Combining Static and Dynamic Optimizations Using Closed-Form Solutions

Daniel Lundén[1] David Broman[1] Lawrence M. Murray[2]

[1]KTH Royal Institute of Technology, Sweden [2]Uppsala University, Sweden

Financially supported by the Swedish Foundation for Strategic Research.

Introduction

- **Probabilistic programming languages** increases *expressiveness* compared to probabilistic graphical models through:
  - Stochastic branching
  - Recursion
- **Importance sampling** methods (including **sequential Monte Carlo methods**) perform inference in probabilistic programming languages through *repeated execution* of programs.
- **Static optimizations** are program transformations performed prior to execution.
  Benefit: we can optimize programs offline with minimal overhead during execution.
- **Dynamic optimizations** are optimizations performed during execution.
  Benefit: we have access to information only available during execution.
- The local **closed-form solutions** we use here are:
  Given $p(x)$, $p(y|x)$ and y, $p(y)$ and $p(x|y)$ can be calculated in closed-form.

Objective of optimization

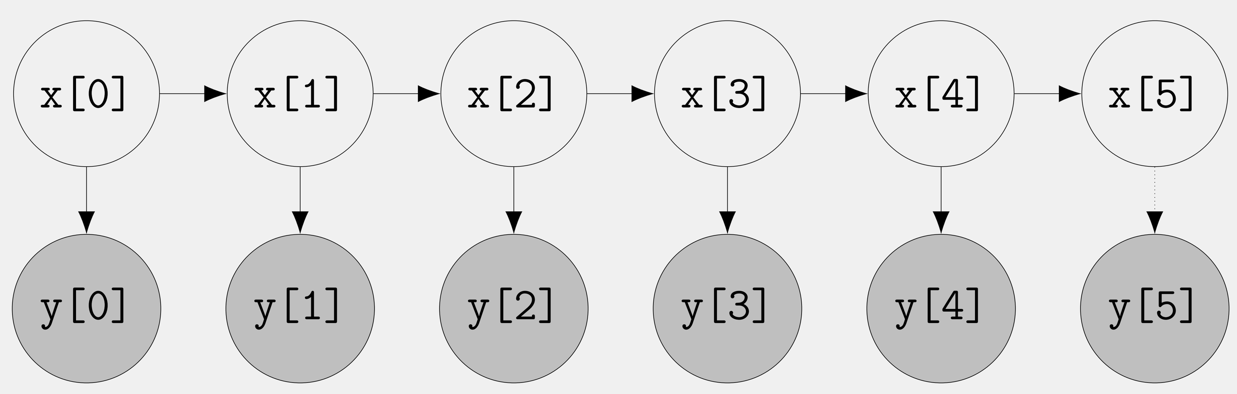
- **Maximizing quality of inference** (e.g. low variance of estimators).
- **Minimizing execution time of inference**.

The need for static optimization

A probabilistic program (Anglican)

```
(defquery static
  (let [data [0.4 0.9 -0.1 -1.3 0.2 2.1]
        x (sample (normal 0 1))]
    (observe (normal x 1) (first data))
    (loop [x x, data (rest data)]
      (if (seq data)
        (let [x (sample (normal x 1))]
          (observe (if (> (count data) 1)
                     (normal x 1) (normal 0 (+ 1 (abs x))))
                   (first data))
          (recur x (rest data))) x))))
```

Corresponding graphical model



Optimized program (partially solved analytically)

```
(defquery static-opt
  (let [data [2.1]
        x (sample (normal -0.157 (sqrt 1.617)))]
    (observe (normal 0 (+ 1 (abs x))) (first data)) x))
```

Conclusion

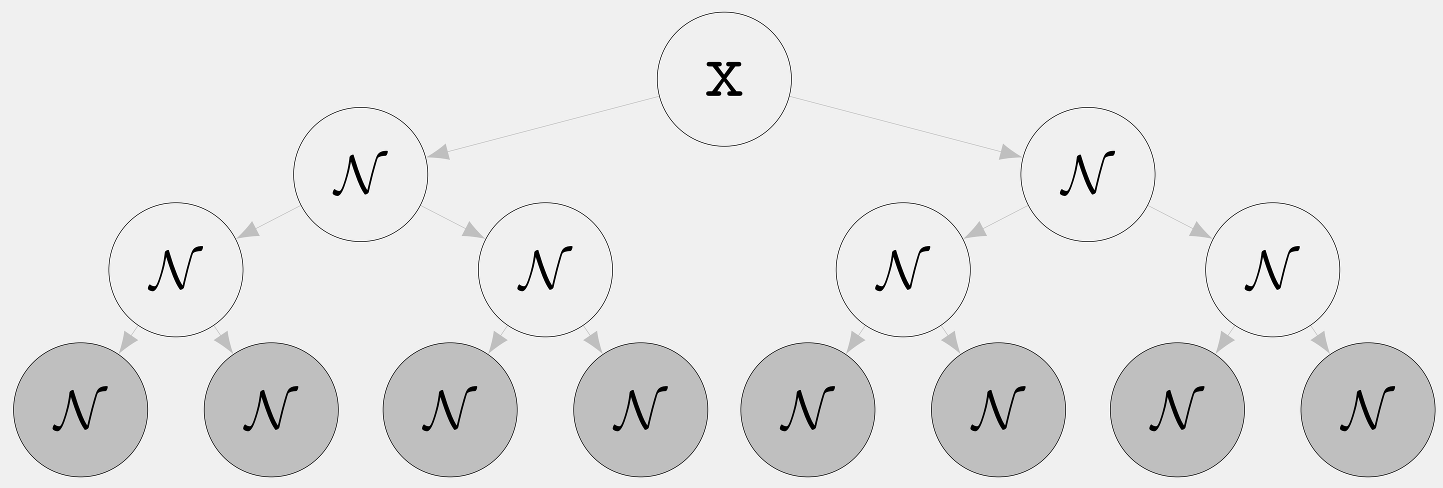
- When our program $\equiv$ a graphical model, we should always do these types of optimizations statically.

The need for dynamic optimization

A probabilistic program (Anglican)

```
(defquery dynamic
  (let [data [0 -1.1 2.4 1.2 -0.1 -1.4 -1.9]
        x (sample (normal 0 1))
        mix (fn [anc]
              (if (sample (flip 0.5))
                (normal anc 1)
                (normal 0 (+ 1 (abs anc)))))
        foo (fn foo [root depth]
              (let [left (mix root)
                    right (mix root)]
                (if (= depth 1)
                  [left right]
                  (concat (foo (sample left)
                               (- depth 1))
                          (foo (sample right)
                               (- depth 1))))))
        leaves (foo x 3)]
    (map (fn [dist obs] (observe dist obs))
         leaves data) x))
```

Corresponding graphical model

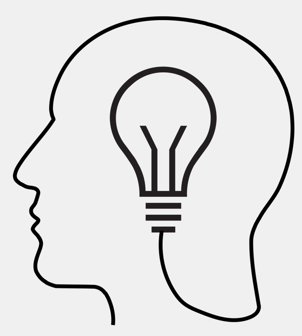


Conclusion

- Static optimization not a good idea due to stochastic branching
- **Delayed sampling** is a dynamic approach for these situations (see references below)

Challenges

- When do we use which approach?
- Can we get the best of both worlds by **combining** the two approaches **within** a single program?
- Can we define and find **optimal** combinations?



References

- Lawrence M. Murray, Daniel Lundén, Jan Kudlicka, David Broman, and Thomas B. Schön. 2017. *Delayed Sampling and Automatic Rao–Blackwellization of Probabilistic Programs*. To appear in proceedings of AISTATS 2018.
- Daniel Lundén. 2017. *Delayed sampling in the probabilistic programming language Anglican*. Master's thesis. KTH Royal Institute of Technology.

This project is financially supported by the Swedish Foundation for Strategic Research (ASSEMBLE RIT15-0012).