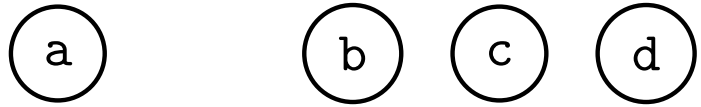
Optimal Shuffle Code with Permutation Instructions

Sebastian Buchwald, Manuel Mohr, Ignaz Rutter

Chair for Programming Paradigms & Chair for Algorithmics, Karlsruhe Institute of Technology (KIT)

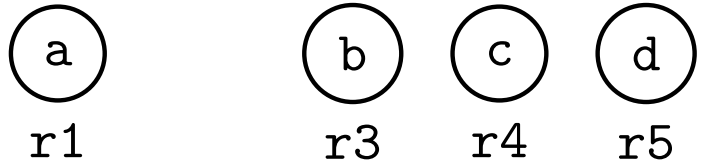
Register Allocation and Shuffle Code




```
a = 10;
b = 20;
c = 30;
d = 40;
...
print(a, a, c, b, d);
```

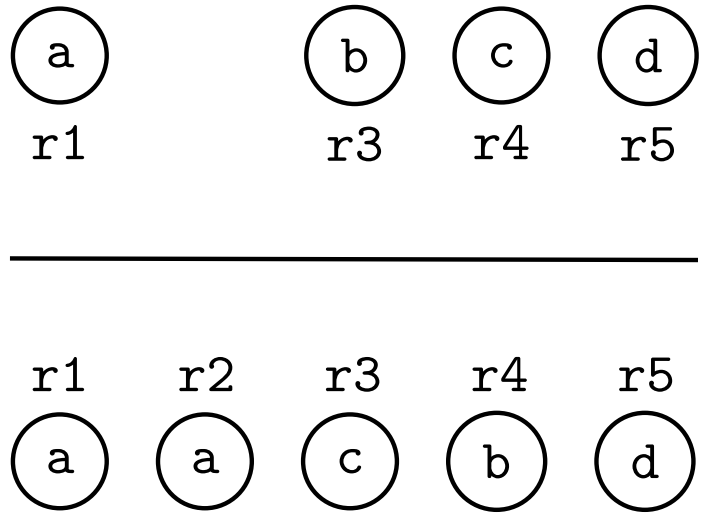
Register Allocation and Shuffle Code

```
a = 10;
b = 20;
c = 30;
d = 40;
...
print(a, a, c, b, d);
```



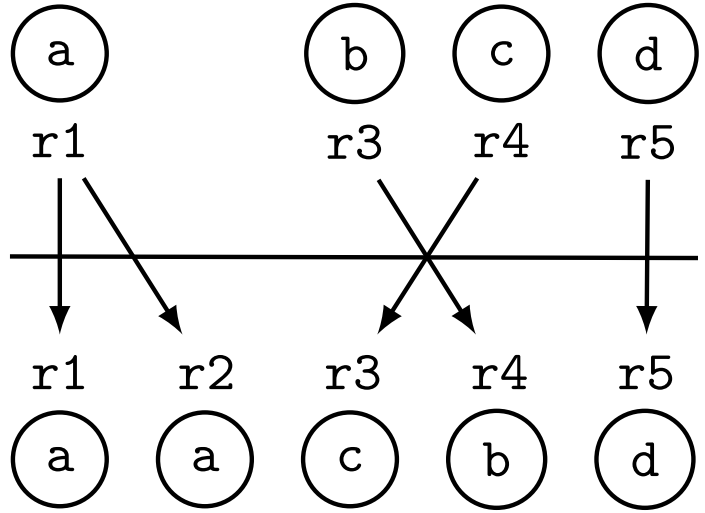
Register Allocation and Shuffle Code

```
a = 10;
b = 20;
c = 30;
d = 40;
...
print(a, a, c, b, d);
```



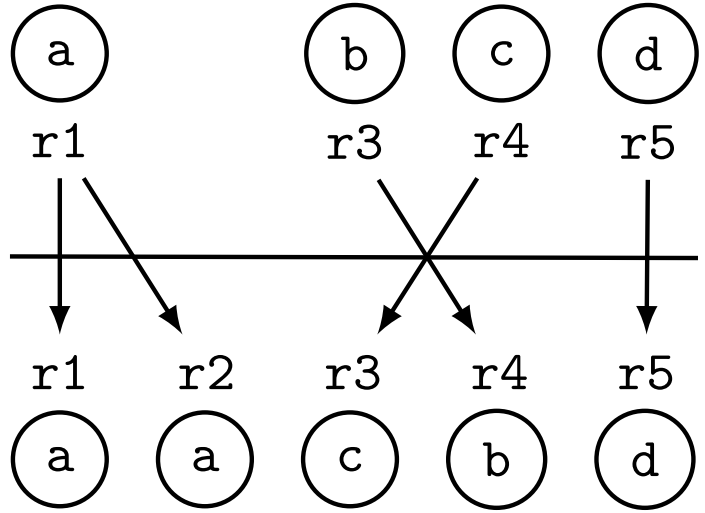
Register Allocation and Shuffle Code

```
a = 10;
b = 20;
c = 30;
d = 40;
...
print(a, a, c, b, d);
```

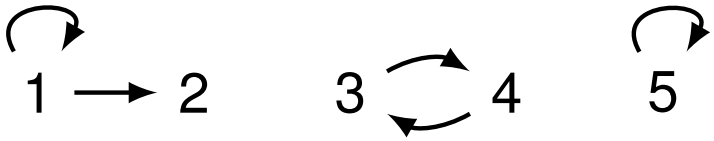


Register Allocation and Shuffle Code

```
a = 10;
b = 20;
c = 30;
d = 40;
...
print(a, a, c, b, d);
```

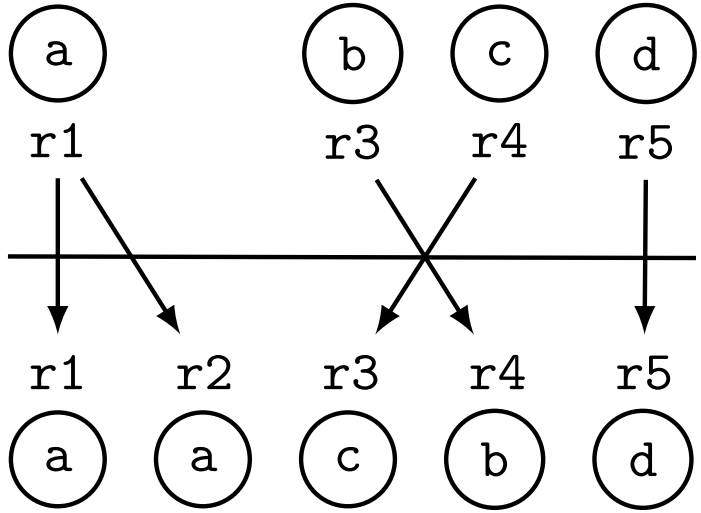


Register Transfer Graph (RTG)

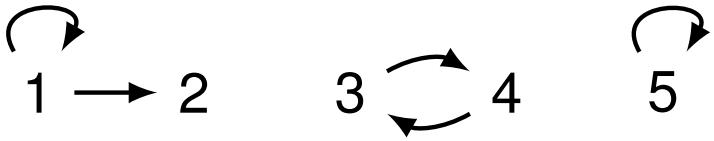


Register Allocation and Shuffle Code

```
a = 10;
b = 20;
c = 30;
d = 40;
...
print(a, a, c, b, d);
```



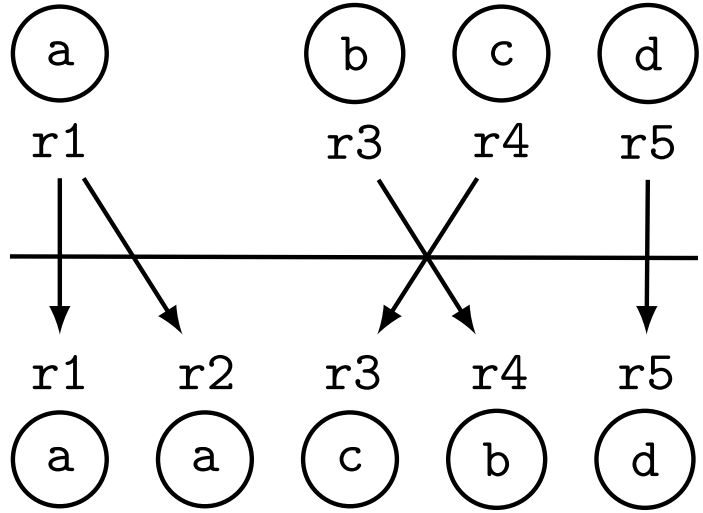
Register Transfer Graph (RTG)



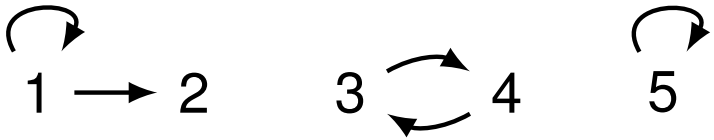

```
copy r1, r2
swap r3, r4
```

Register Allocation and Shuffle Code

```
a = 10;
b = 20;
c = 30;
d = 40;
...
print(a, a, c, b, d);
```



Register Transfer Graph (RTG)




```
copy r1, r2
swap r3, r4
```

Shuffle Code Generation Problem

Find a shortest shuffle code that **implements** a given RTG.

Permutation Instructions

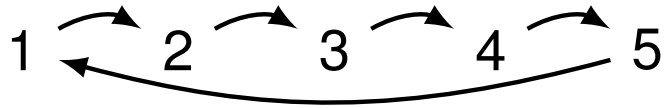
- For large RTGs, implementations can get lengthy

$\Rightarrow$ Proposed more powerful permutation instructions [Mohr et al. 2013]

Permutation Instructions

- For large RTGs, implementations can get lengthy

⇒ Proposed more powerful permutation instructions [Mohr et al. 2013]



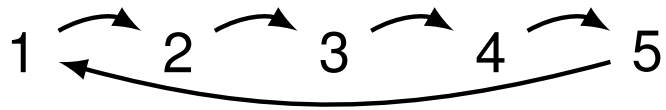

```
permi5 r1, r2, r3, r4, r5
```


```
permi5 r1, r2
```

Permutation Instructions

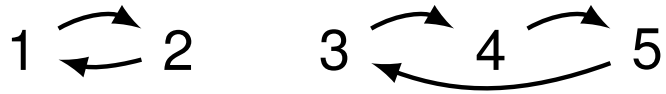
- For large RTGs, implementations can get lengthy

⇒ Proposed more powerful permutation instructions [Mohr et al. 2013]




```
permi5 r1, r2, r3, r4, r5
```


```
permi5 r1, r2
```



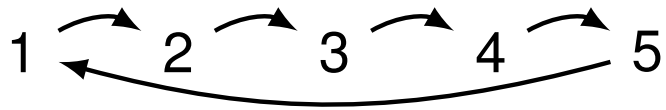

```
permi23 r1, r2, r3, r4, r5
```


```
permi23 r1, r2, r3, r4
```

Permutation Instructions

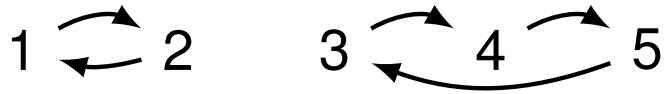
- For large RTGs, implementations can get lengthy

⇒ Proposed more powerful permutation instructions [Mohr et al. 2013]




```
permi5 r1, r2, r3, r4, r5
```

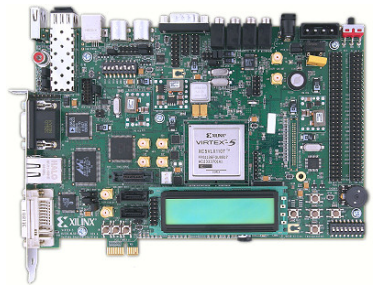

```
permi5 r1, r2
```




```
permi23 r1, r2, r3, r4, r5
```


```
permi23 r1, r2, r3, r4
```

- Exists as FPGA-based prototype processor with modified compiler [Mohr et al. 2013]
  - Improves performance in practice
  - Uses greedy heuristic



Shuffle Code with Permutation Instructions

Karlsruhe Institute of Technology

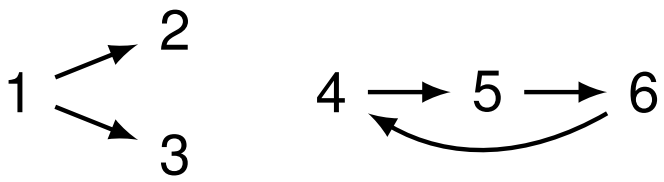
SCGP with permutation instructions

Given RTG, find a shortest shuffle code using `permi5`, `permi23` and `copy`.

Shuffle Code with Permutation Instructions

SCGP with permutation instructions

Given RTG, find a shortest shuffle code using `permi5`, `permi23` and `copy`.



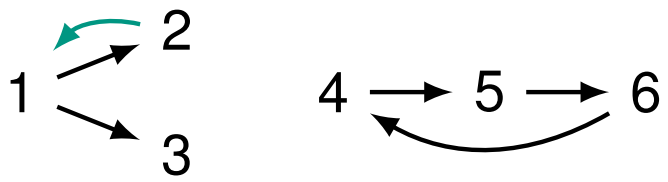
Naive implementation:

```
copy r1, r2
copy r1, r3
permi5 r4, r5, r6
```

Shuffle Code with Permutation Instructions

SCGP with permutation instructions

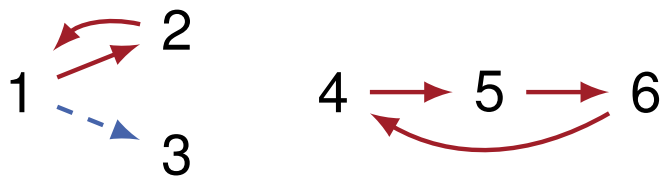
Given RTG, find a shortest shuffle code using `permi5`, `permi23` and `copy`.



Shuffle Code with Permutation Instructions

SCGP with permutation instructions

Given RTG, find a shortest shuffle code using `permi5`, `permi23` and `copy`.



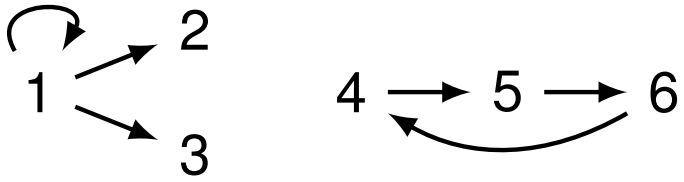
Optimal implementation:

```
permi23 r1, r2, r4, r5, r6
copy r2, r3
```

Shuffle Code with Permutation Instructions

SCGP with permutation instructions

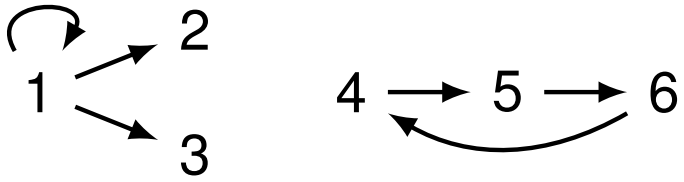
Given RTG, find a shortest shuffle code using `permi5`, `permi23` and `copy`.



Shuffle Code with Permutation Instructions

SCGP with permutation instructions

Given RTG, find a shortest shuffle code using `permi5`, `permi23` and `copy`.



Optimal implementation:

```
copy r1, r2
copy r1, r3
permi5 r4, r5, r6
```

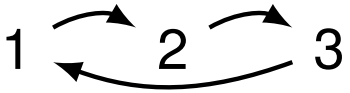
Shuffle Code Generation as a Graph Problem

Shuffle Code Generation as a Graph Problem

Effect of a **permutation**:

$$\pi \bullet (V, E) = (V, \{(\pi(u), v) \mid (u, v) \in E\})$$

$$[(1\ 2) \circ (2\ 3)] \bullet$$



$\xrightarrow{?}$

Shuffle Code Generation as a Graph Problem

Effect of a **permutation**:

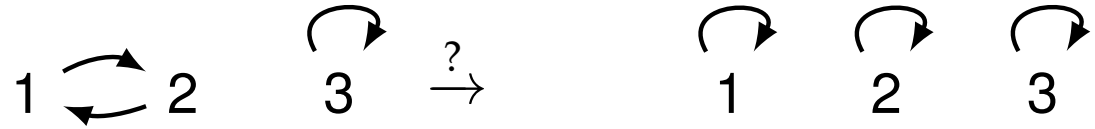
$$\pi \bullet (V, E) = (V, \{(\pi(u), v) \mid (u, v) \in E\})$$

$$[(1\ 2) \circ (2\ 3)] \bullet \quad 1 \rightleftarrows 2 \rightleftarrows 3 \quad \overset{?}{\rightarrow} \quad \circlearrowright 1 \quad \circlearrowright 2 \quad \circlearrowright 3$$

Shuffle Code Generation as a Graph Problem

Effect of a **permutation**:

$$\pi \bullet (V, E) = (V, \{(\pi(u), v) \mid (u, v) \in E\})$$

$$[(1\ 2)] \bullet$$ 

Shuffle Code Generation as a Graph Problem

Effect of a **permutation**:

$$\pi \bullet (V, E) = (V, \{(\pi(u), v) \mid (u, v) \in E\})$$

$$[\ \ ] \bullet \circlearrowright 1 \quad \circlearrowright 2 \quad \circlearrowright 3 = \circlearrowright 1 \quad \circlearrowright 2 \quad \circlearrowright 3$$

Shuffle Code Generation as a Graph Problem

Effect of a **permutation**:

$$\pi \bullet (V, E) = (V, \{(\pi(u), v) \mid (u, v) \in E\})$$

$$[\ 2\ 3\ 1\ ] \bullet \circlearrowright_1\ \circlearrowright_2\ \circlearrowright_3 = \circlearrowright_2\ \circlearrowright_3\ \circlearrowright_1$$

- Defines group action of permutations on RTGs
- For permutation RTGs (PRTGs): make given PRTG trivial

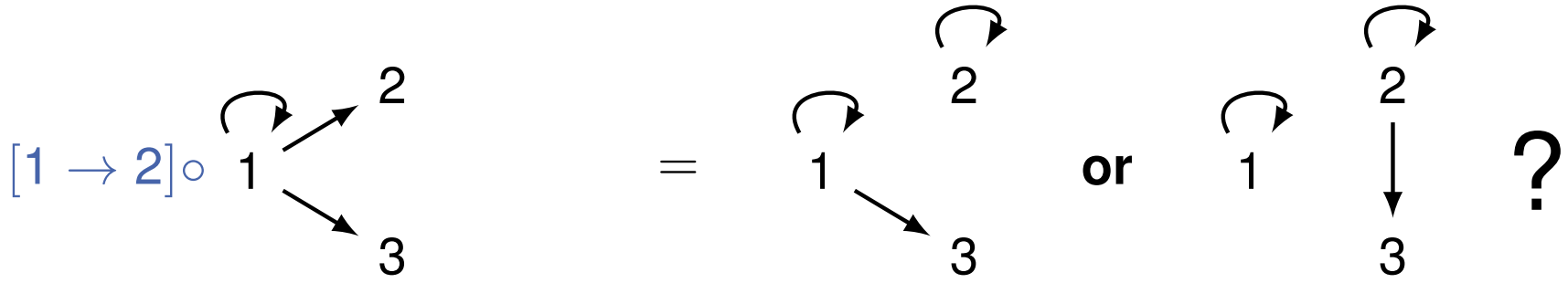
Shuffle Code Generation as a Graph Problem

Effect of a **permutation**:

$$\pi \bullet (V, E) = (V, \{(\pi(u), v) \mid (u, v) \in E\})$$

- Defines group action of permutations on RTGs
- For permutation RTGs (PRTGs): make given PRTG trivial

Effect of a **copy**:



- Copies not expressible in RTGs

Observations on Shuffle Code

Observations on Shuffle Code

Copy $a \to b$ followed by transposition $\tau = (c\ d)$

rewrite to

Transposition $(c\ d)$ followed by copy $\tau(a) \to \tau(b)$

Observations on Shuffle Code

Copy $a \to b$ followed by transposition $\tau = (c\ d)$

rewrite to

Transposition $(c\ d)$ followed by copy $\tau(a) \to \tau(b)$

Observations on Shuffle Code

Copy $a \to b$ followed by transposition $\tau = (c\ d)$

rewrite to

Transposition $(c\ d)$ followed by copy $\tau(a) \to \tau(b)$

Observations on Shuffle Code

Copy $a \to b$ followed by transposition $\tau = (c\ d)$

rewrite to

Transposition $(c\ d)$ followed by copy $\tau(a) \to \tau(b)$

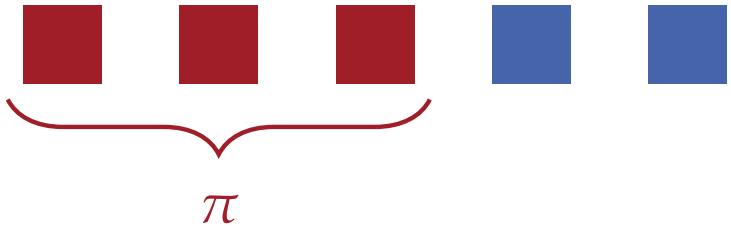
Observations on Shuffle Code

Copy $a \to b$ followed by transposition $\tau = (c\ d)$

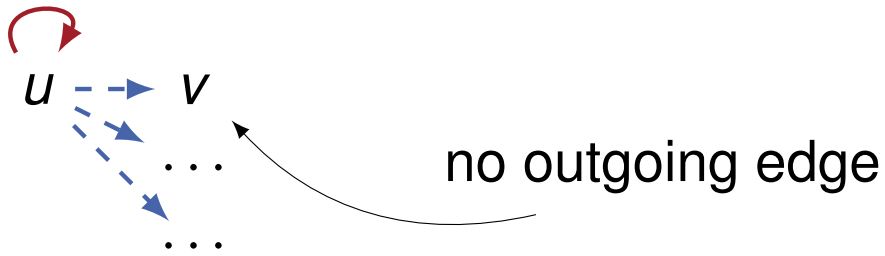
rewrite to

Transposition $(c\ d)$ followed by copy $\tau(a) \to \tau(b)$

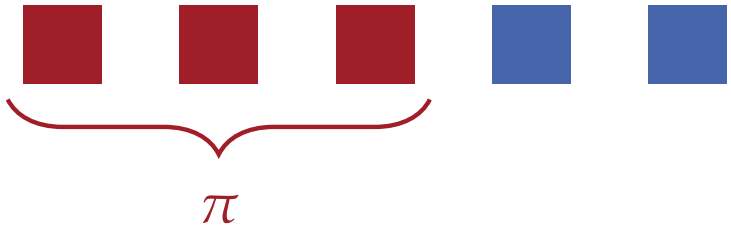
Observations on Shuffle Code



Can transform shuffle code such that in πG, for every copy $u \to v$:

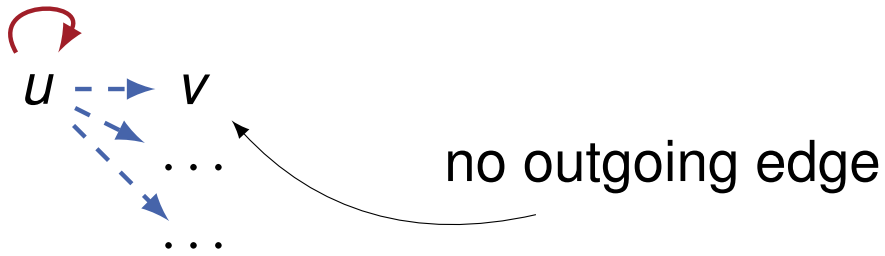


Observations on Shuffle Code

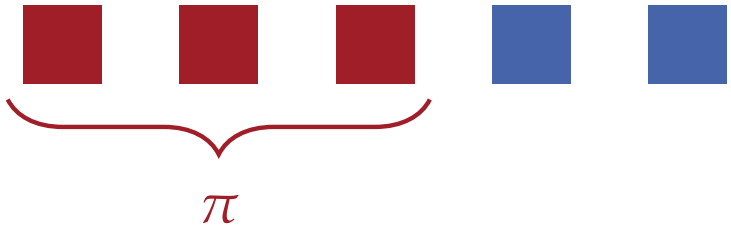


Normalized

Can transform shuffle code such that in πG, for every copy $u \to v$:

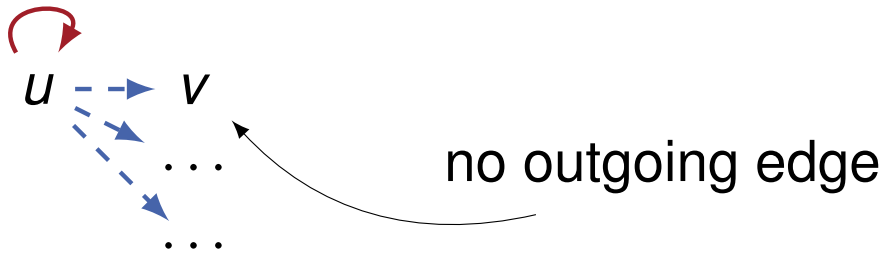


Observations on Shuffle Code



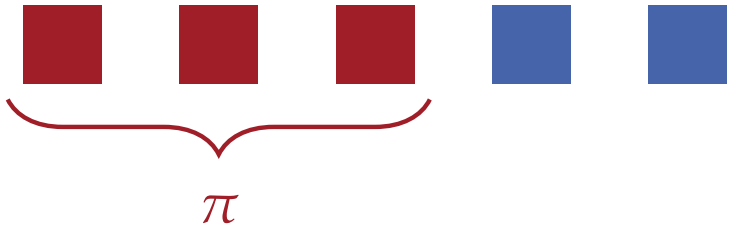
Normalized

Can transform shuffle code such that in πG, for every copy $u \to v$:



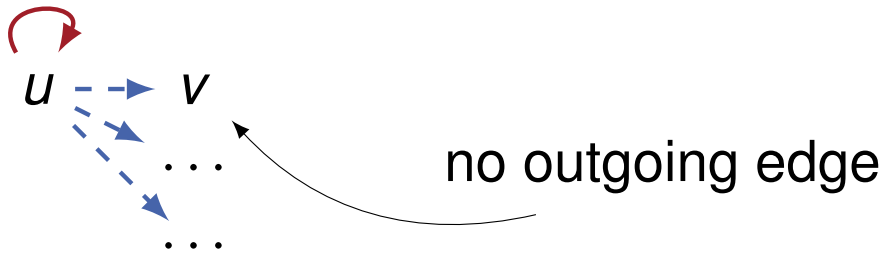
- All outgoing edges of u in πG are copies

Observations on Shuffle Code



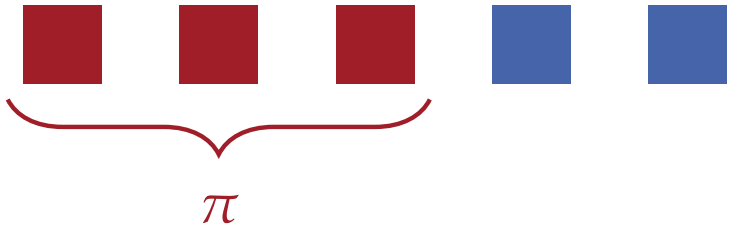
Normalized

Can transform shuffle code such that in πG, for every copy $u \to v$:



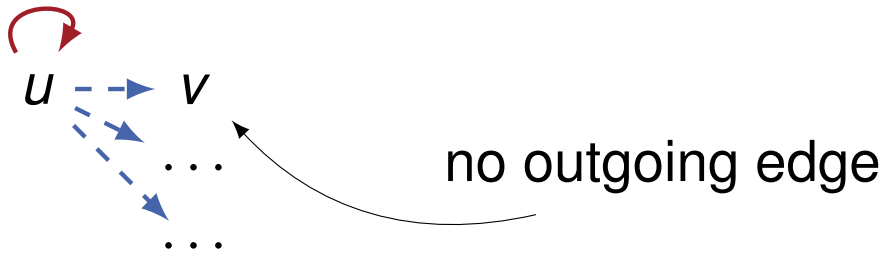
- All outgoing edges (except one) of u in πG are copies

Observations on Shuffle Code



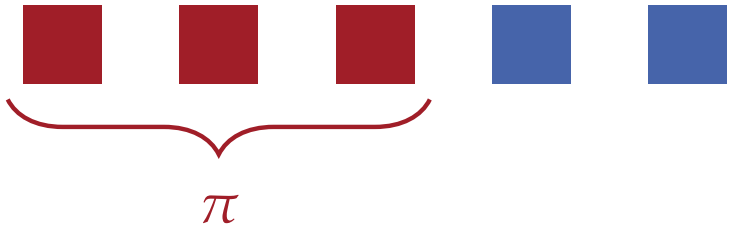
Normalized

Can transform shuffle code such that in πG, for every copy $u \to v$:



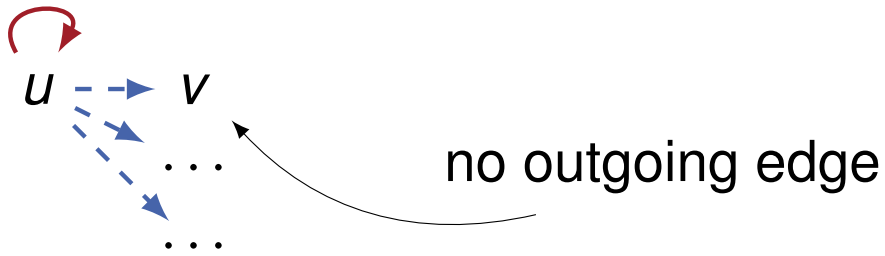
- All outgoing edges (except one) of u in πG are copies
- π permutes edge sources in G

Observations on Shuffle Code



Normalized

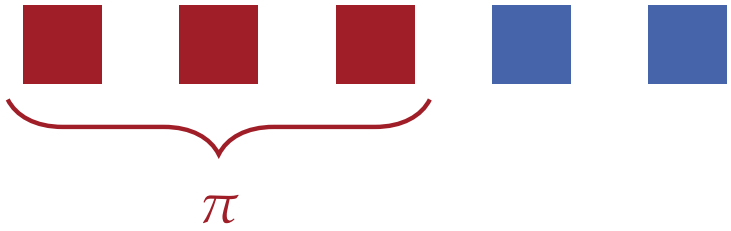
Can transform shuffle code such that in πG, for every copy $u \to v$:



- All outgoing edges (except one) of u in πG are copies
- π permutes edge sources in G

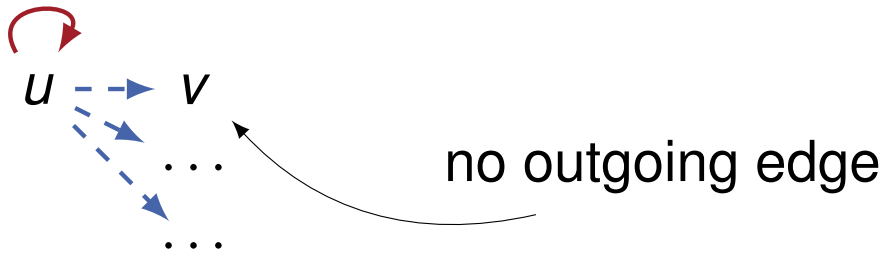
$\Rightarrow$ All outgoing edges (except one) of u in G correspond to copies

Observations on Shuffle Code



Normalized

Can transform shuffle code such that in πG, for every copy $u \to v$:



- All outgoing edges (except one) of u in πG are copies
- π permutes edge sources in G

$\Rightarrow$ All outgoing edges (except one) of u in G correspond to copies
$\to$ **copy set**

Strategy & Outline

Strategy & Outline

Step 1: Pick a copy set C.
(For each vertex, color one outgoing edge red, rest blue.)

Strategy & Outline

Step 1: Pick a copy set C.
(For each vertex, color one outgoing edge red, rest blue.)

Strategy & Outline

Step 1: Pick a copy set C.
(For each vertex, color one outgoing edge red, rest blue.)

Step 2: Find an optimal shuffle code for the red RTG $G - C$.
(Has maximum out-degree 1.)

Strategy & Outline

Step 1: Pick a copy set C.
(For each vertex, color one outgoing edge red, rest blue.)

Step 2: Find an optimal shuffle code for the red RTG $G - C$.
(Has maximum out-degree 1.)

Strategy & Outline

Step 1: Pick a copy set C.
(For each vertex, color one outgoing edge red, rest blue.)

Step 2: Find an optimal shuffle code for the red RTG $G - C$.
(Has maximum out-degree 1.)

Step 3: Implement blue edges using copy operations.

Strategy & Outline

Choice of copy set is crucial:

Step 1: Pick a copy set C.
(For each vertex, color one outgoing edge red, rest blue.)

Step 2: Find an optimal shuffle code for the red RTG $G - C$.
(Has maximum out-degree 1.)

Step 3: Implement blue edges using copy operations.

Strategy & Outline

Choice of copy set is crucial:

Step 1: Pick a copy set C.
(For each vertex, color one outgoing edge red, rest blue.)

Step 2: Find an optimal shuffle code for the red RTG $G - C$.
(Has maximum out-degree 1.)

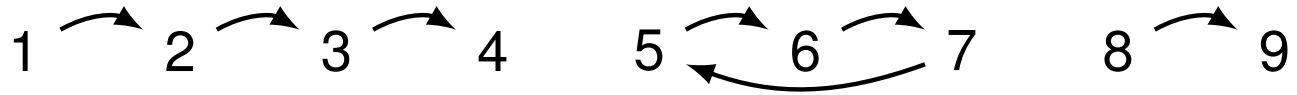
Step 3: Implement blue edges using copy operations.

Problem 1: Given copy set C, compute optimal shuffle code for $G - C$.

Problem 2: Find copy set C where $G - C$ requires fewest instructions.

The GREEDY Algorithm (adapted from [Mohr et al. 2013])

- Complete each directed path into directed cycle

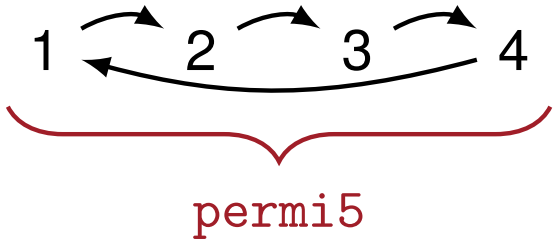


The GREEDY Algorithm (adapted from [Mohr et al. 2013])

- Complete each directed path into directed cycle

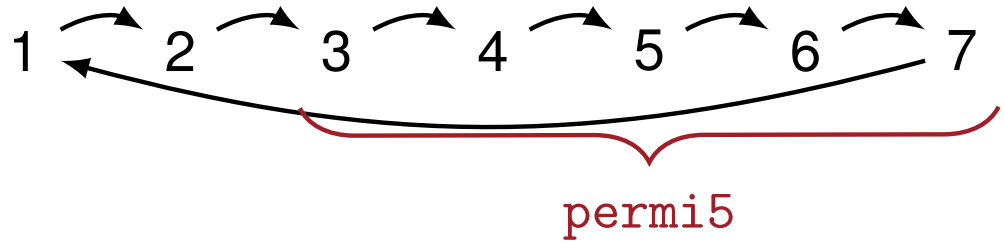
The GREEDY Algorithm (adapted from [Mohr et al. 2013])

- Complete each directed path into directed cycle
- **Phase 1**
  - While there is a cycle K of size 4 or more: use `permi5` to reduce K's size



The GREEDY Algorithm (adapted from [Mohr et al. 2013])

- Complete each directed path into directed cycle
- **Phase 1**
  - While there is a cycle K of size 4 or more: use `permi5` to reduce K's size

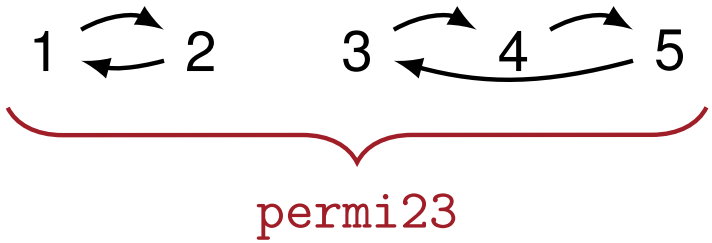


The GREEDY Algorithm (adapted from [Mohr et al. 2013])

- Complete each directed path into directed cycle
- **Phase 1**
  - While there is a cycle K of size 4 or more: use `permi5` to reduce K's size
- **Phase 2**: Only cycles of size ≤ 3 left

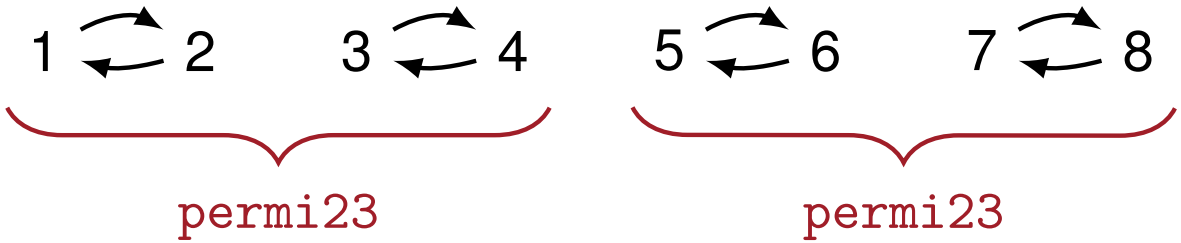
The GREEDY Algorithm (adapted from [Mohr et al. 2013])

- Complete each directed path into directed cycle
- **Phase 1**
  - While there is a cycle K of size 4 or more: use `permi5` to reduce K's size
- **Phase 2**: Only cycles of size ≤ 3 left
  - 2-cycle and 3-cycle available: resolve using `permi23`



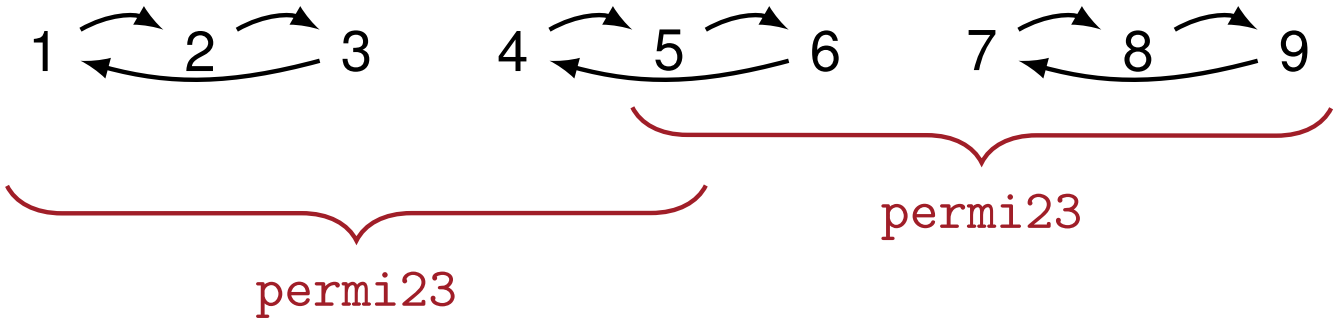
The GREEDY Algorithm (adapted from [Mohr et al. 2013])

- Complete each directed path into directed cycle
- **Phase 1**
  - While there is a cycle K of size 4 or more: use `permi5` to reduce K's size
- **Phase 2**: Only cycles of size ≤ 3 left
  - 2-cycle and 3-cycle available: resolve using `permi23`
  - Only 2-cycles available: resolve pairs using `permi23`



The GREEDY Algorithm (adapted from [Mohr et al. 2013])

- Complete each directed path into directed cycle
- **Phase 1**
  - While there is a cycle K of size 4 or more: use `permi5` to reduce K's size
- **Phase 2**: Only cycles of size ≤ 3 left
  - 2-cycle and 3-cycle available: resolve using `permi23`
  - Only 2-cycles available: resolve pairs using `permi23`
  - Only 3-cycles available: resolve groups of three using `permi23`



The GREEDY Algorithm (adapted from [Mohr et al. 2013])

- Complete each directed path into directed cycle
- **Phase 1**
  - While there is a cycle K of size 4 or more: use `permi5` to reduce K's size
- **Phase 2**: Only cycles of size ≤ 3 left
  - 2-cycle and 3-cycle available: resolve using `permi23`
  - Only 2-cycles available: resolve pairs using `permi23`
  - Only 3-cycles available: resolve groups of three using `permi23`

The GREEDY Algorithm (adapted from [Mohr et al. 2013])

- Complete each directed path into directed cycle
- **Phase 1**
  - While there is a cycle K of size 4 or more: use `permi5` to reduce K's size
- **Phase 2**: Only cycles of size ≤ 3 left
  - 2-cycle and 3-cycle available: resolve using `permi23`
  - Only 2-cycles available: resolve pairs using `permi23`
  - Only 3-cycles available: resolve groups of three using `permi23`

Signature $\text{sig}(G) = (X, a_2, a_3)$

- $X = \Sigma_{\sigma \in G} \lfloor \text{size}(\sigma)/4 \rfloor$
- $a_i = |\{\sigma \in G \mid \text{size}(\sigma) = i \mod 4\}|$

The GREEDY Algorithm (adapted from [Mohr et al. 2013])

- Complete each directed path into directed cycle
- **Phase 1**
  - While there is a cycle K of size 4 or more: use `permi5` to reduce K's size
- **Phase 2**: Only cycles of size ≤ 3 left
  - 2-cycle and 3-cycle available: resolve using `permi23`
  - Only 2-cycles available: resolve pairs using `permi23`
  - Only 3-cycles available: resolve groups of three using `permi23`

Signature $\text{sig}(G) = (X, a_2, a_3)$

- $X = \Sigma_{\sigma \in G} \lfloor \text{size}(\sigma)/4 \rfloor$
- $a_i = |\{\sigma \in G \mid \text{size}(\sigma) = i \mod 4\}|$

Cost function $\text{GREEDY}(G) = X + \begin{cases} \lceil (a_2 + a_3)/2 \rceil & \text{if } a_2 \geq a_3 \\ \lceil (a_2 + 2a_3)/3 \rceil & \text{if } a_2 < a_3 \end{cases}$

The GREEDY Algorithm (adapted from [Mohr et al. 2013])

- Complete each directed path into directed cycle
- **Phase 1**
  - While there is a cycle K of size 4 or more: use `permi5` to reduce K's size
- **Phase 2**: Only cycles of size ≤ 3 left
  - 2-cycle and 3-cycle available: resolve using `permi23`
  - Only 2-cycles available: resolve pairs using `permi23`
  - Only 3-cycles available: resolve groups of three using `permi23`

Signature $\text{sig}(G) = (X, a_2, a_3)$

- $X = \Sigma_{\sigma \in G} \lfloor \text{size}(\sigma)/4 \rfloor$
- $a_i = |\{\sigma \in G \mid \text{size}(\sigma) = i \mod 4\}|$

Cost function $\text{GREEDY}(G) = X + \begin{cases} \lceil (a_2 + a_3)/2 \rceil & \text{if } a_2 \geq a_3 \\ \lceil (a_2 + 2a_3)/3 \rceil & \text{if } a_2 < a_3 \end{cases}$

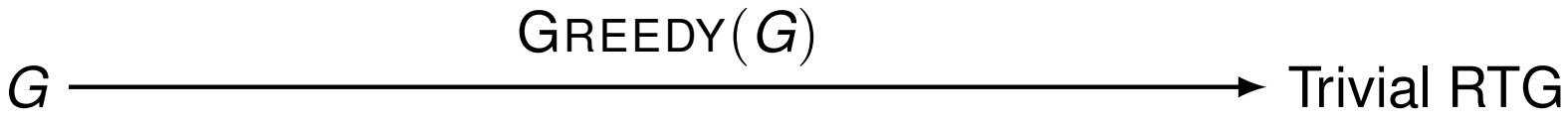
Theorem

GREEDY is optimal for PRTGs.

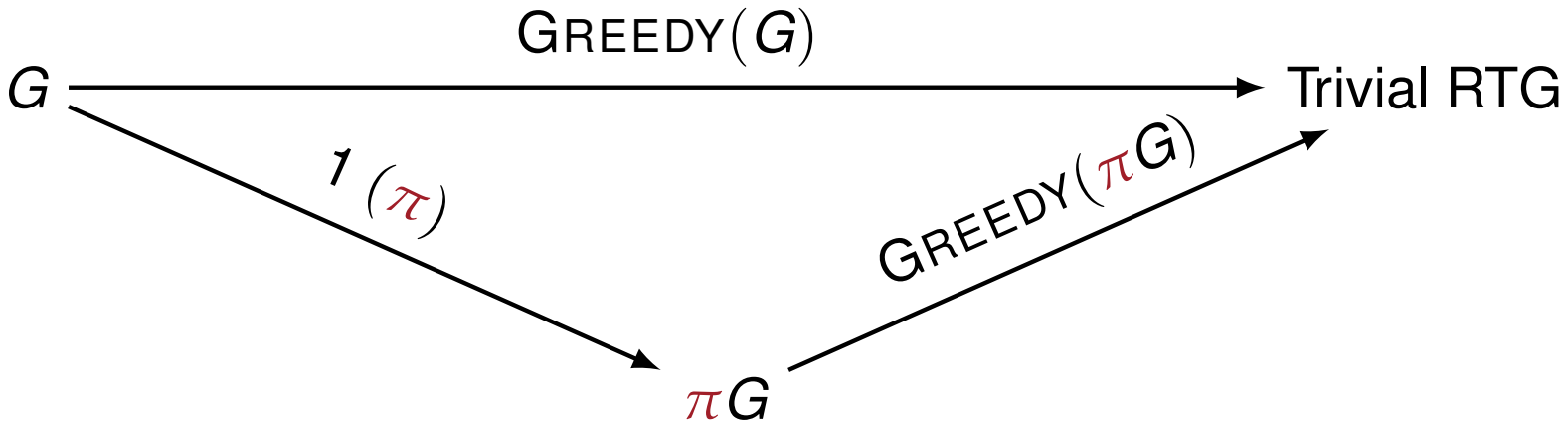
Proof Outline

G

Proof Outline

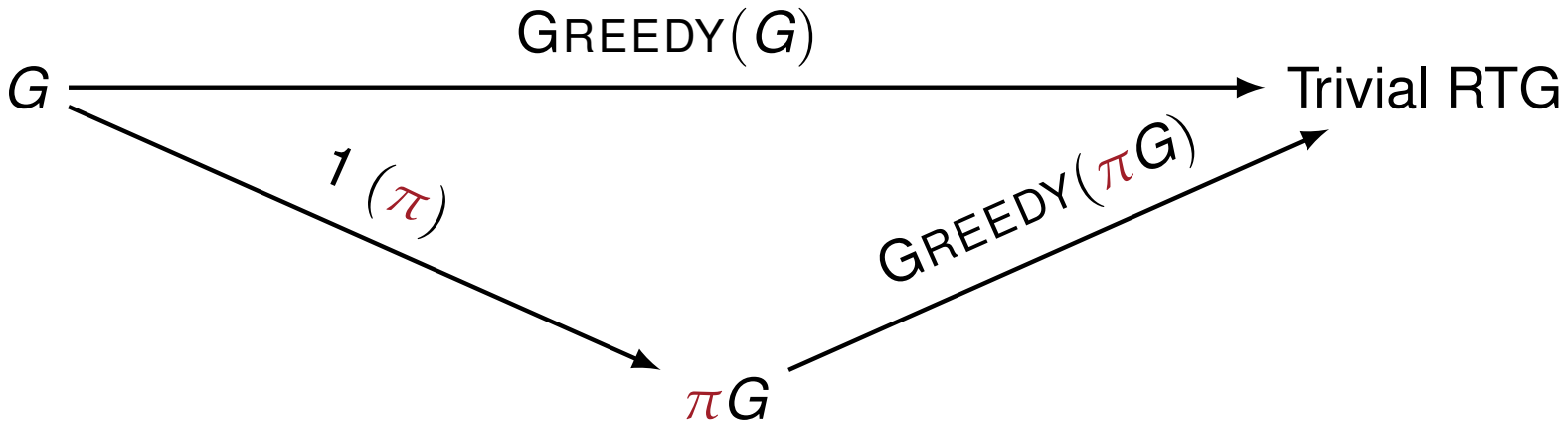


Proof Outline



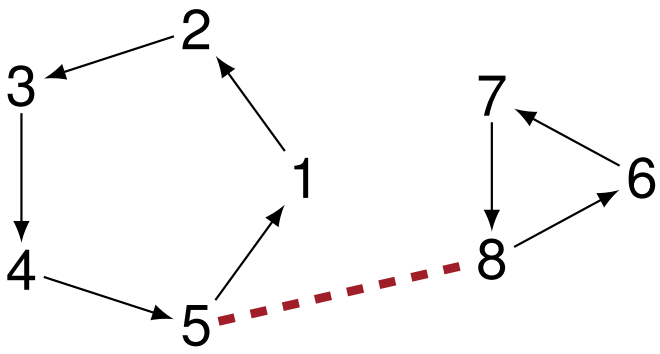
- Take arbitrary permutation instruction π

Proof Outline

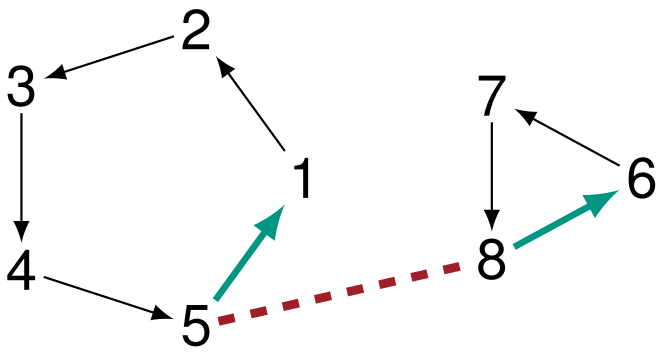


- Take arbitrary permutation instruction π
- **Show** that **always** $\textsc{Greedy}(G) \leq \textsc{Greedy}(\pi G) + 1$
  $\Leftrightarrow \textsc{Greedy}(G) - \textsc{Greedy}(\pi G) \leq 1$

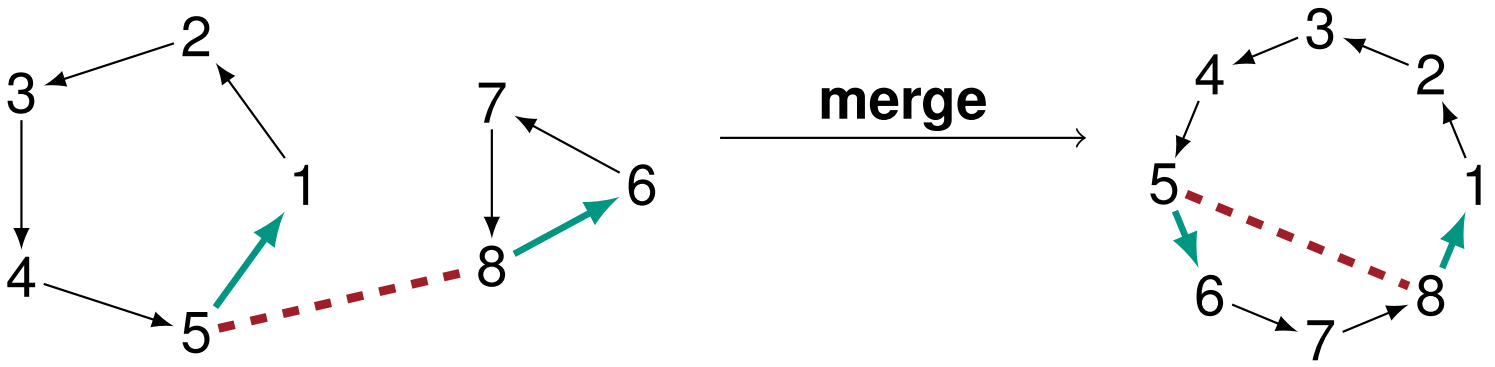
Transpositions



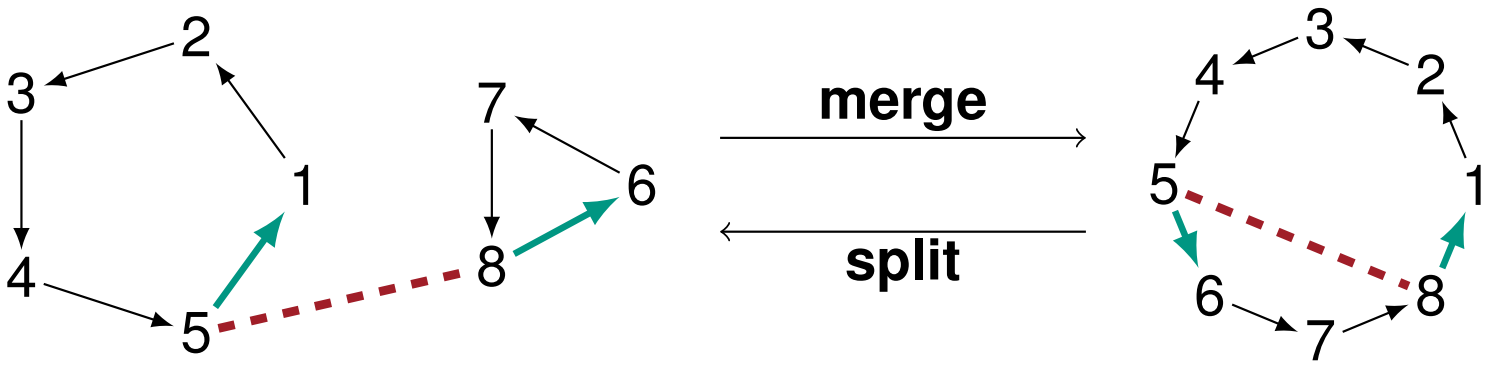
Transpositions



Transpositions



Transpositions



Merges

- Look at all possible signature changes $(\Delta_X, \Delta_2, \Delta_3)$

Merges

Cycle sizes modulo 4

	0	1	2	3
0	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$
1		$(0,1,0)$	$(0,-1,1)$	$(1,0,-1)$
2			$(1,-2,0)$	$(1,-1,-1)$
3				$(1,1,-2)$

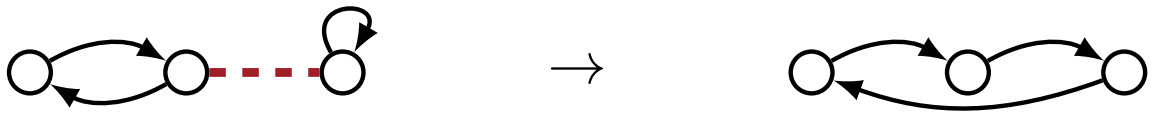
- Look at all possible signature changes $(\Delta_X, \Delta_2, \Delta_3)$

Merges

Cycle sizes modulo 4

	0	1	2	3
0	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$
1		$(0,1,0)$	$(0,-1,1)$	$(1,0,-1)$
2			$(1,-2,0)$	$(1,-1,-1)$
3				$(1,1,-2)$

- Look at all possible signature changes $(\Delta_X, \Delta_2, \Delta_3)$

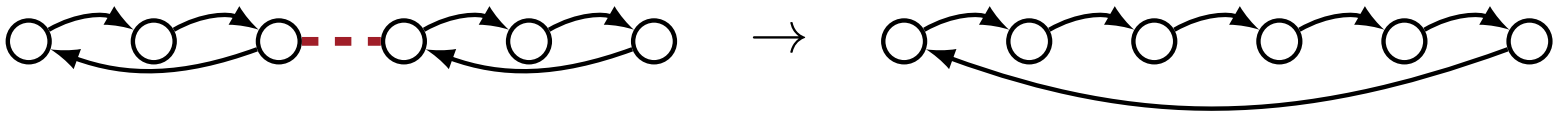


Merges

Cycle sizes modulo 4

	0	1	2	3
0	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$
1		$(0,1,0)$	$(0,-1,1)$	$(1,0,-1)$
2			$(1,-2,0)$	$(1,-1,-1)$
3				$(1,1,-2)$

- Look at all possible signature changes $(\Delta_X, \Delta_2, \Delta_3)$



Merges

Cycle sizes modulo 4

	0	1	2	3
0	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$
1		$(0,1,0)$	$(0,-1,1)$	$(1,0,-1)$
2			$(1,-2,0)$	$(1,-1,-1)$
3				$(1,1,-2)$

- Look at all possible signature changes $(\Delta_X, \Delta_2, \Delta_3)$
- Cost change **only** depends on signature change

Merges

Cycle sizes modulo 4

	0	1	2	3
0	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$
1		$(0,1,0)$	$(0,-1,1)$	$(1,0,-1)$
2			$(1,-2,0)$	$(1,-1,-1)$
3				$(1,1,-2)$

- Look at all possible signature changes $(\Delta_X, \Delta_2, \Delta_3)$
- Cost change **only** depends on signature change

	0	1	2	3
0	0	0	0	0
1		½	0	½
2			0	0
3				½

$a_2 \geq a_3$

	0	1	2	3
0	0	0	0	0
1		⅓	⅓	⅓
2			⅓	0
3				0

$a_2 < a_3$

Merges

Cycle sizes modulo 4

	0	1	2	3
0	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$
1		$(0,1,0)$	$(0,-1,1)$	$(1,0,-1)$
2			$(1,-2,0)$	$(1,-1,-1)$
3				$(1,1,-2)$

- Look at all possible signature changes $(\Delta_X, \Delta_2, \Delta_3)$
- Cost change **only** depends on signature change

	0	1	2	3
0	0	0	0	0
1		½	0	½
2			0	0
3				½

$a_2 \geq a_3$

	0	1	2	3
0	0	0	0	0
1		⅓	⅓	⅓
2			⅓	0
3				0

$a_2 < a_3$

Merges

Cycle sizes modulo 4

	0	1	2	3
0	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$
1		$(0,1,0)$	$(0,-1,1)$	$(1,0,-1)$
2			$(1,-2,0)$	$(1,-1,-1)$
3				$(1,1,-2)$

- Look at all possible signature changes $(\Delta_X, \Delta_2, \Delta_3)$
- Cost change **only** depends on signature change

	0	1	2	3
0	0	0	0	0
1		½	0	½
2			0	0
3				½

$a_2 \geq a_3$

	0	1	2	3
0	0	0	0	0
1		⅓	⅓	⅓
2			⅓	0
3				0

$a_2 < a_3$

$\Rightarrow$ **Merges are never worthwhile!**

Splits

	0	1	2	3
0	0	0	0	0
1		½	0	½
2			0	0
3				½

$a_2 \geq a_3$

	0	1	2	3
0	0	0	0	0
1		⅓	⅓	⅓
2			⅓	0
3				0

$a_2 < a_3$

Splits

	0	1	2	3
0	0	0	0	0
1		$-\frac{1}{2}$	0	$-\frac{1}{2}$
2			0	0
3				$-\frac{1}{2}$

$a_2 \geq a_3$

	0	1	2	3
0	0	0	0	0
1		$-\frac{1}{3}$	$-\frac{1}{3}$	$-\frac{1}{3}$
2			$-\frac{1}{3}$	0
3				0

$a_2 < a_3$

Splits

	0	1	2	3
0	0	0	0	0
1		$-\frac{1}{2}$	0	$-\frac{1}{2}$
2			0	0
3				$-\frac{1}{2}$

$a_2 \geq a_3$

	0	1	2	3
0	0	0	0	0
1		$-\frac{1}{3}$	$-\frac{1}{3}$	$-\frac{1}{3}$
2			$-\frac{1}{3}$	0
3				0

$a_2 < a_3$

- `permi5` can implement 4 transpositions $\Rightarrow -\frac{1}{2} \cdot 4 = -2$?

Splits

	0	1	2	3
0	0	0	0	0
1		$-\frac{1}{2}$	0	$-\frac{1}{2}$
2			0	0
3				$-\frac{1}{2}$

$a_2 \geq a_3$

	0	1	2	3
0	0	0	0	0
1		$-\frac{1}{3}$	$-\frac{1}{3}$	$-\frac{1}{3}$
2			$-\frac{1}{3}$	0
3				0

$a_2 < a_3$

- `permi5` can implement 4 transpositions $\Rightarrow -\frac{1}{2} \cdot 4 = -2$?
- However: because of structure of instructions, not every transposition can reduce costs

Splits

	0	1	2	3
0	0	0	0	0
1		$-\frac{1}{2}$	0	$-\frac{1}{2}$
2			0	0
3				$-\frac{1}{2}$

$a_2 \geq a_3$

	0	1	2	3
0	0	0	0	0
1		$-\frac{1}{3}$	$-\frac{1}{3}$	$-\frac{1}{3}$
2			$-\frac{1}{3}$	0
3				0

$a_2 < a_3$

- `permi5` can implement 4 transpositions $\Rightarrow -\frac{1}{2} \cdot 4 = -2$?
- However: because of structure of instructions, not every transposition can reduce costs

GREEDY is optimal for PRTGs

GREEDY computes an optimal shuffle code for PRTGs in linear time.

Finding optimal copy sets

Given G, have to find copy set C, s.t. $\textsc{Greedy}(G - C)$ is minimal

Finding optimal copy sets

Given G, have to find copy set C, s.t. $\textsc{Greedy}(G - C)$ is minimal

- Equivalent to minimizing

$$\textsc{Greedy}'(G - C) = \begin{cases} X + \frac{a_2}{2} + \frac{a_3}{2} & \text{if } a_2 \geq a_3 \\ X + \frac{a_2}{3} + \frac{2a_3}{3} & \text{if } a_2 < a_3 \end{cases}$$

- Distinguish cases with $\text{diff}(G - C) = a_2 - a_3$

Finding optimal copy sets

Given G, have to find copy set C, s.t. $\textsc{Greedy}(G - C)$ is minimal

- Equivalent to minimizing

$$\textsc{Greedy}'(G - C) = \begin{cases} X + \frac{a_2}{2} + \frac{a_3}{2} =: cost^1(G - C) & \text{if } a_2 \geq a_3 \\ X + \frac{a_2}{3} + \frac{2a_3}{3} =: cost^2(G - C) & \text{if } a_2 < a_3 \end{cases}$$

- Distinguish cases with $\text{diff}(G - C) = a_2 - a_3$

Finding optimal copy sets

Given G, have to find copy set C, s.t. $\textsc{Greedy}(G - C)$ is minimal

- Equivalent to minimizing

$$\textsc{Greedy}'(G - C) = \begin{cases} X + \frac{a_2}{2} + \frac{a_3}{2} =: cost^1(G - C) & \text{if } a_2 \geq a_3 \\ X + \frac{a_2}{3} + \frac{2a_3}{3} =: cost^2(G - C) & \text{if } a_2 < a_3 \end{cases}$$

- Distinguish cases with $\text{diff}(G - C) = a_2 - a_3$

Dynamic program with tables $T^1[\cdot]$, $T^2[\cdot]$

- $T^i[d] = \min\{\text{cost}^i(G - C) \mid C \text{ copyset with } \text{diff}(G - C) = d\}$

Finding optimal copy sets

Given G, have to find copy set C, s.t. $\textsc{Greedy}(G - C)$ is minimal

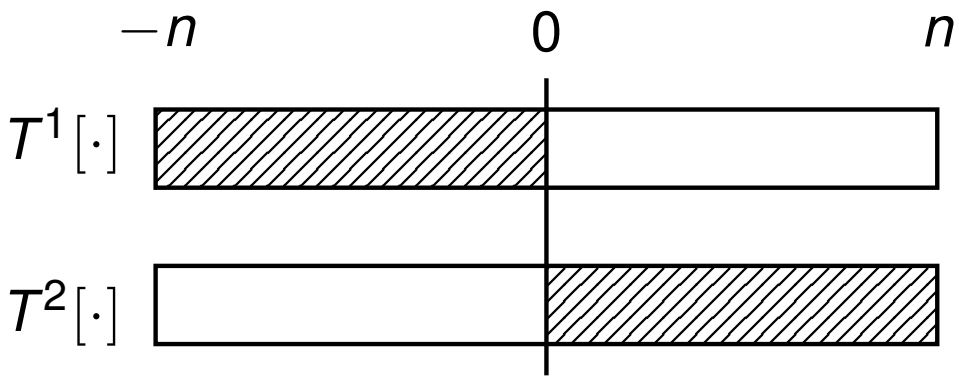
- Equivalent to minimizing

$$\textsc{Greedy}'(G - C) = \begin{cases} X + \frac{a_2}{2} + \frac{a_3}{2} =: cost^1(G - C) & \text{if } a_2 \geq a_3 \\ X + \frac{a_2}{3} + \frac{2a_3}{3} =: cost^2(G - C) & \text{if } a_2 < a_3 \end{cases}$$

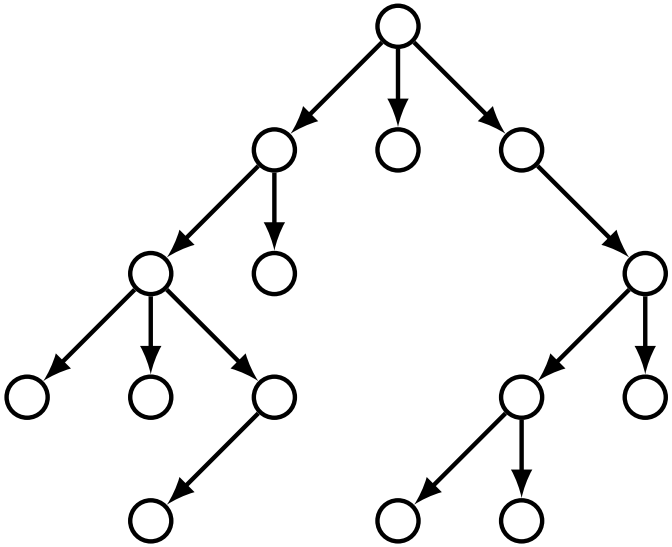
- Distinguish cases with $\text{diff}(G - C) = a_2 - a_3$

Dynamic program with tables $T^1[\cdot]$, $T^2[\cdot]$

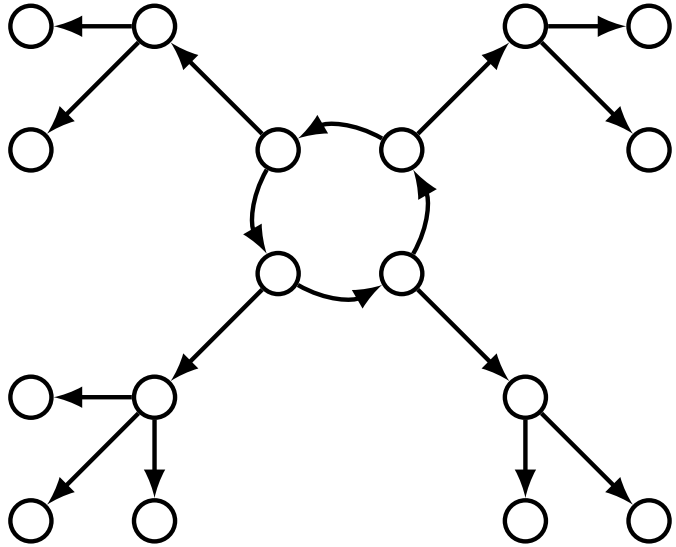
- $T^i[d] = \min\{\text{cost}^i(G - C) \mid C \text{ copyset with } \text{diff}(G - C) = d\}$



Computing Tables for Different RTG Shapes

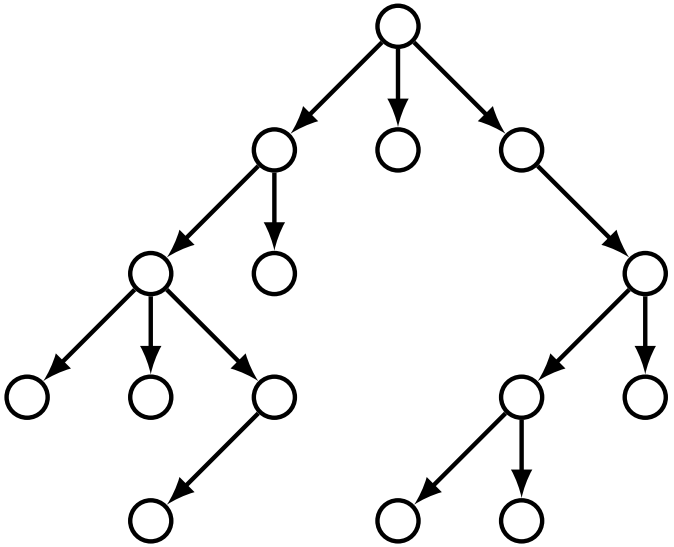


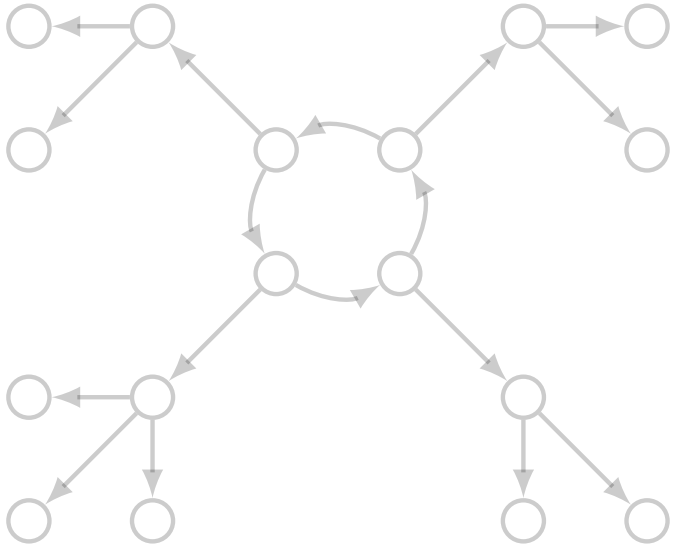
Disconnected RTGs



$O(n^2)$

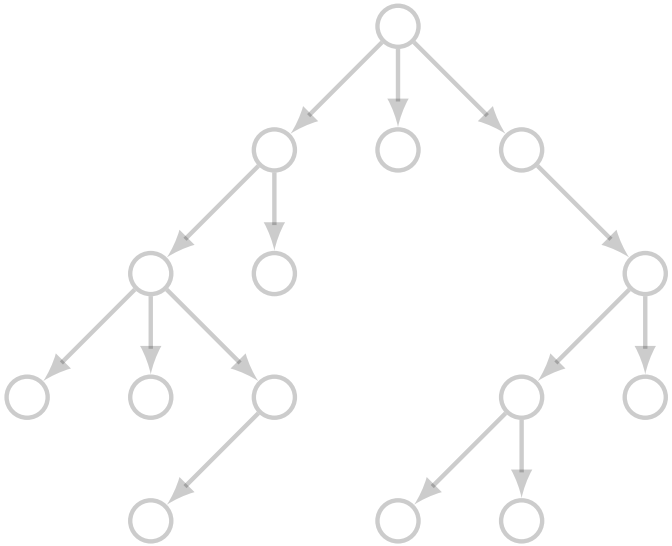
Computing Tables for Different RTG Shapes

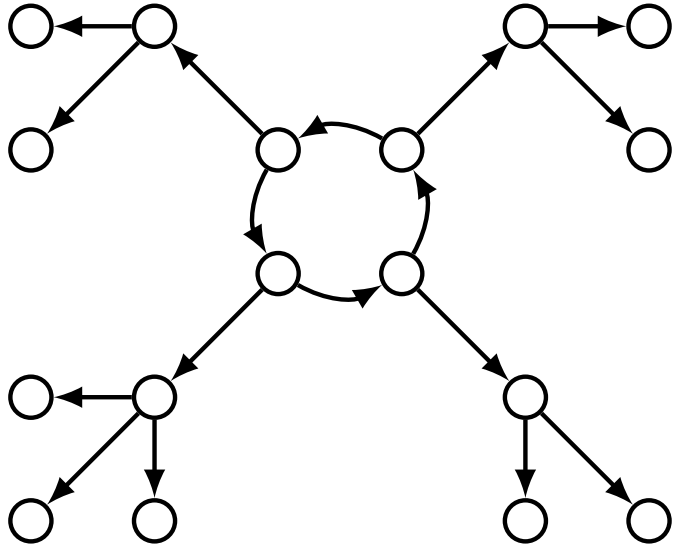




Disconnected RTGs	$O(n^2)$
Tree RTGs	$O(n^3)$

Computing Tables for Different RTG Shapes





Disconnected RTGs	$O(n^2)$
Tree RTGs	$O(n^3)$
Connected RTGs containing cycle	$O(n^4)$

Conclusion

Problem 1: Given copy set C, compute optimal shuffle code for $G - C$.

- Shown that GREEDY is optimal for PRTGs, runs in $O(n)$ time
- Equivalent to factoring permutation into shortest product of permutations of maximum size 5

Conclusion

Problem 1: Given copy set C, compute optimal shuffle code for $G - C$.

- Shown that GREEDY is optimal for PRTGs, runs in $O(n)$ time
- Equivalent to factoring permutation into shortest product of permutations of maximum size 5

Problem 2: Find copy set C s.t. $G - C$ requires fewest instructions.

- Used dynamic programming to solve optimally
- Runs in $O(n^4)$ time

Conclusion

Problem 1: Given copy set C, compute optimal shuffle code for $G - C$.

- Shown that GREEDY is optimal for PRTGs, runs in $O(n)$ time
- Equivalent to factoring permutation into shortest product of permutations of maximum size 5

Problem 2: Find copy set C s.t. $G - C$ requires fewest instructions.

- Used dynamic programming to solve optimally
- Runs in $O(n^4)$ time

Future work:

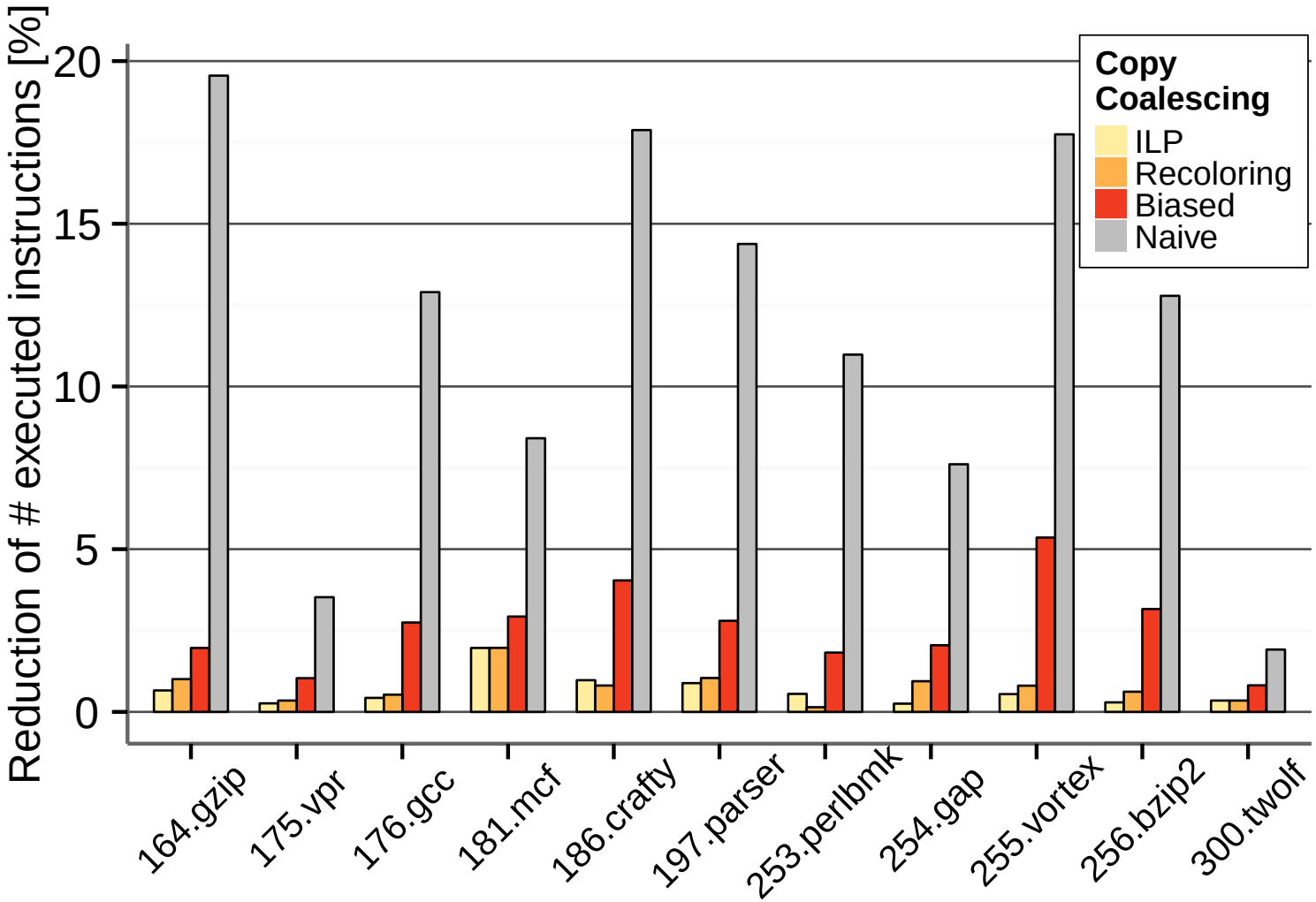
- Works for $k = 3, 4, 5, 6$, what about larger sizes?
- Problem NP-complete if permutation size is part of input

$\Rightarrow$ FPT-algorithm?

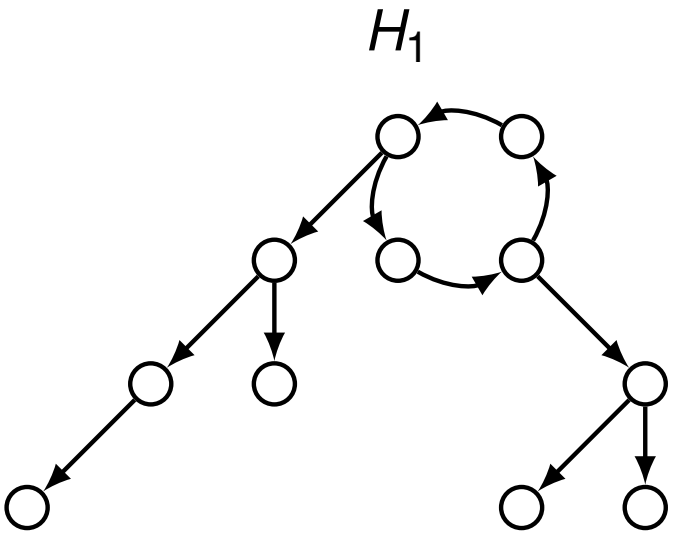
Karlsruhe Institute of Technology

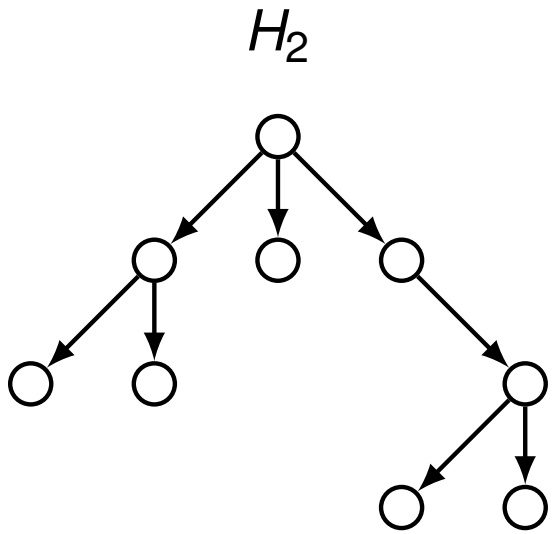
Backup Slides

Speedup Measurements

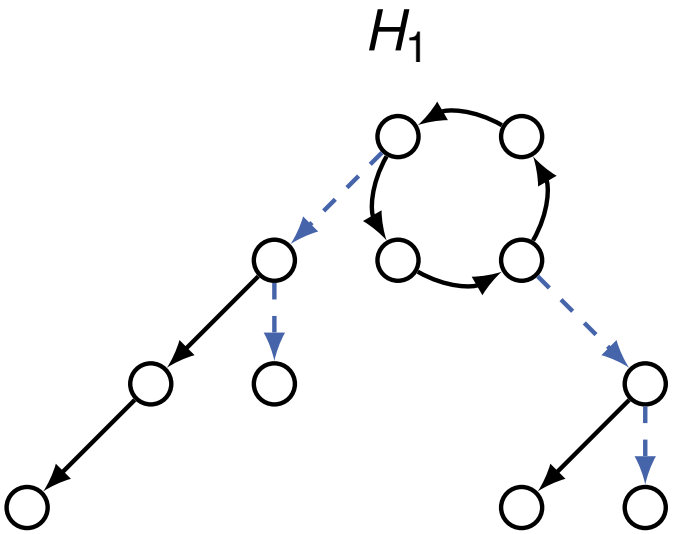


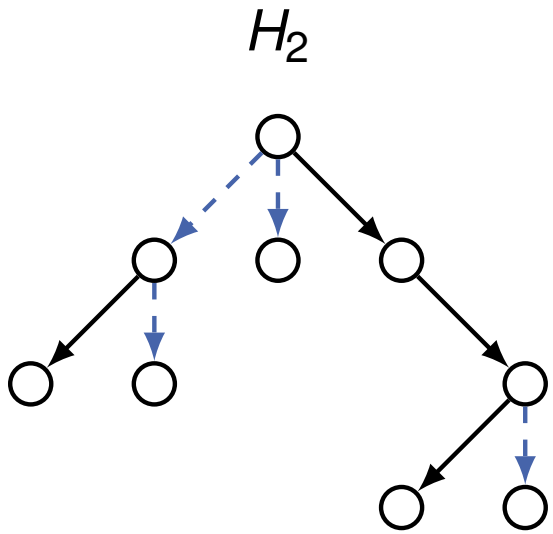
Disconnected RTGs



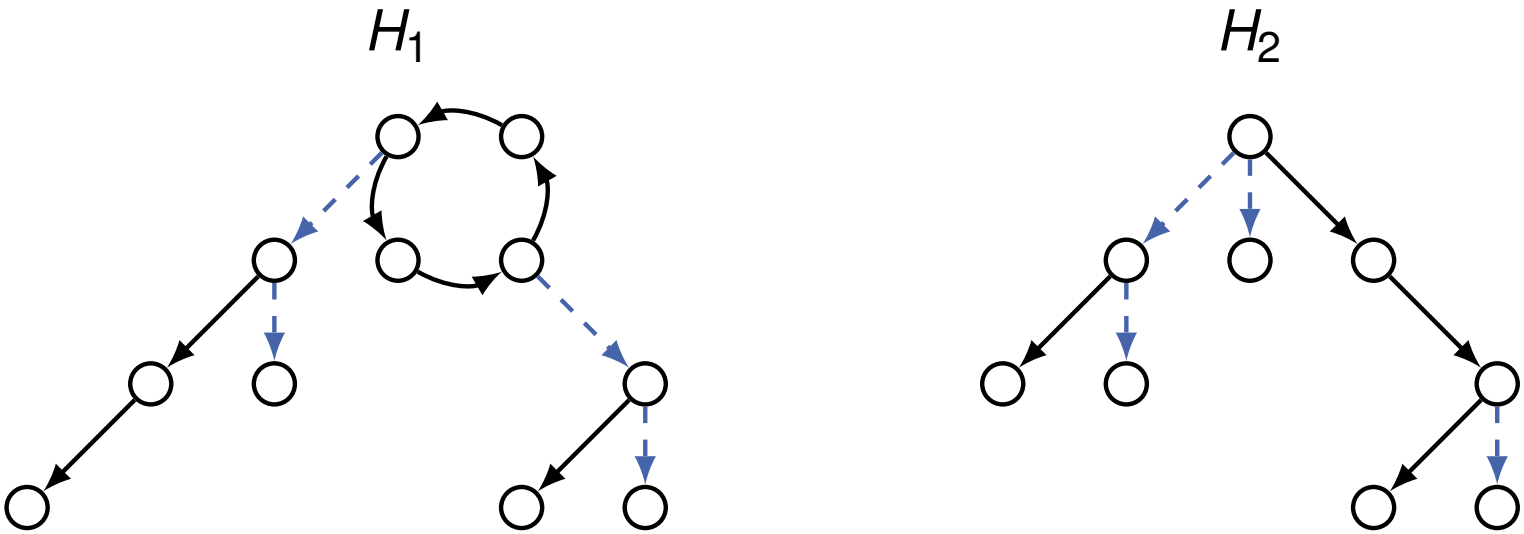


Disconnected RTGs



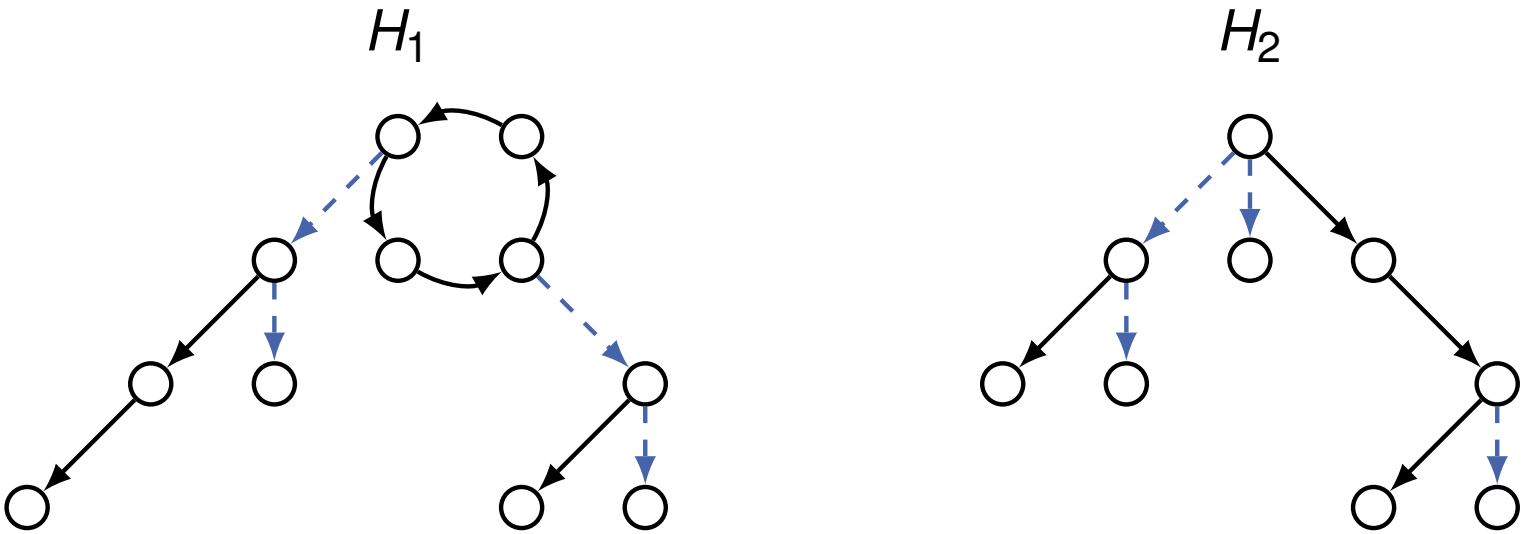


Disconnected RTGs



- $C := C_1 \cup C_2$
- $\text{cost}(G - C) = \text{cost}(H_1 - C_1) + \text{cost}(H_2 - C_2)$
- $\text{diff}(G - C) = \text{diff}(H_1 - C_1) + \text{diff}(H_2 - C_2)$

Disconnected RTGs

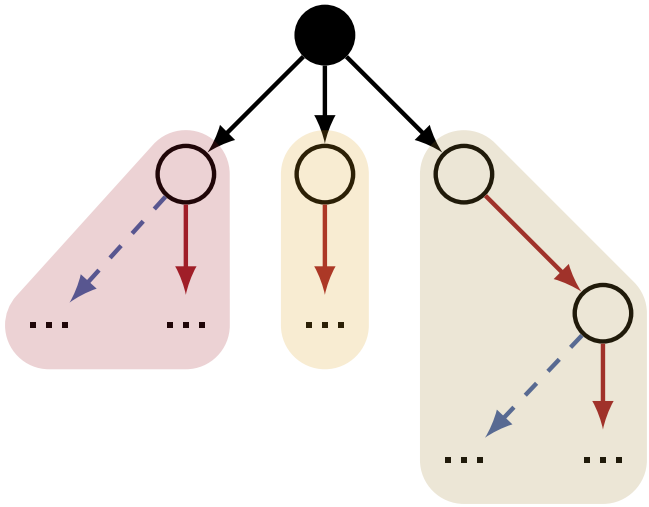


- $C := C_1 \cup C_2$
- $\text{cost}(G - C) = \text{cost}(H_1 - C_1) + \text{cost}(H_2 - C_2)$
- $\text{diff}(G - C) = \text{diff}(H_1 - C_1) + \text{diff}(H_2 - C_2)$

$\Rightarrow$ Can find optimal copy set C by trying all pairs C_1, C_2

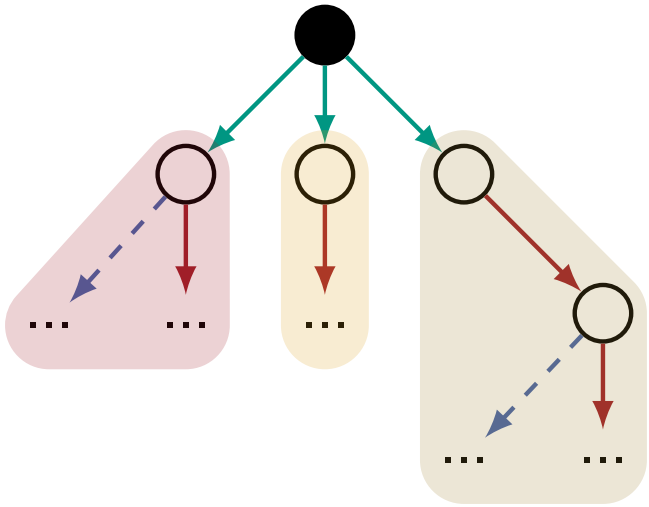
$\Rightarrow$ Can be computed in $O(n^2)$ time

Tree RTGs



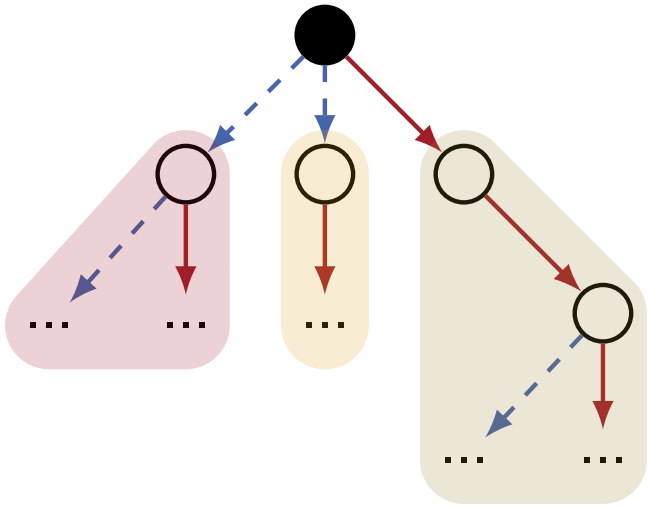
- Process tree RTGs in bottom-up fashion

Tree RTGs



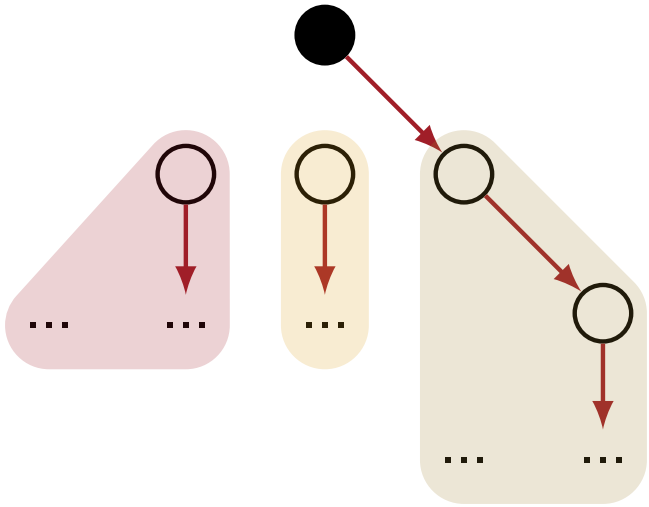
- Process tree RTGs in bottom-up fashion

Tree RTGs



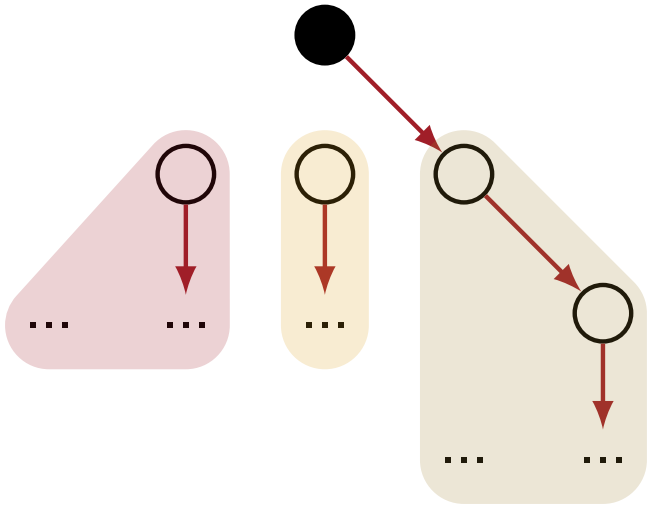
- Process tree RTGs in bottom-up fashion

Tree RTGs



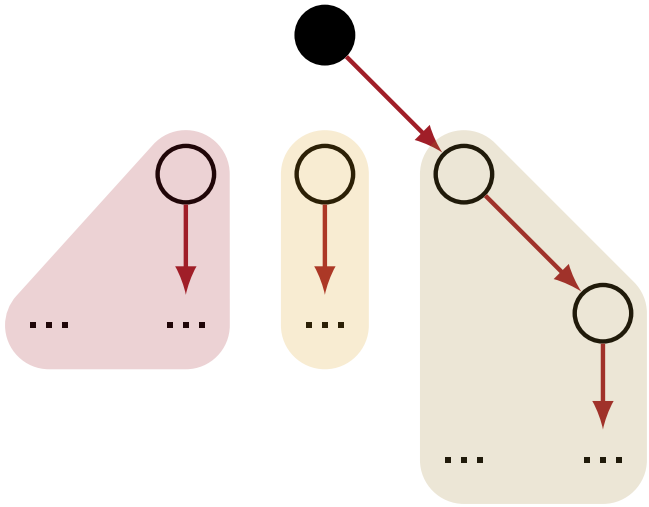
- Process tree RTGs in bottom-up fashion

Tree RTGs



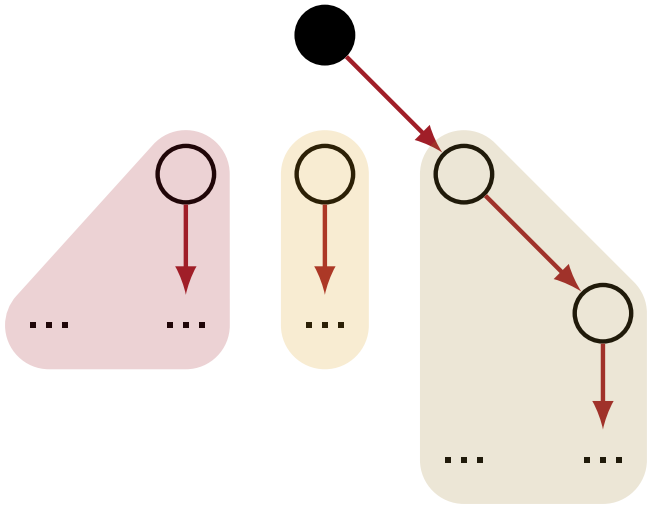
- Process tree RTGs in bottom-up fashion
- Extend table to track length of path starting at root

Tree RTGs



- Process tree RTGs in bottom-up fashion
- Extend table to track length of path (modulo 4) starting at root

Tree RTGs

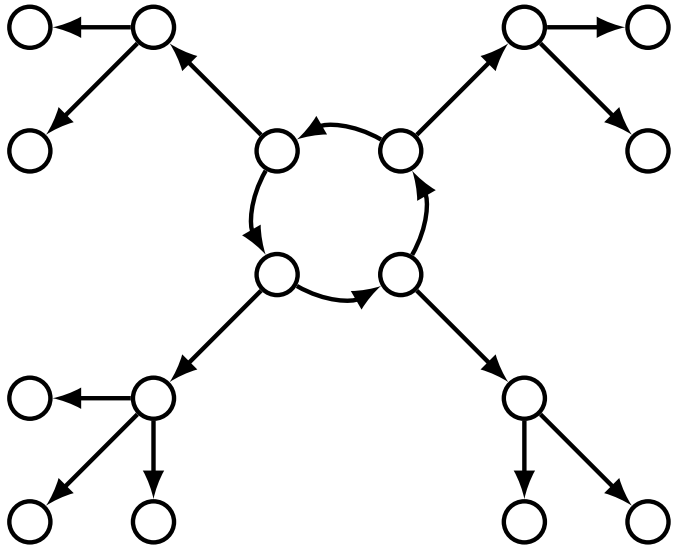


- Process tree RTGs in bottom-up fashion
- Extend table to track length of path (modulo 4) starting at root

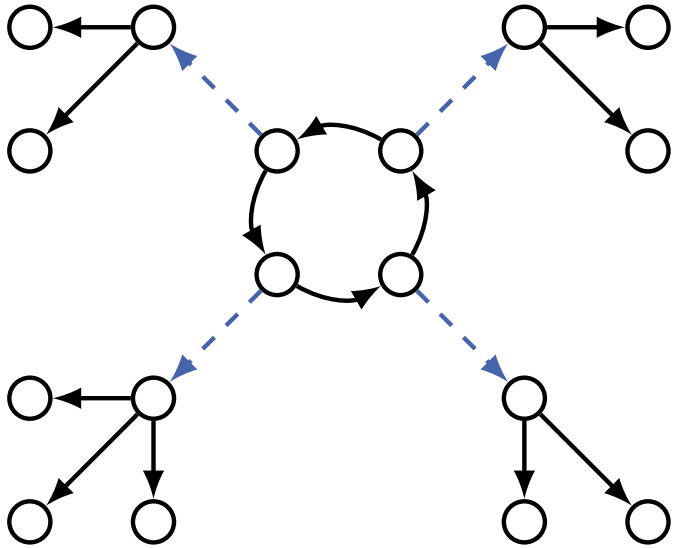
$\Rightarrow$ Can find optimal copy set C by trying all outgoing edges of root

$\Rightarrow$ Can be computed in time $O(n^3)$

Connected RTGs

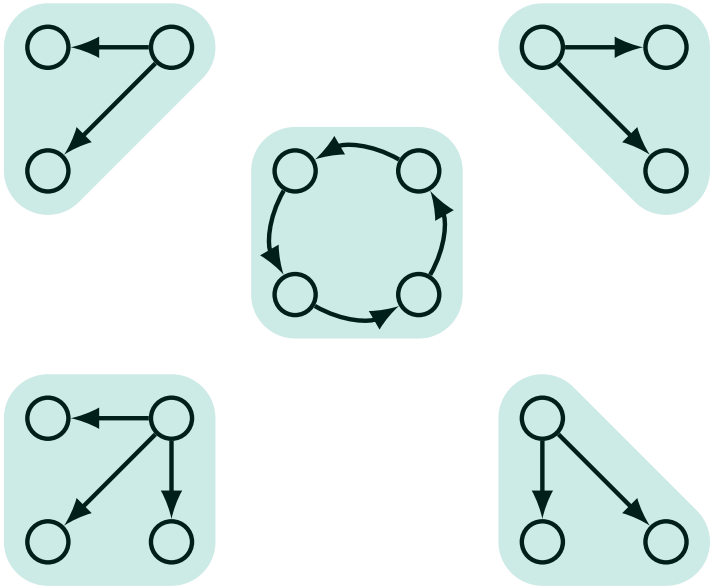


Connected RTGs



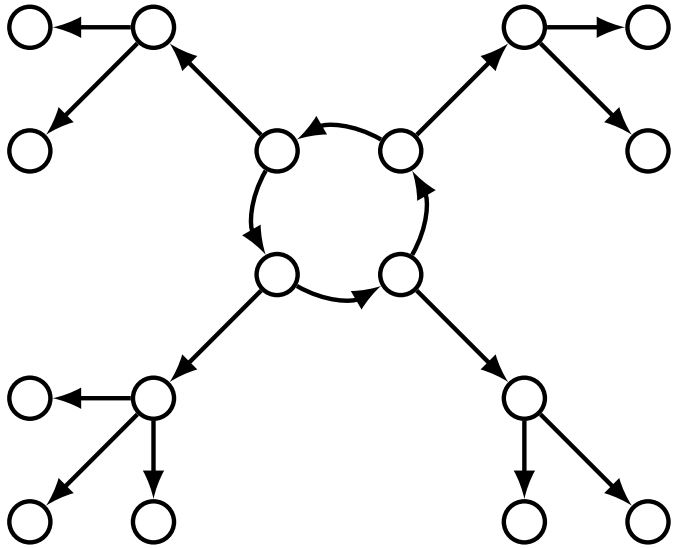
- Either keep cycle K and put edges leaving K into copy set

Connected RTGs



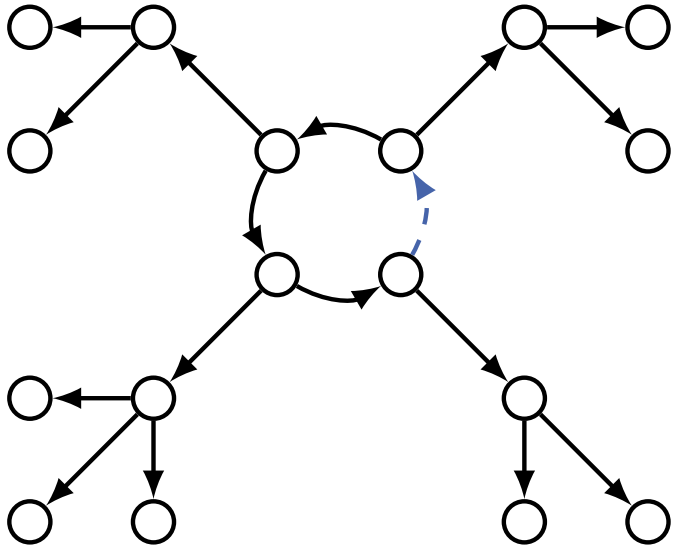
- Either keep cycle K and put edges leaving K into copy set

Connected RTGs



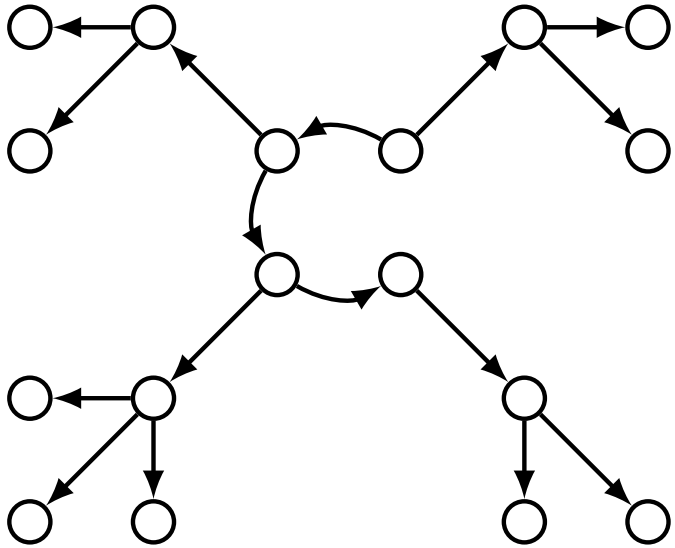
- Either keep cycle K and put edges leaving K into copy set

Connected RTGs



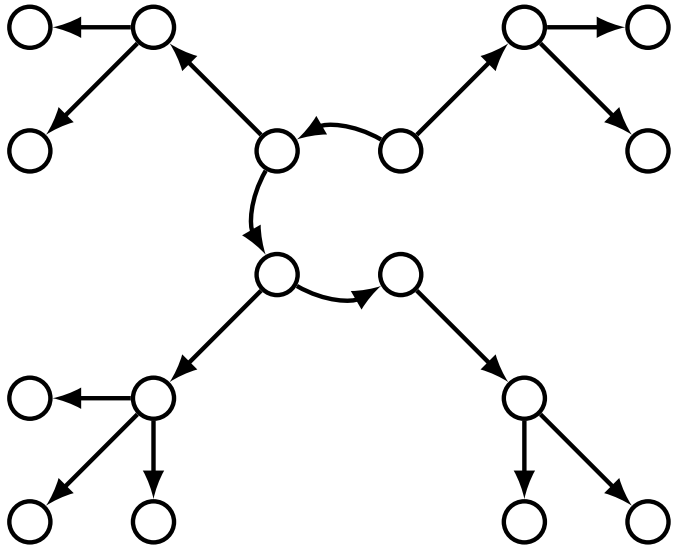
- Either keep cycle K and put edges leaving K into copy set
- Or cut K, $G - C$ is a tree

Connected RTGs



- Either keep cycle K and put edges leaving K into copy set
- Or cut K, $G - C$ is a tree

Connected RTGs



- Either keep cycle K and put edges leaving K into copy set
- Or cut K, $G - C$ is a tree

$\Rightarrow$ Can find optimal copy set C by trying all of K's edges (or keeping K)

$\Rightarrow$ Can be computed in time $O(n^4)$