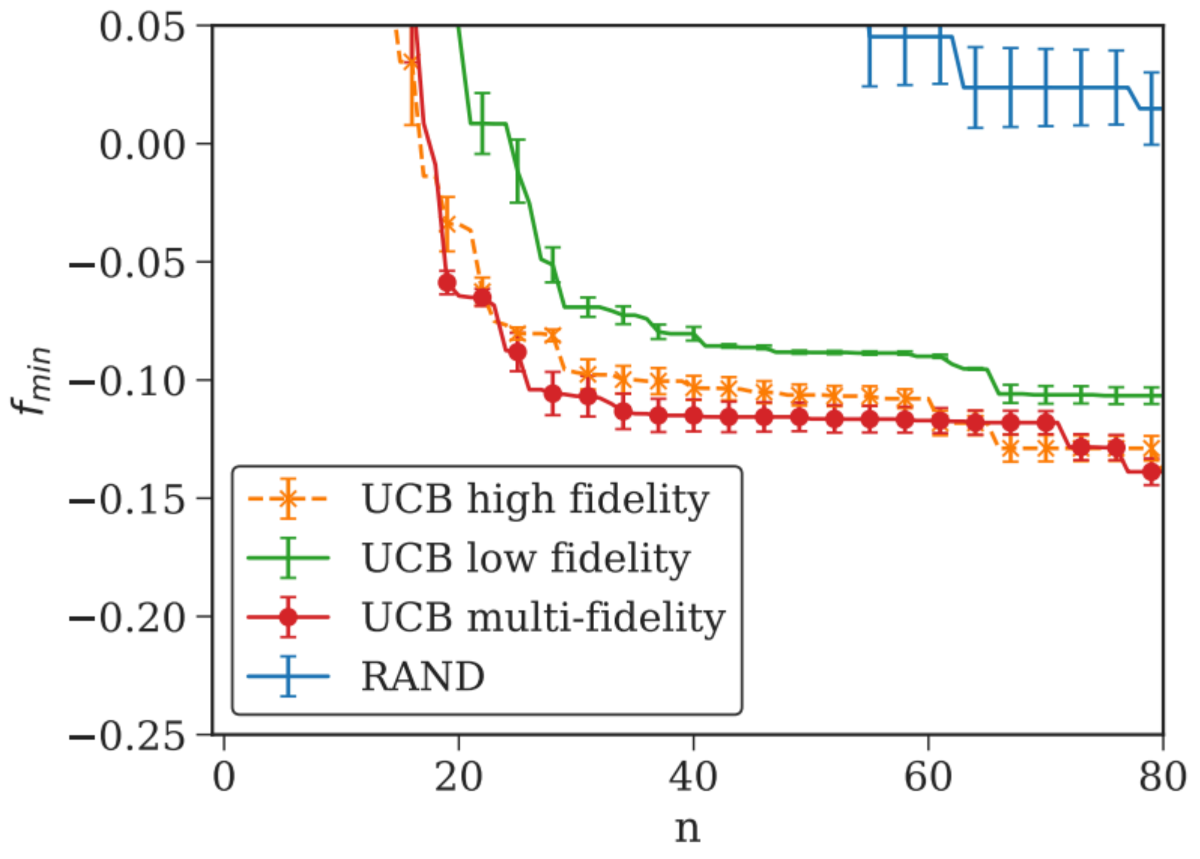
# Evaluation

- 3x better performance than high-fidelity in terms of calls to `gen()`
- High and multi fidelity have comparable performance
  - Converges to very similar value

(a) EI




(b) UCB




(c) Calls to **gen**

| PPL acquisition method $a(x)$ | Avg. number **gen**/$a(x)$ |
|---|---|
| EI high-fidelity | 1000 |
| EI multi-fidelity | 347.89 |
| EI low-fidelity | 10 |
| UCB high-fidelity | 1000 |
| UCB multi-fidelity | 324.65 |
| UCB low-fidelity | 10 |

# Review

- Difficult paper to understand
- Implementation of ProBO not provided
  - Questions remain of how ProBO is integrated into existing PPLs
- Good idea for providing uniform way of performing BO across PPLs