Algorithms for Algebraic Path Properties in Concurrent Systems of Constant Treewidth Components

Krishnendu Chatterjee, Amir Kafshdar Goharshady,
Rasmus Ibsen-Jensen, **Andreas Pavlogiannis**

POPL 2016

IST AUSTRIA
Institute of Science and Technology

Typical Program Analysis Paradigm

A typical paradigm in program analysis is to reduce the problem to a standard graph problem P:

Input: Program

1. Extract control flow graph G
2. Annotate G
3. Run best general graph algorithm for P on G

Typical Program Analysis Paradigm

A typical paradigm in program analysis is to reduce the problem to a standard graph problem P:
Input: Program

1. Extract control flow graph G
2. Annotate G
3. **Run best ~~general~~ graph algorithm for P on G**
   - by exploiting special structure of CFGs

Static Analysis of Concurrent Programs

- Static *dataflow/quantitative* analysis of concurrent systems
- System consists of CFGs of local threads + sync vars
- A node of the concurrent system specifies the local state of each thread (+ sync)
- System transitions wrt interleaving semantics

Algebraic Paths in Static Analysis

- Concurrent system annotated with a (complete, closed) *semiring*.
- Variety of properties expressible
  - (Generalized) reachability
  - Distributive dataflow analysis problems
  - Quantitative problems (quality measures / quantitative verification)
  - Algebraic relaxations for interprocedural analysis

Algebraic Paths in Static Analysis

- Concurrent system annotated with a (complete, closed) *semiring*.
- Variety of properties expressible
  - (Generalized) reachability
  - Distributive dataflow analysis problems
  - Quantitative problems (quality measures / quantitative verification)
  - Algebraic relaxations for interprocedural analysis

Formal Setting

- Complete, closed **semiring** $S = (\Sigma, \oplus, \otimes, \overline{\mathbf{0}}, \overline{\mathbf{1}})$
- k local graphs $G_i = (V_i, E_i)$, $|V_i| \leq n$
- Compose a **concurrent system** $G = (V, E, \mathrm{wt})$
  - Nodes of the form $v = \langle v_1, \ldots, v_k \rangle$
  - $E \subseteq \mathrm{product}(E_1, \ldots, E_k)$
    - i.e. $(\langle u_1, \ldots, u_k \rangle, \langle v_1, \ldots, v_k \rangle) \in E$
  - Global weight function $\mathrm{wt} : E \to \Sigma$
- **Weight** of a path $P : x_1, x_2, \ldots, x_m$:

$$\otimes(P) = \mathrm{wt}(x_1, x_2) \otimes \mathrm{wt}(x_2, x_3) \cdots \otimes \mathrm{wt}(x_{k-1}, x_m)$$

Formal Setting

- Complete, closed **semiring** $S = (\Sigma, \oplus, \otimes, \overline{\mathbf{0}}, \overline{\mathbf{1}})$
- k local graphs $G_i = (V_i, E_i)$, $|V_i| \leq n$
- Compose a **concurrent system** $G = (V, E, \mathrm{wt})$
  - Nodes of the form $v = \langle v_1, \ldots, v_k \rangle$
  - $E \subseteq \mathrm{product}(E_1, \ldots, E_k)$
    - i.e. $(\langle u_1, \ldots, u_k \rangle, \langle v_1, \ldots, v_k \rangle) \in E$
  - Global weight function $\mathrm{wt} : E \to \Sigma$
- **Weight** of a path $P : x_1, x_2, \ldots, x_m$:

$$\otimes(P) = \mathrm{wt}(x_1, x_2) \otimes \mathrm{wt}(x_2, x_3) \cdots \otimes \mathrm{wt}(x_{k-1}, x_m)$$

Formal Setting

- Complete, closed **semiring** $S = (\Sigma, \oplus, \otimes, \overline{\mathbf{0}}, \overline{\mathbf{1}})$
- k local graphs $G_i = (V_i, E_i)$, $|V_i| \leq n$
- Compose a **concurrent system** $G = (V, E, \mathrm{wt})$
    - Nodes of the form $v = \langle v_1, \ldots, v_k \rangle$
    - $E \subseteq \mathrm{product}(E_1, \ldots, E_k)$
        - i.e. $(\langle u_1, \ldots, u_k \rangle, \langle v_1, \ldots, v_k \rangle) \in E$
    - Global weight function $\mathrm{wt} : E \to \Sigma$
- **Weight** of a path $P : x_1, x_2, \ldots, x_m$:

$$\otimes(P) = \mathrm{wt}(x_1, x_2) \otimes \mathrm{wt}(x_2, x_3) \cdots \otimes \mathrm{wt}(x_{k-1}, x_m)$$

Graph Problem: Semiring Distances

Weight of a path $P : x_1, x_2, \ldots, x_m$:

$$\otimes(P) = \mathrm{wt}(x_1, x_2) \otimes \mathrm{wt}(x_2, x_3) \cdots \otimes \mathrm{wt}(x_{k-1}, x_m)$$

Semiring distance from u to v:

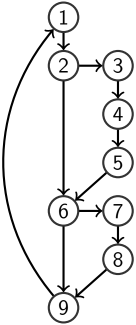
$$d(u, v) = \bigoplus_{P : u \rightsquigarrow v} \otimes(P)$$

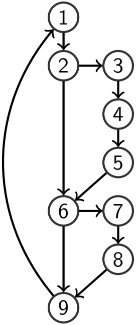
Method 1: Thread 1

```
while 1 do
    if turn = −1 then
        lock(ℓ)
        turn ← my_id
        unlock(ℓ)
    if turn = my_id then
        /* do stuff */
        turn ← −1
end
```

Method 2: Thread 2

```
while 1 do
    if turn = −1 then
        lock(ℓ)
        turn ← my_id
        unlock(ℓ)
    if turn = my_id then
        /* do stuff */
        turn ← −1
end
```



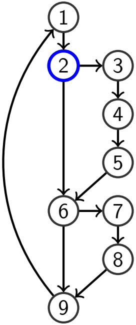


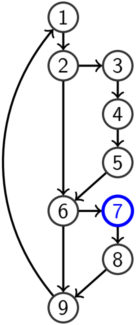
Method 1: Thread 1

```
while 1 do
    if turn = −1 then
        lock(ℓ)
        turn ← my_id
        unlock(ℓ)
    if turn = my_id then
        /* do stuff */
        turn ← −1
end
```

Method 2: Thread 2

```
while 1 do
    if turn = −1 then
        lock(ℓ)
        turn ← my_id
        unlock(ℓ)
    if turn = my_id then
        /* do stuff */
        turn ← −1
end
```





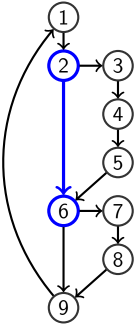
Nodes V: $\langle 2, 7 \rangle$

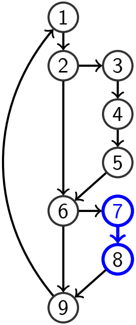
Method 1: Thread 1

```
while 1 do
    if turn = −1 then
        lock(ℓ)
        turn ← my_id
        unlock(ℓ)
    if turn = my_id then
        /* do stuff */
        turn ← −1
end
```

Method 2: Thread 2

```
while 1 do
    if turn = −1 then
        lock(ℓ)
        turn ← my_id
        unlock(ℓ)
    if turn = my_id then
        /* do stuff */
        turn ← −1
end
```





Nodes V: $\langle 2, 7\rangle$

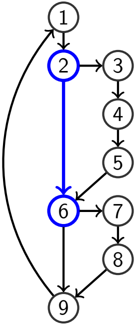
Edges E: $(\langle 2, 7\rangle, \langle 6, 8\rangle)$

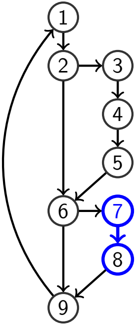
Method 1: Thread 1

```
while 1 do
    if turn = −1 then
        lock(ℓ)
        turn ← my_id
        unlock(ℓ)
    if turn = my_id then
        /* do stuff */
        turn ← −1
end
```

Method 2: Thread 2

```
while 1 do
    if turn = −1 then
        lock(ℓ)
        turn ← my_id
        unlock(ℓ)
    if turn = my_id then
        /* do stuff */
        turn ← −1
end
```





Nodes V: $\langle 2, 7 \rangle$

Edges E: $(\langle 2, 7 \rangle, \langle 6, 8 \rangle)$

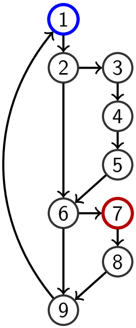
Weights $\mathrm{wt} : E \to \Sigma : \mathrm{wt}(\langle 2, 7 \rangle, \langle 6, 8 \rangle) = \alpha$

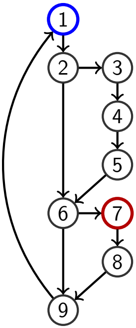
Method 1: Thread 1

```
1 while 1 do
2     if turn = −1 then
3         lock(ℓ)
4         turn ← my_id
5         unlock(ℓ)
6     if turn = my_id then
7         /* do stuff */
8         turn ← −1
9 end
```

Method 2: Thread 2

```
1 while 1 do
2     if turn = −1 then
3         lock(ℓ)
4         turn ← my_id
5         unlock(ℓ)
6     if turn = my_id then
7         /* do stuff */
8         turn ← −1
9 end
```





Nodes V: $\langle 2, 7\rangle$

Edges E: $(\langle 2, 7\rangle, \langle 6, 8\rangle)$

Weights $\mathrm{wt} : E \to \Sigma : \mathrm{wt}(\langle 2, 7\rangle, \langle 6, 8\rangle) = \alpha$

Pair query: $d(\langle 1, 1\rangle, \langle 7, 7\rangle)$

Existing Algorithmic Approach

- Construct the product graph G from the local components G_1, G_2
- Compute transitive closure (all-pairs) on G
  - Warshall-Floyd-Kleene algorithm, cubic complexity
  - $O\left(\left(n^2\right)^3\right) = O\left(n^6\right)$

Existing Algorithmic Approach

- Construct the product graph G from the local components G_1, G_2
- Compute transitive closure (all-pairs) on G
  - Warshall-Floyd-Kleene algorithm, cubic complexity
  - $O\left(\left(n^2\right)^3\right) = O\left(n^6\right)$

Our Improvements for Semiring Distances

- Component graphs have special structure, i.e., low-treewidth graphs
  - Measures similarity of a graph to a tree
  - Well-known property of control-flow graphs

$$\downarrow$$

A faster algorithm for the transitive closure $O\left(n^6\right) \rightarrow O\left(n^{4+\epsilon}\right)$

Our Improvements for Semiring Distances

- Component graphs have special structure, i.e., low-treewidth graphs
  - Measures similarity of a graph to a tree
  - Well-known property of control-flow graphs

$$\downarrow$$

A faster algorithm for the transitive closure $O\left(n^6\right) \to O\left(n^{4+\epsilon}\right)$

Our Improvements for Semiring Distances

- On demand analysis: *preprocess* vs *query*

Preprocessing spectrum

No Preprocessing

Transitive Closure

- Answering a few queries does not require the transitive closure
- On demand analysis: preprocess more only if expecting many queries

Our Improvements for Semiring Distances

- On demand analysis: *preprocess* vs *query*

Preprocessing spectrum

No Preprocessing

Transitive Closure

- Answering a few queries does not require the transitive closure
- On demand analysis: preprocess more only if expecting many queries

Our Improvements for Semiring Distances

- 2 components: product size $n \times n$, transitive closure has n^4 entries
- Existing $O(n^6)$ time for transitive closure

Preprocessing spectrum

No Preprocessing

Transitive Closure

- (1): Transitive closure almost optimal
- (3): Conditionally optimal

Our Improvements for Semiring Distances

- 2 components: product size $n \times n$, transitive closure has n^4 entries
- Existing $O(n^6)$ time for transitive closure

Preprocessing spectrum

(1) $n^{4+\epsilon} + i$

$n^{3+\epsilon}$

No Preprocessing

Transitive Closure

- (1): Transitive closure almost optimal
- (3): Conditionally optimal

Our Improvements for Semiring Distances

- 2 components: product size $n \times n$, transitive closure has n^4 entries
- Existing $O(n^6)$ time for transitive closure

Preprocessing spectrum

	(2) $n^{3+\epsilon} + i \cdot n$	(1) $n^{4+\epsilon} + i$

$n^{1+\epsilon}$ $n^{3+\epsilon}$

No Preprocessing

Transitive Closure

- (1): Transitive closure almost optimal
- (3): Conditionally optimal

Our Improvements for Semiring Distances

- 2 components: product size $n \times n$, transitive closure has n^4 entries
- Existing $O(n^6)$ time for transitive closure

Preprocessing spectrum

(3) $n^3 + i \cdot n^2$	(2) $n^{3+\epsilon} + i \cdot n$	(1) $n^{4+\epsilon} + i$

$n^{1+\epsilon}$ $n^{3+\epsilon}$

No Preprocessing Transitive Closure

- (1): Transitive closure almost optimal
- (3): Conditionally optimal

Our Improvements for Semiring Distances

- 2 components: product size $n \times n$, transitive closure has n^4 entries
- Existing $O(n^6)$ time for transitive closure

Preprocessing spectrum

(3) $n^3 + i \cdot n^2$	(2) $n^{3+\epsilon} + i \cdot n$	(1) $n^{4+\epsilon} + i$

$n^{1+\epsilon}$ $n^{3+\epsilon}$

No Preprocessing

Transitive Closure

- (1): Transitive closure almost optimal
- (3): Conditionally optimal

Our Improvements for Semiring Distances

- 2 components: product size $n \times n$, transitive closure has n^4 entries
- Existing $O(n^6)$ time for transitive closure

Preprocessing spectrum

(3) $n^3 + i \cdot n^2$	(2) $n^{3+\epsilon} + i \cdot n$	(1) $n^{4+\epsilon} + i$

$n^{1+\epsilon}$ $n^{3+\epsilon}$

No Preprocessing

Transitive Closure

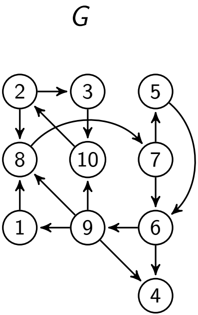
- (1): Transitive closure almost optimal
- (3): Conditionally optimal

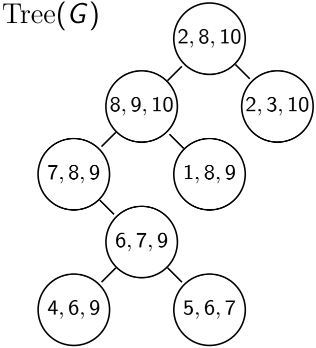
Outline

- Tree decompositions
- Treewidth of the concurrent system
- On demand analysis on the concurrent tree-decomposition
- Experimental results

Definition (Tree decomposition)

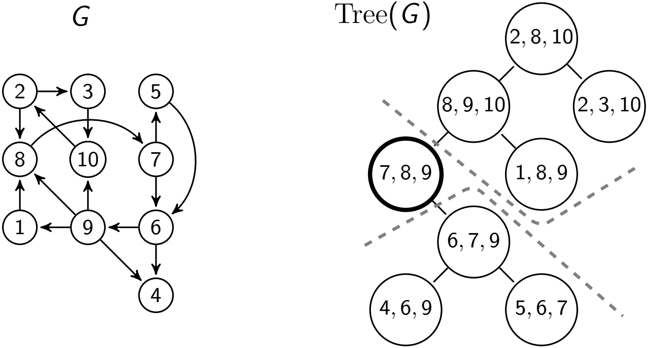
Given a graph $G = (V, E)$, a **tree-decomposition** $\mathrm{Tree}(G) = (V_T, E_T)$ is a tree of bags $B_i \subseteq V$





Definition (Tree decomposition)

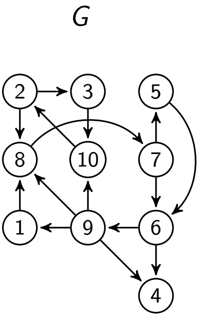
Given a graph $G = (V, E)$, a **tree-decomposition** $\mathrm{Tree}(G) = (V_T, E_T)$ is a tree of bags $B_i \subseteq V$

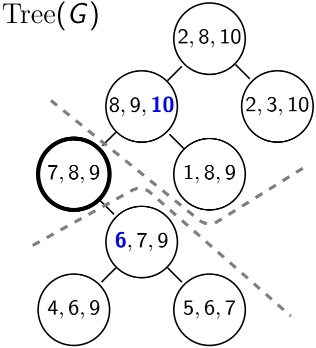


Tree Decompositions

Definition (Tree decomposition)

Given a graph $G = (V, E)$, a **tree-decomposition** $\mathrm{Tree}(G) = (V_T, E_T)$ is a tree of bags $B_i \subseteq V$

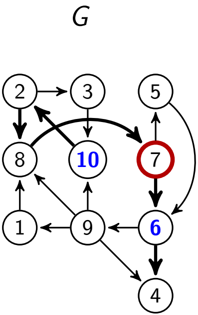


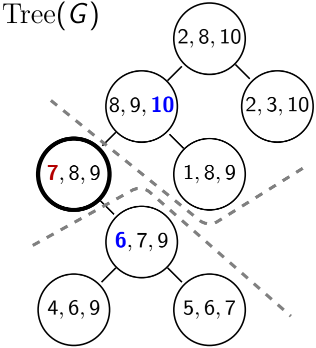


Tree Decompositions

Definition (Tree decomposition)

Given a graph $G = (V, E)$, a **tree-decomposition** $\mathrm{Tree}(G) = (V_T, E_T)$ is a tree of bags $B_i \subseteq V$





Tree Decompositions

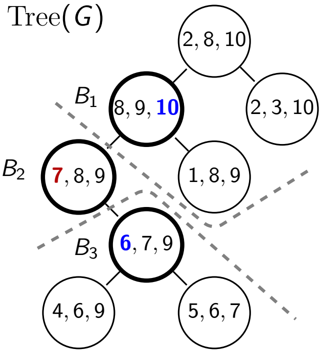
Definition (Tree decomposition)

Given a graph $G = (V, E)$, a **tree-decomposition** $\mathrm{Tree}(G) = (V_T, E_T)$ is a tree of bags $B_i \subseteq V$

$\exists\ x_1 \in B_1 \cap B_2$

$\exists\ x_2 \in B_2 \cap B_3$

d(10,6)=$d(10, x_1) \otimes d(x_1, x_2) \otimes d(x_2, 6)$



Tree Decompositions

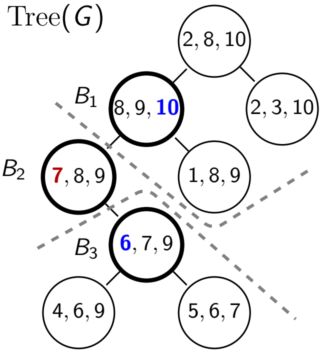
Definition (Tree decomposition)

Given a graph $G = (V, E)$, a **tree-decomposition** $\mathrm{Tree}(G) = (V_T, E_T)$ is a tree of bags $B_i \subseteq V$

$\exists\ x_1 \in B_1 \cap B_2$

$\exists\ x_2 \in B_2 \cap B_3$

d(10,6)=$d(10, x_1) \otimes d(x_1, x_2) \otimes d(x_2, 6)$

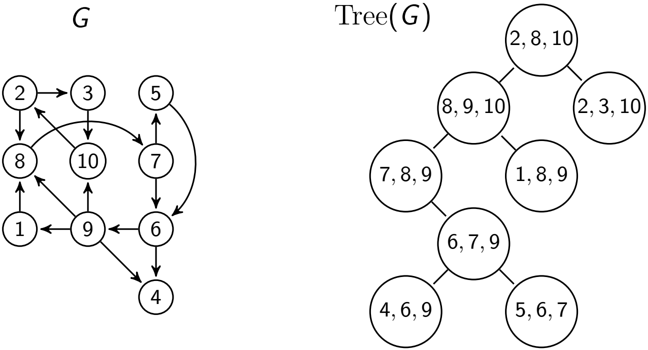


Semiring distances reduce to:

1. Tree decomposition
2. Local Distances

CFGs of typical imperative programs have tree-decompositions of small sized bags

- Theoretically, for goto-free programs
  - Pascal ≤ 4
  - C ≤ 7
- In practice small in imperative programs (e.g. Java ≤ 8)



Treewidth

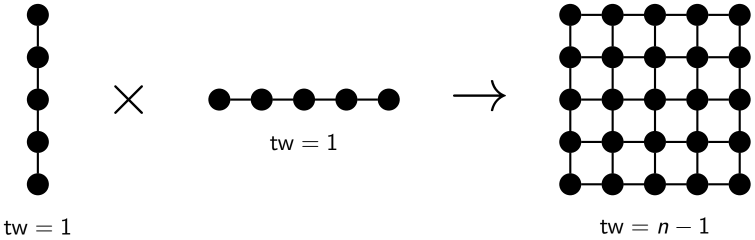
CFGs of typical imperative programs have tree-decompositions of small sized bags

- Theoretically, for goto-free programs
  - Pascal ≤ 4
  - C ≤ 7
- In practice small in imperative programs (e.g. Java ≤ 8)

Theorem (Tree decomposition)

For constant treewidth graphs, $\mathrm{Tree}(G)$ *can be constructed in* $O(n)$ *time.*

Treewidth of the Concurrent System

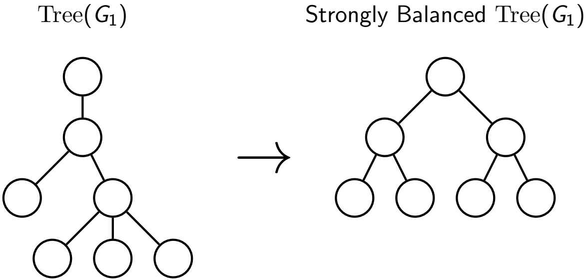


- Components of small treewidth can yield a concurrent system G of very large treewidth!
- Computing an optimal tree decomposition of G is intractable (NP-C)

Strongly Balanced Tree Decompositions

Convert the local tree decompositions to

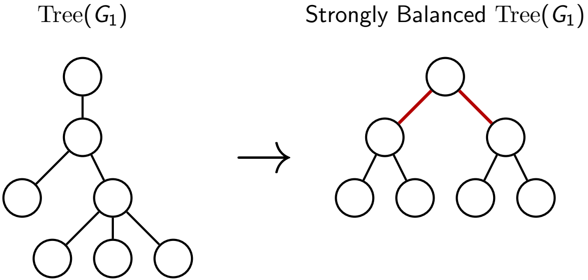
- **Binary:** every bag has two children
- **Strongly balanced:** most bags have two subtrees of approximately equal size



Strongly Balanced Tree Decompositions

Convert the local tree decompositions to

- **Binary:** every bag has two children
- **Strongly balanced:** most bags have two subtrees of approximately equal size

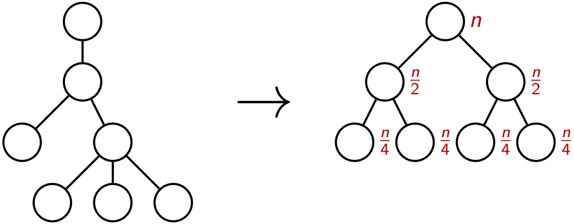


Strongly Balanced Tree Decompositions

Convert the local tree decompositions to

- **Binary:** every bag has two children
- **Strongly balanced:** most bags have two subtrees of approximately equal size

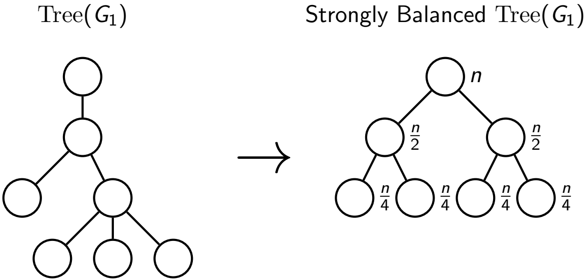
$\mathrm{Tree}(G_1)$ Strongly Balanced $\mathrm{Tree}(G_1)$



Strongly Balanced Tree Decompositions

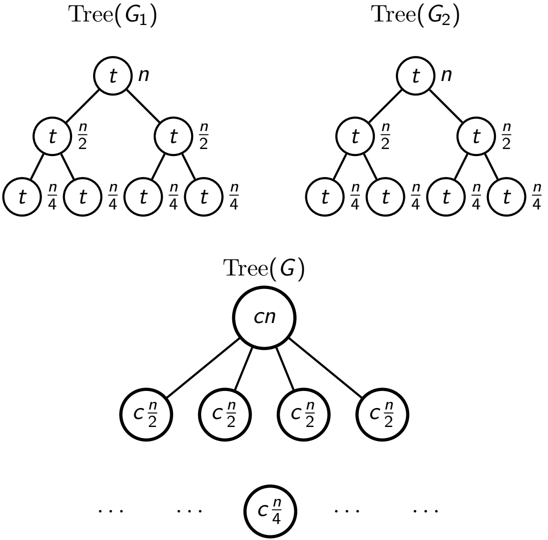
Convert the local tree decompositions to

- **Binary:** every bag has two children
- **Strongly balanced:** most bags have two subtrees of approximately equal size



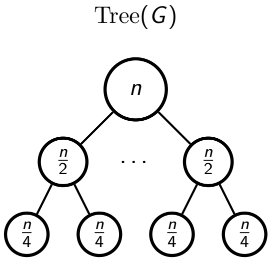
Tree Decomposition of the Concurrent System

$t = O(1)$



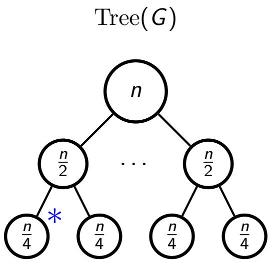
On Demand Analysis: Local Distances

For every two nodes u, v appearing in a bag, compute $d(u, v)$



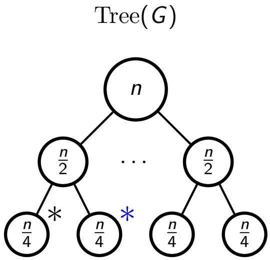
- Transitive closure on the bags instead of the whole system
- Cost decreases geometrically on the levels
  - Transitive closure on the root dominates
  - $O(n^6) \rightarrow O(n^3)$

For every two nodes u, v appearing in a bag, compute $d(u, v)$



- Transitive closure on the bags instead of the whole system
- Cost decreases geometrically on the levels
  - Transitive closure on the root dominates
  - $O(n^6) \rightarrow O(n^3)$

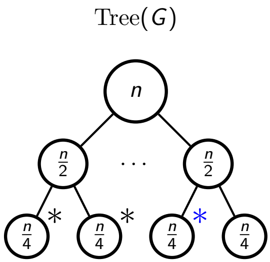
For every two nodes u, v appearing in a bag, compute $d(u, v)$



- Transitive closure on the bags instead of the whole system
- Cost decreases geometrically on the levels
  - Transitive closure on the root dominates
  - $O(n^6) \rightarrow O(n^3)$

On Demand Analysis: Local Distances

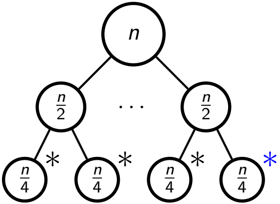
For every two nodes u, v appearing in a bag, compute $d(u, v)$



- Transitive closure on the bags instead of the whole system
- Cost decreases geometrically on the levels
  - Transitive closure on the root dominates
  - $O(n^6) \to O(n^3)$

For every two nodes u, v appearing in a bag, compute $d(u, v)$

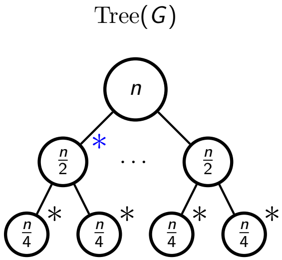
Tree(G)



- Transitive closure on the bags instead of the whole system
- Cost decreases geometrically on the levels
  - Transitive closure on the root dominates
  - $O(n^6) \rightarrow O(n^3)$

On Demand Analysis: Local Distances

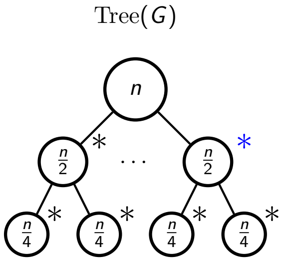
For every two nodes u, v appearing in a bag, compute $d(u, v)$



- Transitive closure on the bags instead of the whole system
- Cost decreases geometrically on the levels
  - Transitive closure on the root dominates
  - $O(n^6) \to O(n^3)$

On Demand Analysis: Local Distances

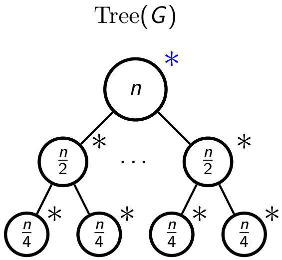
For every two nodes u, v appearing in a bag, compute $d(u, v)$



- Transitive closure on the bags instead of the whole system
- Cost decreases geometrically on the levels
  - Transitive closure on the root dominates
  - $O(n^6) \to O(n^3)$

On Demand Analysis: Local Distances

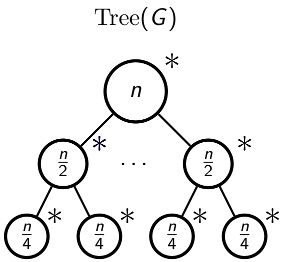
For every two nodes u, v appearing in a bag, compute $d(u, v)$



- Transitive closure on the bags instead of the whole system
- Cost decreases geometrically on the levels
  - Transitive closure on the root dominates
  - $O(n^6) \rightarrow O(n^3)$

On Demand Analysis: Local Distances

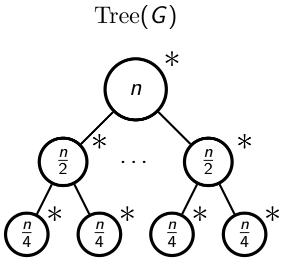
For every two nodes u, v appearing in a bag, compute $d(u, v)$



- Transitive closure on the bags instead of the whole system
- Cost decreases geometrically on the levels
    - Transitive closure on the root dominates
    - $O(n^6) \to O(n^3)$

On Demand Analysis: Local Distances

For every two nodes u, v appearing in a bag, compute $d(u, v)$



- Transitive closure on the bags instead of the whole system
- Cost decreases geometrically on the levels
    - Transitive closure on the root dominates
    - $O(n^6) \to O(n^3)$

Tradeoffs

Preprocessing spectrum

(3) $n^3 + i \cdot n^2$	(2) $n^{3+\epsilon} + i \cdot n$	(1) $n^{4+\epsilon} + i$

$n^{1+\epsilon}$ $\qquad$ $n^{3+\epsilon}$

No Preprocessing $\qquad$ Transitive Closure

Experiments

- Control-flow graphs (CFGs) of methods from java libraries and benchmarks
- Focus on algorithmic comparison with standard transitive closure algorithms
- Used the tropical min-plus semiring on $\mathbb{R} \cup \{\infty\}$

Experiments 1

- CFGs from the DaCapo suit
- n nodes each CFG
- 2-self product: size $n \times n$

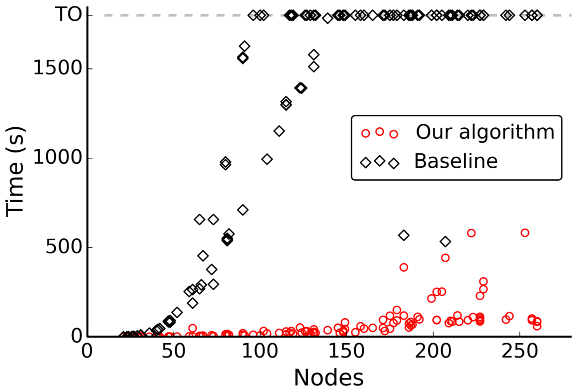
- Random weights in $[-10^3, 10^3]$
- Compute the transitive closure
- Baseline: Bellman-Ford

Experiments 1

- CFGs from the DaCapo suit
- n nodes each CFG
- 2-self product: size $n \times n$
- Random weights in $[-10^3, 10^3]$
- Compute the transitive closure
- Baseline: Bellman-Ford

Experiments 1

- CFGs from the DaCapo suit
- n nodes each CFG
- 2-self product: size $n \times n$
- Random weights in $[-10^3, 10^3]$
- Compute the transitive closure
- Baseline: Bellman-Ford



Experiments 2

- CFGs from container methods in java.util.concurrent
- Bloated with values of locks
- 2-self product: size $n \times n$

- Random weights in $[-10^3, 10^3]$
- Transitive closure

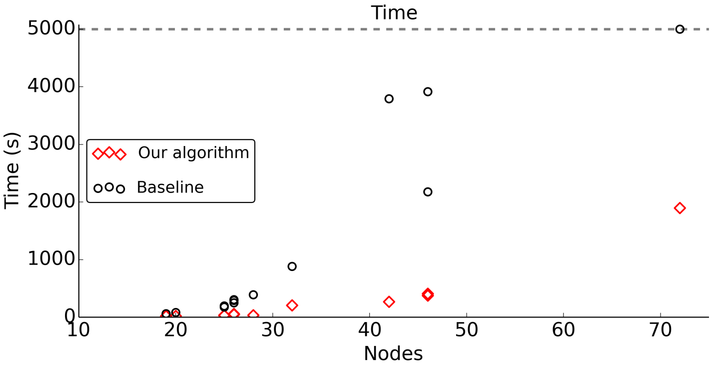
Experiments 2

- CFGs from container methods in java.util.concurrent
- Bloated with values of locks
- 2-self product: size $n \times n$

- Random weights in $[-10^3, 10^3]$
- Transitive closure

Experiments 2

- CFGs from container methods in java.util.concurrent
- Bloated with values of locks
- 2-self product: size $n \times n$
- Random weights in $[-10^3, 10^3]$
- Transitive closure



Thank you!

Questions?

Experiments 2

- CFGs from container methods in java.util.concurrent
- Bloated with values of locks
- 2-self product: size $n \times n$
- Random weights in $[-10^3, 10^3]$
- Transitive closure times:
  - $\mathbf{T_o(s)}$: our algorithm
  - $\mathbf{T_b(s)}$: baseline (Bellman-Ford)

Java method	**n**	$\mathbf{T_o(s)}$	$\mathbf{T_b(s)}$
ArrayBlockingQueue: poll	19	19	60
ArrayBlockingQueue: peek	20	20	81
LinkedBlockingDeque: advance	25	29	195
PriorityBlockingQueue: removeEQ	25	32	176
ArrayBlockingQueue: init	26	47	249
LinkedBlockingDeque: remove	26	49	290
ArrayBlockingQueue: offer	26	56	304
ArrayBlockingQueue: clear	28	33	389
ArrayBlockingQueue: contains	32	205	881
DelayQueue: remove	42	267	3792
ConcurrentHashMap: scanAndLockForPut	46	375	2176
ArrayBlockingQueue: next	46	407	3915
ConcurrentHashMap: put	72	1895	> 8 h

Results

2 components

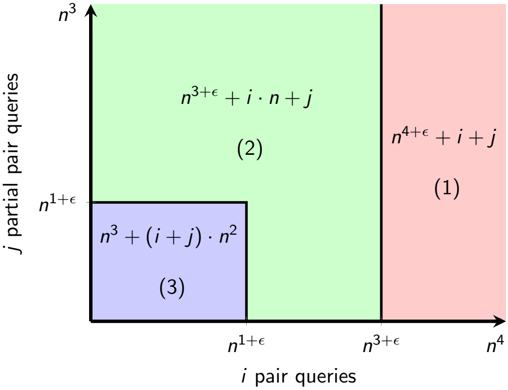
	Preprocess		Query time		
	Time	Space	Single-source	Pair	Partial pair
Existing	$O(n^6)$	$O(n^4)$	$O(n^2)$	$O(1)$	$O(1)$
Our result $(\epsilon > 0)$	$\mathbf{O(n^3)}$	$\mathbf{O(n^{2+\epsilon})}$	$\mathbf{O(n^{2+\epsilon})}$	$\mathbf{O(n^2)}$	$\mathbf{O(n^2)}$
Our result $(\epsilon > 0)$	$\mathbf{O(n^{3+\epsilon})}$	$\mathbf{O(n^3)}$	$\mathbf{O(n^{2+\epsilon})}$	$\mathbf{O(n)}$	$\mathbf{O(1)}$
Our result $(\epsilon > 0)$	$\mathbf{O(n^{4+\epsilon})}$	$\mathbf{O(n^4)}$	$\mathbf{O(n^2)}$	$\mathbf{O(1)}$	$\mathbf{O(1)}$

$k \geq 3$ components

	Preprocess		Query time		
	Time	Space	Single-source	Pair	Partial pair
Existing	$O(n^{3k})$	$O(n^{2k})$	$O(n^k)$	$O(1)$	$O(1)$
Our result	$\mathbf{O(n^{3k-3})}$	$\mathbf{O(n^{2k-1})}$	$\mathbf{O(n^{2(k-1)})}$	$\mathbf{O(n^{k-1})}$	$\mathbf{O(1)}$
Our result	$\mathbf{O(n^{3k-2})}$	$\mathbf{O(n^{2k})}$	$\mathbf{O(n^k)}$	$\mathbf{O(1)}$	$\mathbf{O(1)}$

Results (2 Components)

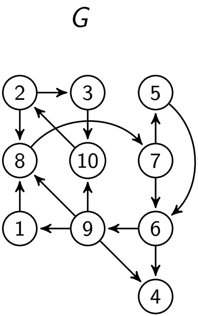
- 2 components: product size $n \times n$, transitive closure has n^4 entries
- Existing $O(n^6)$ time for transitive closure

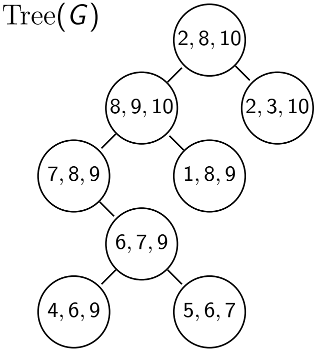


Definition (Tree decomposition)

Given a graph $G = (V, E)$, a **tree-decomposition** $\mathrm{Tree}(G) = (V_T, E_T)$ is a tree of bags $B_i \subseteq V$ such that:

1. Every node of G is contained in a bag
2. Every edge of G is contained in a bag
3. Every node of G appears in a contiguous subtree of $\mathrm{Tree}(G)$.





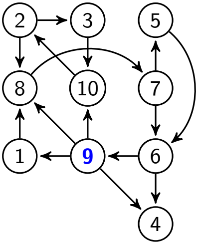
Tree Decompositions

Definition (Tree decomposition)

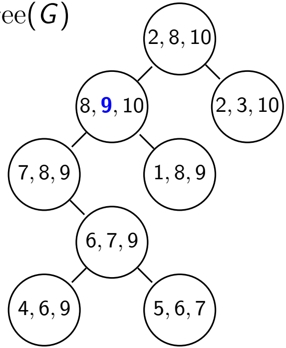
Given a graph $G = (V, E)$, a **tree-decomposition** $\mathrm{Tree}(G) = (V_T, E_T)$ is a tree of bags $B_i \subseteq V$ such that:

1. Every node of G is contained in a bag
2. Every edge of G is contained in a bag
3. Every node of G appears in a contiguous subtree of $\mathrm{Tree}(G)$.

G



$\mathrm{Tree}(G)$



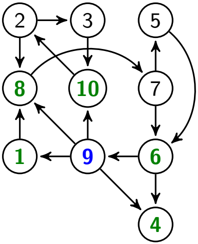
Tree Decompositions

Definition (Tree decomposition)

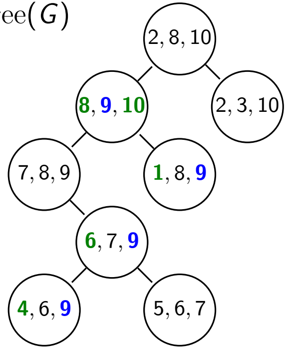
Given a graph $G = (V, E)$, a **tree-decomposition** $\mathrm{Tree}(G) = (V_T, E_T)$ is a tree of bags $B_i \subseteq V$ such that:

1. Every node of G is contained in a bag
2. Every edge of G is contained in a bag
3. Every node of G appears in a contiguous subtree of $\mathrm{Tree}(G)$.

G



$\mathrm{Tree}(G)$

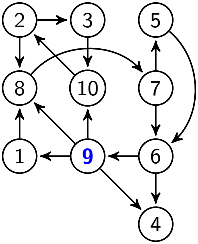


Definition (Tree decomposition)

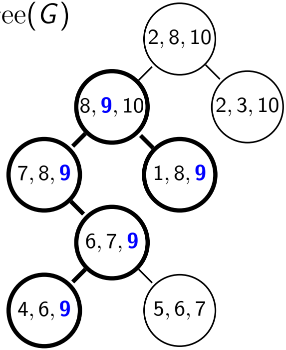
Given a graph $G=(V,E)$, a **tree-decomposition** $\mathrm{Tree}(G)=(V_T,E_T)$ is a tree of bags $B_i \subseteq V$ such that:

1. Every node of G is contained in a bag
2. Every edge of G is contained in a bag
3. Every node of G appears in a contiguous subtree of $\mathrm{Tree}(G)$.

G



$\mathrm{Tree}(G)$



1. Any graph G can be constructed as the product of two constant-treewidth graphs G_1, G_2
2. $\implies$ Semiring distance on the product of G_1, G_2 as hard as on G
3. Widely conjectured $\Omega(n^3)$ lower-bound