Stochastic Search-Based Synthesis

Woosuk Lee

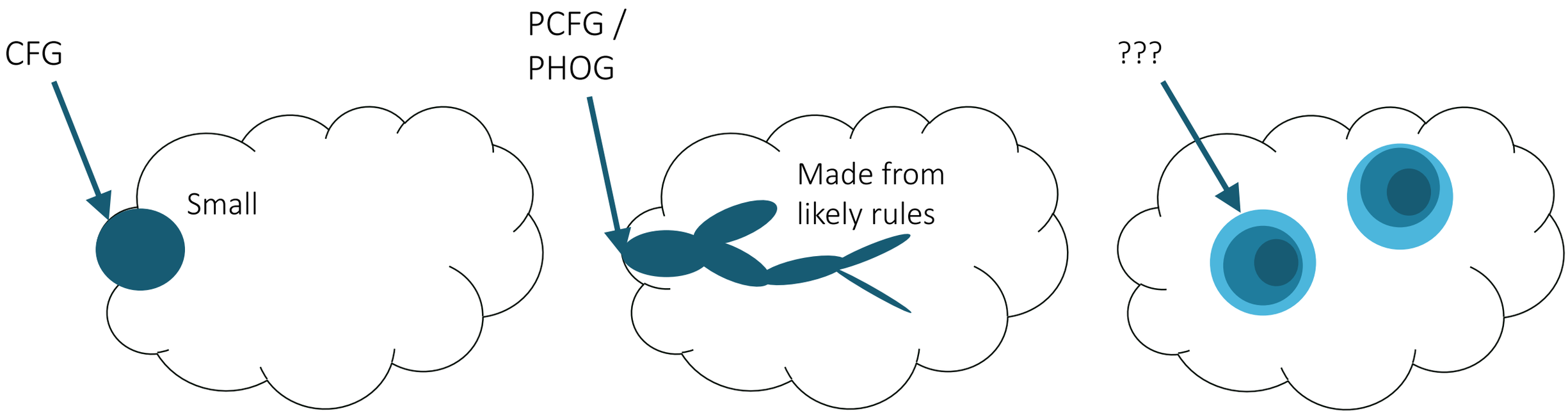
CSE9116 SPRING 2024

Hanyang University

Three Search Strategies

- **Enumerative**: enumeration + optimization
- **Stochastic**: probabilistic walk
- **Constraint-based**: encoding a synthesis problem as a SAT/SMT instance

Searched Regions



Enumerative search

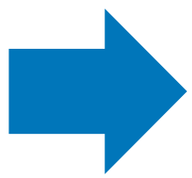
Weighted
enumerative search

Local search

- **Enumerative**: smaller candidates first + probabilistic guidance
- **Stochastic**: variants of an initial random candidate

Application — Superoptimization (STOKE)

```
 1 # gcc -03
 2
 3 movq rsi, r9
 4 movl ecx, ecx
 5 shrq 32, rsi
 6 andl 0xffffffff, r9d
 7 movq rcx, rax
 8 movl edx, edx
 9 imulq r9, rax
10 imulq rdx, r9
11 imulq rsi, rdx
12 imulq rsi, rcx
13 addq rdx, rax
14 jae .L0
15 movabsq 0x100000000, rdx
16 addq rdx, rcx
17 .L0:
18 movq rax, rsi
19 movq rax, rdx
20 shrq 32, rsi
21 salq 32, rdx
22 addq rsi, rcx
23 addq r9, rdx
24 adcq 0, rcx
25 addq r8, rdx
26 adcq 0, rcx
27 addq rdi, rdx
28 adcq 0, rcx
29 movq rcx, r8
30 movq rdx, rdi
```



```
 1 # STOKE
 2
 3 shlq 32, rcx
 4 movl edx, edx
 5 xorq rdx, rcx
 6 movq rcx, rax
 7 mulq rsi
 8 addq r8, rdi
 9 adcq 0, rdx
10 addq rdi, rax
11 adcq 0, rdx
12 movq rdx, r8
13 movq rax, rdi
```

1.6x speed-up

Montgomery multiplication kernel from the OpenSSL RSA library. Compilations shown for gcc –O3 (left) and a stochastic optimizer (right).

http://stoke.stanford.edu

Goal

- Given a source program *s* and test inputs *Tests*,

- Finding another program of better performance and semantically equivalent to *s*

Requirement 1: Cost Function

- Compares program output to reference test cases and measures candidates' performance

- Lower the better (0: best)

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

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

Requirement 2: Move Function

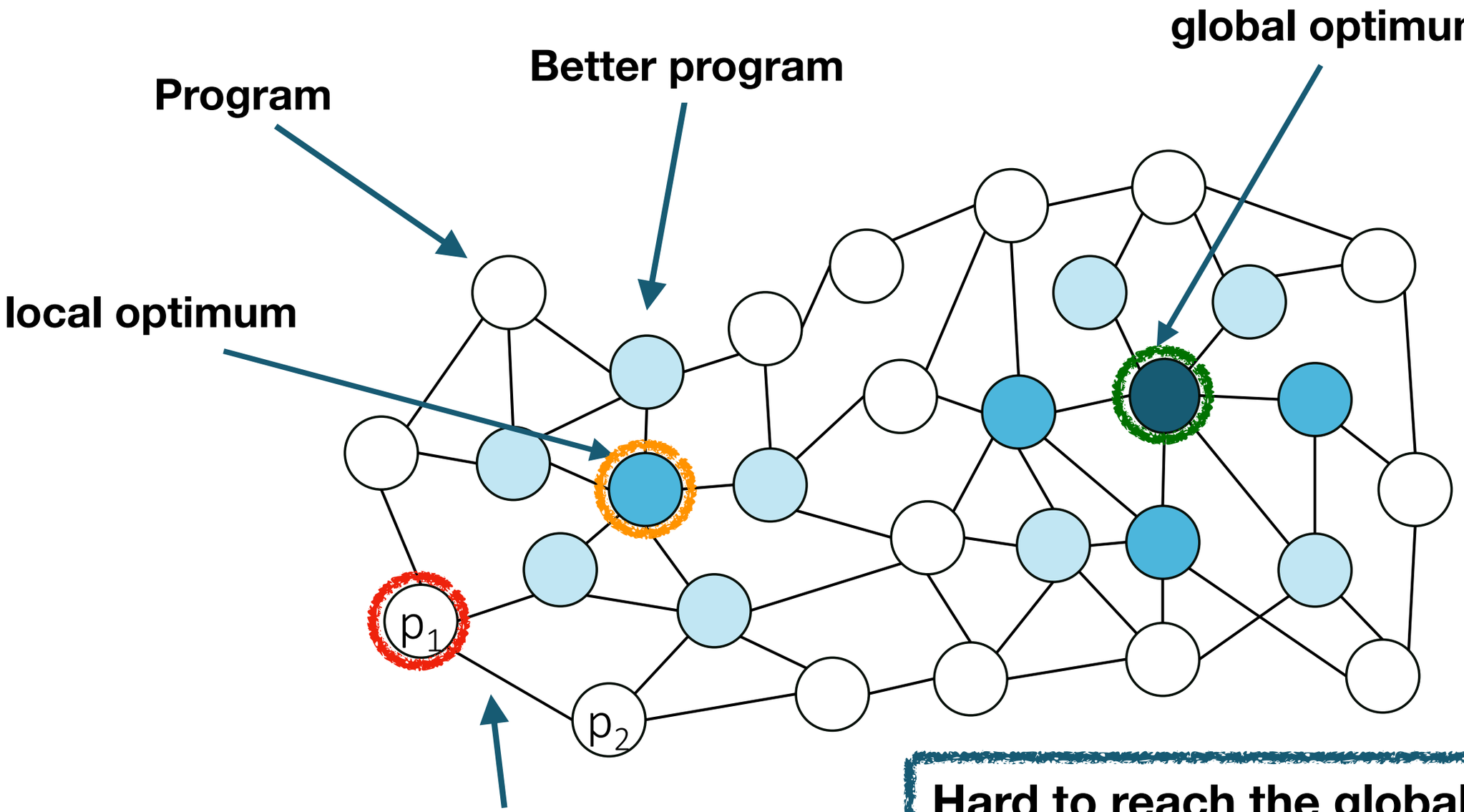
- Makes a small syntactic change to a current program
- Examples
  - Generate a random instruction
  - Remove a randomly chosen instruction
  - Replace opcode (e.g., ADD → MUL) of a randomly chosen instruction with another
  - …

Initial Stochastic Synthesis Algorithm

```
p :=  random()
while cost(p) > 0:
  p' := propose_move(p);
  if (cost(p') < cost(p)):
    p := p'
```

p: best candidate found so far

Local Minima



Any program candidates can be visited by the move function (ergodicity)

Hard to reach the global optimum starting from p1 (due to local minima)

Stochastic Synthesis Algorithm (improved)

```
p :=  random()
while cost(p) > 0:
   p’ := propose_move(p);
   if (random(A(p->p’)):
      p := p’
```

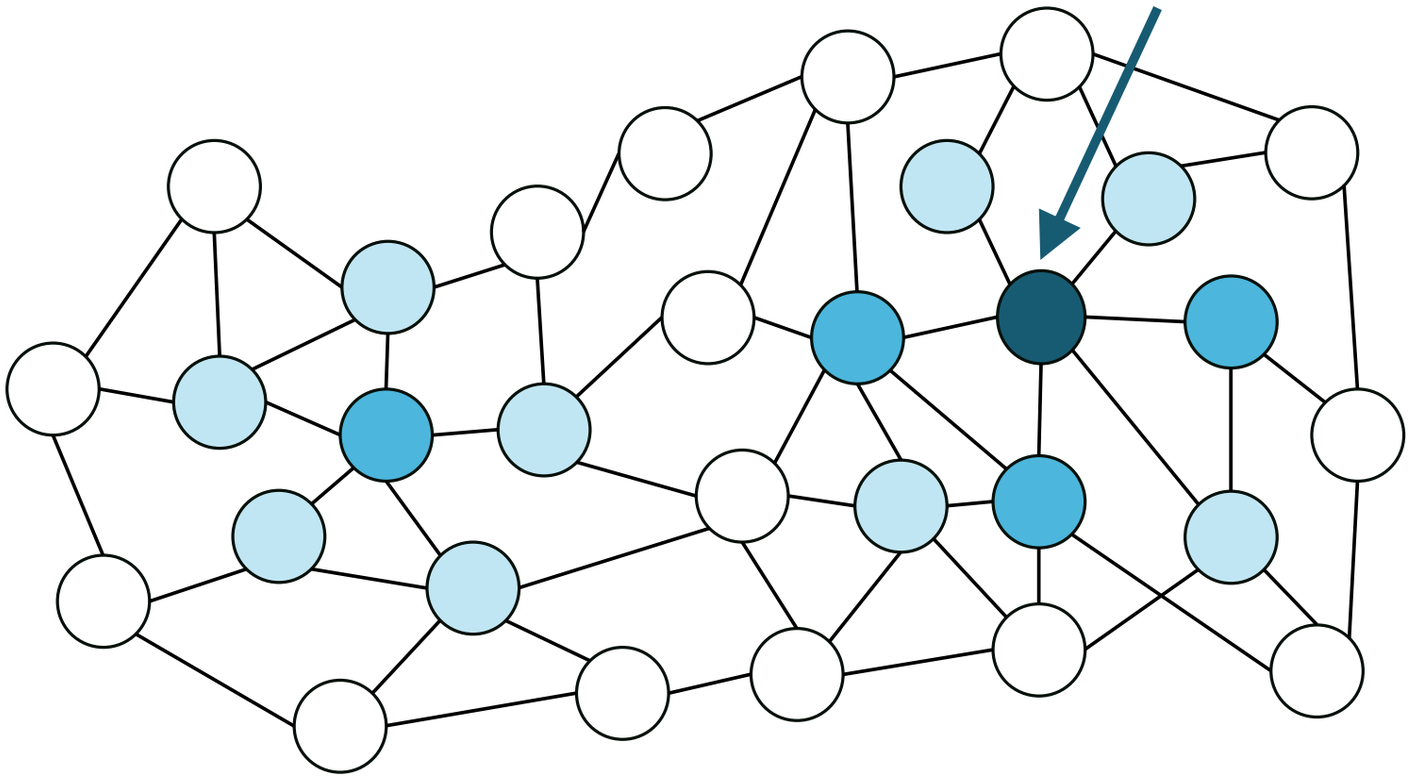
A(p -> p’) : prob. of changing the current best candidate from p to p’

- If cost(p’) < cost(p) then 1
- Otherwise, proportional to cost(p) / cost(p’)

Possible to Reach the Global Optimum

```
.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
```



Guaranteed Property

- For any two candidates p1, p2, if each is reachable from the other by repeatedly applying the move function (p1 ↔ p2, called *ergodicity*)

- a global optimum can be eventually found!

- Through Metropolis-Hastings algorithm

Successful Results

Ray tracing library

```
V delta(V& v1, V& v2, float r1, float r2) {
  // v1  = [(rdi),      4(rdi),      8(rdi)    ]
  // v2  = [(rsi),      4(rsi),      8(rsi)    ]
  // ret = [xmm0[63:32], xmm0[31:0], xmm1[31:0]]

  assert(0.0 <= r1 <= 1.0 && 0.0 <= r2 <= 1.0);

  // gcc -O3:
  return V(99*(v1.x*(r1-0.5))+99*(v2.x*(r2-0.5)),
           99*(v1.y*(r1-0.5))+99*(v2.y*(r2-0.5)),
           99*(v1.z*(r1-.05))+99*(v2.z*(r2-0.5)));
  // STOKE:
  return V(99*(v1.x*(r1-0.5)),
           99*(v1.y*(r1-0.5)),
           99*(v2.z*(r2-0.5)));
}
```

```
 1 # gcc -O3
 2
 3 movl 0.5, eax
 4 movd eax, xmm2
 5 subss xmm2, xmm0
 6 movss 8(rdi), xmm3
 7 subss xmm2, xmm1
 8 movss 4(rdi), xmm5
 9 movss 8(rsi), xmm2
10 movss 4(rsi), xmm6
11 mulss xmm0, xmm3
12 movl 99.0, eax
13 movd eax, xmm4
14 mulss xmm1, xmm2
15 mulss xmm0, xmm5
16 mulss xmm1, xmm6
17 mulss (rdi), xmm0
18 mulss (rsi), xmm1
19 mulss xmm4, xmm5
20 mulss xmm4, xmm6
21 mulss xmm4, xmm3
22 mulss xmm4, xmm2
23 mulss xmm4, xmm0
24 mulss xmm4, xmm1
25 addss xmm6, xmm5
26 addss xmm1, xmm0
27 movss xmm5, -20(rsp)
28 movaps xmm3, xmm1
29 addss xmm2, xmm1
30 movss xmm0, -24(rsp)
31 movq -24(rsp), xmm0
```

```
 1 # STOKE
 2
 3 movl 0.5 eax
 4 movd eax, xmm2
 5 subps xmm2, xmm0
 6 movl 99.0, eax
 7 subps xmm2, xmm1
 8 movd eax, xmm4
 9 mulss 8(rsi), xmm1
10 movss 4(rdi), xmm5
11 mulss xmm0, xmm5
12 mulss (rdi), xmm0
13 mulss xmm4, xmm0
14 mulps xmm4, xmm5
15 punpckldq xmm5, xmm0
16 mulss xmm4, xmm1
```

> 5x speed-up

Successful Results

BLAS (Linear algebra) library

```
void SAXPY(int* x, int* y, int a) {
  x[i]   = a * x[i]   + y[i];
  x[i+1] = a * x[i+1] + y[i+1];
  x[i+2] = a * x[i+2] + y[i+2];
  x[i+3] = a * x[i+3] + y[i+3];
}
```

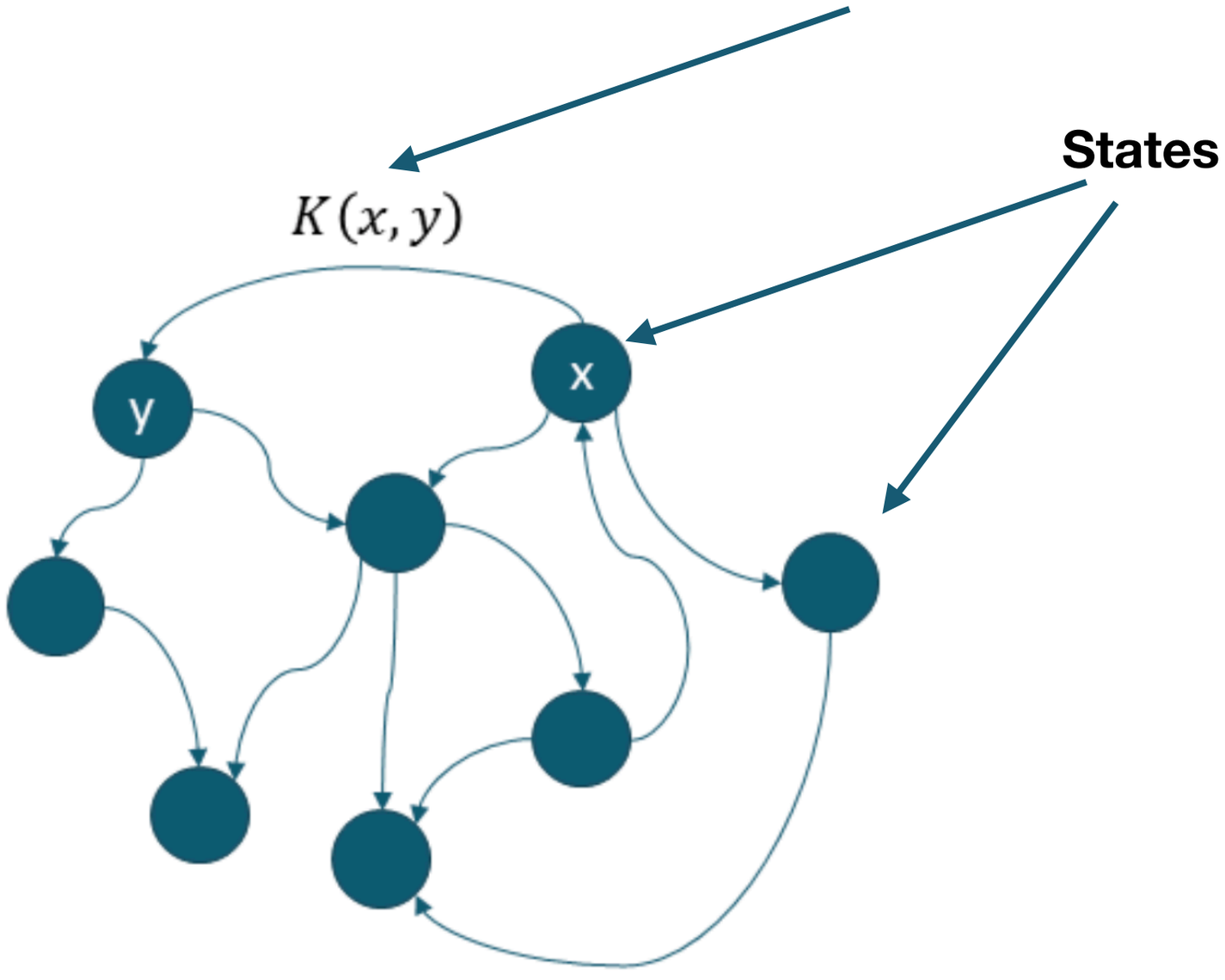
```
1 # gcc -O3
2
3 movslq ecx,rcx
4 leaq (rsi,rcx,4),r8
5 leaq 1(rcx),r9
6 movl (r8),eax
7 imull edi,eax
8 addl (rdx,rcx,4),eax
9 movl eax,(r8)
10 leaq (rsi,r9,4),r8
11 movl (r8),eax
12 imull edi,eax
13 addl (rdx,r9,4),eax
14 leaq 2(rcx),r9
15 addq 3,rcx
16 movl eax,(r8)
17 leaq (rsi,r9,4),r8
18 movl (r8),eax
19 imull edi,eax
20 addl (rdx,r9,4),eax
21 movl eax,(r8)
22 leaq (rsi,rcx,4),rax
23 imull (rax),edi
24 addl (rdx,rcx,4),edi
25 movl edi,(rax)
```

```
1 # STOKE
2
3 movd edi,xmm0
4 shufps 0,xmm0,xmm0
5 movups (rsi,rcx,4),xmm1
6 pmullw xmm1,xmm0
7 movups (rdx,rcx,4),xmm1
8 paddw xmm1,xmm0
9 movups xmm0,(rsi,rcx,4)
```

1.4x speed-up

Markov Chains

Probability of transitioning from state x to state y



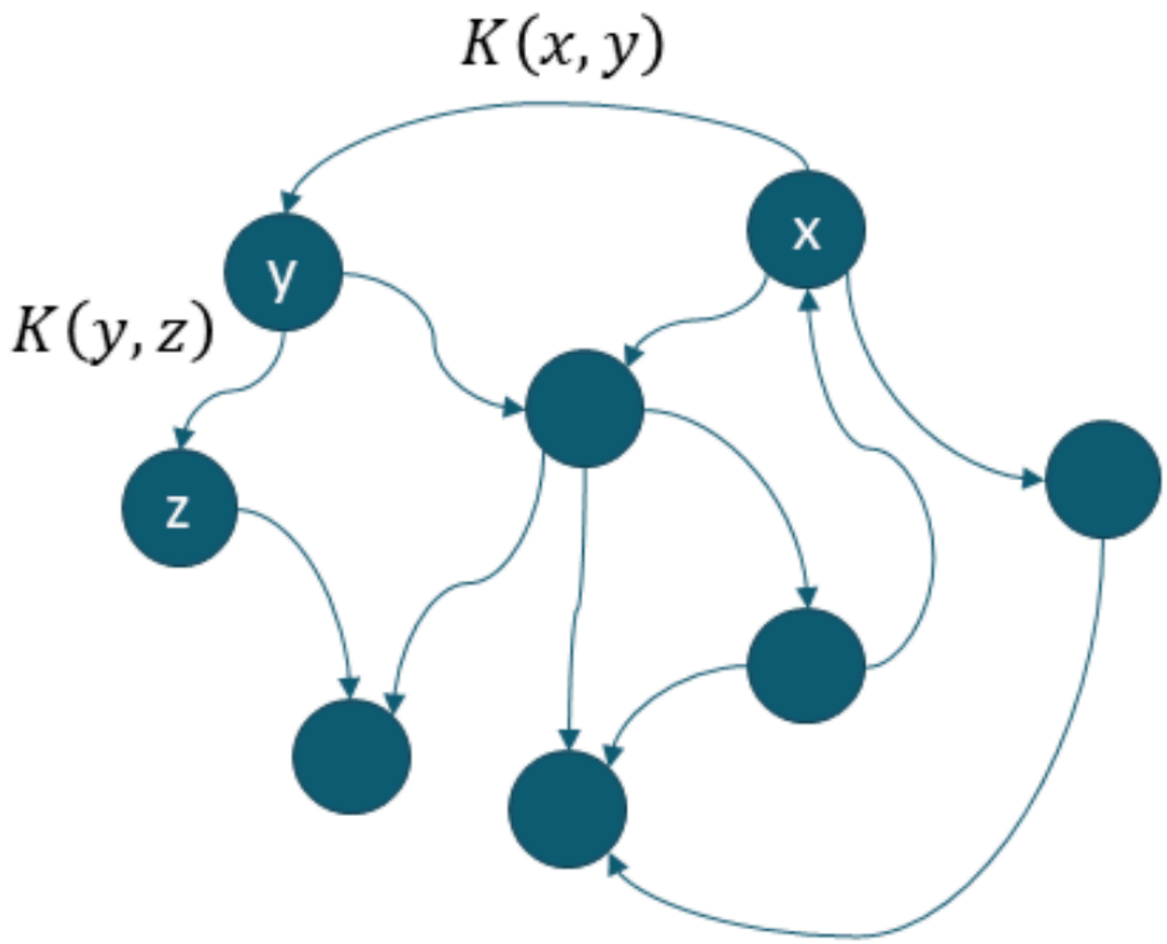
Markov Chains

- A matrix K such that

  x-th row, y-th col

  - $\forall x, y.\ K(x, y) \geq 0$
  - $\forall x.\ \sum_y K(x, y) = 1$
  -

- $K(x, y) \cdot K(y, z)$: probability of transitioning from x to y and to z
- $\sum_y K(x, y) \cdot K(y, z)$: probability of transition from x to z in two steps (denoted $K^2(x, z)$)
- $K^n(x, y)$: probability of transitioning from x to y in exactly *n* steps



Fundamental Theorem of Markov Chains

- If a Markov chain is connected (every state is reachable from every other state) and not *periodic*,

periodic example:

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

1 always moves to 2,

2 always moves to 1

- Then, $\forall x.\ \lim_{n \to \infty} K^n(x, y) = \pi(y)$

Stationary distribution

$$\pi = K \cdot \pi$$

- Intuitively, if a process has been running for a long time, where I started is not important.

In Program Synthesis

- State ≃ program
- We hope to find a K such that
  - If a program x is "good" — $\pi(x)$ is high
  - If a program x is "bad" — $\pi(x)$ is low
- Then,
  - Starting from any random program, keep doing a transition according to K
  - Then we will reach a good program.

A score of each program candidate
(0: worst, 1: best)

How to Find such a *K*?†

$$K(x,y) = \begin{cases} 1 & (A(x,y) \geq 1) \\ A(x,y) & (A(x,y) < 1) \end{cases} \quad \text{where } A(x,y) = \frac{\pi(y)}{\pi(x)}$$

Why? — because $\pi = K\pi$.

Proof) $\pi(x) * K(x,y) = \pi(y) * K(y,x)$.

$\therefore \sum_x \pi(x) * K(x,y) = \sum_x \pi(y) * K(y,x) = \pi(y) \sum_x K(y,x) = \pi(y)$

Therefore $\pi = K\pi$.

†Assuming transition from an arbitrary program to every other program has the same probability

Program Synthesis with Metropolis-Hastings

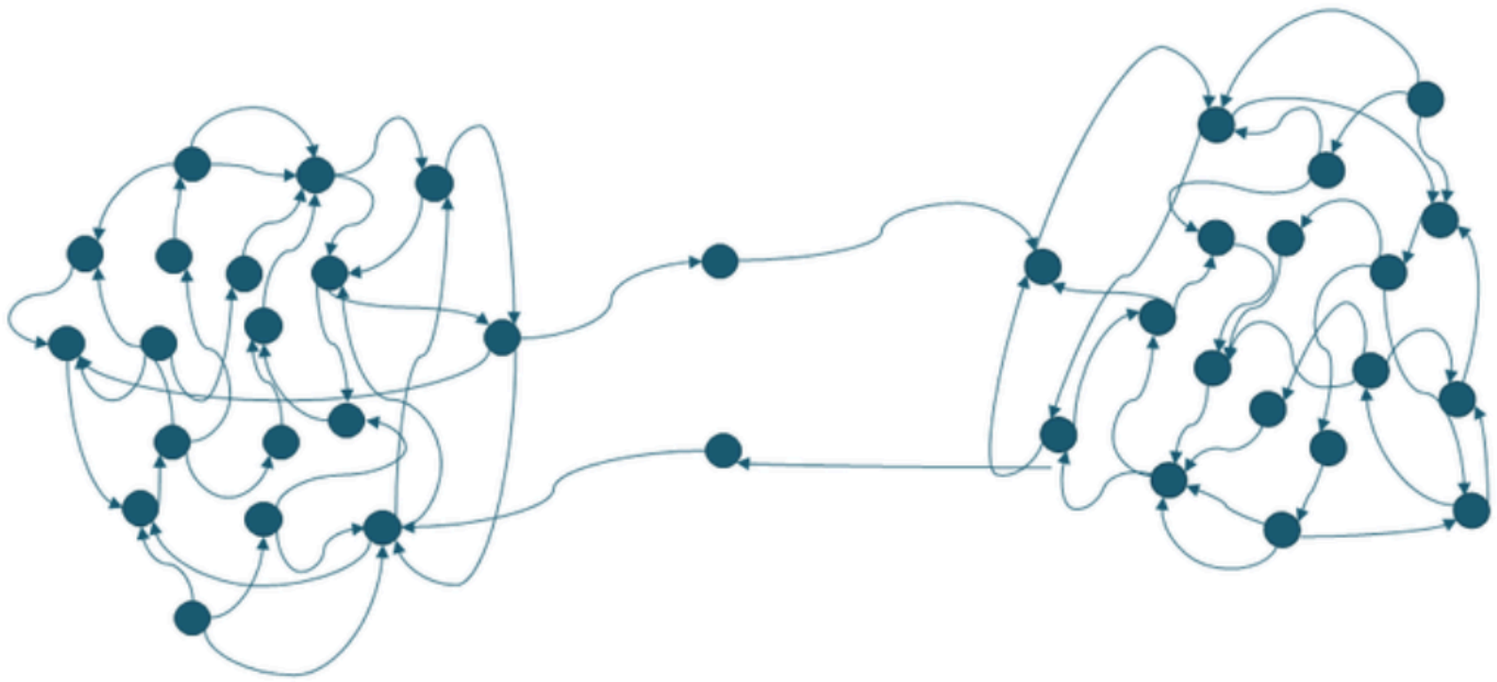
- By the Fundamental Theorem, $\forall x.\ \lim_{n\to\infty} K^n(x,y) = \pi(y)$

- Starting from a random program, if we repeatedly do the following steps for a "long enough" time
  - If a next candidate (obtainable by the move function) is better, move.
  - Otherwise, move with a probability proportional to the ratio between the scores of current and new programs
- Eventually we will reach the best program *p* such that $\pi(p) = 1$.

Property of the Algorithm

- No search bias
  - enumerative search: small programs first
  - But a finite program space is often used (e.g., by bounding size)
- In practice, $A(x, y) = \exp(-\beta \cdot \frac{\pi(y)}{\pi(x)})$ is often used
  - Using a proper β is important
  - β ↑ : pure random search, β ↓ : greedy search
- π should tell us whether a program is getting closer to being correct (e.g., if π returns 0 or 1, it won't work)

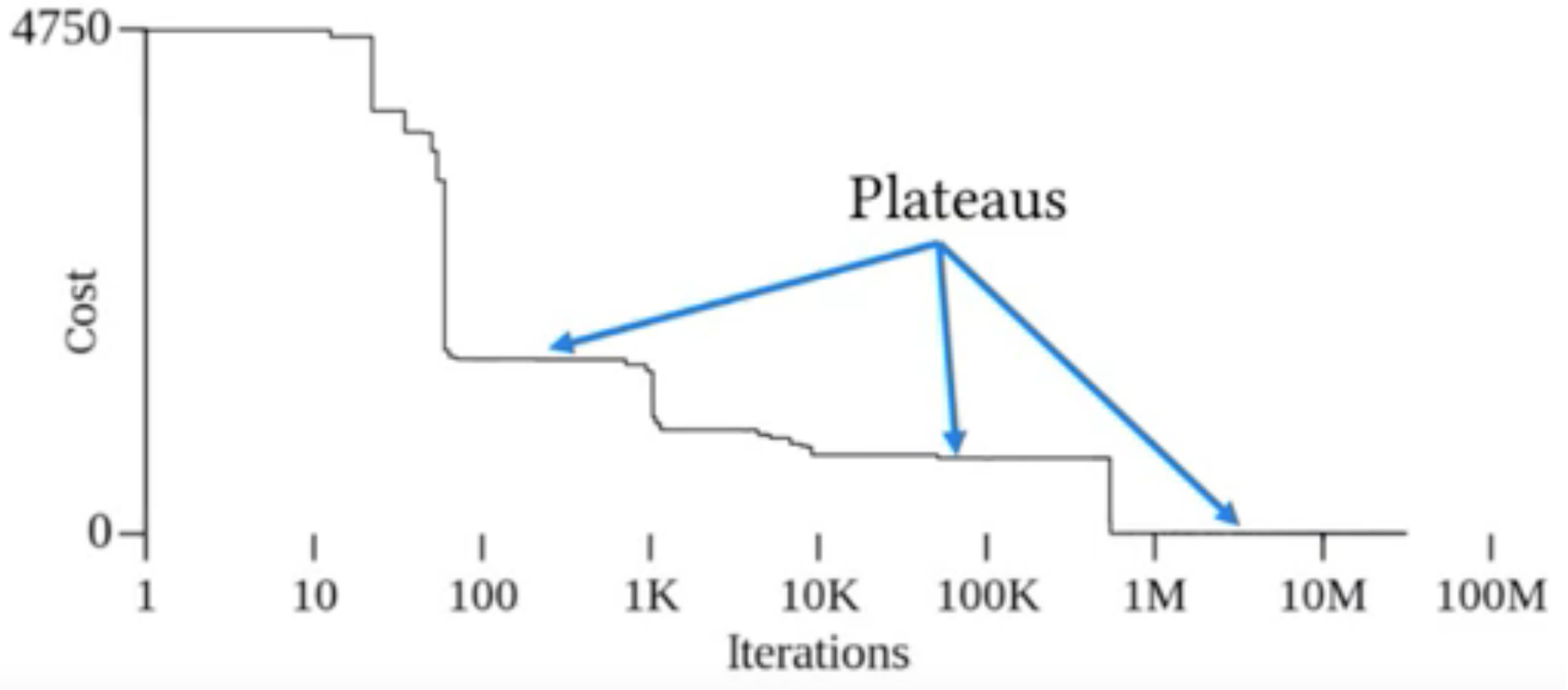
Almost Disjoint Clusters



- Strongly connected components: many candidates have (nearly) equal score due to many semantically equivalent programs

- Starting in one cluster, the prob. of transitioning to the other is extremely low.

Getting Stuck in a Long Search



- Because it is hard to escape from a strongly connected component, it is often beneficial to abandon a search and begin a fresh one.

What is a Good Restart Strategy?

- Let A be a randomized algorithm that always produces the correct solution when it stops.

- Minimizing the expected time required to obtain a solution from A

  - Run A for a fixed amount of time t1 (e.g., 10000 iterations)

  - If a solution isn't found, run A for another fixed amount of time t2, etc

---

Luby et al., Optimal Speedup of Las Vegas Algorithms, 1993

What is a Good Restart Strategy?

- Let $S = (t_1, t_2, \cdots)$ be a restart strategy.
- Let $\ell_A = \min_S T(A, S)$ where $T(A, S)$ is the expected running time of A under strategy S
  - I.e., the expected running time of the optimal strategy

What is a Good Restart Strategy?

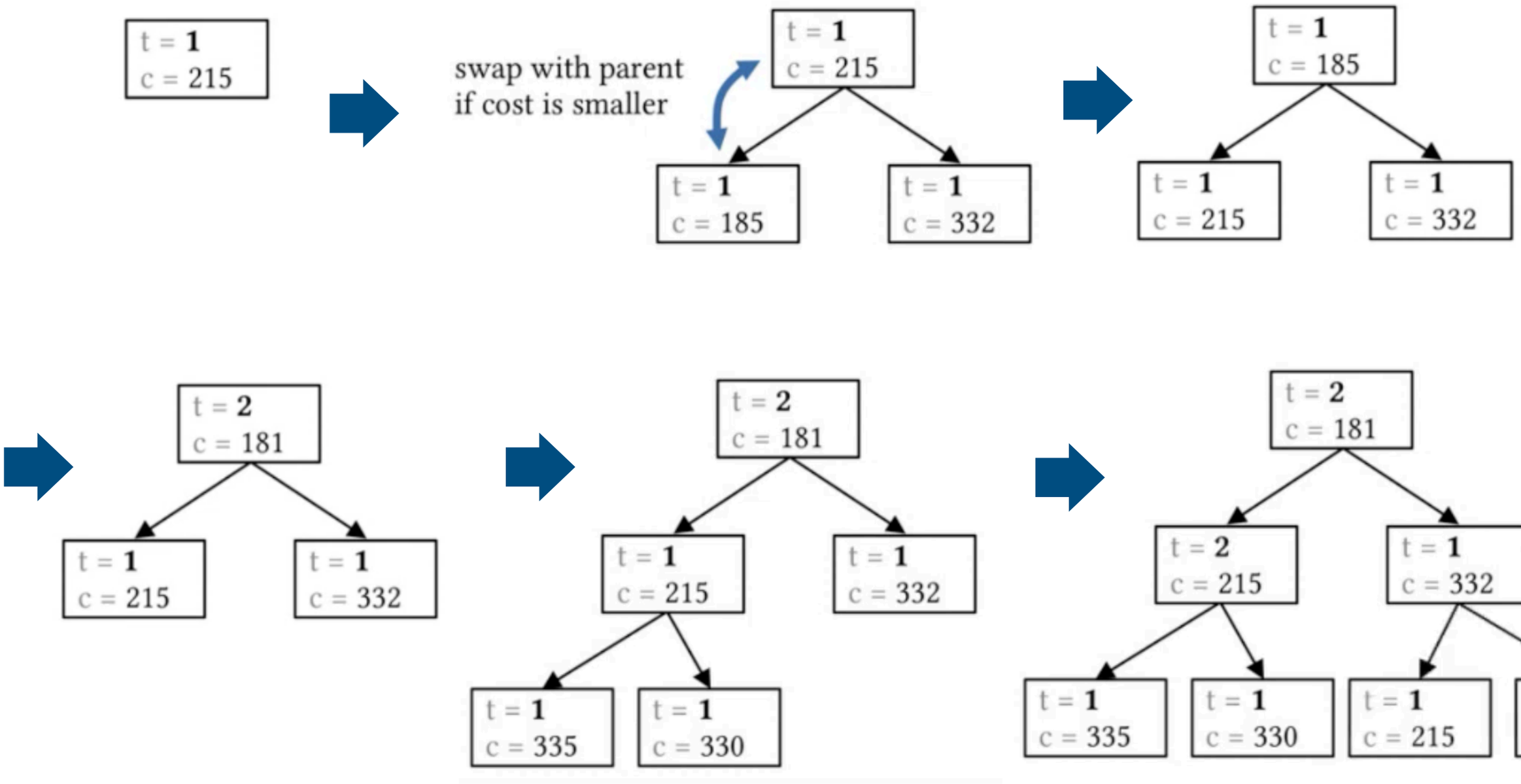
- Let $S^* = (1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8, 1, 1, 2, \cdots)$
  - called Luby sequence L_∞ where $L_0 = 1$ and $L_i = L_{i-1}, L_{i-1}, 2^i$
- S^* leads to the best performance that can be achieved
  - $T(A, S^*) = O(\ell_A \log \ell_A)$
  - No strategy can do better under black-box assumption (no information other than when the algorithm stops is available)
  - A better strategy may exist if we relax black-box assumption

Improving Stochastic Synthesis with Restart Strategies

- Koenig et al., Adaptive Restarts for Stochastic Synthesis, PLDI 2021.

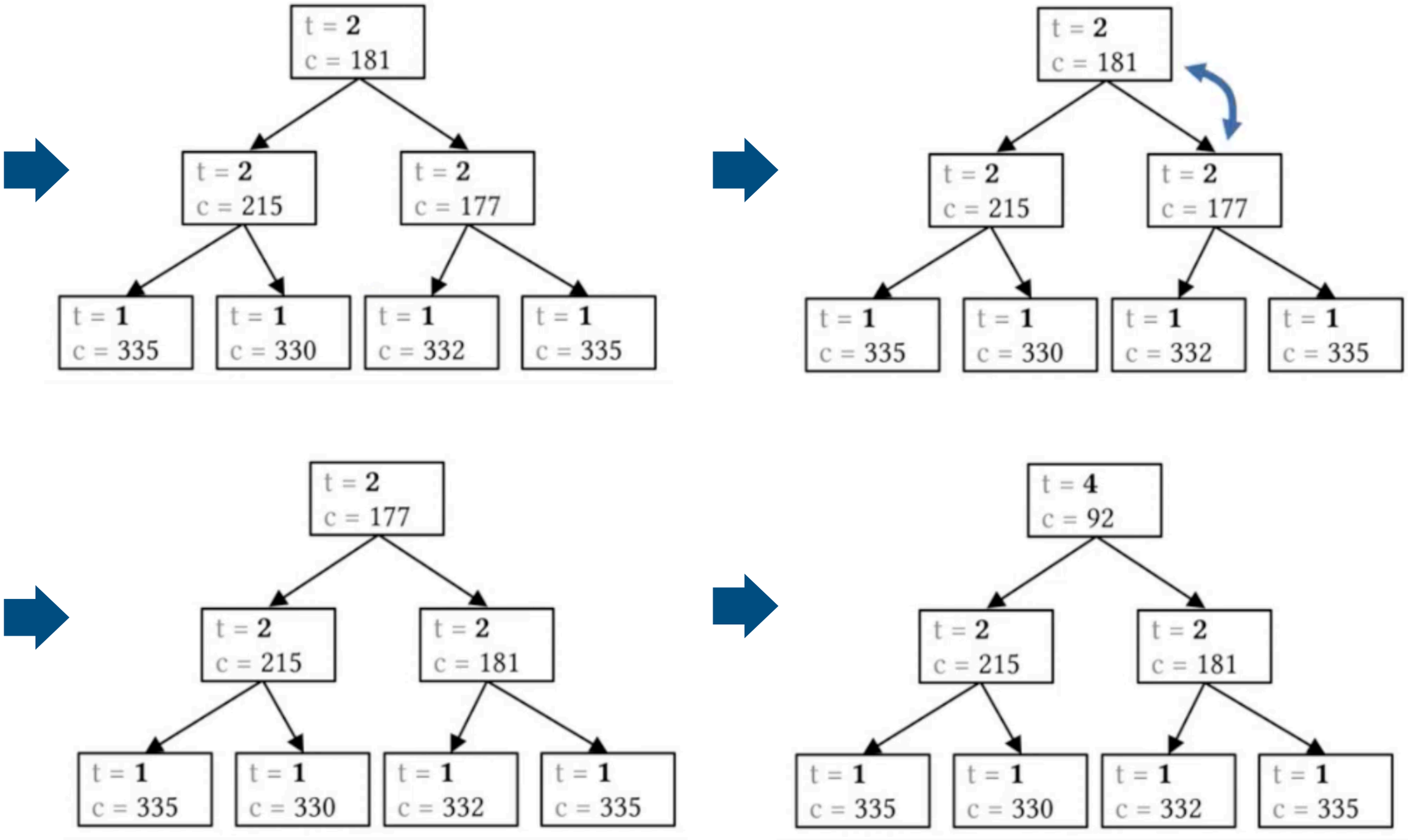
- Key idea: prioritize low cost searches

Adaptive Restart Algorithm



From Jason Koenig's presentation slides at PLDI'21

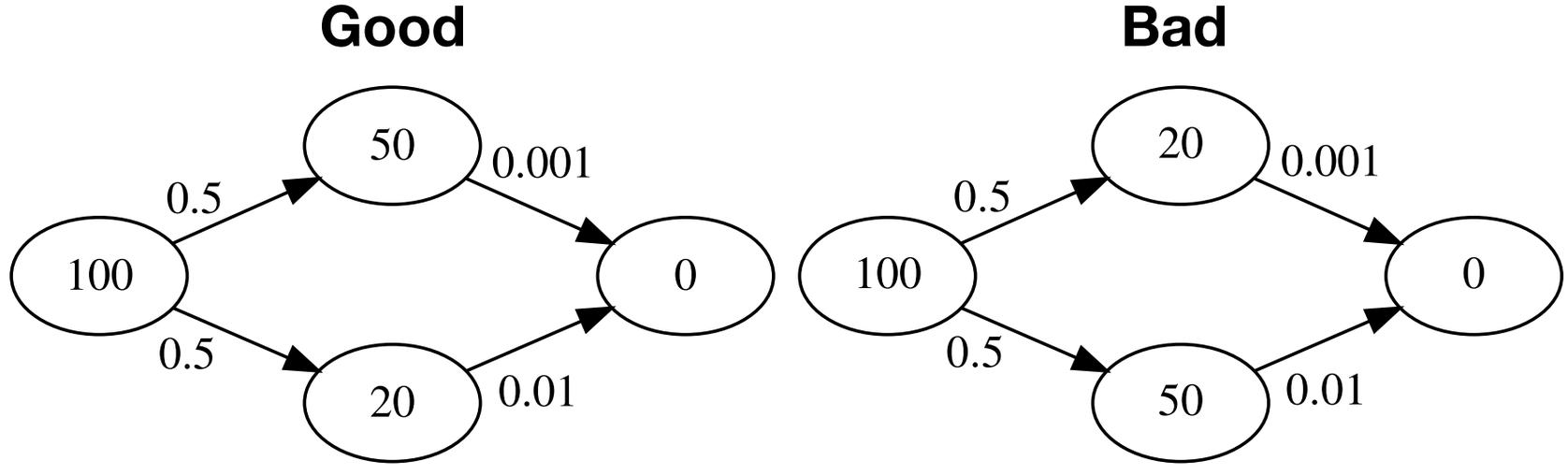
Adaptive Restart Algorithm



From Jason Koenig's presentation slides at PLDI'21

Adaptive Restart Algorithm

- Most time spent on lowest cost searches
- Best when cost accurately predicts time to finish



- 10x faster than the previous algorithm without restarts and 5.5x faster than the Luby restart strategy

From Jason Koenig's presentation slides at PLDI'21