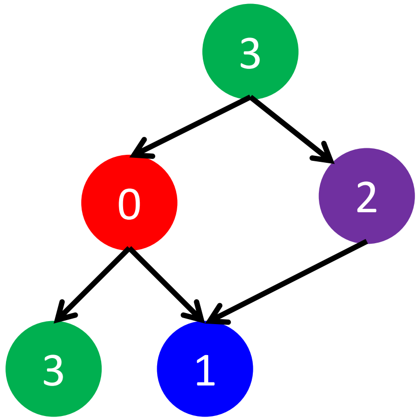



…GTCTAAAACATGATT… 0
…GTCTGAATCATGATT… 1
…GTCTGAAACATGATT… 0
…GTCTGAATCATCATT… 1

# Massively Parallel Depth-First Search and It's Applications

Kazuki Yoshizoe (美添 一樹)
Search and Parallel Computing Unit, RIKEN AIP
Jul. 29rd, 2019, 2nd DOML Workshop @RIKEN AIP Nihombashi

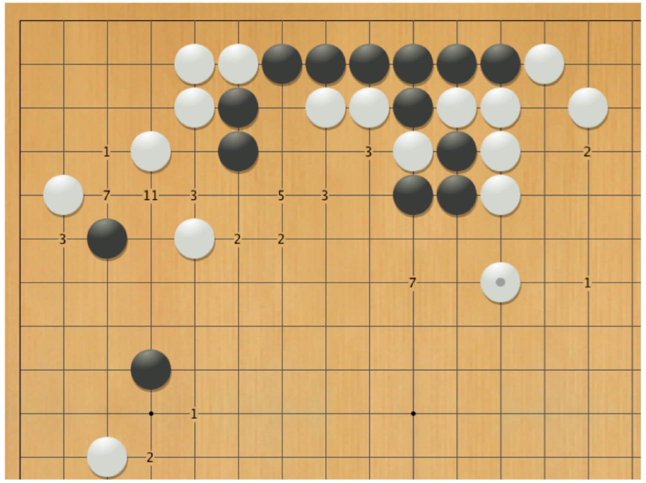
# Who am I?

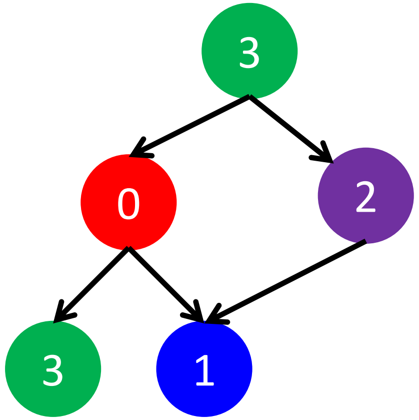
Parallel computing lab
at graduate school

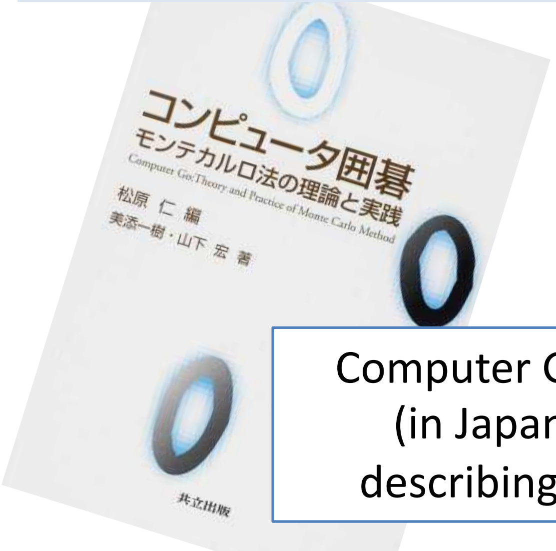
Search Algorithms

Game AI algorithms

Parallel Search Algorithms






Computer Go book
(in Japanese)
describing MCTS

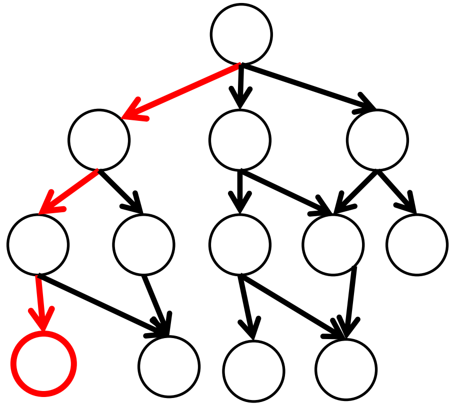
AlphaGo (by DeepMind) uses Deep Learning with Monte-Carlo Tree Search (MCTS)

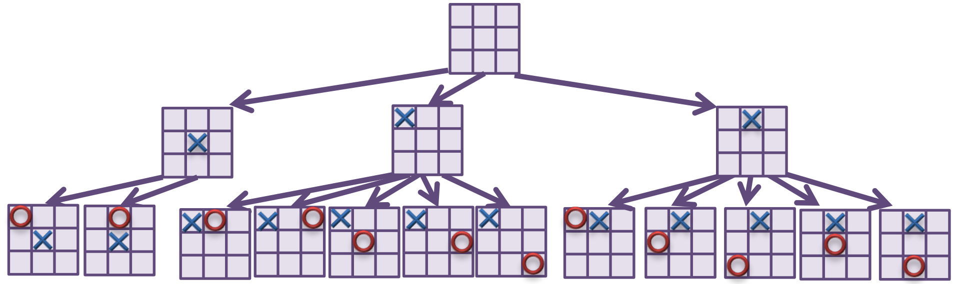
K. Yoshizoe et al., "Scalable Distributed Monte-Carlo Tree Search", Symposium on Combinatorial Search (SoCS), 2011.

Scales up to 3000+ fold speedup

# What is Search?

Here **Search** means finding **node(s) or path(s)** from a given graph





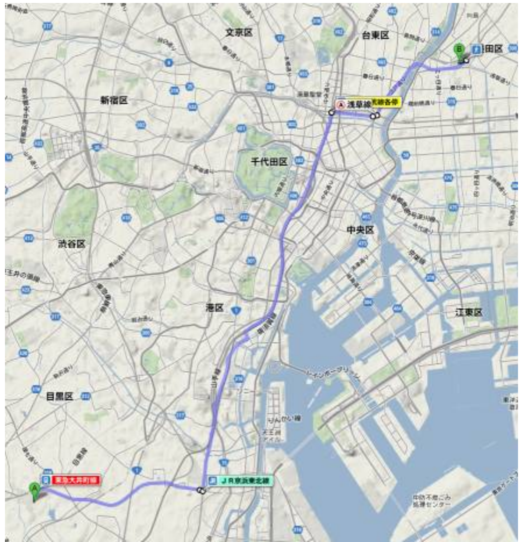
Node or path shows
- ✓ "shortest path"
- ✓ "optimal combination"
- ✓ "best play in games"

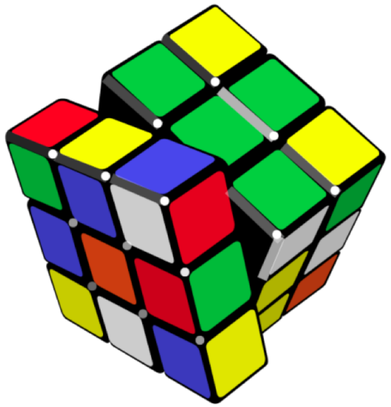
**Explicitly given Graph**

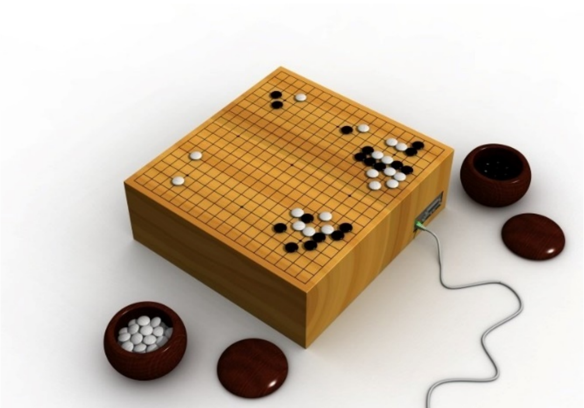
- Road map
- Trains
- social network

**Rule Based Graph**

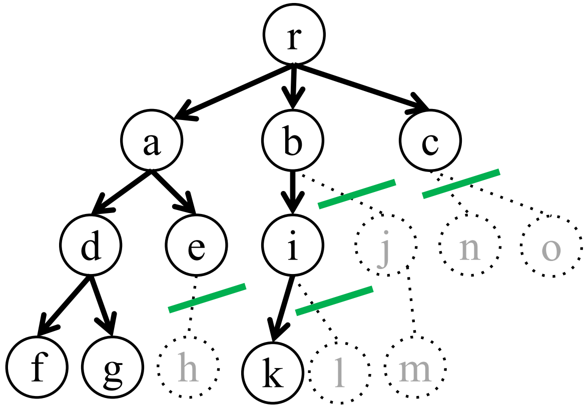
- Combinatorial optimization
- Games
- SAT, CSP



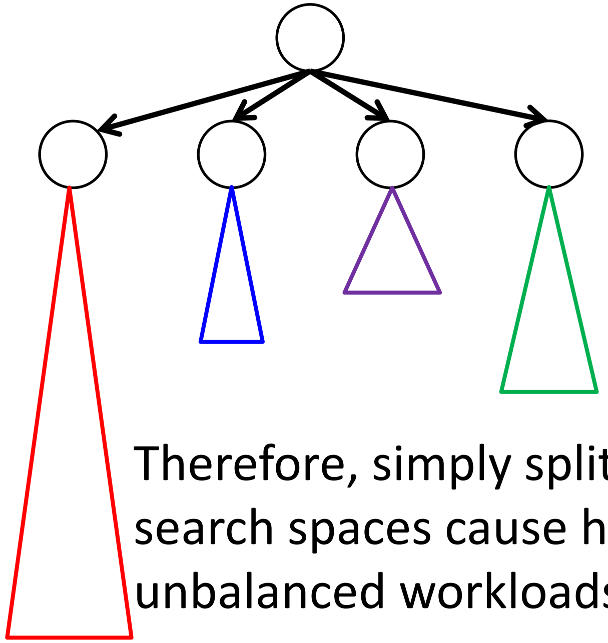




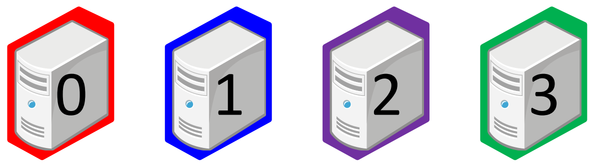
# Parallelizing Search Algorithms




Practical search algorithms "prune" search spaces to focus on promising part.



Therefore, simply splitting search spaces cause highly unbalanced workloads




Parallel Depth-First Search (DFS) and applications

DFS is simple but has many important applications

- Frequent Itemset Mining
- Statistical Pattern Mining
  [Yoshizoe, Terada, Tsuda 2018]
- Constraints Satisfaction
  [Ishii, Yoshizoe, Suzumura 2014]
- Continuous Optimizations
  [Izumi, Yoshizoe, Ishii 2018]

# Depth-First Search Applications




items

transactions

| database | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| A | x | x | x | x | x | x |
| B | | x | x | | x | |
| C | | x | | | x | |
| D | x | x | | x | x | x |
| E | | x | | x | | |
| F | x | | | x | | x |
| G | | | x | x | | x |

Counting / Enumerating
Frequent **itemsets**
from a given database

ex. **itemsets** with freq. 3 or higher

$\{1\}, \{2\}, \{3\}, \{4\}, \{5\}, \{6\}, \{1,4\},$
$\{1,6\}, \{2,4\}, \{2,5\}, \{4,6\}, \mathbf{\{1,4,6\}}$

## Frequent Itemset Mining

or Association Rule Mining



### ex1. Market Basket Analysis

items: products
trans.: customers
x: purchased items

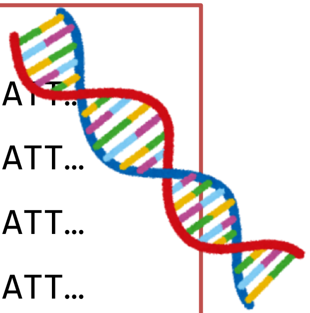
Fundamental problem in data mining

## Statistical Pattern Mining

### ex2. Genomics (GWAS)

items: SNPs
trans.: human
x: SNP

…GTCTAAAACATGATT…
…GTCTGAATCATGATT…
…GTCTGAAACATGATT…
…GTCTGAATCATCATT…



SNP: Single Nucleotide Polymorphism

Example. Finding cause of genetic disease from DNA

[Yoshizoe, Terada, Tsuda 2018] Bioinformatics

# Improving GWAS

Genome Wide Association Studies

# Finding Statistically Significant SNPs

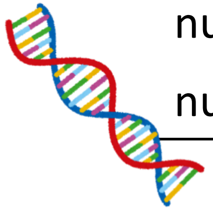
SNP

patients

| | 1 | 2 | 3 | 4 | 5 | 6 | |
|---|---|---|---|---|---|---|---|
| A | x | x | x | x | x | x | + |
| B | | x | x | | x | | + |
| C | | x | | | x | | - |
| D | x | x | | x | x | x | + |
| E | | x | | x | | | - |
| F | x | | | x | | x | - |
| G | | | x | x | | x | + |

Example, Early-onset Alzheimer SNP database

| | |
|---|---|
| nu. SNP | 380,157 |
| nu. patients | 364 |
| nu. Positive | 176 |



Existing GWAS: Finds one or two SNPs
Our tool : Finds **any number** of SNPs

A genetic disease can be caused by a combination of multiple SNPs (not by just one or two SNPs)

A naïve approach requires testing all possible $2^{380157}$ combinations of SNPs.
We make it possible with
- a smart pruning of the search space
- large scale parallelization

[Alzheimer] J. A. Webster, J. R. Gibbs, J. Clarke et al., “Genetic control of human brain transcript expression in Alzheimer disease.” Am J Hum Genet., vol. 84, no. 4, pp. 445–58, 2009.

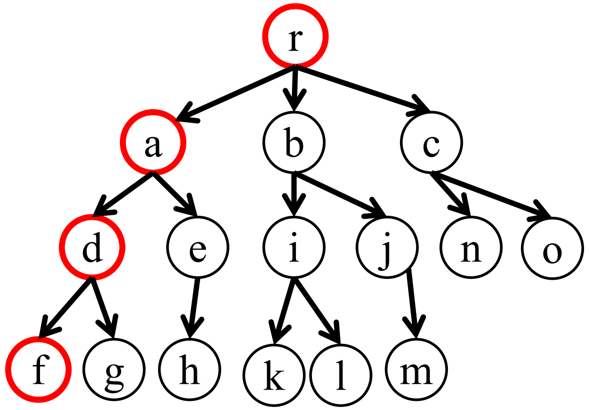
# Depth First Search (w/o threshold)

## Back tracking DFS

```
DFS() {
  Recur(r)
}
Recur(node n) {
   foreach (child c of n) {
      // do something for c
      Recur(c)
   }
}
```

back tracking can be naturally implemented with *recursive* call

Simply traverses all nodes in the tree




Memory usage $O(d)$
Only current path is needed

Depth-First Search is

- suitable for trees
- require small memory
- **Parallelization friendly**

Frequent Itemset Mining can be solved using DFS w/o threshold

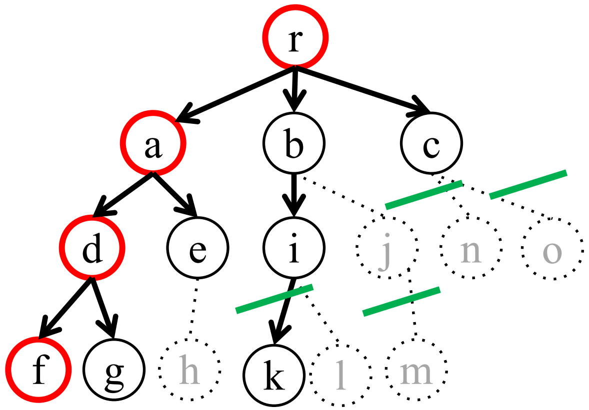
# Depth First Search with threshold update

## DFS with threshold

```
DFS() {
  Recur(r)
}
Recur(node n) {
   foreach (child c of n) {
      // do something for c
      if (c is within threshold) Recur(c)
      UpdateThreshold()
   }
}
```

Prune search space by
**dynamically updating threshold**

Update threshold during search.
More branches in the right are pruned.
(Search progresses from left to right.)




Ex. finding top-k nodes

With threshold,
more difficult to
parallelize, but possible
without large overhead

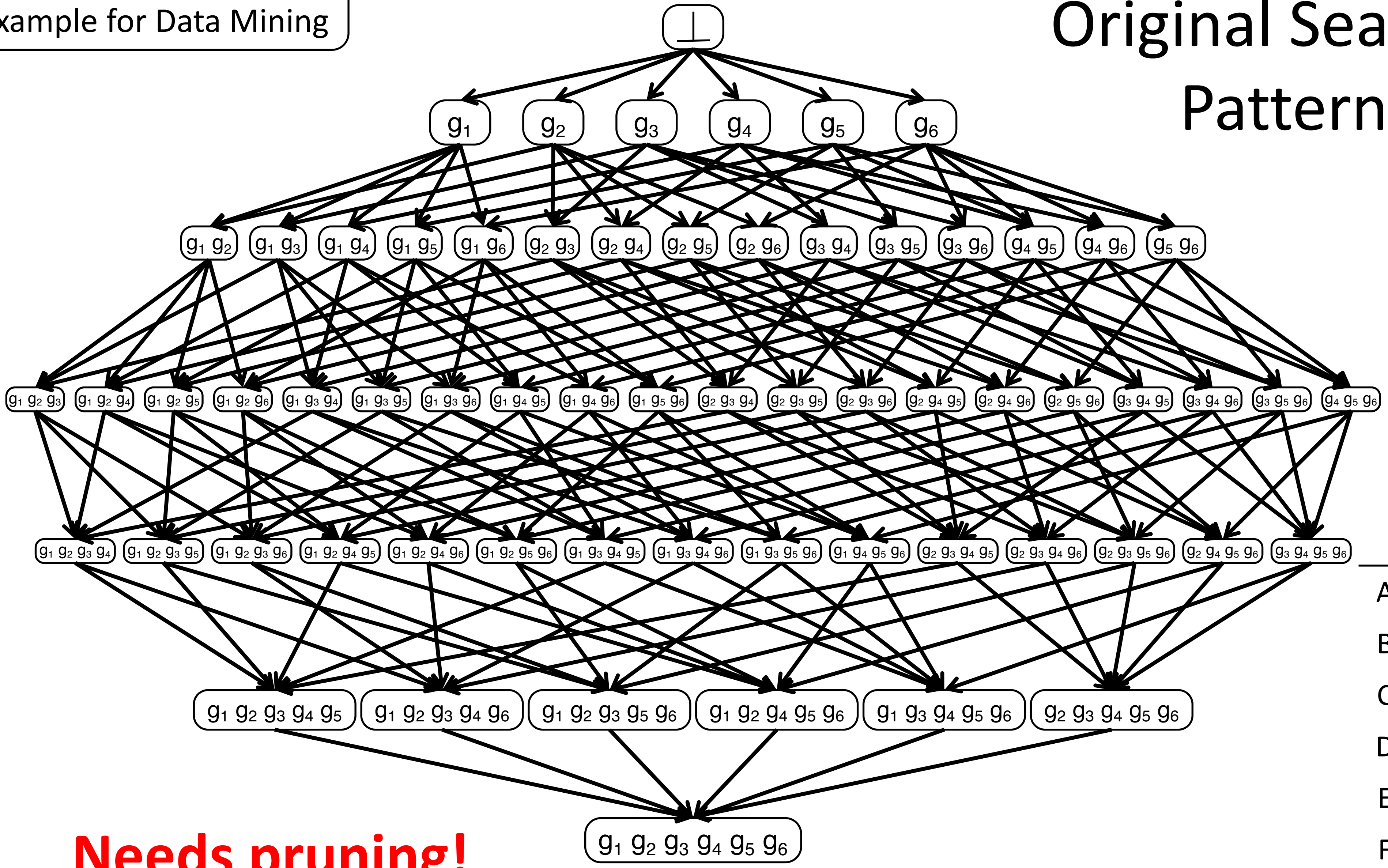
[Yoshizoe, Terada, Tsuda 2018]

[Ishii, Yoshizoe, Suzumura 2014]

Statistical Patten Mining can be implemented in DFS with threshold.
Significance threshold (P-Value lower bound) is updated and propagated.

Depth-First Search
Example for Data Mining

# Original Search space of Pattern Mining




**Needs pruning!**

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| A | x | x | x | x | x | x |
| B | | x | x | | x | |
| C | | x | | | x | |
| D | x | x | | x | x | x |
| E | | x | | x | | |
| F | x | | | x | | x |
| G | | | x | x | | x |

[Pasquier, Bastide, Taouil, Lakhal 1999]

# Closed Itemset

| transaction \ item | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| **A** | **x** | x | x | **x** | x | **x** |
| B | | x | x | | x | |
| C | | x | | | x | |
| **D** | **x** | x | | **x** | x | **x** |
| E | | x | | x | | |
| **F** | **x** | | | **x** | | **x** |
| G | | | x | x | | x |

Enumeration of itemsets (freq. >=3)
{1}, {2}, {3}, {4}, {5}, {6},{1,4}, {1,6}, {2,4}, {2,5}, {4,6}, **{1,4,6}**

Enumeration of closed itemsets (freq. >=3)
{2}, {3}, {4}, {2,4}, {2,5}, {4,6}, **{1,4,6}**

*closed itemset* is a compressed way of enumeration

## def: Closed Itemset

An itemset $I$ is a *closed itemset* if there is no item $j$ such that
$$j \notin I \wedge \mathrm{freq}(I \cup \{j\}) = \mathrm{freq}(I)$$

(frequency decreases by adding any other item to a closed itemset)

**{1, 4, 6}** is a *closed itemset*

implicitly includes {1}, {1, 4}, {1, 6}

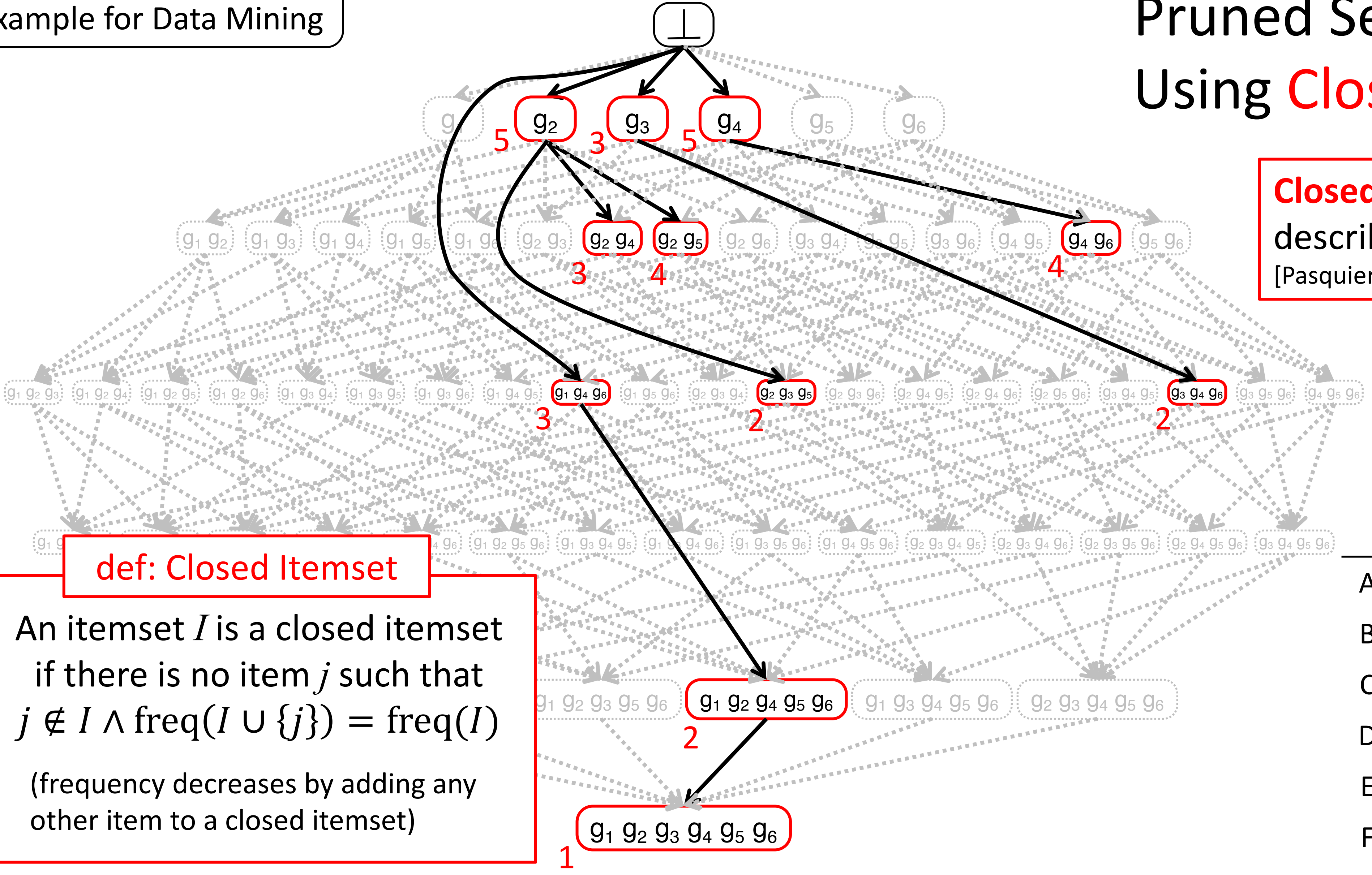
freq{1,4,6}=freq{1,4}=freq{1,6}=freq{1}=3

Note: freq{4, 6}=4, so {4,6} is not included

Depth-First Search
Example for Data Mining

# Pruned Search Space Using Closed Itemset

**Closed Itemsets** (red nodes) describe all frequent itemsets
[Pasquier, Bastide, Taouil, Lakhal 1999]




**def: Closed Itemset**

An itemset $I$ is a closed itemset if there is no item $j$ such that

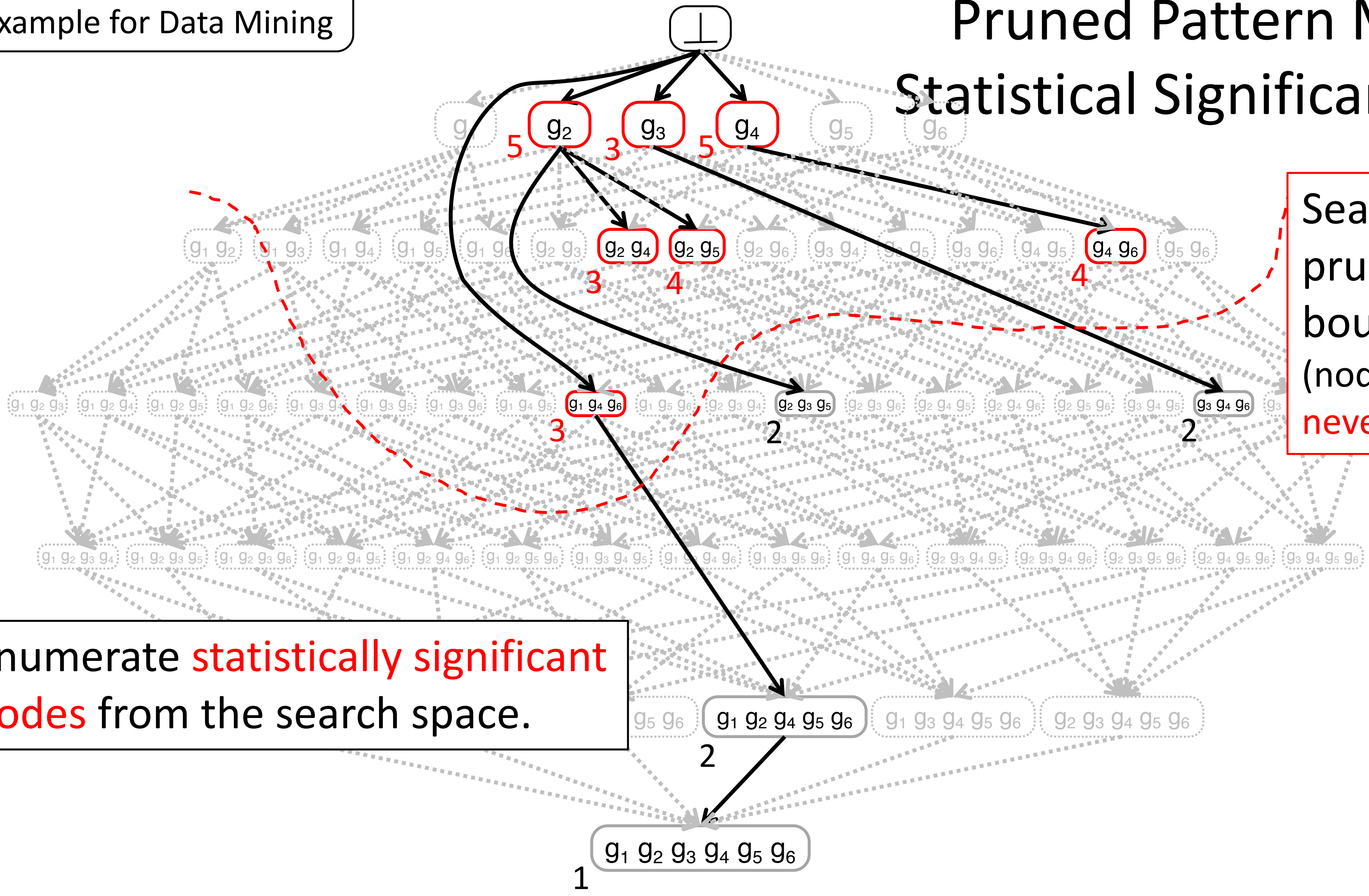
$$j \notin I \land \text{freq}(I \cup \{j\}) = \text{freq}(I)$$

(frequency decreases by adding any other item to a closed itemset)

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| A | x | x | x | x | x | x |
| B | | x | x | | x | |
| C | | x | | | x | |
| D | x | x | | x | x | x |
| E | | x | | x | | |
| F | x | | | x | | x |
| G | | | x | x | | x |

Depth-First Search
Example for Data Mining

# Pruned Pattern Mining with Statistical Significance threshold




Search space can be pruned using the lower bound of Fisher P-Value (nodes below this line will never be significant)

| | 1 | 2 | 3 | 4 | 5 | 6 | |
|---|---|---|---|---|---|---|---|
| A | x | x | x | x | x | x | + |
| B | | x | x | | x | | + |
| C | | x | | | x | | - |
| D | x | x | | x | x | x | + |
| E | | x | | x | | | - |
| F | x | | | x | | x | - |
| G | | | x | x | | x | + |

Enumerate statistically significant nodes from the search space.


[Terada, Okada-Hatakeyama, **Tsuda**, Sese, 2013]

[Minato, Uno, **Tsuda**, Terada, Sese 2014]

# Massive Parallel Statistical Pattern Mining

# For solving GWAS and others

Frequent Itemset Mining based on Closed Itemset

[Pasquier, Bastide, Taouil, Lakhal 1999]

Apply reverse search technique (**LCM algorithm**)

[Avis, Fukuda 1996]

[Uno, Kiyomi, Arimura 2004]

Applied to Statistical Pattern Mining

**LAMP algorithm**

[Terada, Okada-Hatakeyama, **Tsuda**, Sese, 2014]

Faster **LAMP** using DFS with threshold

[Minato, Uno, **Tsuda**, Terada, Sese 2014]

**Massive Parallel LAMP (MP-LAMP)**

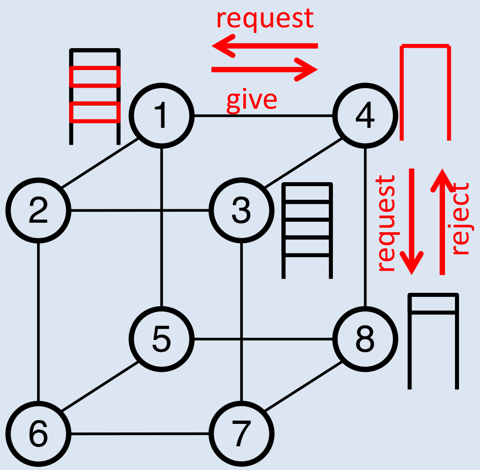
[**Yoshizoe**, Terada, **Tsuda** 2018]

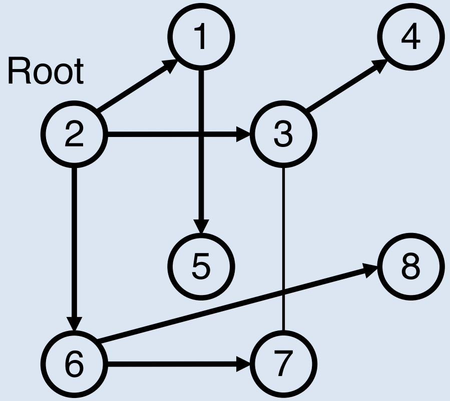
## Parallelization Method

Reformulate algorithm from recursive call to stack + loop

hardware/middleware aware algorithm and implementation

### Work stealing and broadcast/reduce



# Preparation for Parallelization: Reformulate the Algorithm

recursive function call (original)

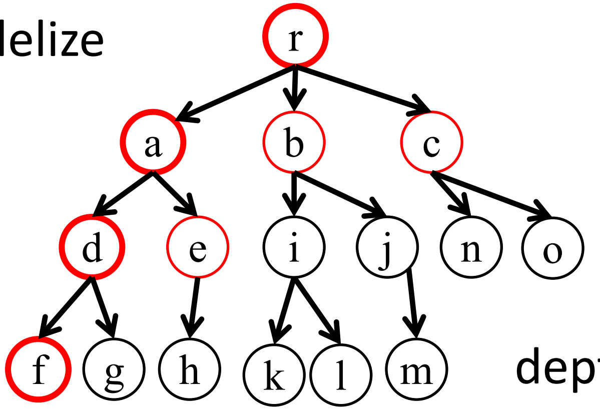
```
DFS() {
  Recur(r)
}
Recur(node n) {
    foreach (child c of n) {
        // do something for c
        if (c is within threshold) Recur(c)
        UpdateThreshold()
    }
}
```

reformulate

implementation using stack and loop

```
StackDFS() {
    push(r)
    Loop()
}
Loop() {
    while(stack not empty) {
        pop n from stack
        foreach (child c of n) {
            // do something for c
              if (c is within threshold) push(c)
            UpdateThreshold()
        }
    }
}
```

Pros: O($d$) memory
Cons: difficult to parallelize




depth $d$, branching factor $b$

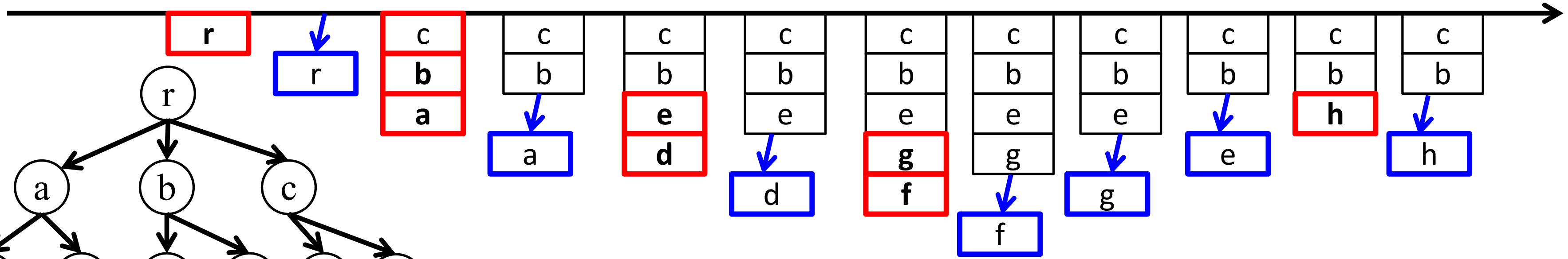
Cons: O($d$ $b$) memory
Pros: easy to parallelize

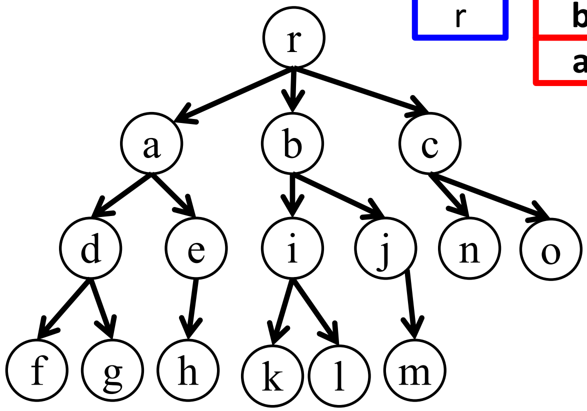
```
DFS() {
  Recur(r)
}
Recur(node n) {
   foreach (child c of n) {
      // do something for c
       if (c is within threshold) Recur(c)
      UpdateThreshold()
   }
}
```

recursion to stack + loop

```
StackDFS() {
   push(r)
   Loop()
}
Loop() {
   while(stack not empty) {
      pop n from stack
      foreach (child c of n) {
         // do something for c
          if (c is within threshold) push(c)
         UpdateThreshold()
      }
   }
}
```

foreach in reverse order






child nodes are pushed in the reverse order for DFS behavior

Note: if pushed in the normal order, it will be breadth-first search which requires large amount of memory

# Parallelization Method

## work stealing

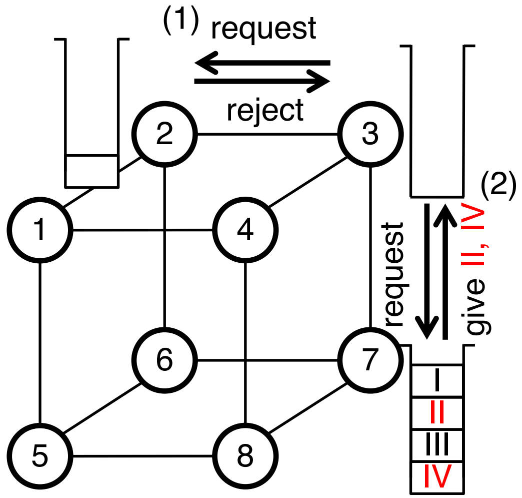
If the stack is empty, find a ***victim*** and ***steal*** workloads from it

## Broadcast

Threshold info is sent on a spanning tree

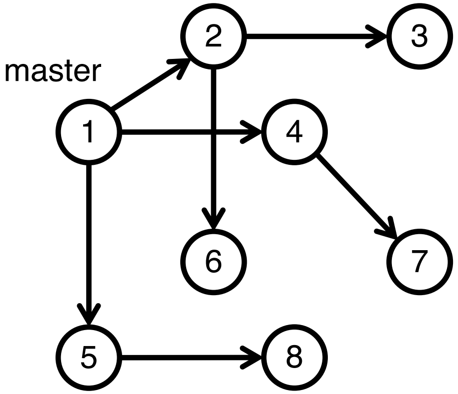
## Termination Detection

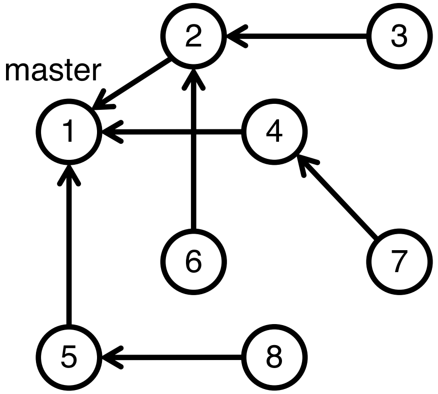
It is not straight forward in a distributed environment. Correctly detect the termination of all processes.




workload request is sent to neighbor process in a virtual Hypercube based communication graph

[Saraswat et al. 2011] Hypercube + random edge

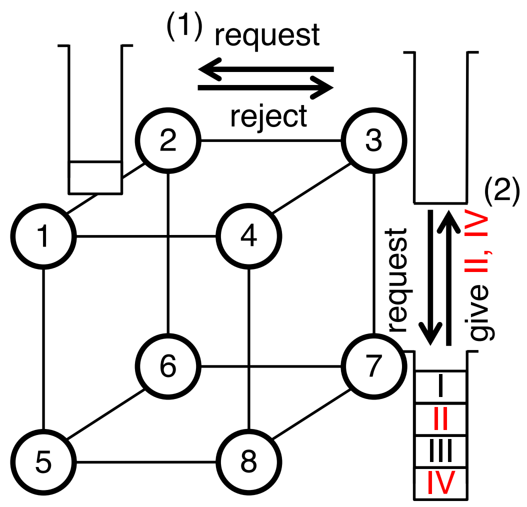



Implementation based on C++ and MPI library

[Mattern 1990] distributed termination detection on spanning tree

# Work Stealing based parallelization




## Receiver initiated Work stealing

Compute nodes with empty stack (empty job)
1, select a ***victim*** node
2, send job request to the ***victim***

the ***victim*** will either
1, give half of the stack (if has enough job)
2, reject the request

victims are
- randomly selected once, then
- selected from neighbors in communication graph

Virtual graph based on
Hypercube and random edges

Lifeline graph [Saraswat et al. 2011]

# DTD Distributed Termination Detection

If all stacks are empty, terminate the algorithm

There might be unfinished communication

If number of send and recv are equal, terminate the algorithm

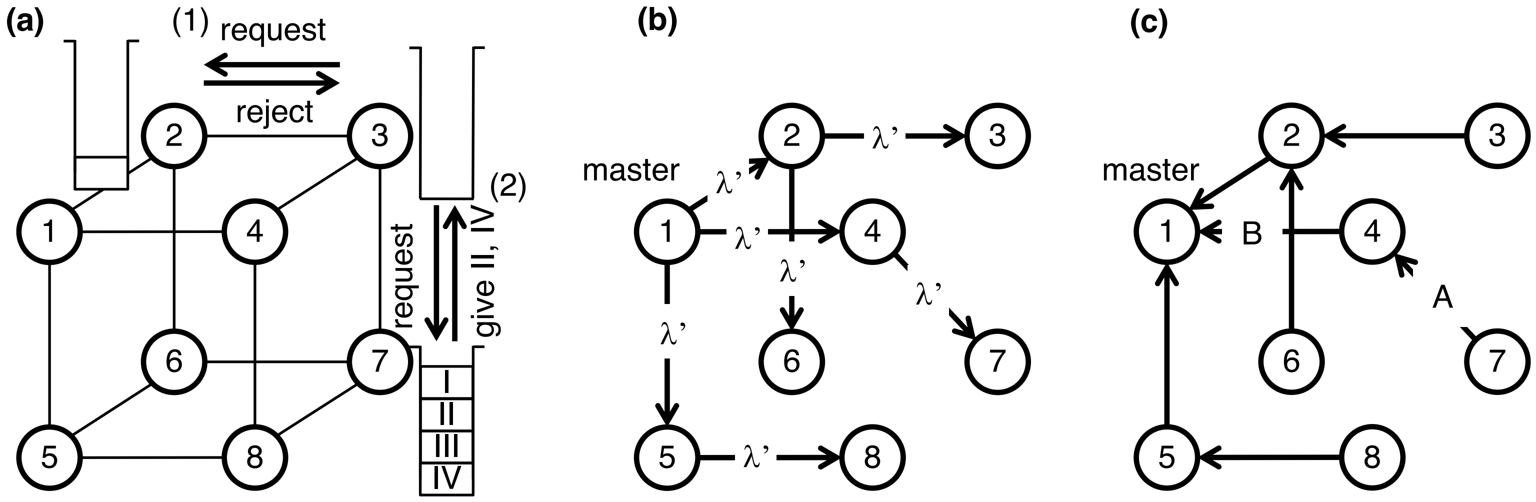
Counting is not synchronized so
same number of send and recv can be overlooked

If any node received a message, restart counting send/recv

It is proved to be correct

A famous token based algorithm by [Dijkstra and Scholten 1980]
Much improved DTD on spanning tree [Mattern 1990] (not well known)

# Using two communication patterns

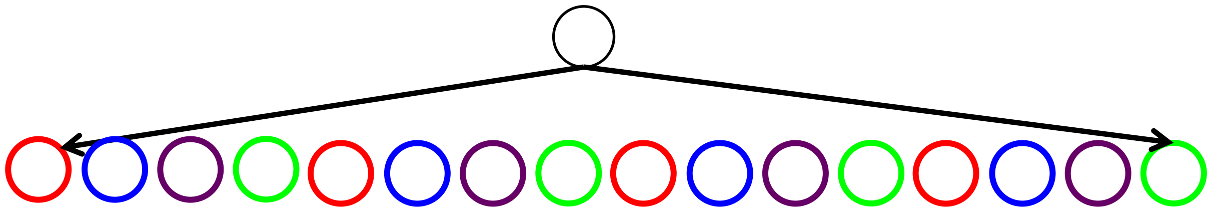



Work stealing between stacks

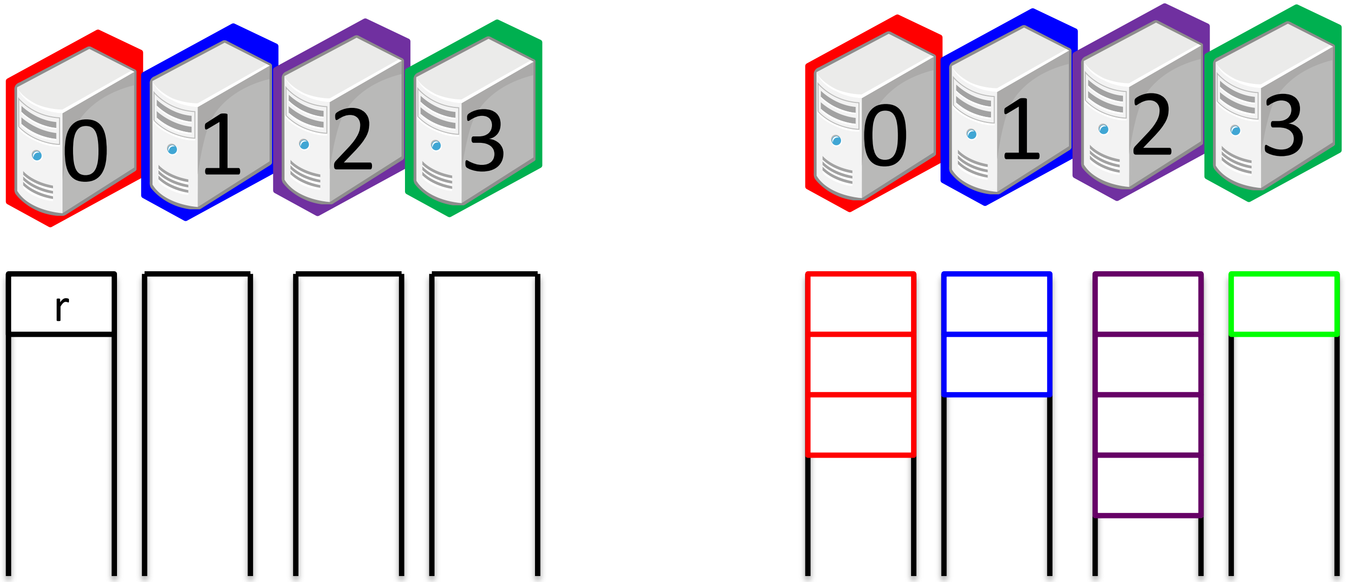
DTD and threshold broadcast/reduce

Distributed Termination Detection

# Combine with Static load balancing



distribute depth 1 nodes at the initialization




Original algorithm starts with only the root node pushed

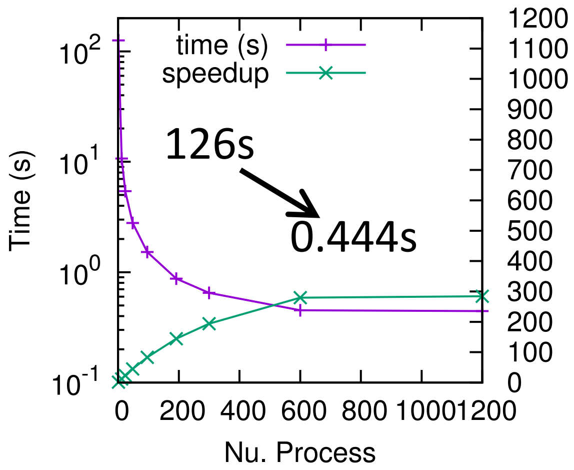
If initialized like this, load balancing can be easier

# Statistical Pattern Mining: Speedup

Speedup for Finding combination of SNPs related to Alz. or Japanese from subset of SNPs

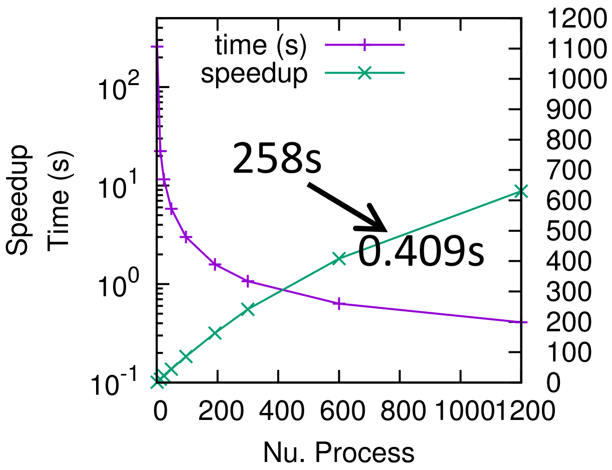
### JPN dom. 10%

item: 11253, trans:697, dens:1.0%



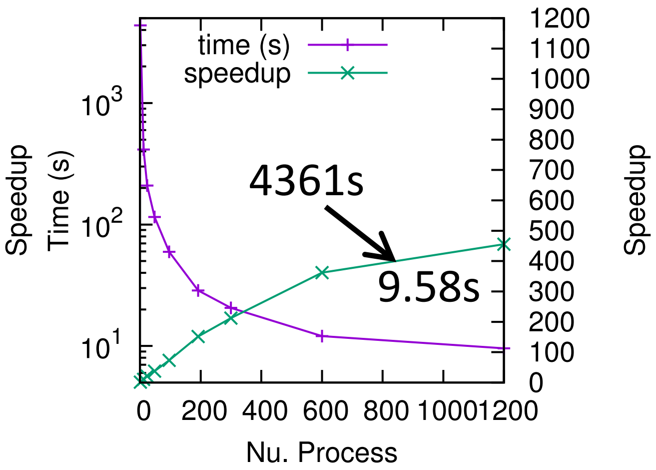

### Alz. dom. 5%

item:44052, trans:364, dens:5.4%



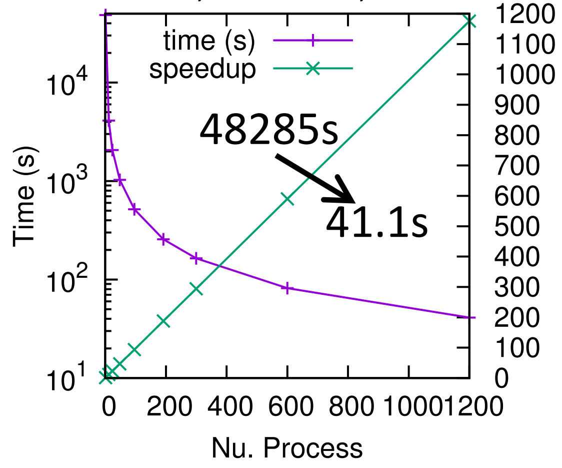

### Alz. rec. 30%

item: 250120, trans:364, dens:2.9%



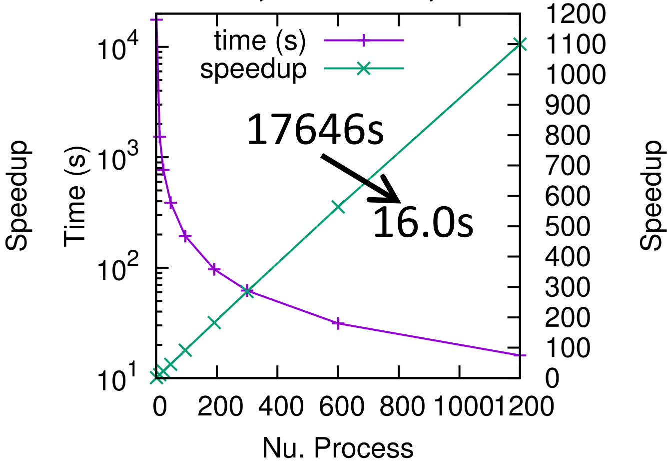

### JPN dom. 20%

item:11914, trans:697, dens:1.9%




### Alz. dom. 10%

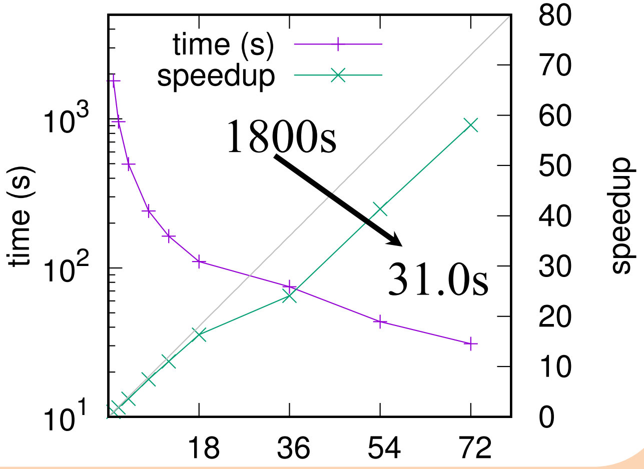
item: 91126, trans: 364, dens:9.8%




EC2 c4.8xlarge
Xeon E5-2666v3
(2.9GHz, 18cores)
10G ethernet,
using ***cfncluster***

### 1,000 Gen JPN dom. 15%

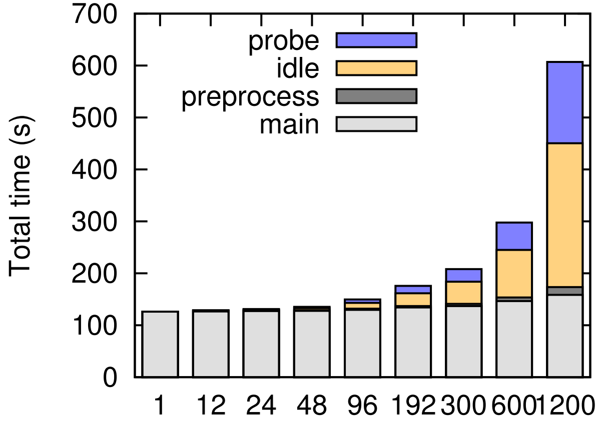
SNP:11676, dens:1.5%



# Breakdown of Execution time (Search, Communication, Idle)

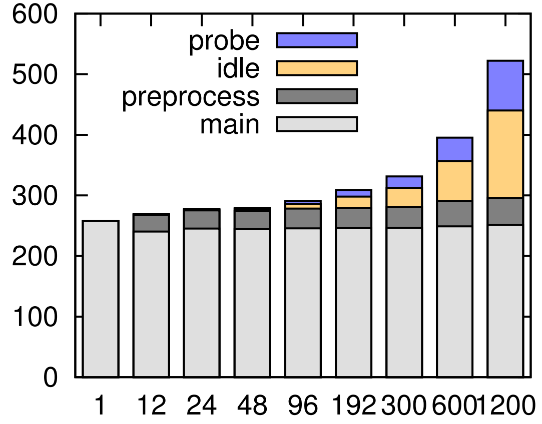
## JPN dom. 10%

item: 11253, trans:697, dens:1.0%



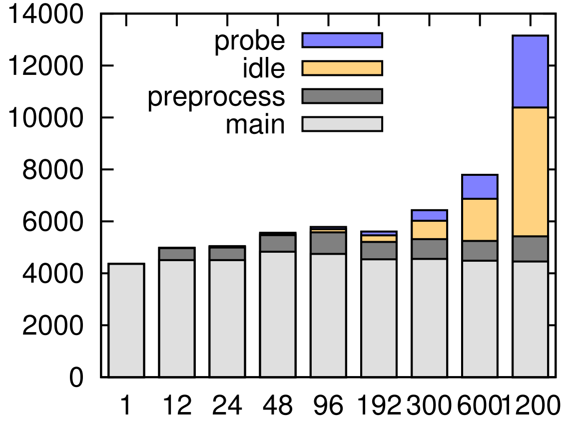

## Alz. dom. 5%

item:44052, trans:364, dens:5.4%



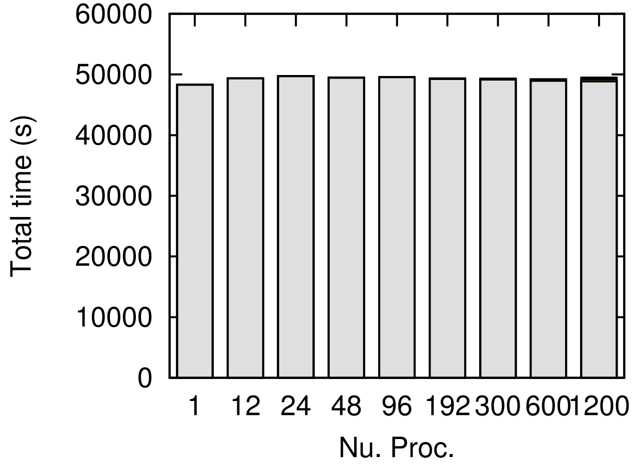

## Alz. rec.30%

item: 250120, trans:364, dens:2.9%



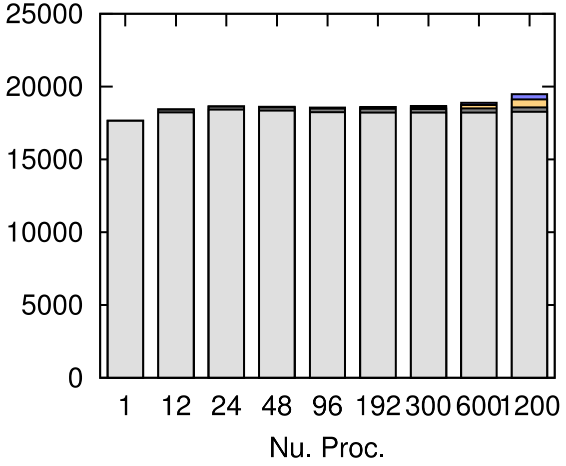

## HapMap dom. 20%

item:11914, trans:697, dens:1.9%




## Alz. dom. 10%

item: 91126, trans: 364, dens:9.8%




For more difficult problems, communication delay is hidden

# Efficient Speedup on K-Computer

Used up to **140K CPU cores** on K-computer, achieved estimated speed up of **110-120K fold.**
(collaboration with RIKEN R-CCS, unpublished)
Other existing work for parallel pattern mining, speedups are limited to 30-fold or less.


One of the discovered SNP combinations from a subset of SNPs for Alzheimer

| | |
|---|---|
| rs1057079 | mTOR, mTOR-AS1 |
| rs1064261 | mTOR |
| rs2075800 | HSPA1L (Hsp70) |
| rs429358 | APOE |

P-value after correction for these 4 SNPs was 0.00031777

We are also working on
- Parallel Itemset Mining Library
- Algorithms for other mining problems
  (includes continuous feature data)

[Alzheimer] J. A. Webster, J. R. Gibbs, J. Clarke et al., "Genetic control of human brain transcript expression in Alzheimer disease." Am J Hum Genet., vol. 84, no. 4, pp. 445–58, 2009.

# Statistical Pattern Mining and Feature Selection

nu. of features roughly 10K to 1M

nu. of data roughly 1000

| database | 1 | 2 | 3 | 4 | 5 | 6 | ? | |
|---|---|---|---|---|---|---|---|---|
| A | x | x | x | x | x | x | | + |
| B | | x | x | | x | | x | + |
| C | | x | | | x | | | - |
| D | x | x | | x | x | x | x | + |
| E | | x | | x | | | x | - |
| F | x | | | x | | x | x | - |
| G | | | x | x | | x | | + |
| ? | | x | | x | x | x | | - |

High dimensional (binary) features

relatively small nu. of samples

…GTCTAAAACATGATT…

…GTCTGAATCATGATT…

…GTCTGAAACATGATT…

…GTCTGAATCATCATT…

Our method finds statistically significant combinations of features

We have methods both for binary and continuous features (some limitations).

# Conclusions

- Parallel Search Algorithms are not straight forward but possible.
  – Depth-First Search algorithms are relatively easy to parallelize.
- We parallelized DFS based Pattern mining and observed highly efficient parallel speedup.
  – speedup measured on a real application for GWAS
  – Run on K-computer (in RIKEN R-CCS), achieved 110K+ speedup
- We can apply this approach to other DFS based mining algorithms (and hopefully to other DFS in general)
  – We implemented a parallel pattern mining library (preparing to make it public)
- Looking for collaboration with ML people in high dimensional feature selection