Lecture 6
Stochastic Search

Nadia Polikarpova

BlinkFill

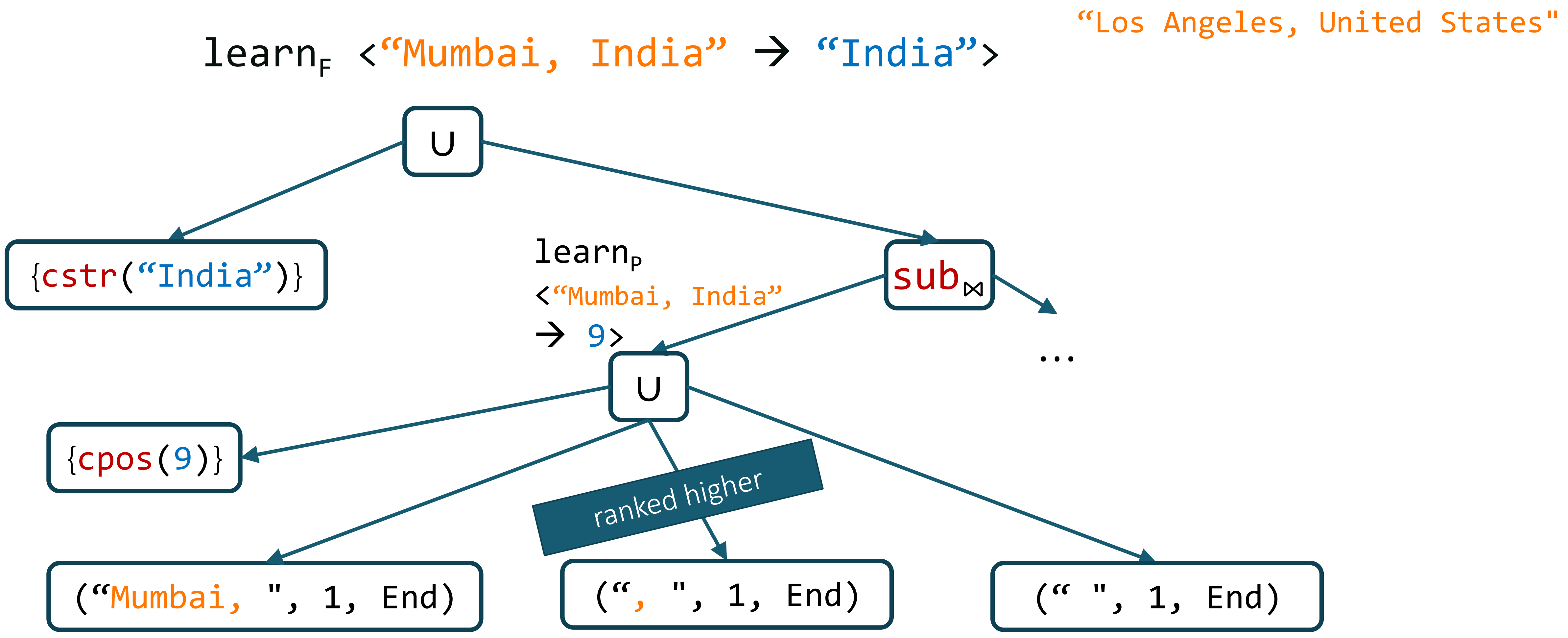
What does BlinkFill use as behavioral constraints? Structural constraints? Search strategy?

- input-output examples; custom string DSL; VSA

What is the main technical insight of BlinkFill wrt FlashFill?

- BlinkFill uses the available inputs (with no outputs) to infer structure (segmentation) common to all inputs
- it uses this structure to shrink the DAG and to rank substring expressions

Example



“Los Angeles, United States"
learn_F <“Mumbai, India” → “India”>
∪
{cstr(“India”)}
sub⋈
…
learn_P <“Mumbai, India” → 9>
∪
{cpos(9)}
ranked higher
(“Mumbai, ", 1, End)
(“, ", 1, End)
(“ ", 1, End)

BlinkFill

Write a BlinkFill program that satisfies:

- `"Programming Language Design and Implementation (PLDI), 2019, Phoenix AZ" -> "PLDI 2019"`
- `"Principles of Programming Languages (POPL), 2020, New Orleans LA" -> "POPL 2020"`

- Between first parentheses and between first and last comma:

```
Concat(SubStr(v1, ("(", 1, End), (")",1, Start)),
       SubStr(v1, (",", 1, End), (",", -1, Start)))
```

BlinkFill

Could we extend the algorithm to support sequences of tokens?

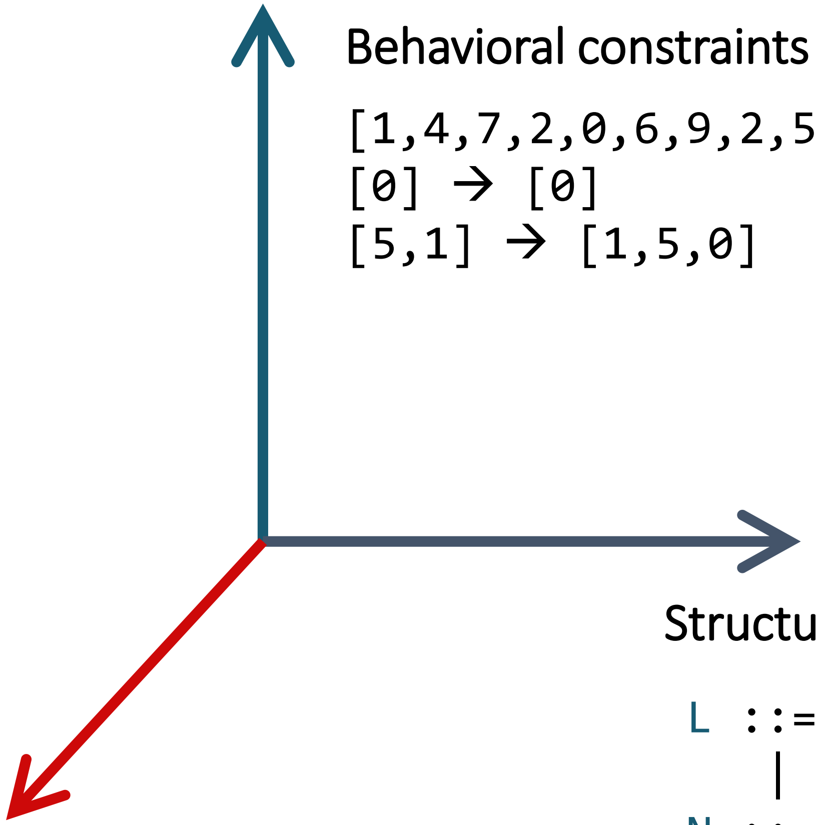
- Each edge of the single-string IDG would have more labels
- Extra edges from 0 and to the last node
- More edges left after intersection (might be a problem, but unclear)
- Need fewer primitive tokens (only lower, upper, digit, space)
- More expressive:
  - "Programming Language Design and Implementation: PLDI 2019" -> "PLDI 2019"
  - “POPL 2020 started on January 22" -> "POPL 2020“
  - `SubStr(v1, (C+ ws d+, 1, Start), (C+ ws d+, 1, End))`

BlinkFill

Strengths? Weaknesses?

- differences between FlashFill and BlinkFill language? which one is more expressive?

The problem statement



Behavioral constraints = examples

```
[1,4,7,2,0,6,9,2,5]  →  [1,2,4,7,0]
[0] → [0]
[5,1] → [1,5,0]
```

Search strategy?

Enumerative
Stochastic
Representation-based
Constraint-based

Structural constraints = grammar

```
L ::= sort(L)  |  L[N..N]
    |  L + L  |  [N]  |  x
N ::= find(L,N)  |  0
```

Stochastic search in synthesis

Weimer, Nguyen, Le Goues, Forrest. *Automatically Finding Patches Using Genetic Programming*. ICSE’09

Schkufza, Sharma, Aiken: *Stochastic superoptimization.* ASPLOS 2013

Shi, Steinhardt, Liang: *FrAngel: Component-Based Synthesis with Control Structures.* POPL'19

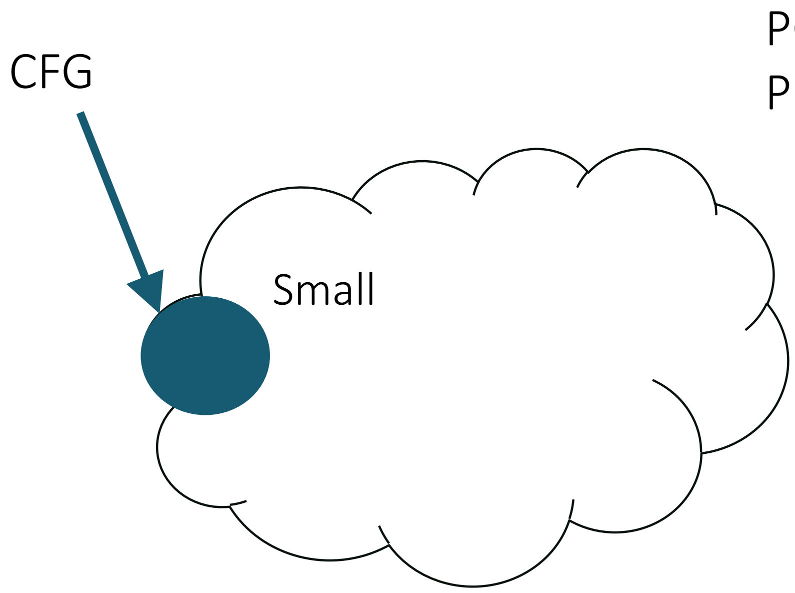
Stochastic search in synthesis

Weimer, Nguyen, Le Goues, Forrest. *Automatically Finding Patches Using Genetic Programming*. ICSE'09

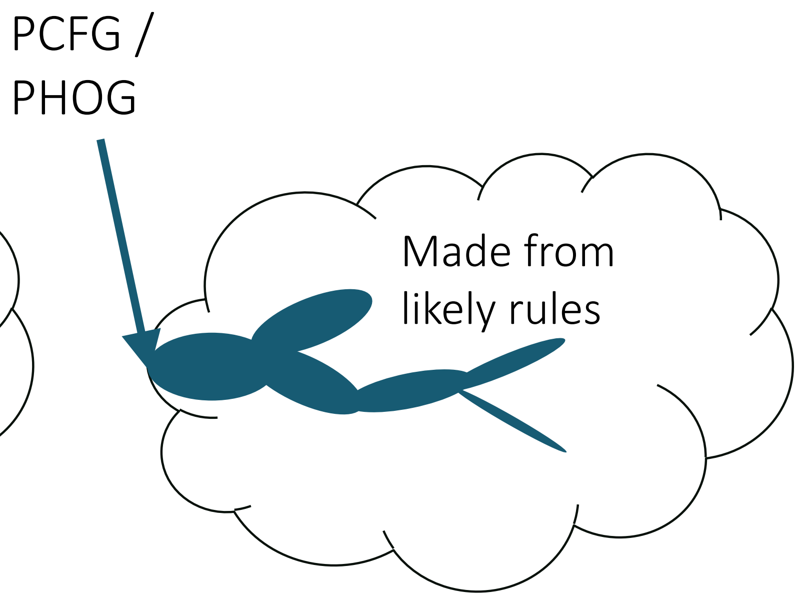
Schkufza, Sharma, Aiken: *Stochastic superoptimization.* ASPLOS 2013

Shi, Steinhardt, Liang: *FrAngel: Component-Based Synthesis with Control Structures.* POPL'19

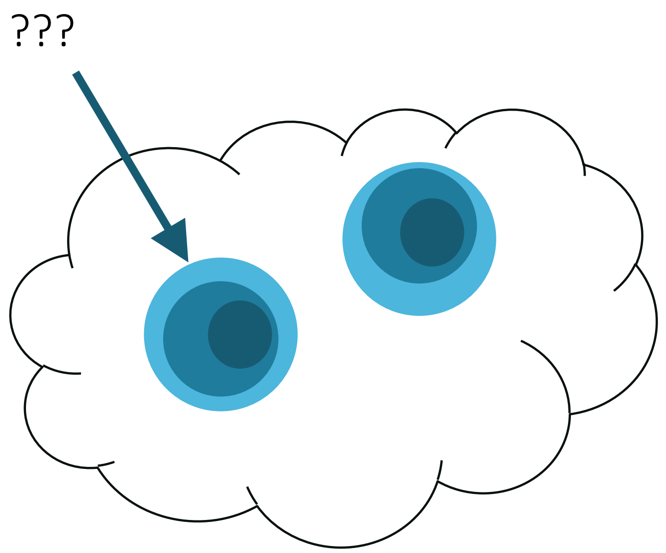
Search space



Enumerative search



Weighted
enumerative search



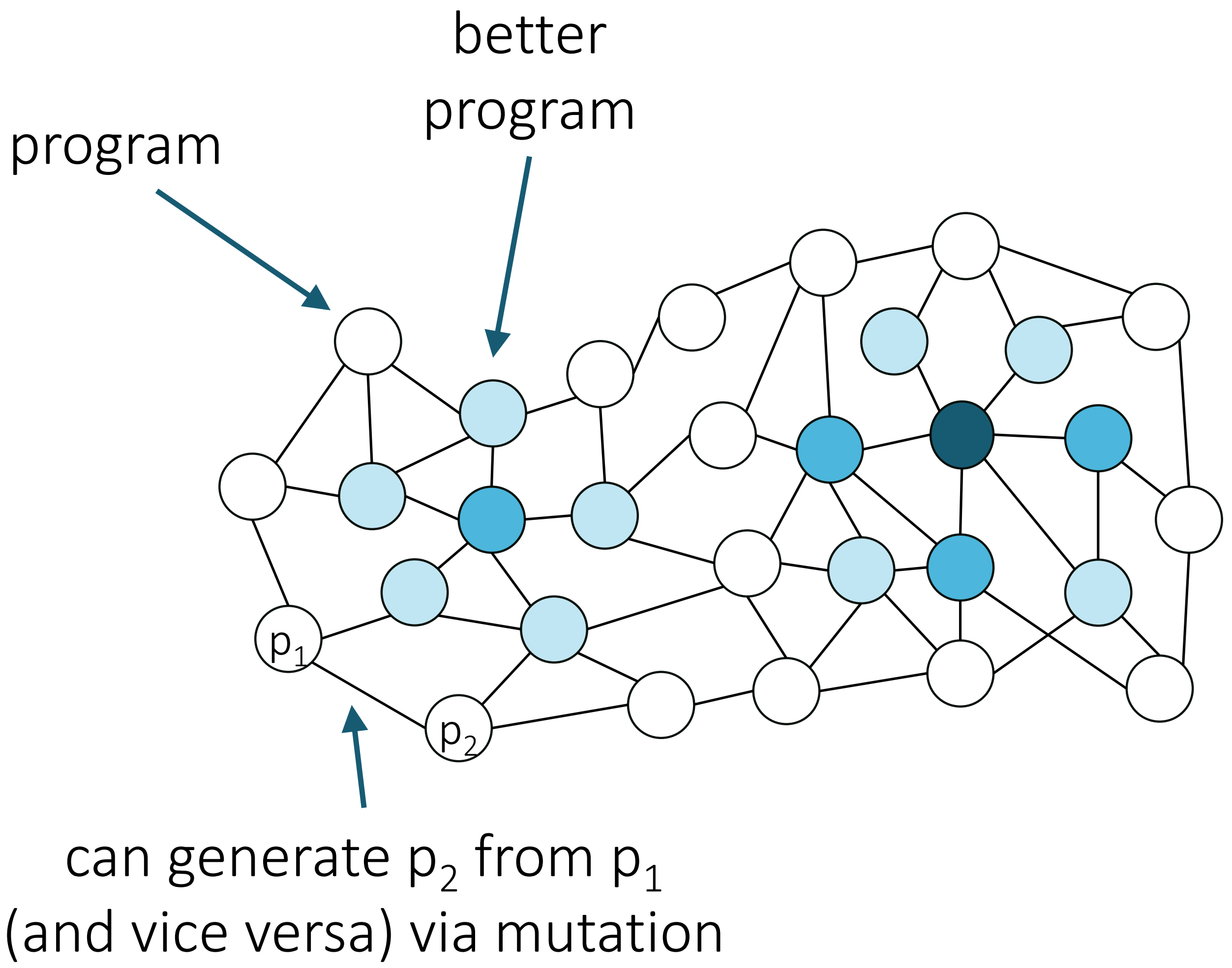
Local search

Naïve local search

To find the best program:

```
p := random()
while (true) {
  p' := mutate(p);
  if (cost(p') < cost(p))
    p := p';
}
```

Will never get to ● from p_1!



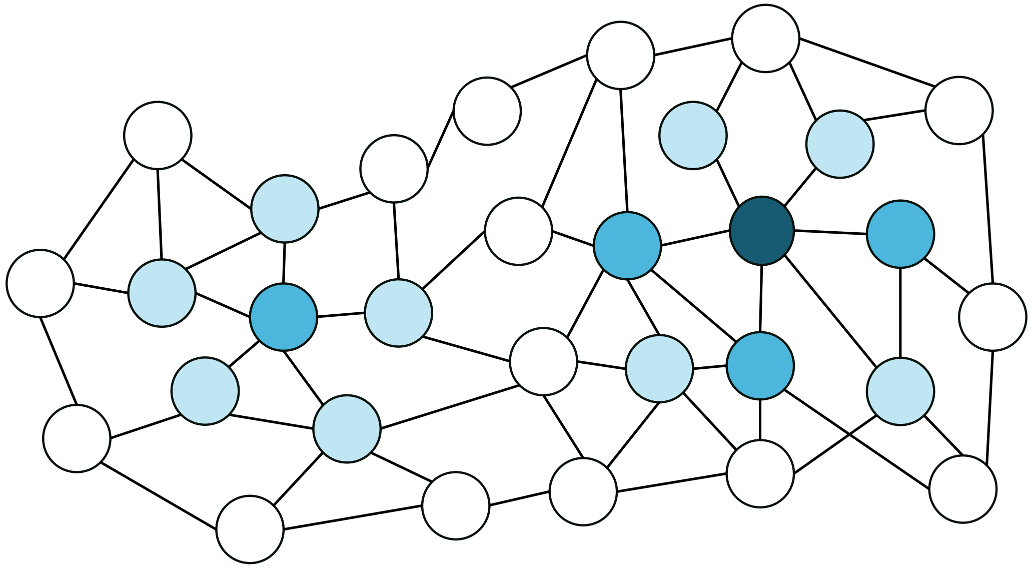
MCMC sampling

Avoid getting stuck in local minima:

```
p := random()
while (true) {
  p’ := mutate(p);
  if (random(A(p -> p’))
    p := p’;
}
```

where

- if p’ is better than p: $A(p \to p') = 1$
- otherswise: $A(p \to p')$ decreases with difference in cost between p’ and p



MCMC sampling

Metropolis algorithm:

$$A(p \to p') = \min(1, e^{-\beta(C(p')-C(p))})$$

The theory of Markov chains tells us that in the limit we will be sampling with the probability proportional to

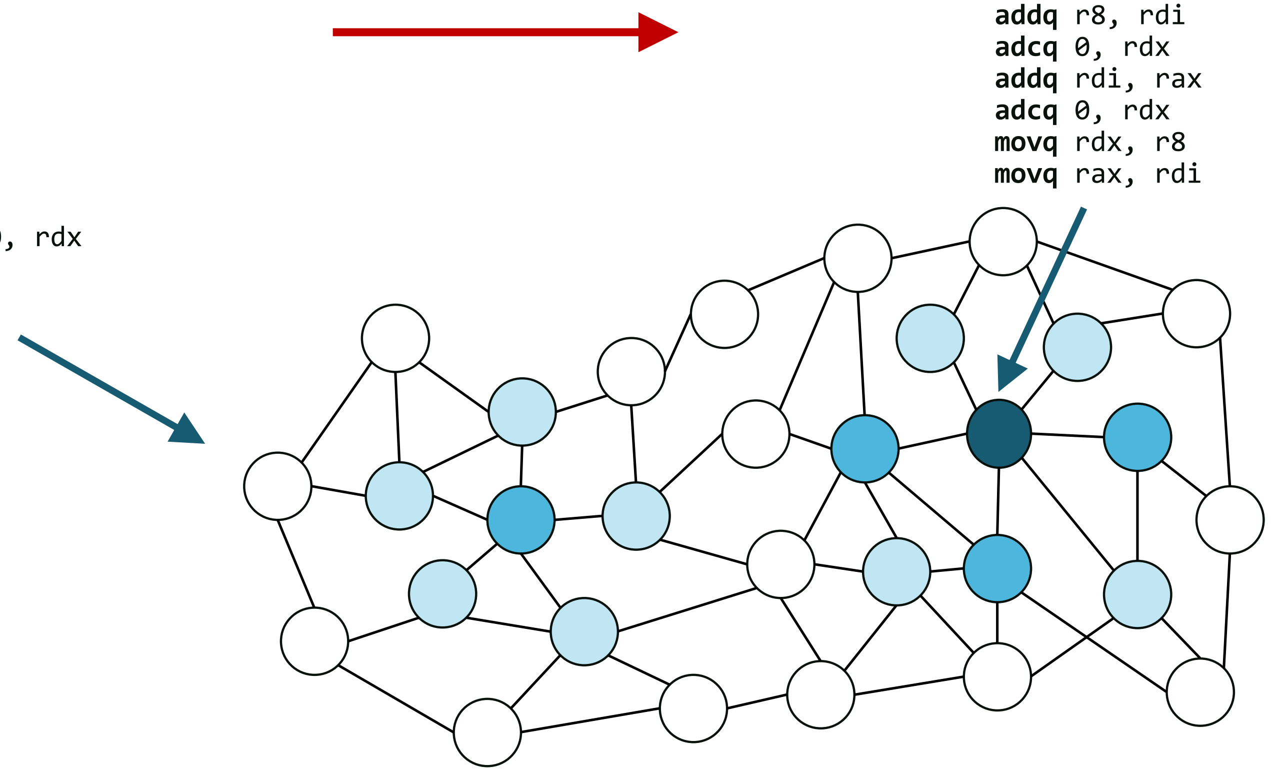
$$e^{-\beta * C(p)}$$

MCMC for superoptimization

[Schkufza, Sharma, Aiken ‘13]

```
.L0:
movq rsi, r9
movl ecx, ecx
shrq 32, rsi
andl 0xffffffff, r9d
movq rcx, rax
movl edx, edx
imulq r9, rax
imulq rdx, r9
imulq rsi, rdx
imulq rsi, rcx
addq rdx, rax
jae .L2
movabsq 0x100000000, rdx
addq rdx, rcx
.L2:
movq rax, rsi
movq rax, rdx
shrq 32, rsi
salq 32, rdx
addq rsi, rcx
addq r9, rdx
adcq 0, rcx
addq r8, rdx
adcq 0, rcx
addq rdi, rdx
adcq 0, rcx
movq rcx, r8
movq rdx, rdi
```

```
.L0:
shlq 32, rcx
movl edx, edx
xorq rdx, rcx
movq rcx, rax
mulq rsi
addq r8, rdi
adcq 0, rdx
addq rdi, rax
adcq 0, rdx
movq rdx, r8
movq rax, rdi
```



Cost function

$$C_s(p) = \mathrm{eq}_s(p) + \mathrm{perf}(p)$$

source program

penalty for wrong results

penalty for being slow

$$\mathrm{eq}_s(p)=\sum_{t\in Tests} \mathrm{reg}_s(p,t) + \mathrm{mem}_s(p,t) + \mathrm{err}(p,t)$$

# of different bits in registers/memory

# of segfaults etc

when $\mathrm{eq}_s(p) = 0$, use a symbolic validator

Cost function

$$C_s(p) = \mathrm{eq}_s(p) + \mathrm{perf}(p)$$

source program

penalty for wrong results

penalty for being slow

$$\mathrm{perf}(p) = \sum_{i \in \mathrm{instr}(p)} \mathrm{latency}(i)$$

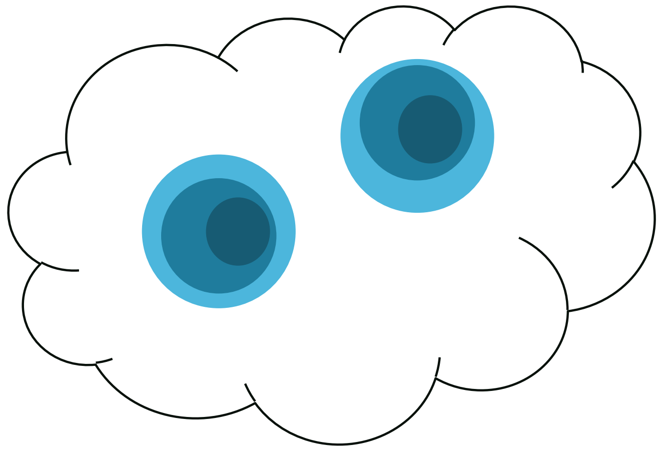
Local search: discussion

Strengths:

- can explore program spaces with no a-priori bias

Limitations?

- only applicable when there is a cost function that faithfully approximates correctness
- Counterexample: round to next power of two



Stochastic search in synthesis

Weimer, Nguyen, Le Goues, Forrest. *Automatically Finding Patches Using Genetic Programming*. ICSE'09

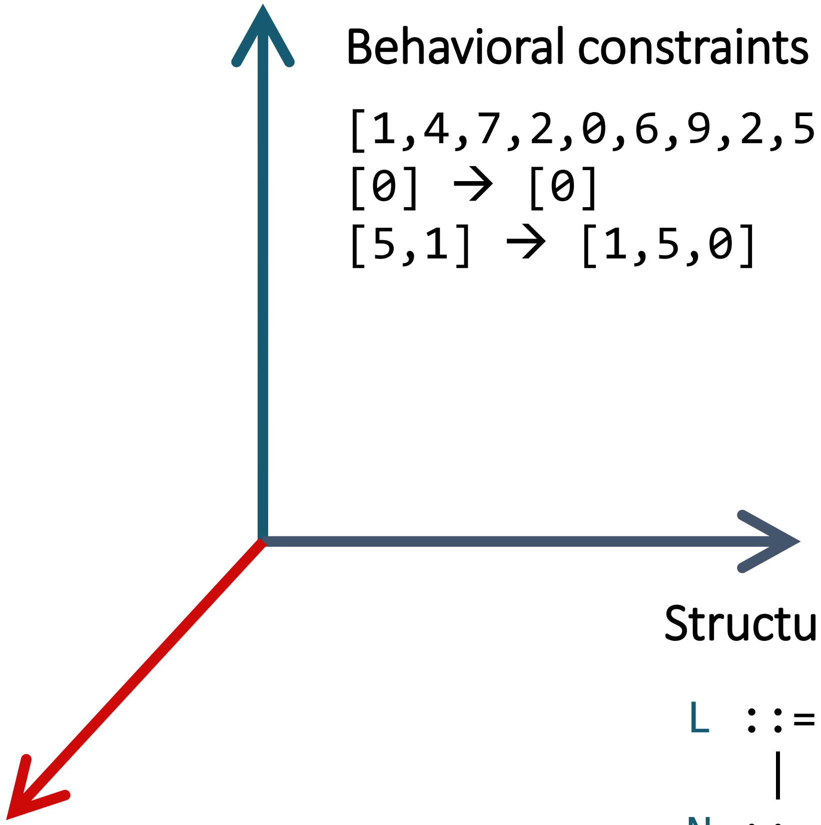
- Similar but for program repair, uses genetic programming

Schkufza, Sharma, Aiken: *Stochastic superoptimization.* ASPLOS 2013

Shi, Steinhardt, Liang: *FrAngel: Component-Based Synthesis with Control Structures.* POPL'19

- Samples from a grammar with bias towards partial solutions
- I assume they use stochastic just for ease of sampling

Next



Behavioral constraints = examples

```
[1,4,7,2,0,6,9,2,5]  →  [1,2,4,7,0]
[0] → [0]
[5,1] → [1,5,0]
```

Structural constraints = grammar

```
L ::= sort(L)  |  L[N..N]
   |  L + L  |  [N]  |  x
N ::= find(L,N)  |  0
```

Search strategy?

Enumerative
Stochastic
Representation-based
Constraint-based