







Efficient Parallel Subgraph Counting using G-Tries

Pedro Ribeiro, Fernando Silva and Luís Lopes

CRACS & INESC-Porto LA
University of Porto, Portugal



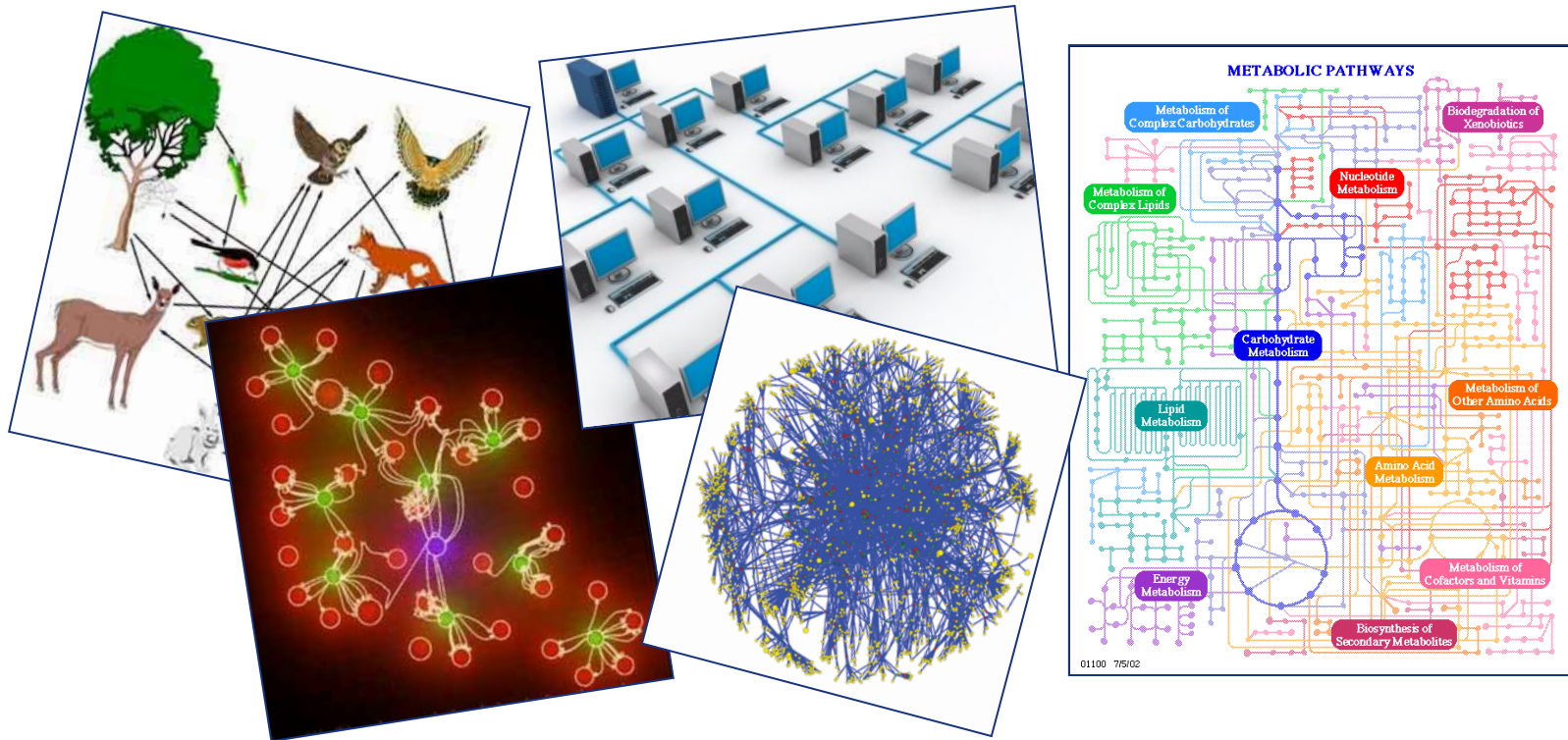
Contents

-  Motivation: Networks and Motifs
-  Subgraph Counting Problem
-  The G-Trie Data Structure
-  Parallel Algorithm
-  Results
-  Conclusion and Future Work

Complex Networks

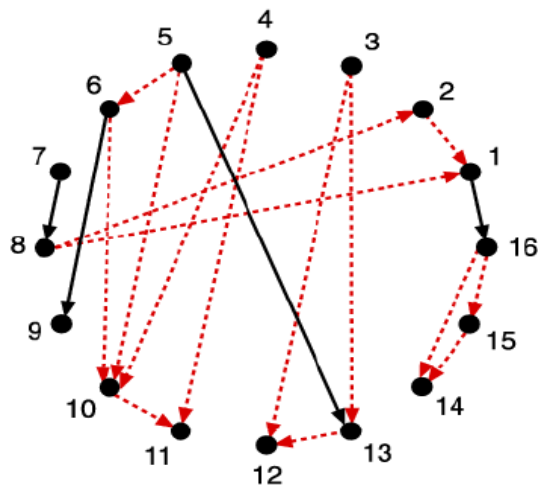
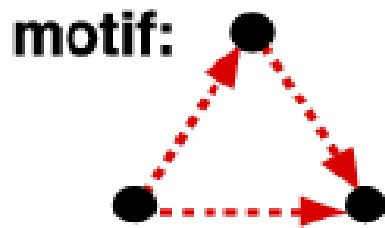
❖ Networks are everywhere!

- Biology, Physics, Mathematics, Sociology, Engineering, Computer Science, Etc

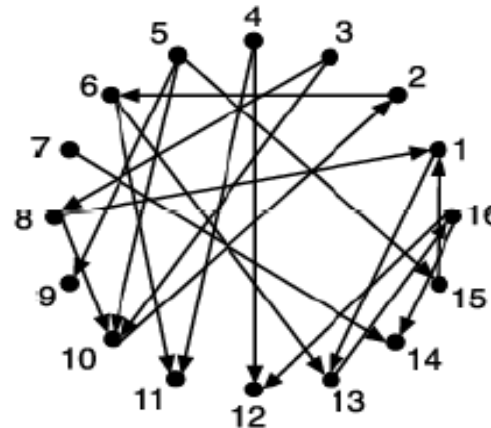
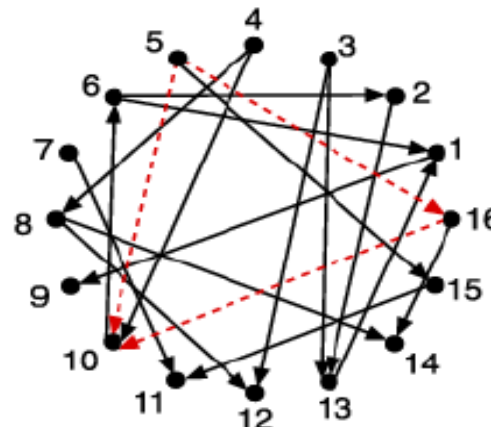


Network Motifs Concept

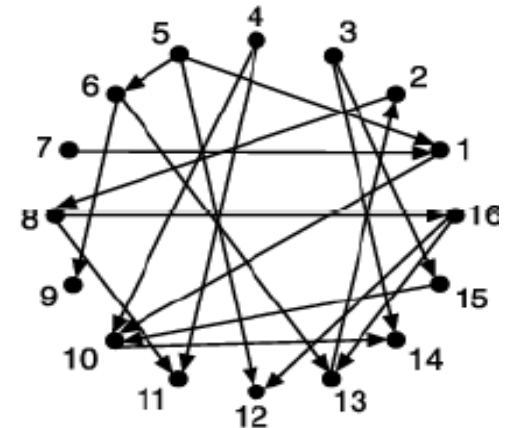
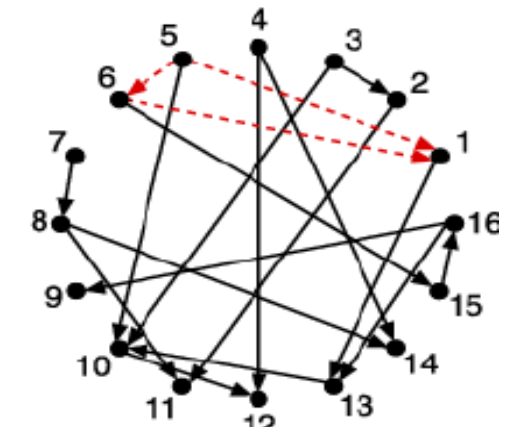
❖ Overrepresented Subgraphs



Original Network



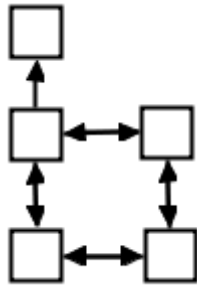
Similar Random Networks



Subgraph Concepts

❖ Counting

- Allow overlapping



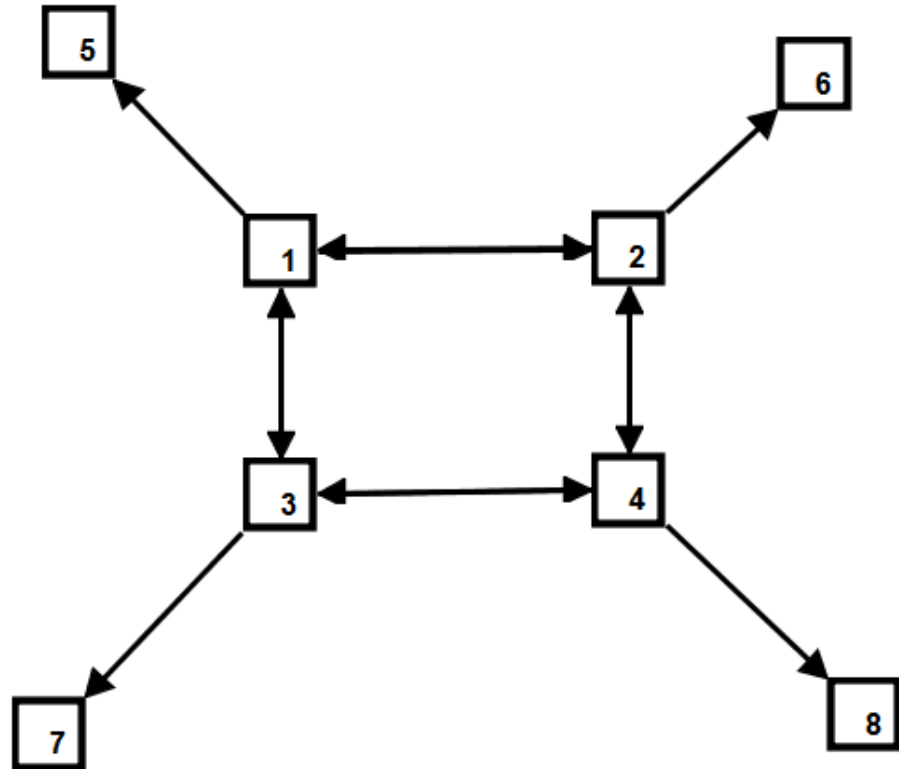
4 occurrences:

$\{1,2,3,4,5\}$

$\{1,2,3,4,6\}$

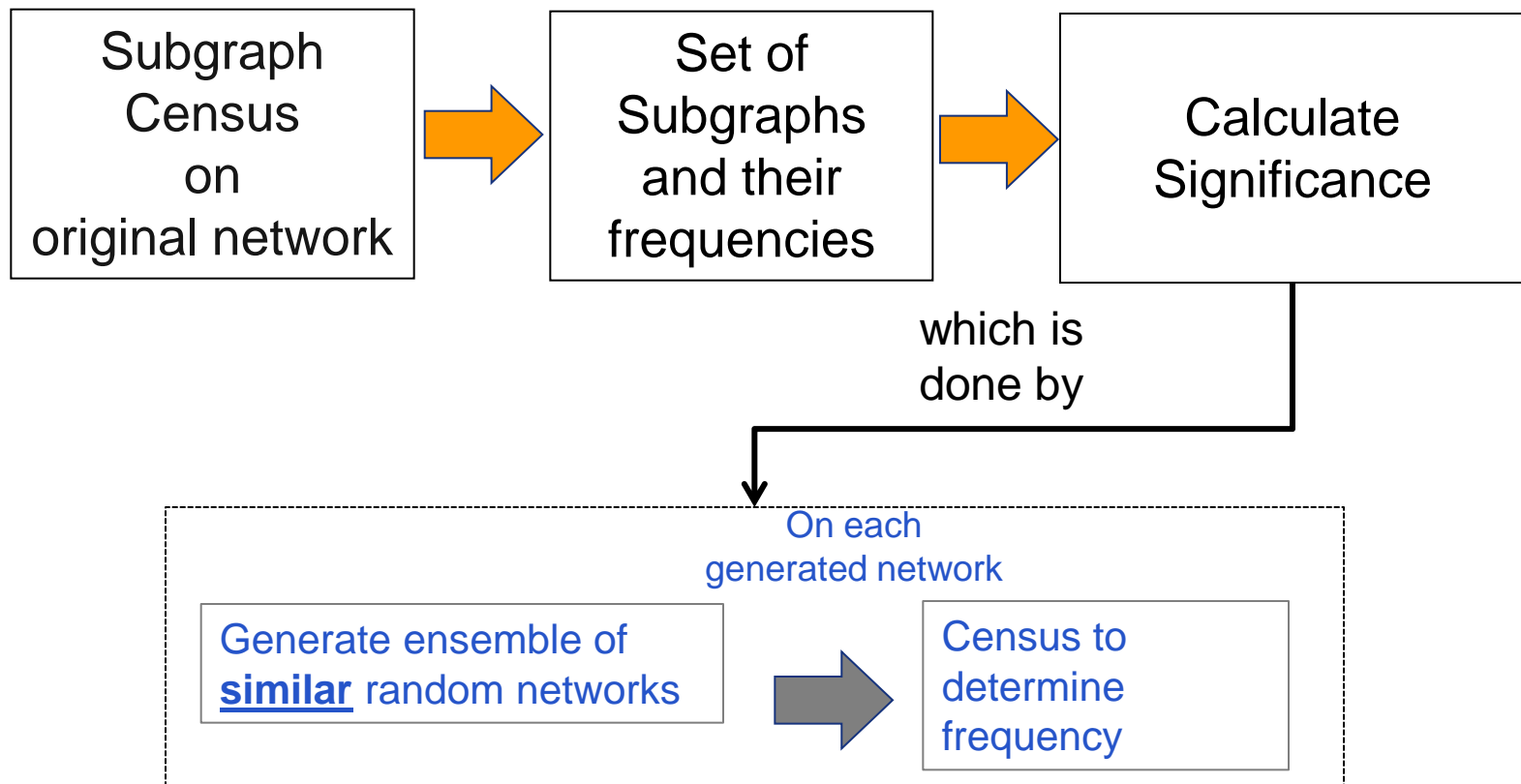
$\{1,2,3,4,7\}$

$\{1,2,3,4,8\}$



Motif Discovery

❖ How to discover network motifs?

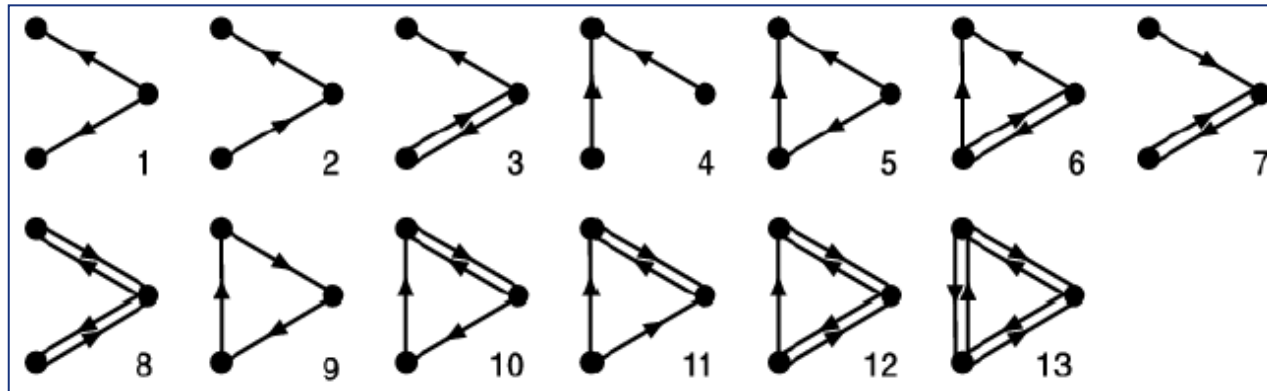


Subgraph Census

❖ **Census is the bottleneck (>95% time)**

❖ **General Problem Definition:**

- **Input:** list of subgraphs S and a bigger graph G
- **Output:** frequency count of each subgraph of S in G



Adapted from (Milo et al 2002)

Algorithms for Frequency Count

❖ Two main approaches

- **Network-centric:** enumerate all subgraphs
- **Graph-centric:** match a single subgraph

❖ Problem is computationally hard

- Execution time grows exponentially with motif size and network size

❖ Improved methods are needed

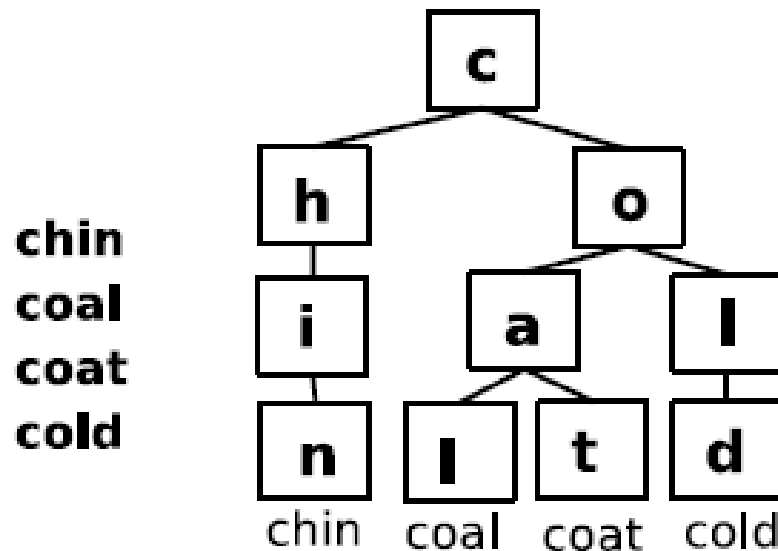
- Current results limited by required time



(Robbert van der Steeg)

G-Tries: Motivation and Concept

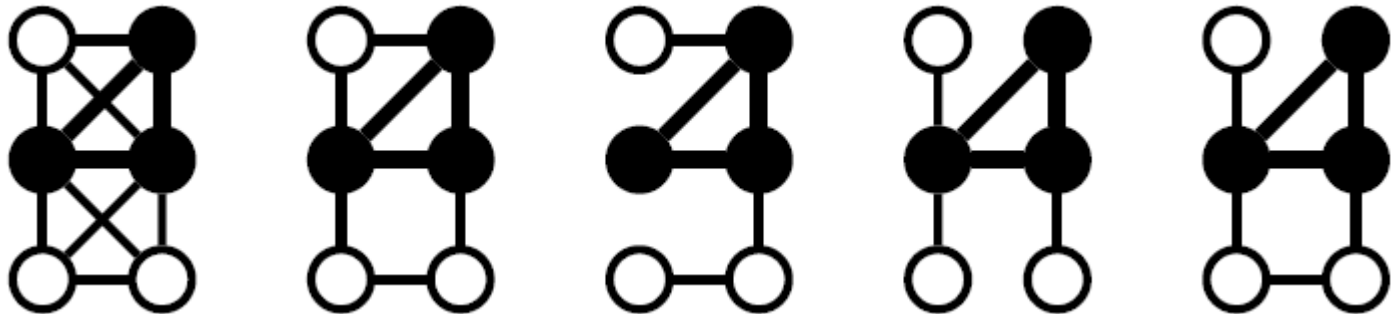
❖ Sequences and prefix trees



❖ Can the concept be extended?

G-Tries: Motivation and Concept

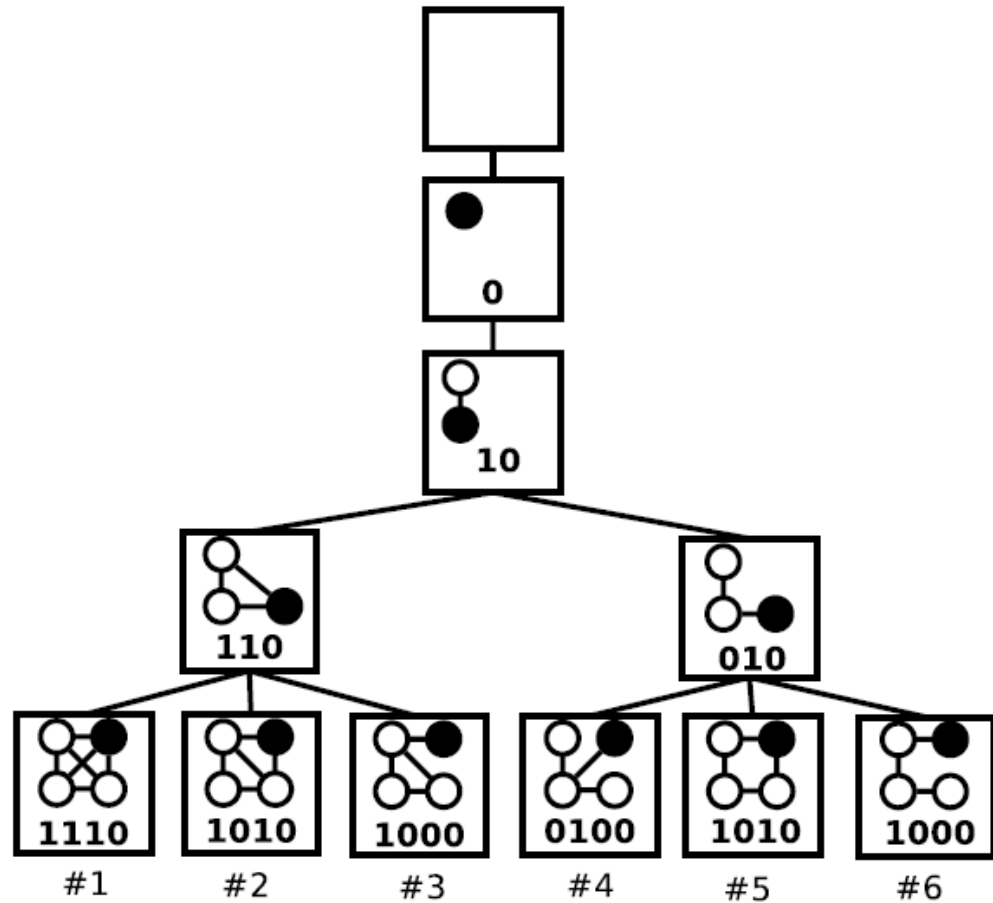
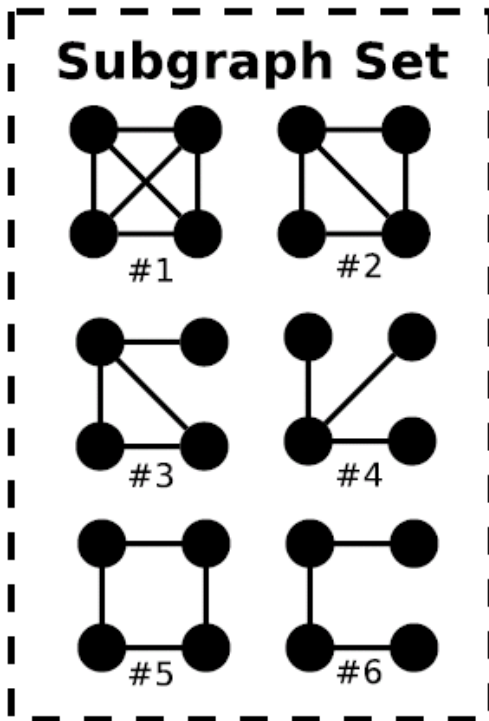
❖ **Subgraphs have common substructure!**



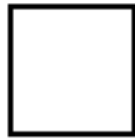
❖ **Create a tree where each tree node corresponds to a single graph vertex**

- **G-Tries** (etymology “Graph reTRIEval”)

G-Trie Example

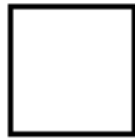


G-Tries: Creation



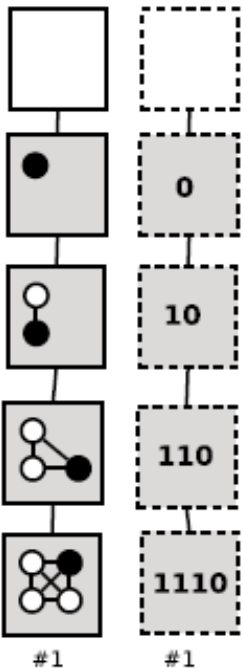
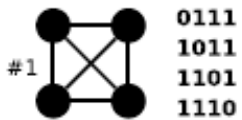
Start with an **empty g-trie**

G-Tries: Creation



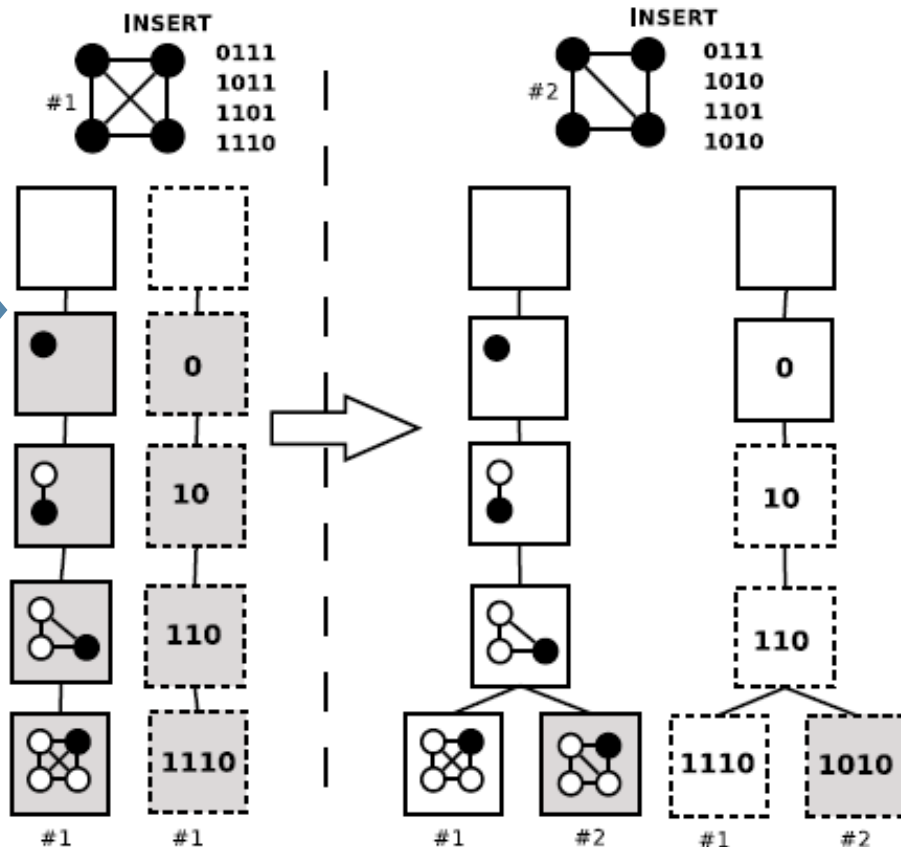
Start with an **empty g-trie**

INSERT

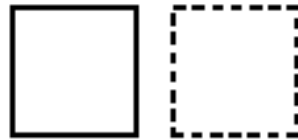


G-Tries: Creation

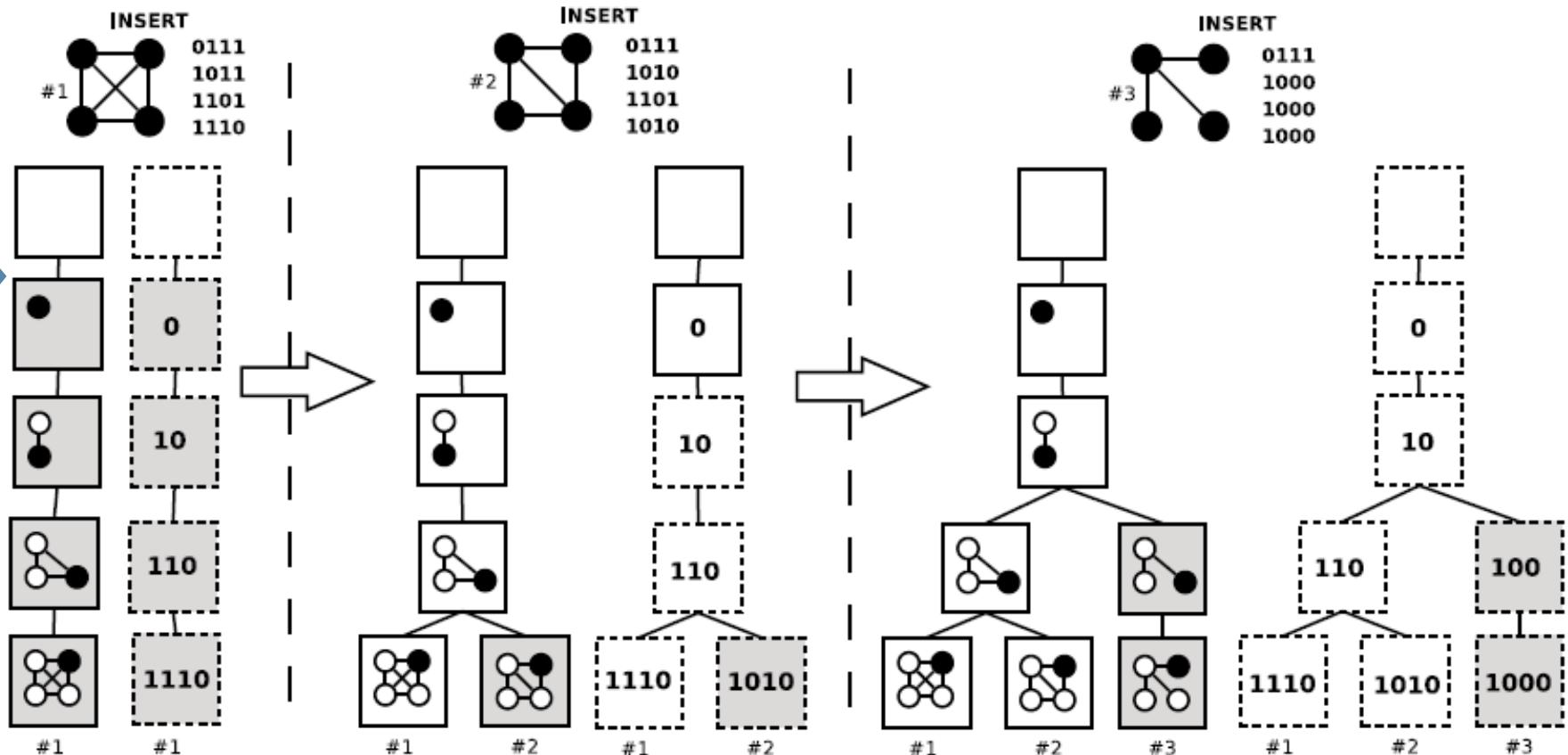
Start with an empty g-trie



G-Tries: Creation



Start with an **empty g-trie**



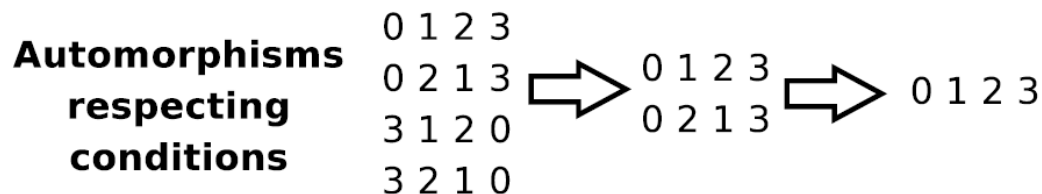
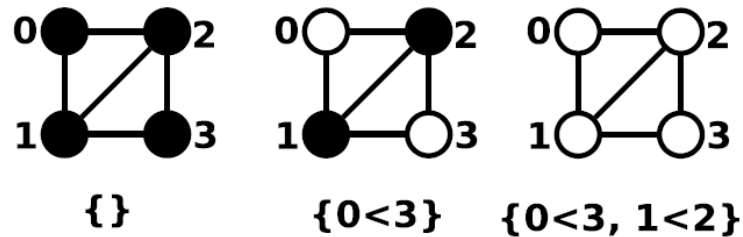
G-Tries: overview of details

❖ Many representations for the same subgraph: automorphisms

- Use **canonical representation** that maximizes common substructure

❖ Symmetry Breaking conditions

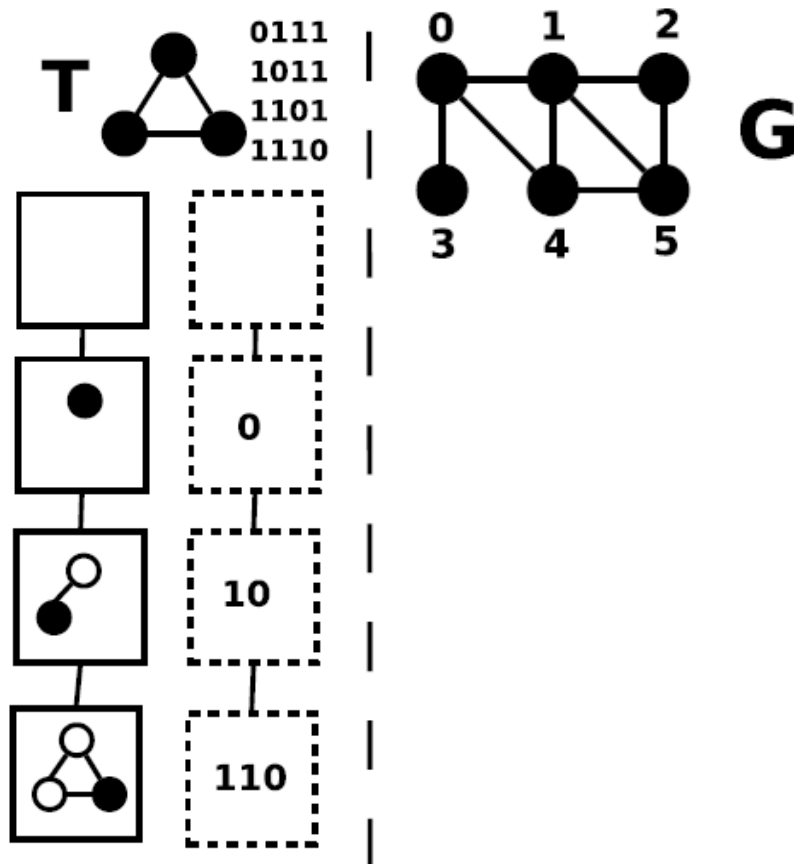
- Each subgraph is found only once



G-Tries: overview of details

❖ Matching algorithm with backtracking

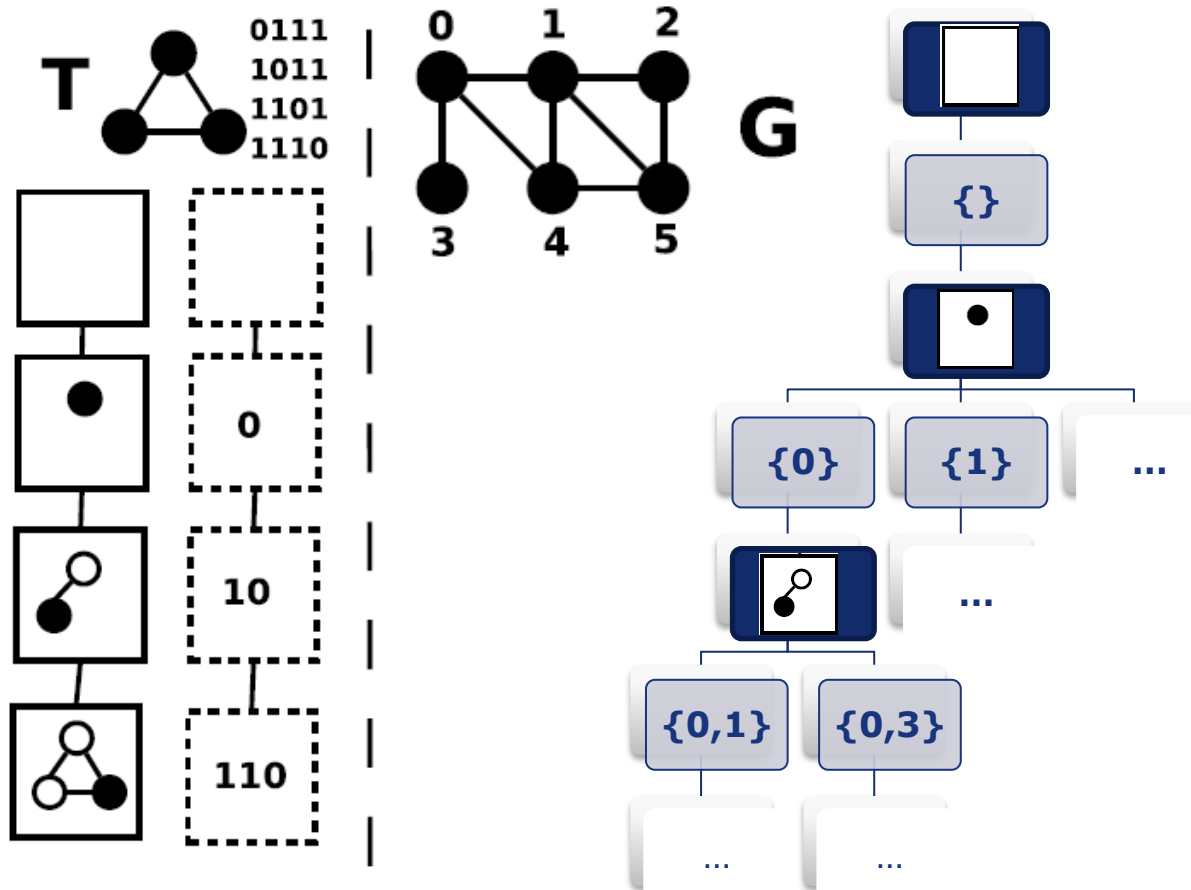
- Creating set of subgraphs following a g-trie path



G-Tries: overview of details

❖ Matching algorithm with backtracking

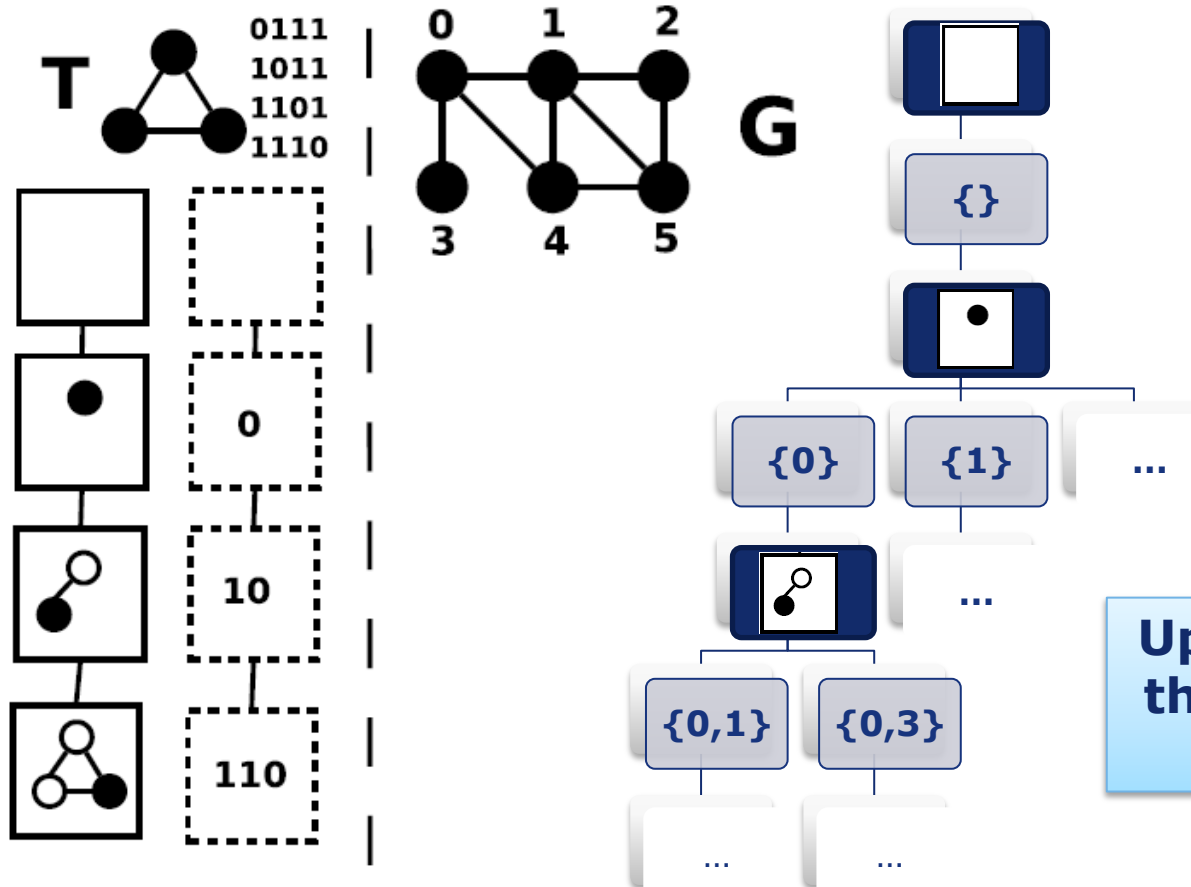
- Creating set of subgraphs following a g-trie path



G-Tries: overview of details

❖ Matching algorithm with backtracking

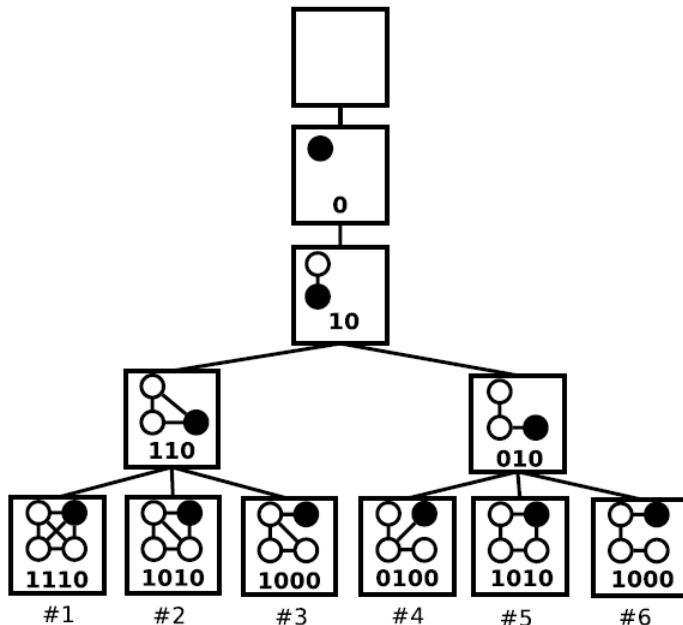
- Creating set of subgraphs following a g-trie path



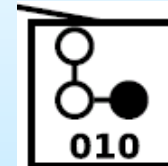
**Up to 100x times faster
than previous methods
(exact count)**

Opportunities for Parallelization

- ❖ Sequential algorithm produces a **tree-shaped search space**
- ❖ Search tree nodes are **independent from each other!**



$\{0,1,3\}$



If we know where we are,
we can continue from there

Tree Nodes -> Work Units

Parallel Problem Definition

- ❖ **Goal:** efficiently distribute work units among processors
- ❖ **Target:** distributed memory with message passing
- ❖ **Constraints:** Tree highly unbalanced
 - Pre-determined static allocation is hard!
 - Requires dynamic load balancing

Receiver-Initiated Strategy

1. Get initial share of work units

2. While computation not ended

- If work units available
 - Process work unit
 - **Someone asked for work?**
 - » Stop my computation and store search position
 - » Divide work in two halves
 - » Send half to requester
 - » Return to computation
- Else
 - Request work units from other processor

3. Aggregate Results

Initial Share of Work

❖ **“Equal” number of vertices for each processor**

**Graph with
15 vertices,
6 processors**

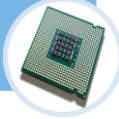
Initial Share of Work

❖ “Equal” number of vertices for each processor

Graph with
15 vertices,
6 processors

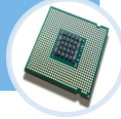
• 0,6,12

Processor
1



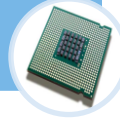
• 1,7,13

Processor
2



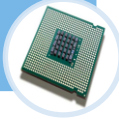
• 2,8,14

Processor
3



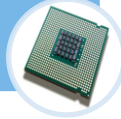
• 3,9,15

Processor
4



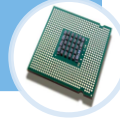
• 4,10

Processor
5



• 5,11

Processor
6



Round Robin
Strategy

Work Requests

- ❖ **Check for work request messages every time a threshold is reached**
 - Time Spent
 - Number of Units processed

- ❖ **A work request stops computation**

Running Computation

Example Computation



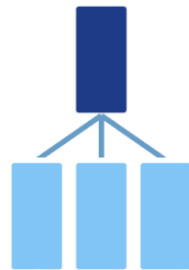
G-Trie Node



Graph Vertex

Running Computation

Example Computation



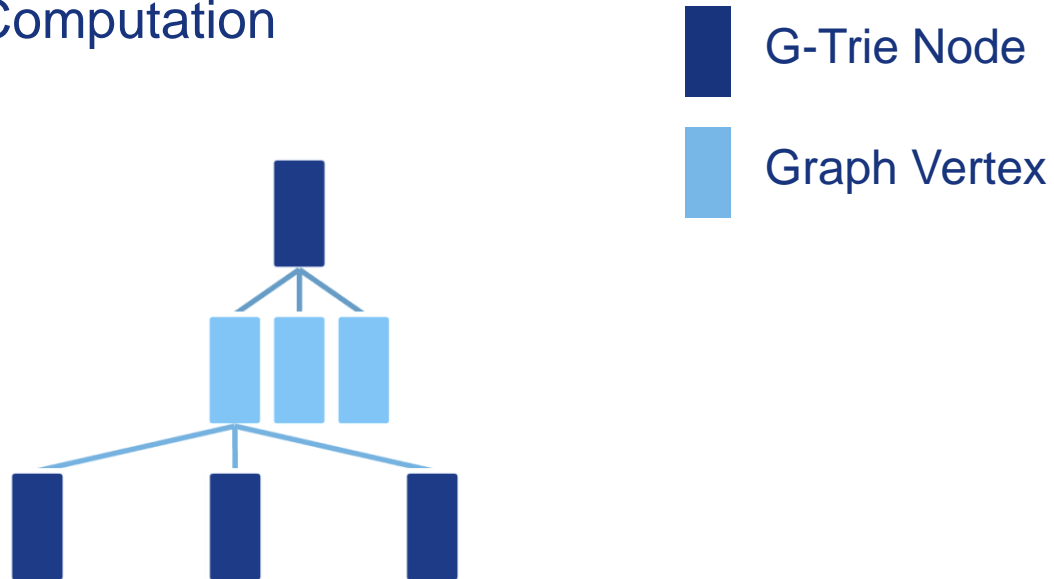
G-Trie Node



Graph Vertex

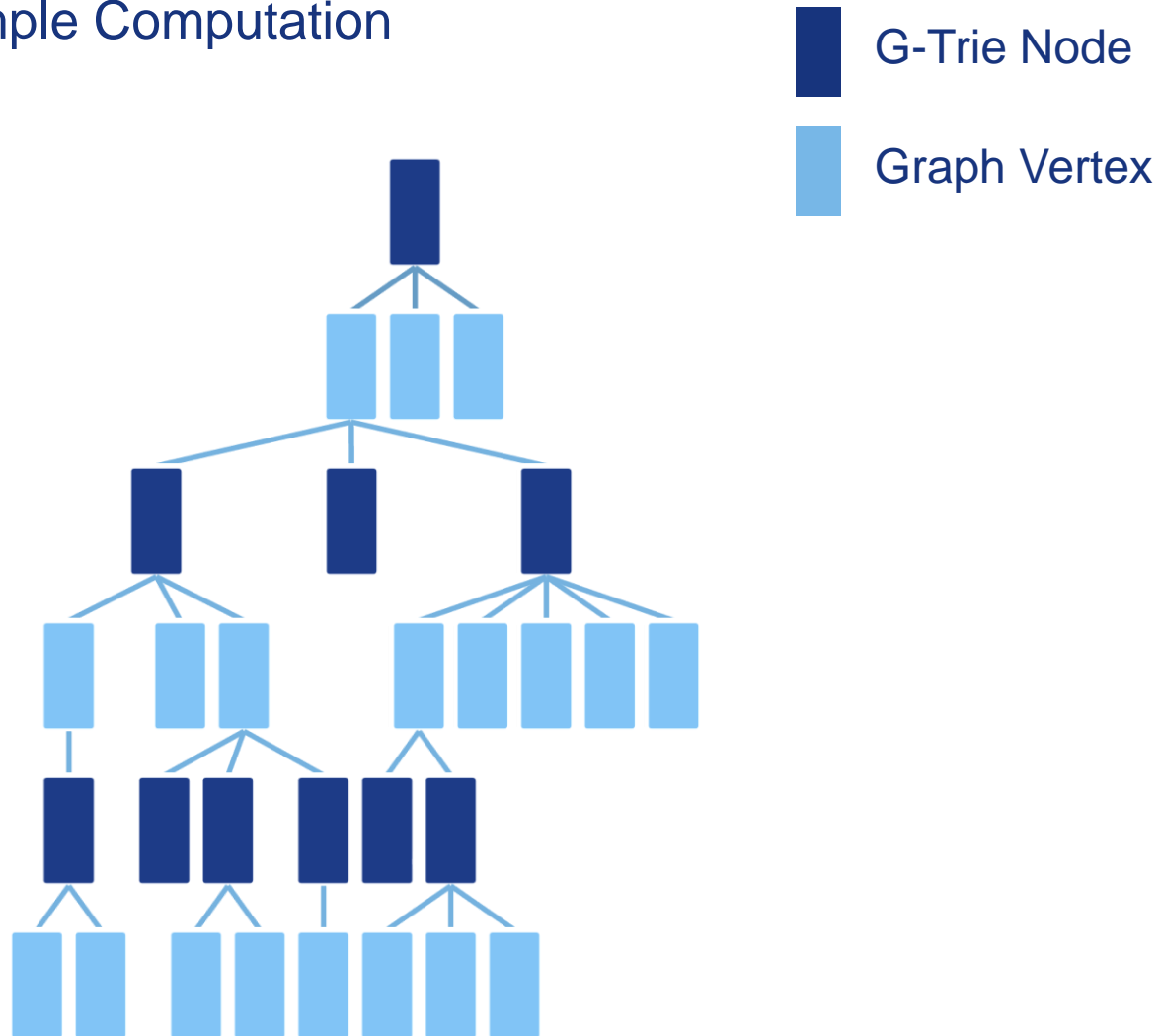
Running Computation

Example Computation



Running Computation

Example Computation

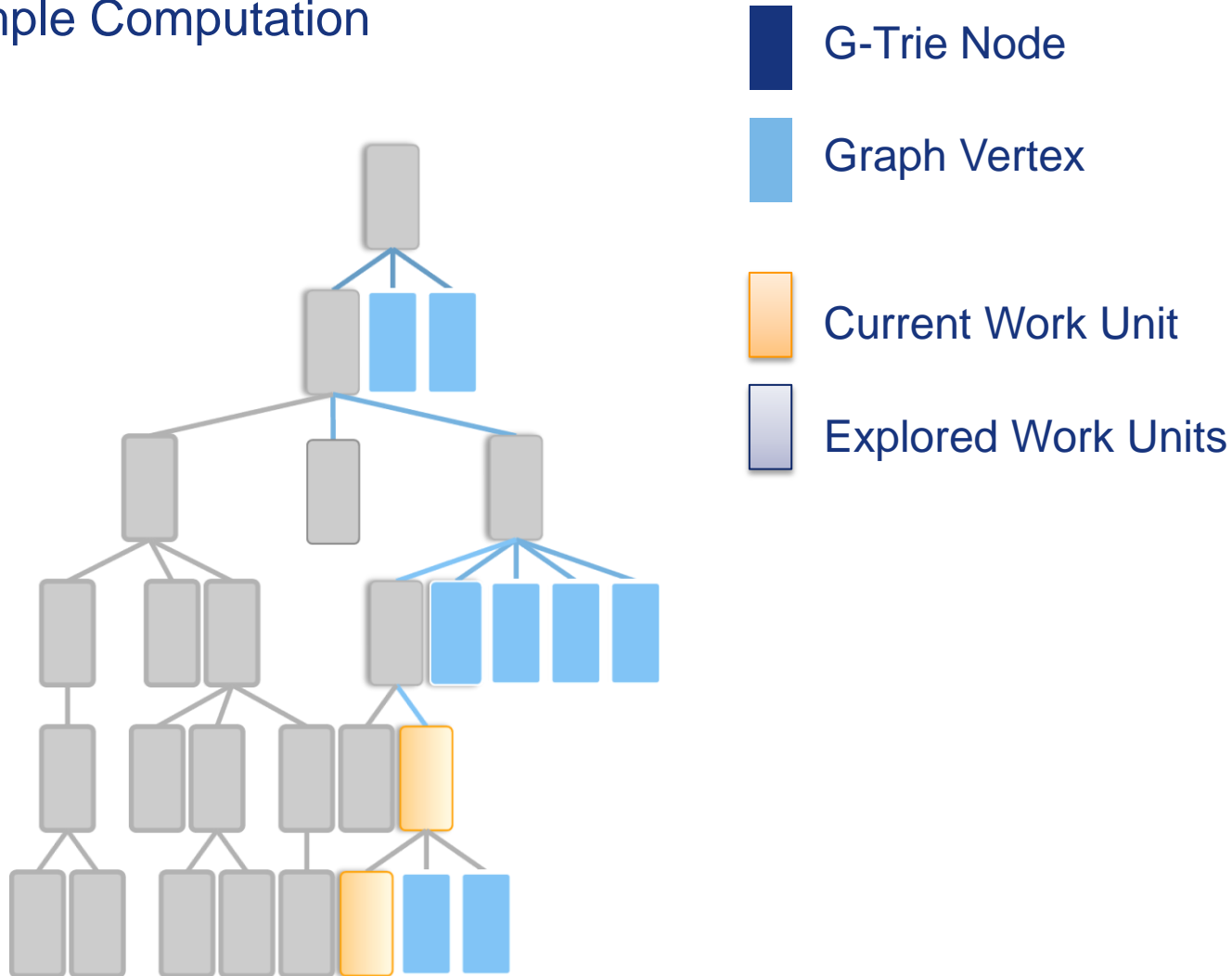


Dividing Computation

- ❖ **Goal:** divide work in two halves
- ❖ **We do a diagonal split**
 - **Keep** current and even unexplored graph vertices
 - **Give** odd unexplored graph vertices
- ❖ **We stop dividing when units are too small**
 - Threshold in distance to search tree leaf

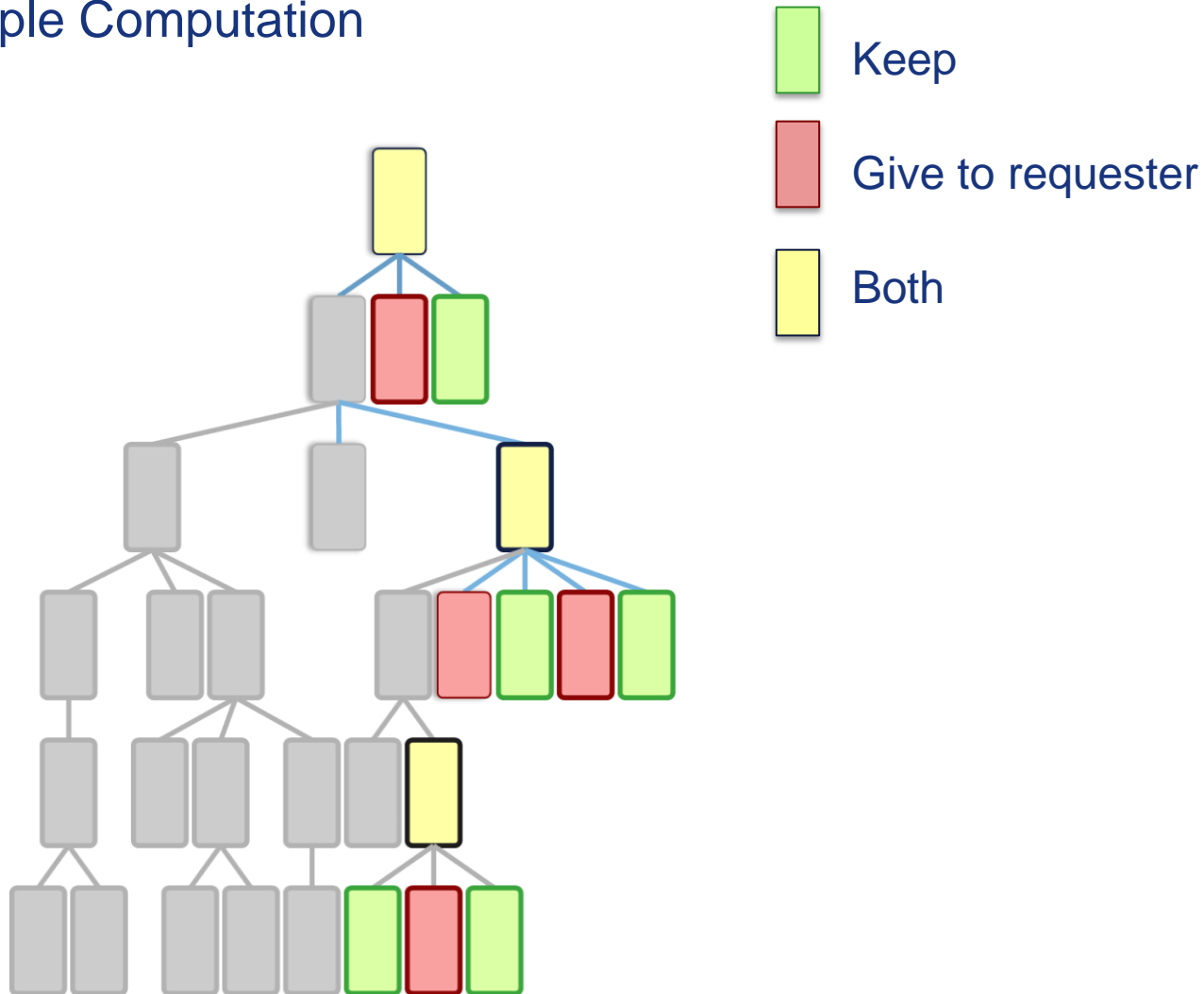
Dividing Computation

Example Computation



Dividing Computation

Example Computation



Sending Work

❖ Compact representation of search state

- Current depth
- Current tree path (g-trie nodes and graph vertices)
- Unexplored graph vertices at each depth

❖ Minimize data to send

❖ Efficiently resume computation

Work Request

- ❖ **When we do not have work, which processor should we contact?**
 - No data locality
 - Search trees completely unbalanced

Work Request

- ❖ **When we do not have work, which processor should we contact?**
 - No data locality
 - Search trees completely unbalanced
- ✓ **Ask a random processor!**
 - Random polling ([Sanders 1994])

Work Request

❖ When we do not have work, which processor should we contact?

- No data locality
- Search trees completely unbalanced

✓ Ask a random processor!

- Random polling ([Sanders 1994])

❖ Which thresholds

- Need to balance
- Static or dynamic

Work Request

❖ When we do not have work, which processor should we contact?

- No data locality
- Search trees completely unbalanced

✓ Ask a random processor!

- Random polling ([Sanders 1994])

❖ Which thresholds

- Need to balance
- Static or dynamic

✓ Constant was enough (efficient MPI_Probe)

Results Overview

❖ Networks used

Network	Nodes	Edges	Edges/Nodes	Description
Foodweb	128	2106	16,45	Food Web of Florida Bay
Neural	297	2148	7,23	Neural network of <i>C. Elegans</i>
Netsc	1589	2742	1,73	Co-authorship of network publications
Protein	2361	7182	3,04	Protein-Protein Interaction
Foldoc	13356	91471	6,85	Relation between terms in dictionary

❖ Environment used

- Dedicated cluster, 12 SuperMicro TwinView Server (2 quad Core Xeon 5335, 12GB Ram), 3.8TB space, Max 128 processors
- Infiniband, OpenMPI

Sequential Algorithm

- ❖ **Census of all undirected k -subgraphs**
- ❖ **Increase K until execution time $> 1h$**

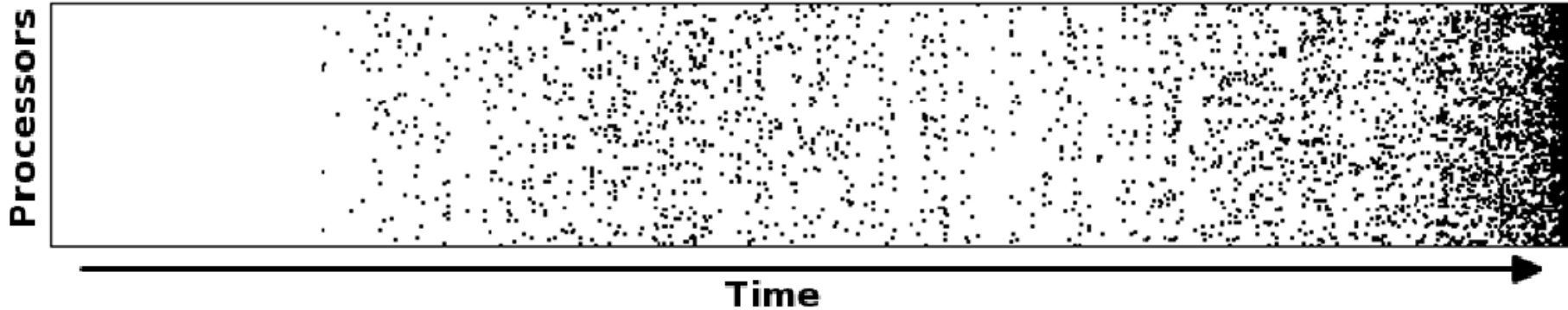
Network	Subgraph Size	Sequential Time (s)	Average Growth	# Different Subgraphs
Foodweb	7	38,233.89	44.9+/-8.8	853
Neural	7	843,212.02	55.8+/-15.2	853
Netsc	9	6,141.13	11.6+/-0.7	261,080
Protein	7	187,671.34	31.6+/-6.2	853
Foldoc	5	5,949.54	307.8+/-272.2	21

Speedup

- ❖ **Absolute Speedup**
- ❖ **Check Threshold:** 100000 nodes ($\sim 0.1s$)
- ❖ **No Division Threshold:** 2 levels

Network	Subgraph Size	#CPUs: Speedup				
		8	16	32	64	128
Foodweb	7	7.94	15.87	31.73	63.76	127.03
Neural	7	7.90	15.87	31.43	61.69	122.78
Netsc	9	7.90	14.76	31.40	62.40	124.02
Protein	7	7.91	15.90	31.41	63.43	124.36
Foldoc	5	7.87	15.70	30.28	61.63	121.36

Communication



Subgraphs of size 7, protein network

Example Application

❖ We experimented to find motifs:

- Use ESU to enumerate subgraphs on networks
- Build G-Trie with the set of found subgraphs
- Apply Parallel G-Trie sampling on randomized networks

❖ Tested on netsc network:

- Combine power of g-tries with parallelism
- Consistently achieve speedups of 2000 relative to sequential network-centric approach (ESU) and 500 relative to subgraph-centric one (Grochow and Kellis)

Conclusions

- ❖ **Created novel parallel algorithm for subgraph counting with g-tries**
 - Dynamic Load Balancing
 - Receiver-Initiated Strategy
 - Random Polling
 - Efficient search state representation
- ❖ **Implemented it and obtained almost linear speedup up to 128**
- ❖ **Tested on motif discovery problem**
 - Previously unfeasible sizes may now be reachable

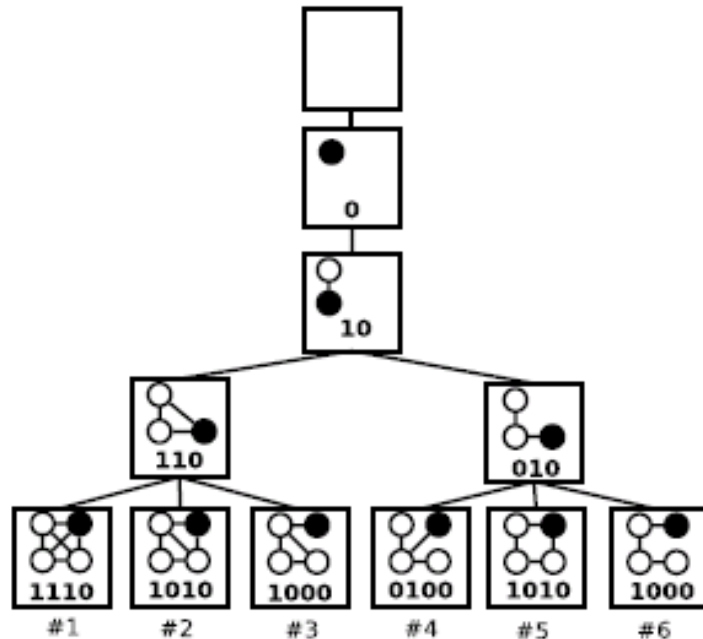
Future Directions

- ❖ **Automatic choice of parameters**
- ❖ **Parallelize sampling version of g-tries algorithms**
- ❖ **Parallelize whole motif computation**
- ❖ **Applications on large-scale datasets**

The End

❖ Thank you! Questions?

<http://www.dcc.fc.up.pt/~pribeiro/>
pribeiro@dcc.fc.up.pt



Efficient Parallel Subgraph Counting using G-Tries

P. Ribeiro, F. Silva and L. Lopes
 CRACS & INESC-Porto LA
 University of Porto
 Portugal

