CS5733 Program Synthesis

#11.Stochastic Search

Ashish Mishra, September 13, 2024

Recap: Representation-based search

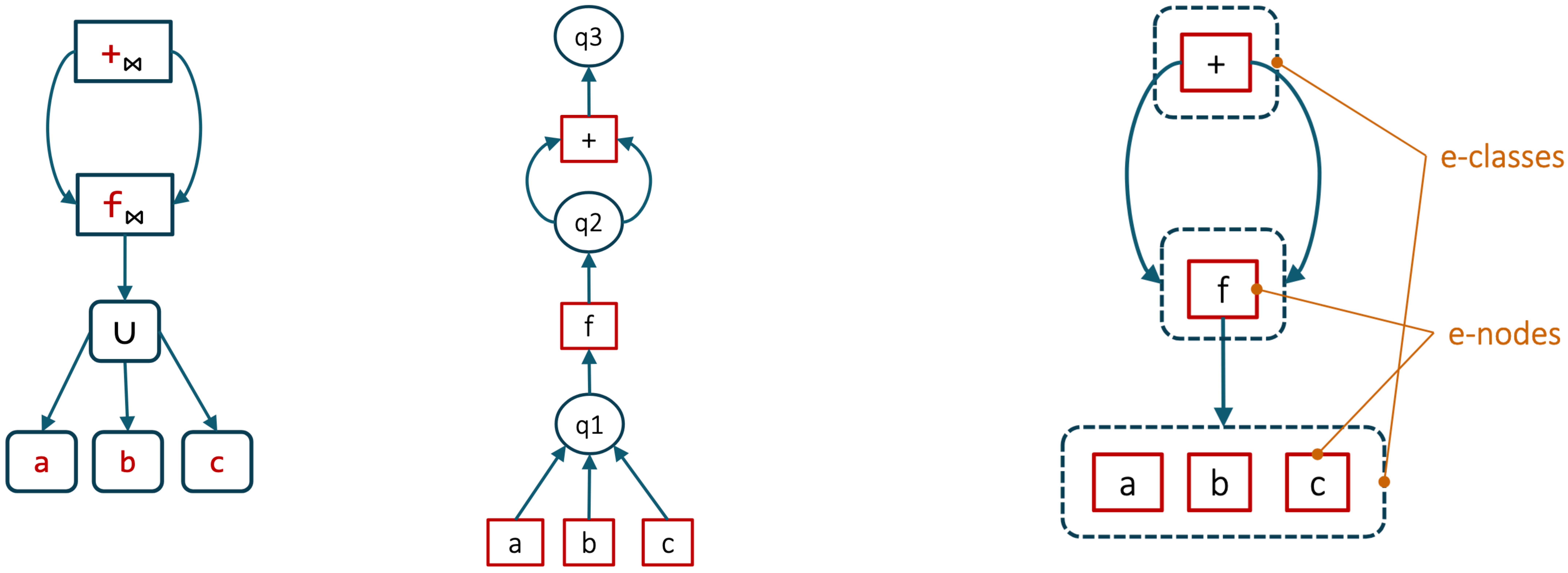
and/or graph

Version Space Algebras (VSA)	Finite Tree Automata (FTA)	Equivalence Graphs (e-grpahs)
ops: learn-1, intersect, extract **DSL:** efficiently invertible **similar to:** top-down prop, but can infer constants	**ops:** learn-1, intersect, extract **DSL:** efficiently enumerable **similar to:** bottom-up with OE, but can store all programs (and add examples incrementally)	**ops:** rewrite, extract **similar to:** term rewriting, but can store all programs

state: represents a set of observationally-equivalent programs

e-class: represents a set of programs equivalent up to rewrites

VSA vs FTA vs E-Graphs



Stochastic Search

The Synthesis Problem

Behavioral constraints = examples

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

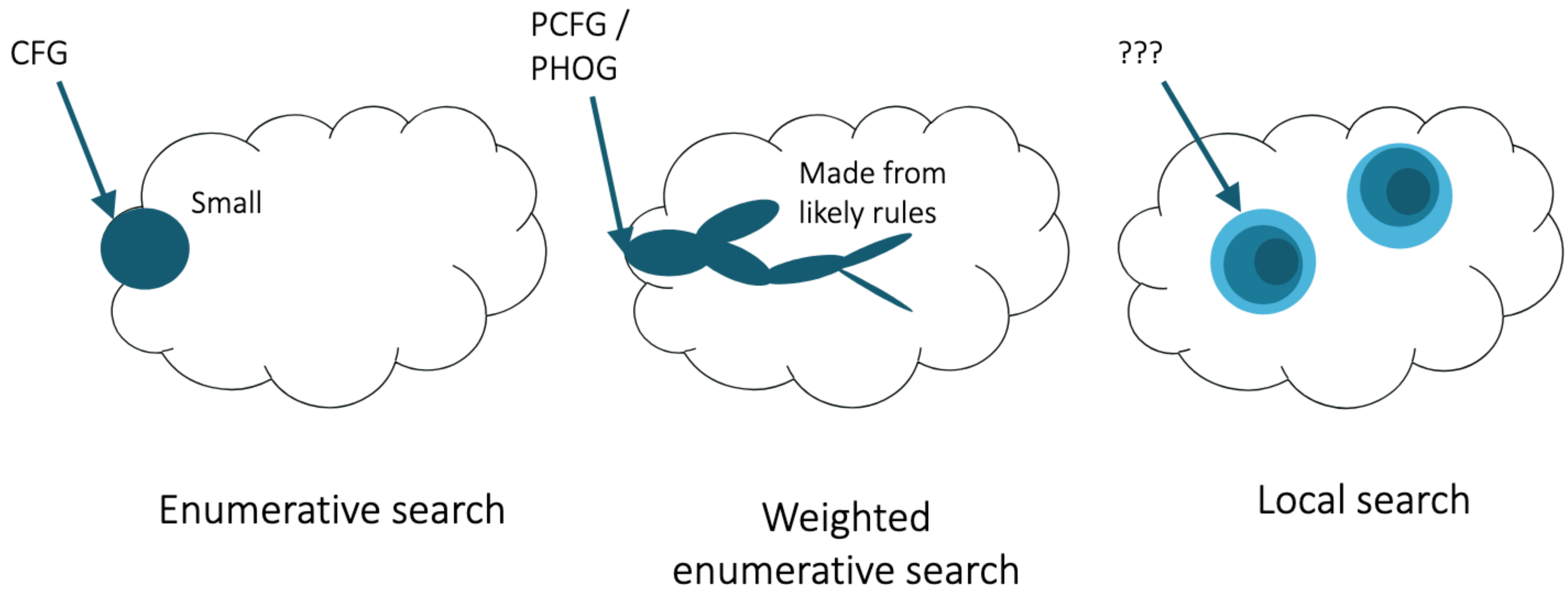
Structural constraints = grammar

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

Search strategy?

Enumerative
Representation-based
Stochastic
Constraint-based

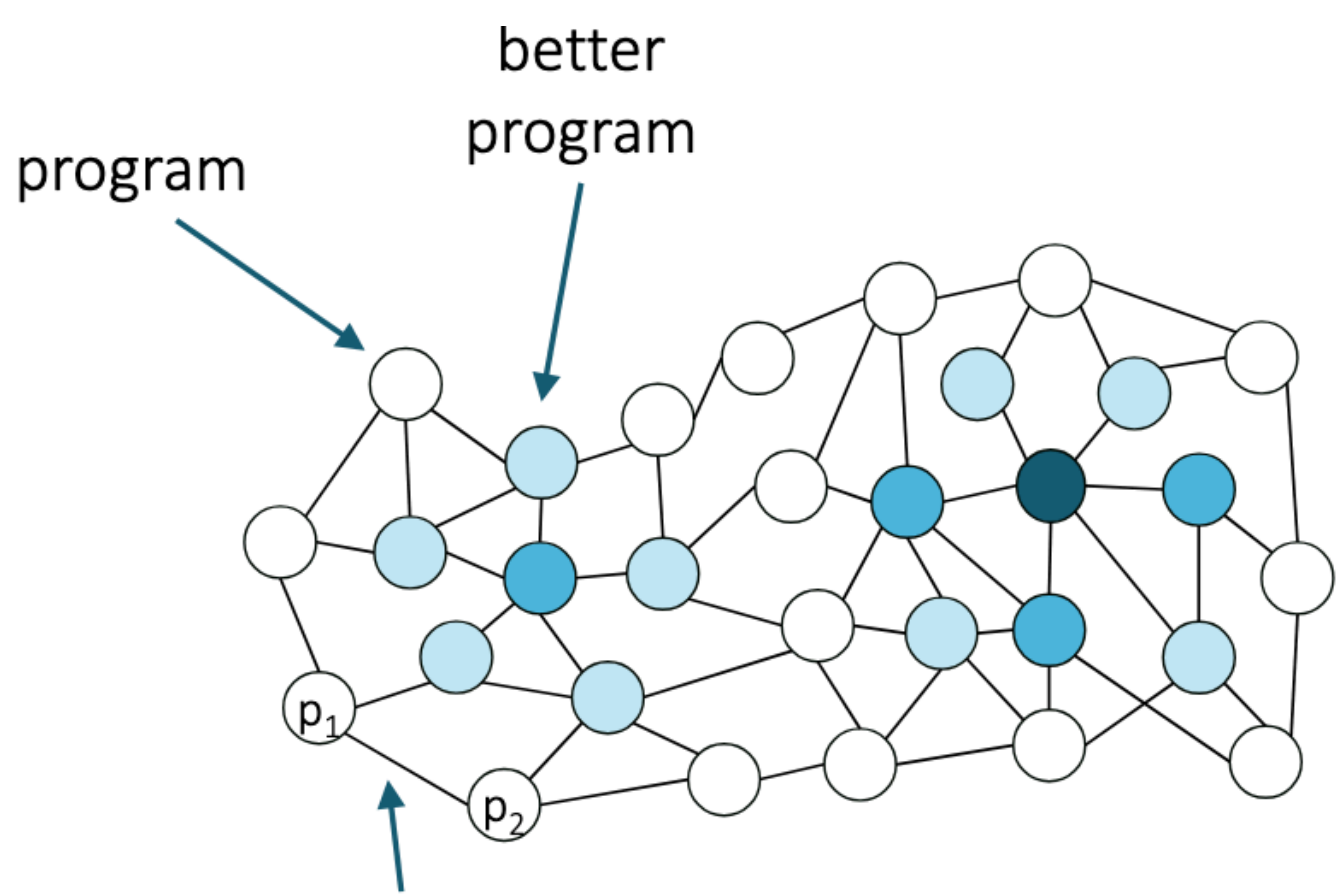
Search space



Naïve local search

To find the best program:

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



can generate p_2 from p_1
(and vice versa) via mutation

Will never get to ● from p_1!

We need a more advanced search! Stochastic search is one such appraoch.

Stochastic search in synthesis

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

Gissurarson, Applis, Panichella, van Deursen, Sands. *PropR: Property-Based Automatic Program Repair.* ICSE’22

Schkufza, Sharma, Aiken: *Stochastic superoptimization.* ASPLOS’13

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

Gissurarson, Applis, Panichella, van Deursen, Sands. *PropR: Property-Based Automatic Program Repair.* ICSE'22

Schkufza, Sharma, Aiken: *Stochastic superoptimization.* ASPLOS'13

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

Example:

[Schkufza, Sharma, Aiken ‘13]

Montgomery multiplication kernel from the OpenSSL big number library

```
.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
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
```

16 lines shorter and 1.6x faster

Uses a different assembly level algorithm than the original, something not possible with traditional compiler optimizations.

MCMC and Metropolis Search

Based on “The Markov Chain Monte Carlo Revolution”
Persi Diaconis

Markov Chains

MP : A probabilistic process.

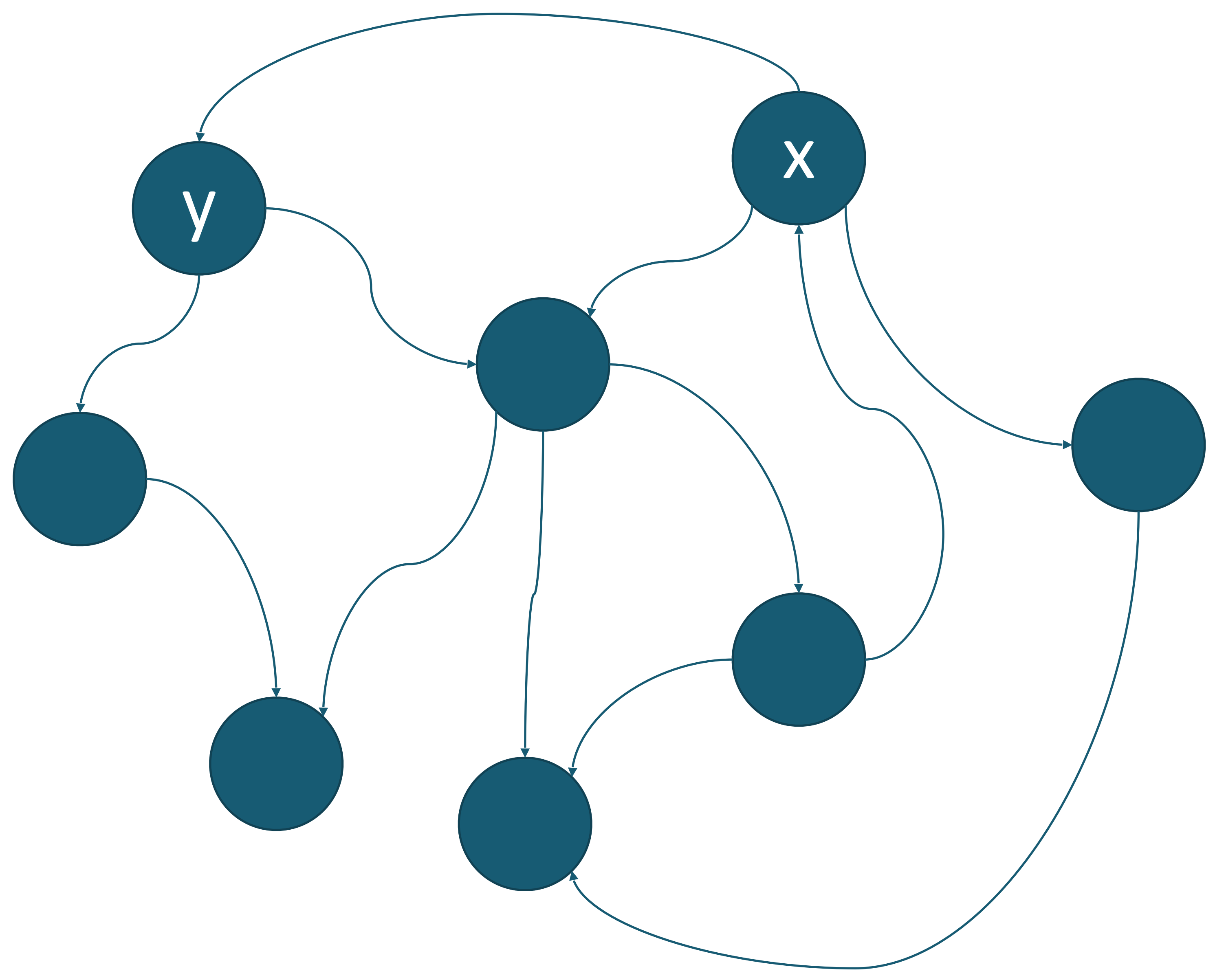
Probability of transitioning from state x to state y.

Let χ be a finite set

A Markov chain is defined by a matrix $K(x, y) : \chi \times \chi \to \mathbb{R}$

- $K(x, y) \geq 0$
- $\sum_y K(x, y) = 1$
- Probability of transitioning from x to y

$K(x, y)$



Markov Chains

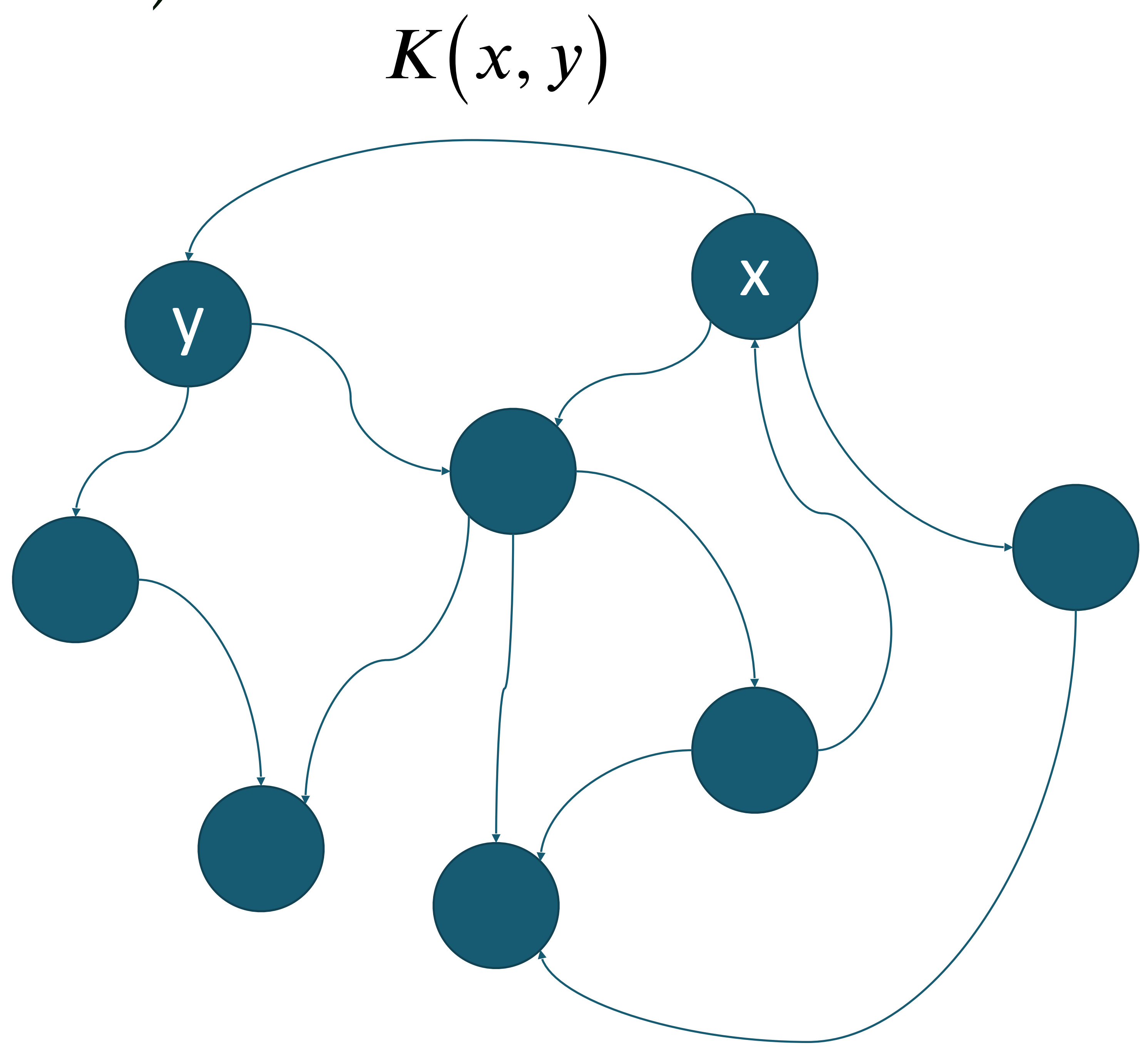
Let χ be a finite set

A Markov chain is defined by a matrix $K(x, y): \chi \times \chi \to \mathbb{R}$

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

Probability of a series $X_0,\ X_1,\ X_2 \ldots$

- $P(X_1 = y \mid X_0 = x) = K(x, y)$



Markov Chains

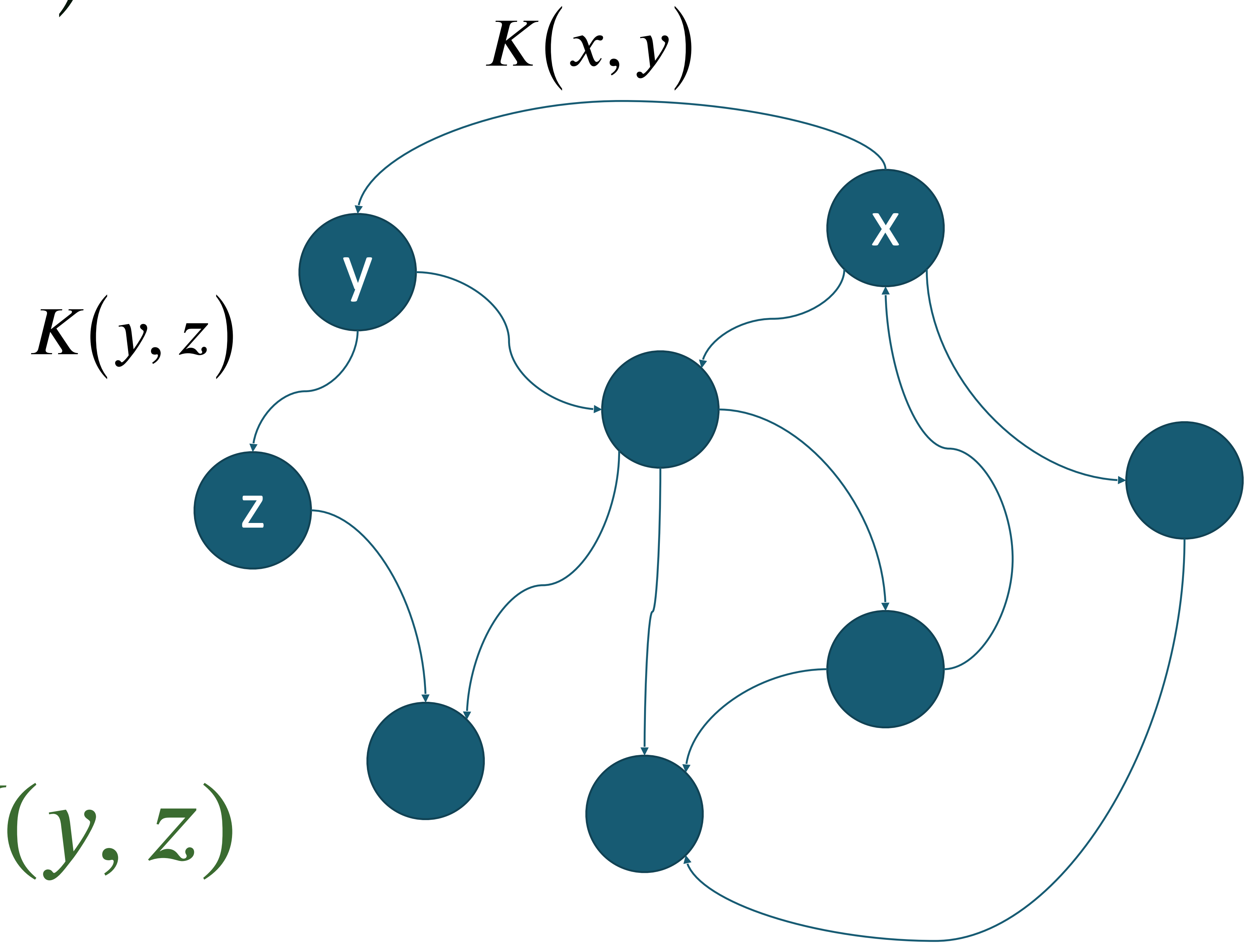
Let χ be a finite set

A Markov chain is defined by a matrix $K(x, y): \chi \times \chi \to \mathbb{R}$

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

Probability of a series $X_0, X_1, X_2 \ldots$

- $P(X_1 = y \mid X_0 = x) = K(x, y)$
- $P(X_1 = y, X_2 = z \mid X_0 = x) = K(x, y)K(y, z)$



Markov Chains

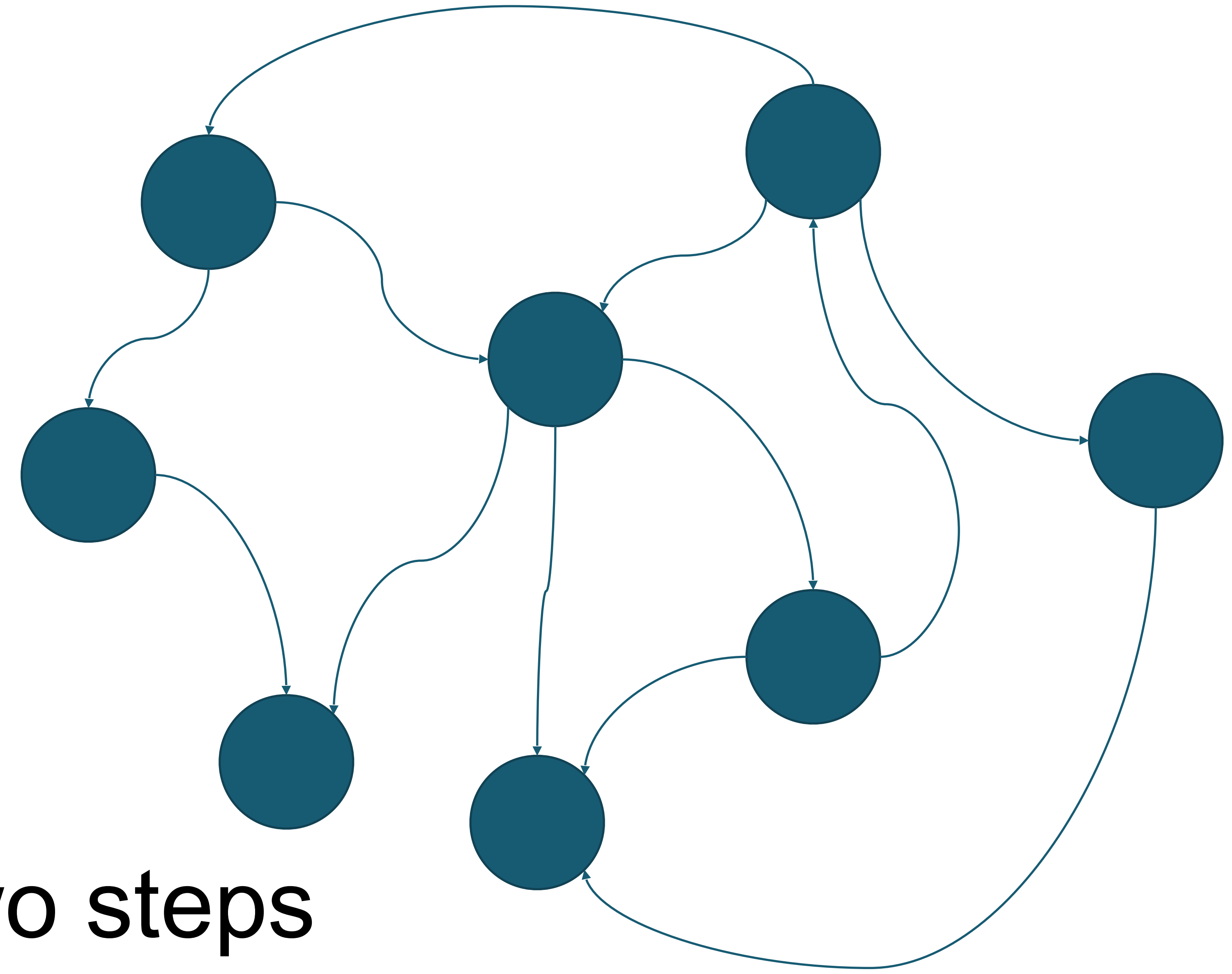
Let χ be a finite set

A Markov chain is defined by a matrix $K(x, y) : \chi \times \chi \to \mathbb{R}$

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

Probability of a series $X_0,\ X_1,\ X_2 \ldots$

- $P(X_1 = y \mid X_0 = x) = K(x, y)$
- $P(X_1 = y,\ X_2 = z \mid X_0 = x) = K(x, y)K(y, z)$
- $P(X_2 = z \mid X_0 = x) = \sum_y K(x, y)K(y, z)$
  - This is matrix multiplication!



K: prob of transitioning from x to y in one step, K^2 : in two steps

K^n : in n steps

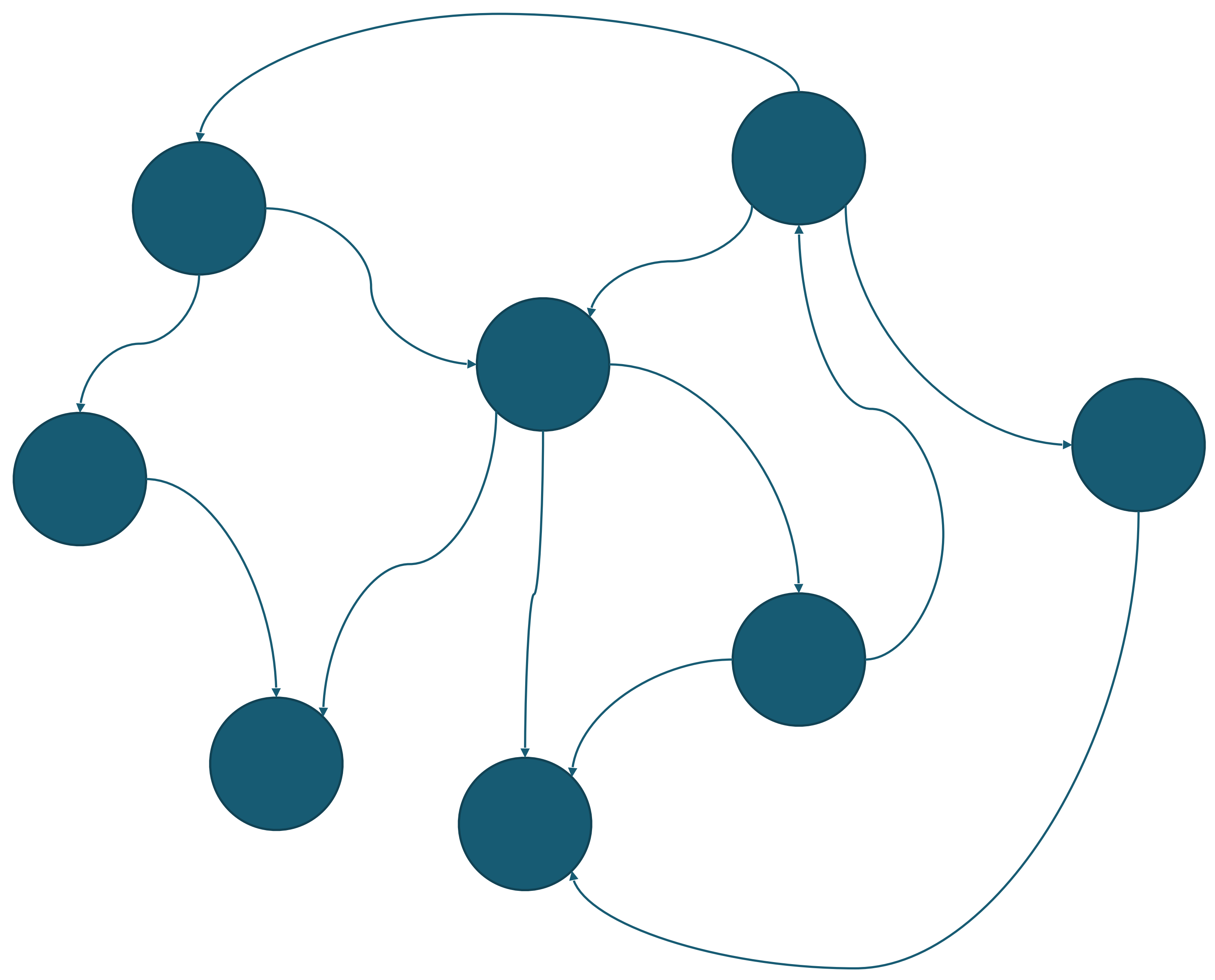
Stationary distribution

What is the probability $\pi(x)$ of being in a node x at some arbitrary step?

- $\pi(x) > 0 \; and \; \sum_x \pi(x) = 1$
- $\pi(y) = \sum_x \pi(x) K(x, y)$
  - i.e. $\pi = \pi K$

pick x from π and take a step from K(x, y); the chance of being at y is π(y)

SO stationary distribution is an eigenVector of K with eigenValue 1.



Fundamental theorem of (finite) Markov chains

If there is an n_0 s.t. $\forall x, y . \quad n > n_0 \Rightarrow K^n\left(x, y\right) \geq 0$

- i.e. the matrix is connected.
- the matrix must also be aperiodic, e.g. rules out processes like $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$

Then K has a unique stationary distribution, π

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

- The n'th step of a run starting at x has probability close to $\pi(y)$ of being at y if n is large.
- we can compute the stationary distribution by starting at some state and then running the markov process for a long time.
  - Where we start doesn't matter

MCMC Based synthesis

Approach:

- Let χ be the space of programs
- Engineer a $K(x, y)$ such that $\pi(x)$ is high for "good programs" and low for "bad programs"
- Pick a random start state x_0
- Simulate the markov process for n steps for some large n.
- By the fundamental theorem, the probability of x_n is a good program will be higher than the probability that it is a bad program

Key step: Engineer K that has desired property for $\pi(x)$

Metropolis algorithm with symmetric Proposal distribution

- Start with a markov matrix $J(x, y)$ with $J\left(x, y\right) > 0 \leftrightarrow J\left(y, x\right) > 0$ and J(x, y) = J(y, x)
- Initialization: Chose an arbitrary x to be the first observation in the sample based and initialize J to satisfy the above property.
- For each iteration say t.
  - Propose a candidate y for the next sample by picking from J(x_t, y).
  - Calculate the acceptance ratio A = $\pi(y)/\pi(x_t)$, which is used to decide whether to accept or reject the candidate.
  - Generate a uniform random number u $\in [0,1]$.
  - If u <= A then accept y and set x_{t+1} <- y
  - If u > A then reject the candidate y and set x_{t+1} <- x

Metropolis algorithm : Non symmetric case

- Start with a markov matrix $J(x, y)$ with $J\left(x, y\right) > 0 \leftrightarrow J\left(y, x\right) > 0$
- For each iteration say t.
  - Propose a candidate y for the next sample by picking from J(x_t, y).
  - Calculate the acceptance ratio A = $\frac{\pi(y)}{J(x, y)} / \frac{\pi(x_t)}{J(y, x)}$, which is used to decide whether to accept or reject the candidate.
  - If A >= 1 then accept y and set x_{t+1} <- y
  - If 0 < A < 1 then
    - accept candidate y and set x_{t+1} <- y with probability A
    - reject canditate y and set x_{t+1} <- x with probability (1- A)

Key issues: Applying MH to Program Synthesis:

- Define a Program Space
- Define a desired stationary distribution π.
  - Need good estimates of π
  - Need a good proposal distribution J
  - Tempting to use naive uniform distribution as J
    - This does not work well as search.
  - Effective π should allow us to judge if program is getting closer to be correct.
    - J must give priority to programs with similar behaviors to use information learnt from the search.

Many recent synthesis applications

Influential work by Schkufza, Sharma Aiken.

- Focus on program optimization

Wide variety of applications in other areas

- Probabilistic programming
- Cognitive Science
- etc.

Stochastic Superoptimization

```
# gcc -O3

.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
```

```
# STOKE

.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
```

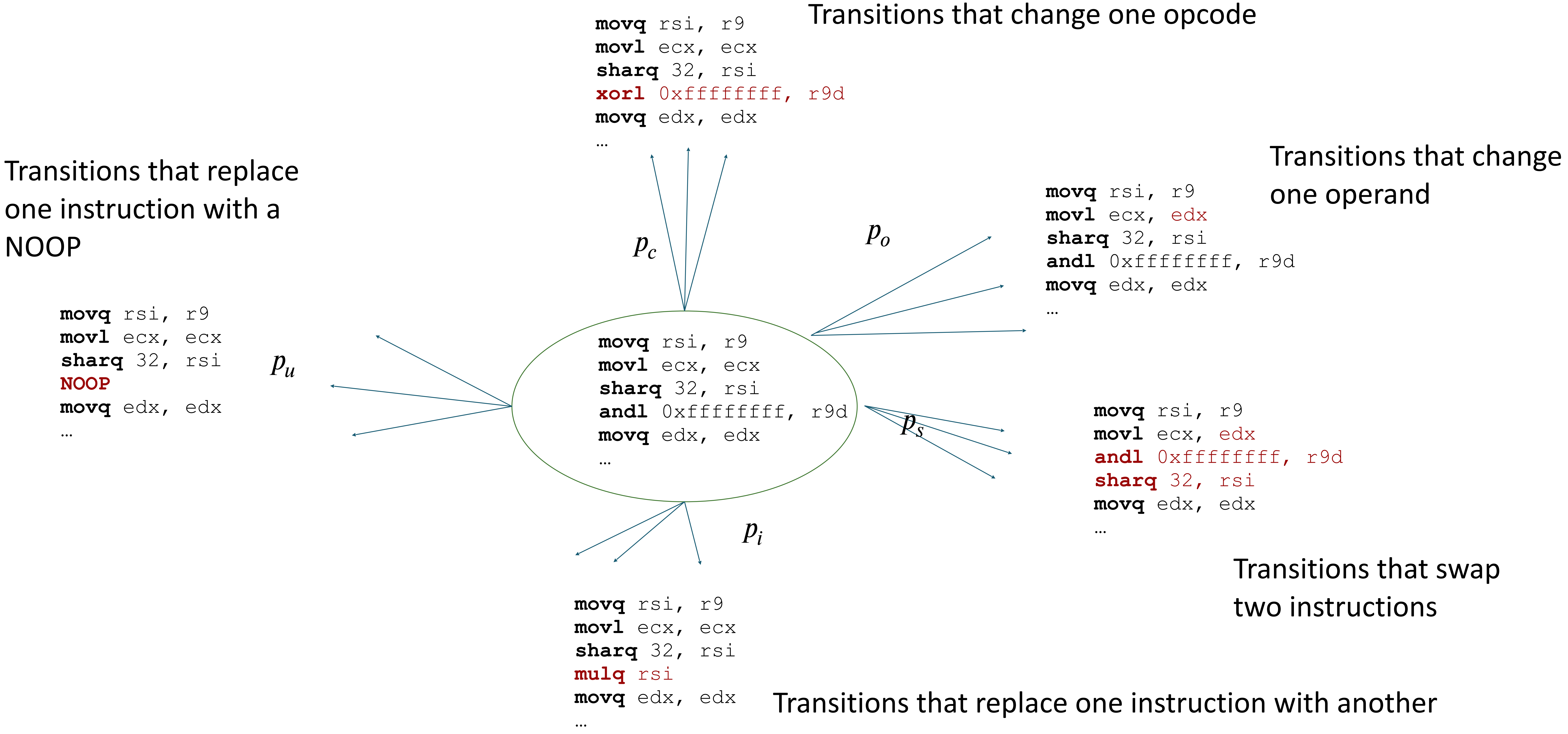
Goal: Synthesize equivalent assembly program that is significantly more efficient.

Example from Schkufza, Sharma and Aiken from ASPLOS 13.

The program space

Sequences of assembly instructions of bounded length

The proposal Distribution J (T R)



The stationary distribution

$$\pi\left(\mathcal{T}\right) = \frac{1}{Z}\, e^{-\beta(eq\left(\mathcal{R},\ \mathcal{T}\right)+perf(\mathcal{R},\ \mathcal{T}))}$$

eq (R , T) correctness component

perf (R , T) performance component

Cost function

$$c(\mathcal{R}; \mathcal{T}) = \text{eq}(\mathcal{R}; \mathcal{T}) + \text{perf}(\mathcal{R}; \mathcal{T})$$

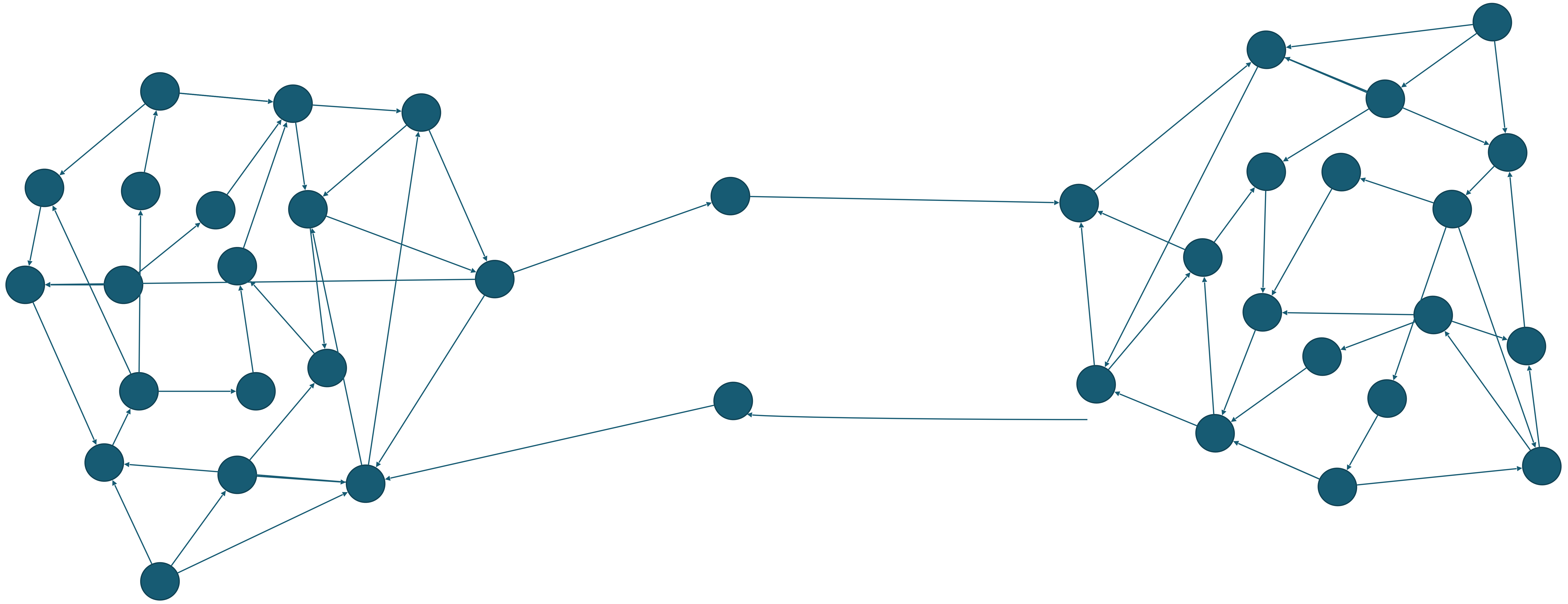
- eq: Calculated by running the candidate program R on the test inputs
and computing a distance between its output and the output of the original program
- perf: computed by evaluating the candidate program through a
performance model that assigns a cost to each instruction.

Improvements

search is conducted in two stages

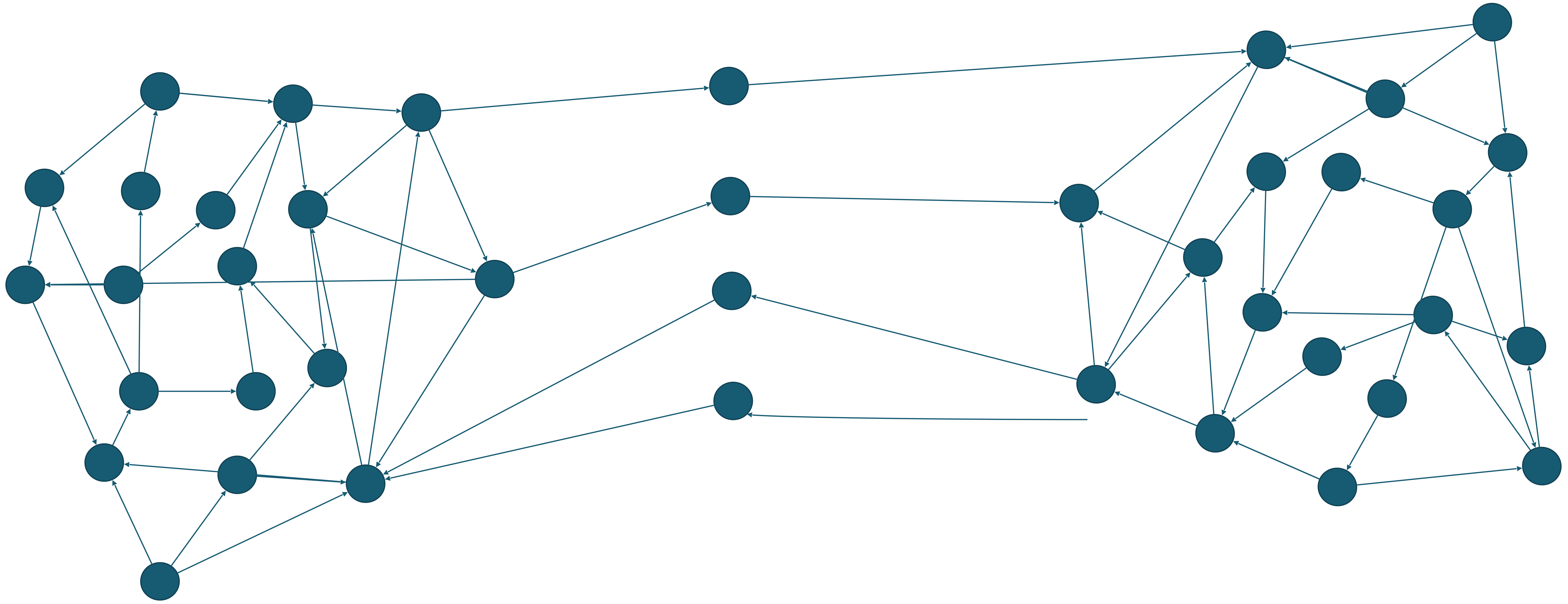
- In the first stage, the performance component is completely ignored, allowing the search to discover correct programs that are very different from the initial one.
- A set of these correct programs discovered in the first phase are used as starting points for a second phase search that includes the performance term.

Almost disjoint clusters



Starting in one cluster, the probability of transitioning to the other is extremely low

Almost disjoint clusters



Introducing more paths, or increasing the probability for the existing ones helps us converge faster to a distribution that is representative of both clusters

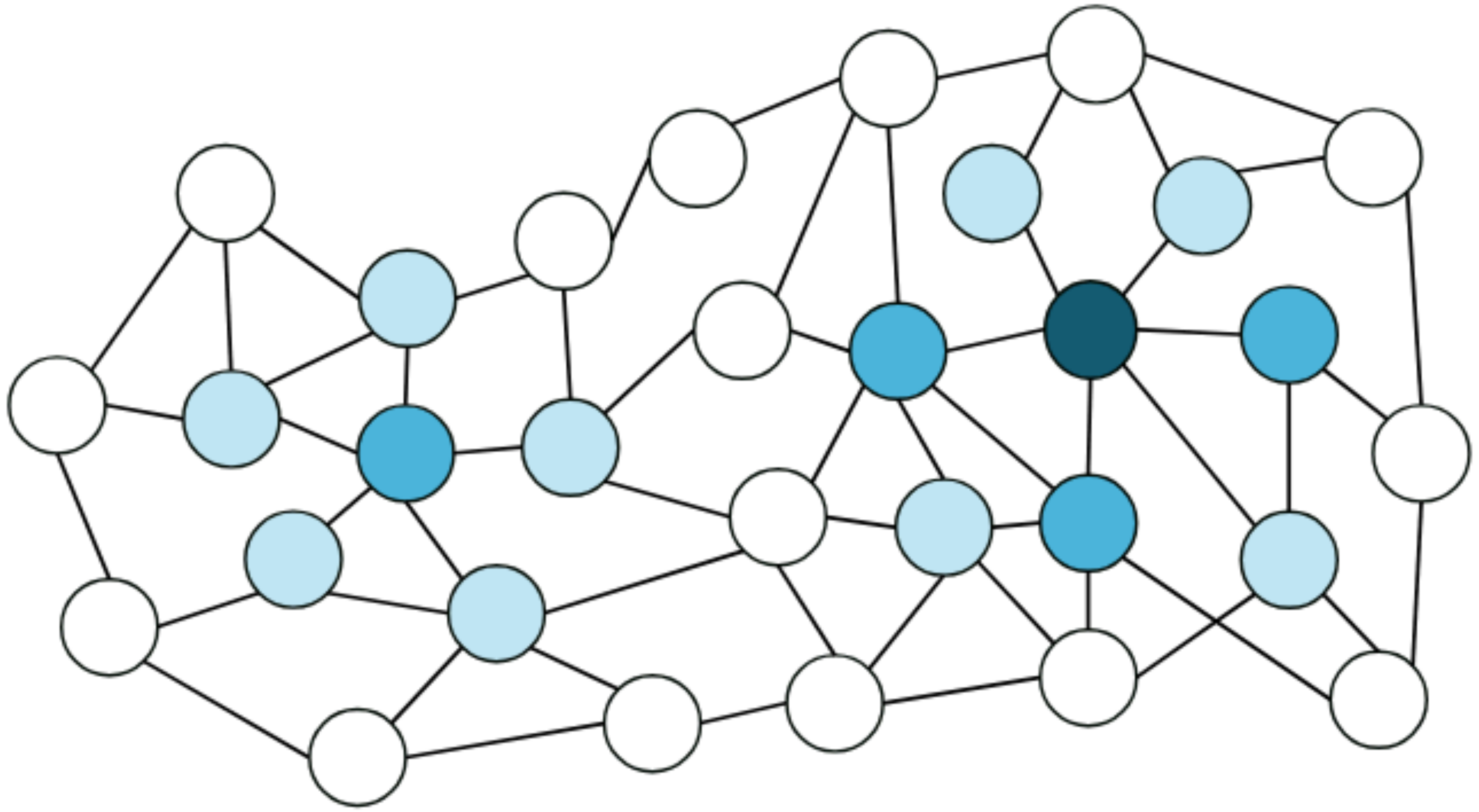
MCMC sampling

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**: $A(p \rightarrow p') = 1$
- otherswise: $A(p \rightarrow p')$ decreases with difference in cost between **p’** and **p**



Stochastic search in synthesis

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

Gissurarson, Applis, Panichella, van Deursen, Sands. *PropR: Property-Based Automatic Program Repair.* ICSE'22

- Similar but for program repair, uses genetic programming

Schkufza, Sharma, Aiken: *Stochastic superoptimization.* ASPLOS'13

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

- Samples from a grammar with bias towards partial solutions

Next: Module II

- Synthesizing Complex Programs
- Rich Specifications:
  - Reference implementation
  - Assertions
  - Pre- and post-condition
  - Fancy types
- Richer Program Space:
  - Recursive programs
  - Imperative programs:
    - Pointer manipulating programs.
    - Programs with effectful libraries.

Logistics

- Milestone 1 deadline: Monday Sept 16.
- Please reach out to me through email or google classroom.
- I will post this weeks reading by EOD.