Learning to Superoptimize Programs

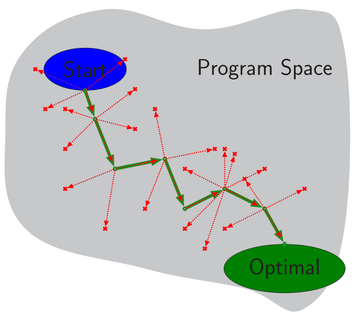
Rudy Bunel, Alban Desmaison, M. Pawan Kumar, Philip H.S. Torr, Pushmeet Kohli

Torr Vision Group - OVAL Group, Department of Engineering Science, University of Oxford, UK

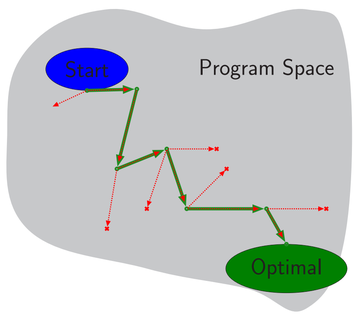
Microsoft Research

Introduction

Stochastic Search-based methods [1] can outperform traditional compilers for Program Optimization.



Uniform proposal distribution



Learned proposal distribution

Proposed transformation — Accepted transformation

We propose to **learn** how to perform the search.

Stochastic Superoptimization [1]

Programs are represented as a series of **instructions**. An instruction is composed of an **opcode** and possibly several **operands**.

```
1   pushq %rbp
2   movq %rsp, %rbp
3   movl %edi, -0x4(%rbp)
4   movl -0x4(%rbp), %edi
5   subl $0x1, %edi
6   movl %edi, -0x8(%rbp)
7   movl -0x4(%rbp), %edi
8   andl -0x8(%rbp), %edi
9   movl %edi, %eax
10  popq %rbp
11  retq
12  nop
13  nop
```

Optimization Starting point: output from `clang`

```
1   blsrl %edi, %eax
2   retq
```

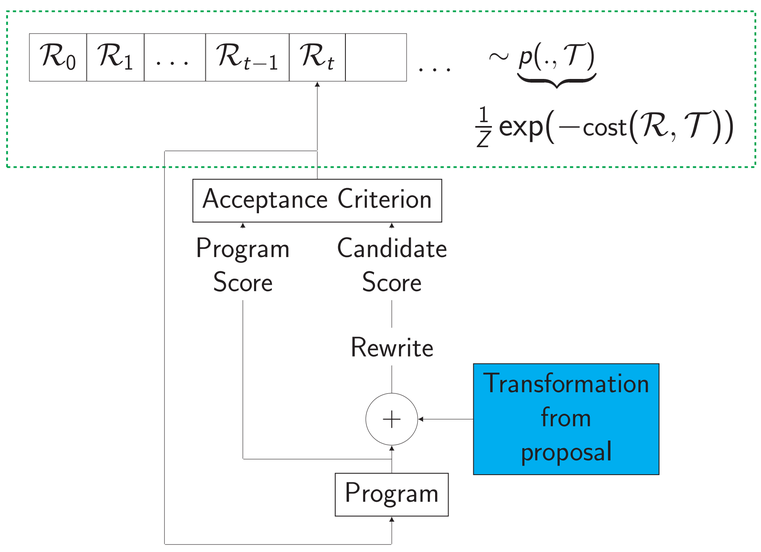
Optimal Rewrite achievable.

Each program $\mathcal{R}$ is associated with a **cost**:

$$\text{cost}(\mathcal{R}, \mathcal{T}) = \omega_e \times \underbrace{\text{eq}(\mathcal{R}, \mathcal{T})}_{\textbf{Correctness}} + \omega_p \times \underbrace{\text{perf}(\mathcal{R})}_{\textbf{Performance}}$$

Program optimization becomes an **unconstrained minimization problem**.

We optimize by running the **Metropolis Algorithm**, a Markov Chain Monte Carlo method:



At each iteration, a transformation is sampled from a **proposal distribution**. Possible types of moves:

- Add empty Instruction
- Delete Instruction
- Replace Instruction
- Transform Opcode
- Change Opcode Width
- Transform Operand
- Swap Instructions
- Shift Instructions

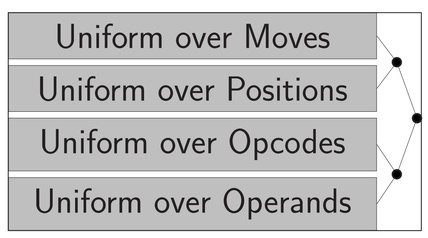
Learning to Superoptimize

Parameterization of the problem

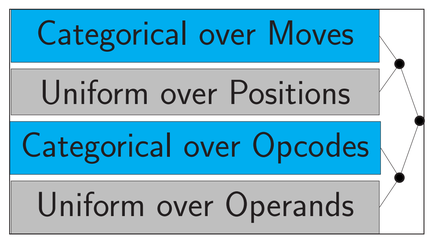
Each transformation corresponds to a series of choice over basic probability distributions:

- What **type of move** is going to be performed?
- Which **positions** in the program are concerned by the move?
- What new **opcodes** should be used?
- What new **operands** should be used?

[1] uses uniform choice over the possible values each time.



Default Proposal distribution



Learnable Proposal distribution

Problem Formulation

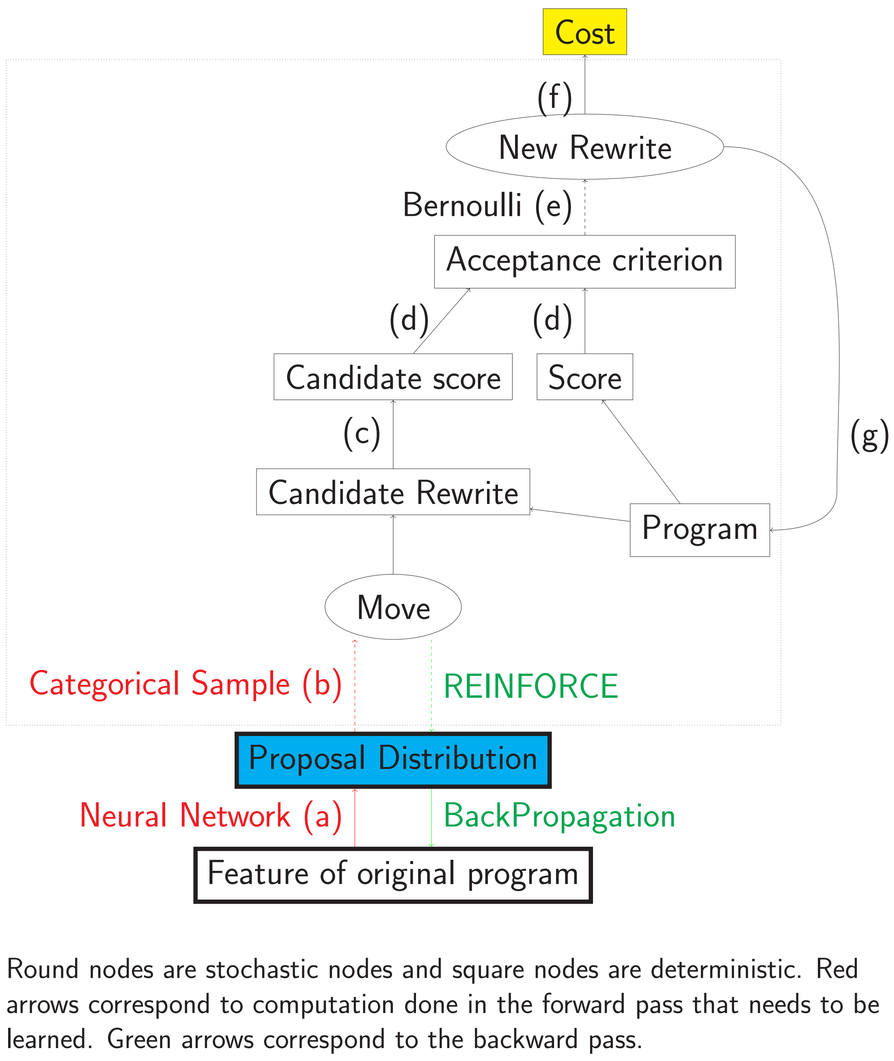
For a given number of MCMC iterations T,
Maximize probability of obtaining good programs at the end.

Minimize expected **ratio** between optimized version and reference.

$$\mathcal{L}(\theta) = \mathbb{E}_{\{\mathcal{R}_t\}\sim q_\theta}\left[\frac{\min_{t=0..T}\text{cost}(\mathcal{R}_t,\mathcal{T})}{\text{cost}(\mathcal{R}_0,\mathcal{T})}\right]$$

Learning the proposal distribution

We unroll the execution of the MCMC algorithm, to understand it as a **Stochastic Computation Graph** [2].
We compute an unbiased estimate of the gradient of our loss function, using the **REINFORCE** [4] algorithm.



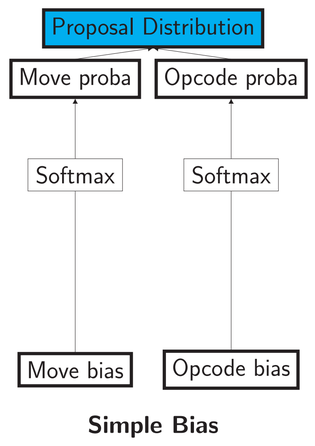
Round nodes are stochastic nodes and square nodes are deterministic. Red arrows correspond to computation done in the forward pass that needs to be learned. Green arrows correspond to the backward pass.

The different steps of the forward pass are:

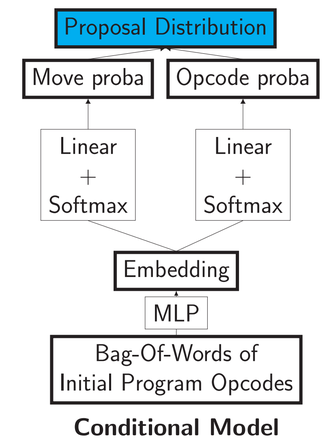
(a) Possibly based on the features of the reference program, the proposal distribution is computed.
(b) A random transformation is sampled from the proposal distribution.
(c) The score of the proposed rewrite is experimentally measured.
(d) The acceptance criterion for the move is computed.
(e) The move is accepted with a probability equal to the acceptance criterion.
(f) The cost is observed, corresponding to the best program obtained during the search.
(g) Moves (b) to (f) are repeated T times.

Experimental Protocol

We instrument the **STOKE** system [1] to be able to estimate the gradients.
We train two types of models:



Simple Bias



Conditional Model

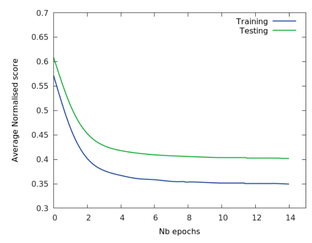
The dataset on which we are training is the **Hacker's Delight** [3] benchmark, a series of 25 small programs.
The models are trained using **Adam**.

Results

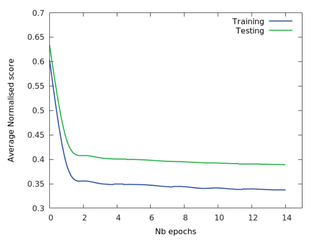
Data augmentation

Training set and **Testing** set are obtained by splitting different Hacker's Delight tasks.
We find equivalent programs with different code from the reference for each task.

Learning the models



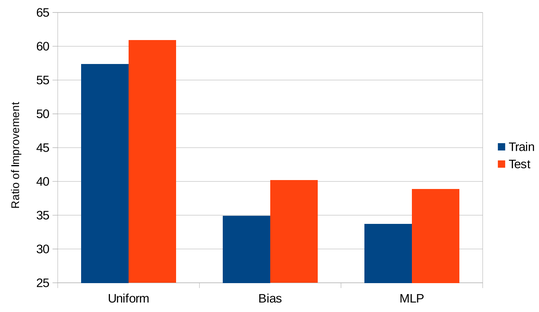
(a) Simple Bias



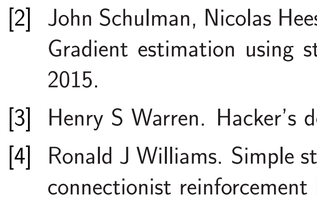
(b) Conditional Model

Evolution of a training run.

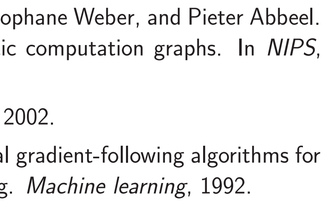
Both models can learn and the improvements achieved generalize to unseen tasks.



Using the learned policy



(a) Before Training



(b) After Training

Optimization traces of a uniform proposal distribution against a learned one.

Discussion

We observe that learning the proposal distribution improves performance. Our method is generic and could be applied to other problems solved using stochastic search.
Further improvements could be achieved by having a richer model to condition over or by also conditioning on the current state of the optimization.

[1] Eric Schkufza, Rahul Sharma, and Alex Aiken. Stochastic superoptimization. *SIGPLAN*, 2013.

[2] John Schulman, Nicolas Heess, Theophane Weber, and Pieter Abbeel. Gradient estimation using stochastic computation graphs. In *NIPS*, 2015.

[3] Henry S Warren. Hacker's delight. 2002.

[4] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 1992.

Supported by Microsoft Research,

rudy@robots.ox.ac.uk

www.robots.ox.ac.uk/~rudy/publications/l2sopt-nampi16.html