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 $f(x)$
- 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:
  - $\text{inf}(D)$ - returns $\text{post}$ (PPL dependent)
  - $\text{post}(s)$ - returns a sample from the posterior distribution
  - $\text{gen}(x, z, s)$ - returns sample from generative distribution
**ProBO Algorithm**

- **Goal**: Return $x^* = \arg \min_{x \in \mathcal{X}} \mathbb{E}_{y \sim s(x)} [f(y)]$

- **Algorithm**:
  - Invoke the PPLs inference procedure via $\text{inf}()$
  - Get new $x$ by optimising acquisition function
  - Observe system at $x$
  - Add new observation to dataset

---

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

1: for $n = 1, \ldots, N$ do  
2:  \hspace{1em} post $\leftarrow \text{inf}(\mathcal{D}_{n-1})$  
3:  \hspace{1em} $x_n \leftarrow \arg \min_{x \in \mathcal{X}} a(x, \text{post}, \text{gen})$  
4:  \hspace{1em} $y_n \sim s(x_n)$  
5:  \hspace{1em} $\mathcal{D}_n \leftarrow \mathcal{D}_{n-1} \cup (x_n, y_n)$  
6: Return $\mathcal{D}_N$.  

- Run inference algorithm to compute $\text{post}$  
- Optimize acquisition using $\text{post}$ and $\text{gen}$  
- Observe system at $x_n$  
- Add new observations to dataset
ProBO - Computation Cost

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

- \texttt{post()} & \texttt{gen()} not analytically differentiable

- Authors explored zeroth-order optimisation of $a_{MF}$

- \texttt{post()} & \texttt{gen()} called $M_f$ times

- Any zeroth-order optimisation algorithm can be used

\begin{algorithm}
\caption{$a_{MF}(x, \text{post}, \text{gen})$}
\begin{algorithmic}[1]
\State $a_{\text{min}} \leftarrow \text{Min value of } a \text{ seen so far}$
\State $\ell = -\infty$, $f = 1$
\While{$\ell \leq a_{\text{min}}$}
\State $\ell \leftarrow \text{LCB-bootstrap(} \text{post}, \text{gen}, M_f \text{)}$
\State $f \leftarrow f + 1$
\EndWhile
\State Return $a(x, \text{post}, \text{gen})$ using $M = M_f$
\end{algorithmic}
\end{algorithm}

\begin{algorithm}
\caption{LCB-bootstrap(\text{post}, \text{gen}, M_f)}
\begin{algorithmic}[1]
\State $y_{1:M_f} \leftarrow \text{Call } \text{post} \text{ and } \text{gen} \text{ } M_f \text{ times}$
\For{$j = 1, \ldots, B$}
\State $\tilde{y}_{1:M_f} \leftarrow \text{Resample}(y_{1:M_f})$
\State $a_j \leftarrow \lambda(\tilde{y}_{1:M_f})$ \Comment{See text for details}
\EndFor
\State Return LCB($a_{1:B}$)
\end{algorithmic}
\end{algorithm}
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 \texttt{gen()}.
- High and multi fidelity have comparable performance.
  - Converges to very similar value.

<table>
<thead>
<tr>
<th>PPL acquisition method ( a(x) )</th>
<th>Avg. number ( \text{gen}/a(x) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>EI high-fidelity</td>
<td>1000</td>
</tr>
<tr>
<td>EI multi-fidelity</td>
<td>347.89</td>
</tr>
<tr>
<td>EI low-fidelity</td>
<td>10</td>
</tr>
<tr>
<td>UCB high-fidelity</td>
<td>1000</td>
</tr>
<tr>
<td>UCB multi-fidelity</td>
<td>324.65</td>
</tr>
<tr>
<td>UCB low-fidelity</td>
<td>10</td>
</tr>
</tbody>
</table>
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