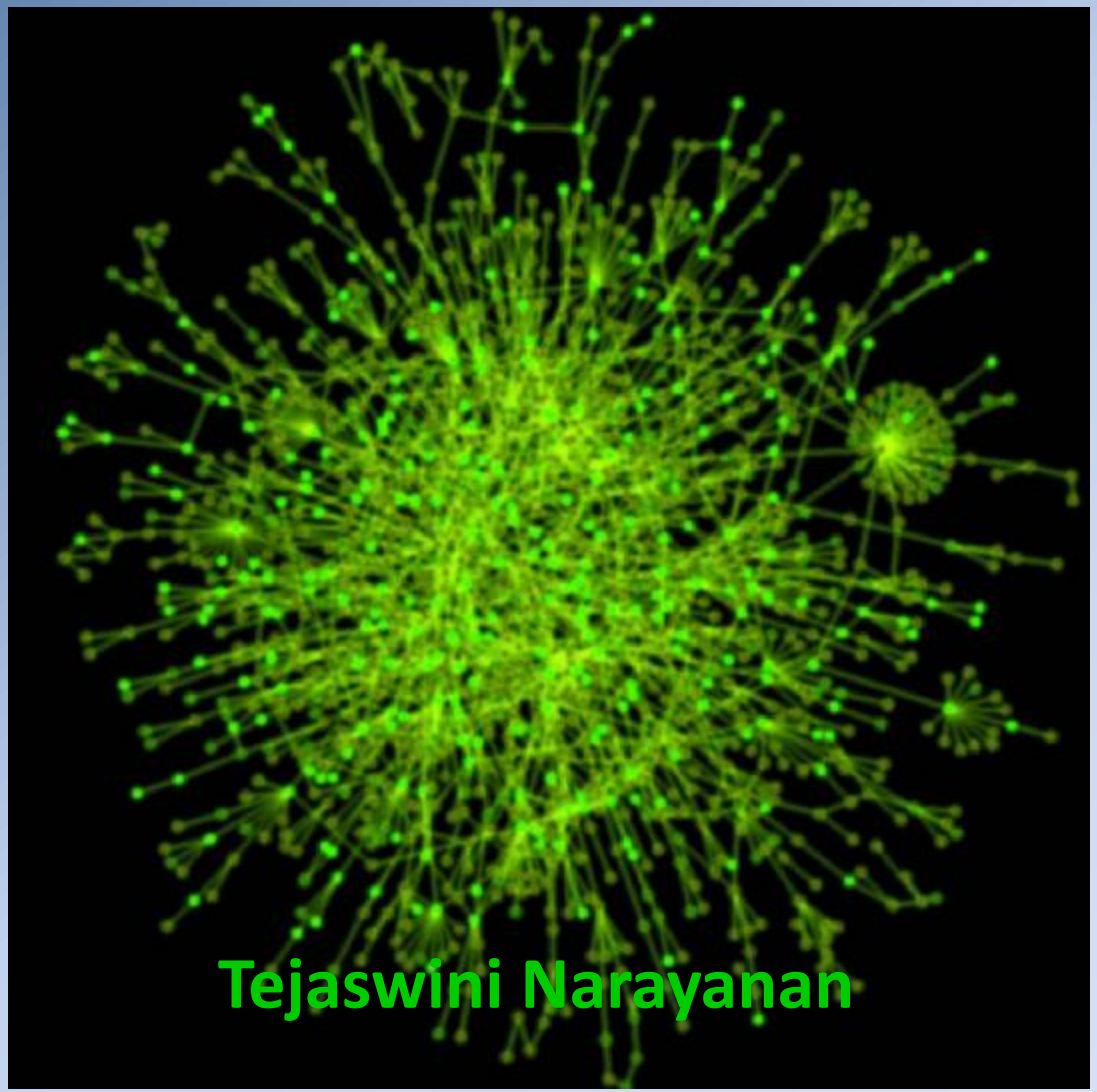
Graph Theoretic Algorithms for Modularity Detection in Biological Networks



5 Mar 2010

Agenda

- Overview
  - Fundamentals of Graph theory
  - Community structures in networks
  - Existing betweenness algorithms
  - Insight into Edge-betweenness algorithm
  - Modularity factor
  - Parallel Implementation
- Work Performed
  - Proposal
  - Results
- Applications
- Over to Merril

1- Fundamentals of *Graph theory*

UCSD

- ***Network:*** Collection of *nodes*, logically connected to each other by *edges*, thus giving some information about the nodes' relationships.

Eg: Social n/w, protein interaction n/w, metabolic reactions n/w, neuronal connectivity n/w,…

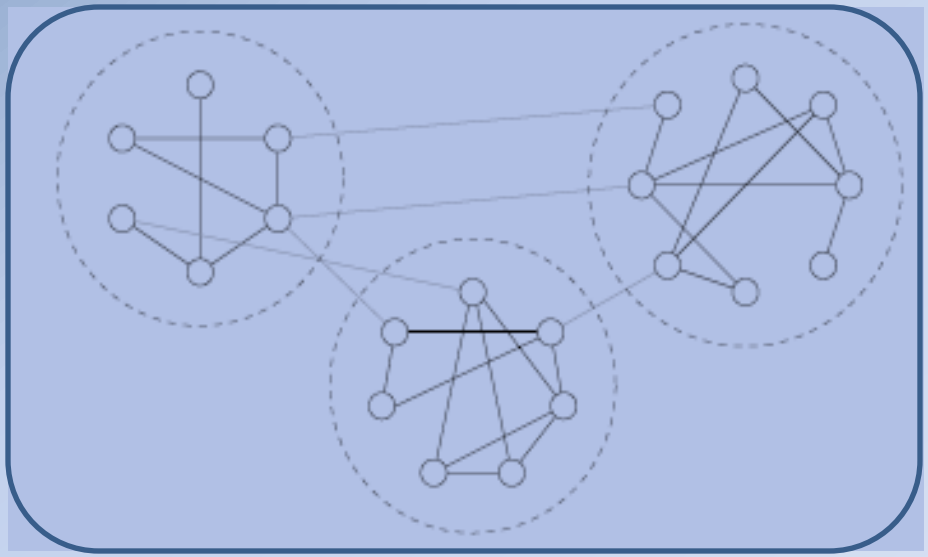
- ***Nodes:*** Individuals, proteins, metabolites, neurons,…

- ***Edges:*** Social Interaction, Protein-protein interaction, reaction between metabolites, axons,…

2- Community structures in network

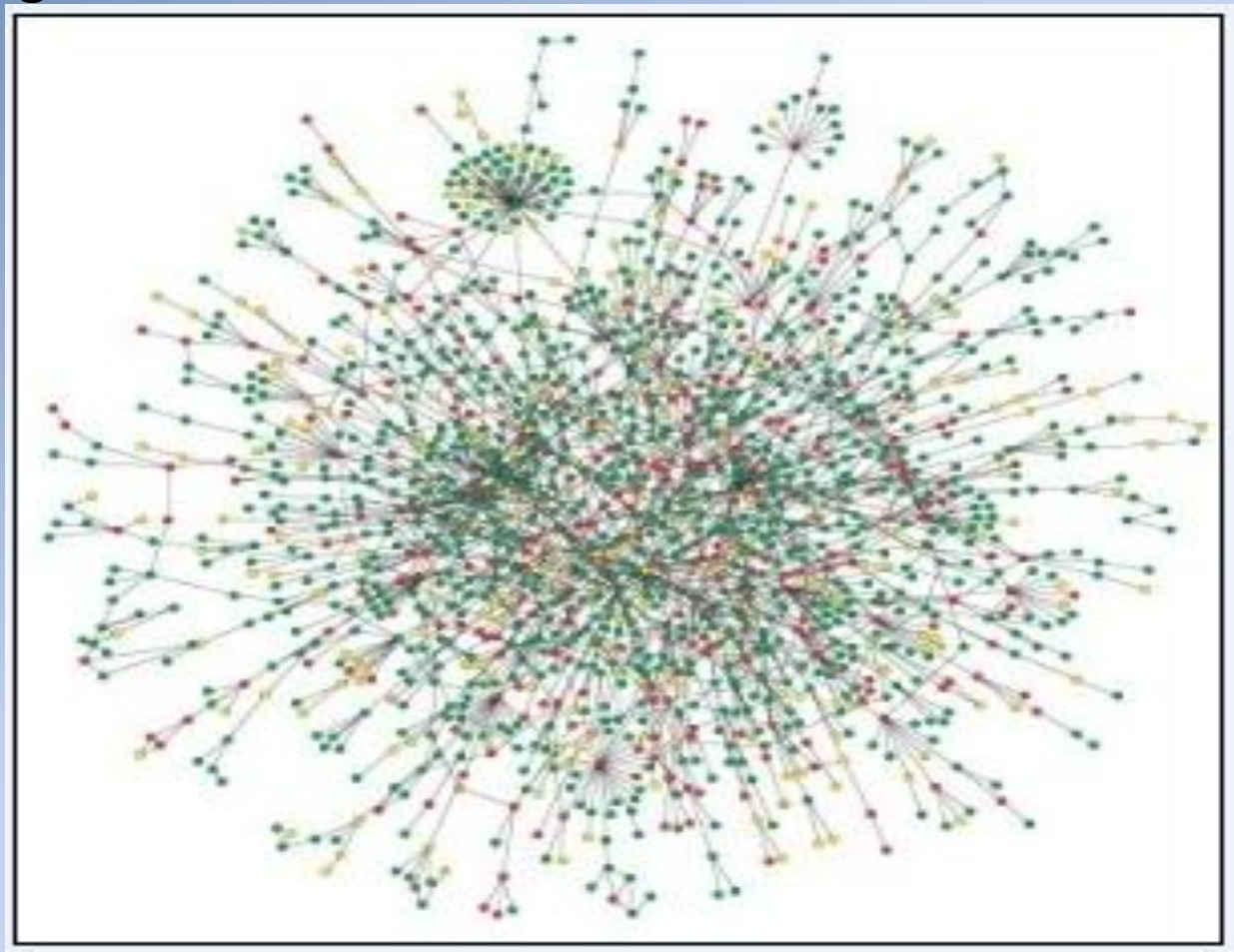
- Definition

"The division of network nodes into groups within which the network connections are dense, but between which they are sparser"



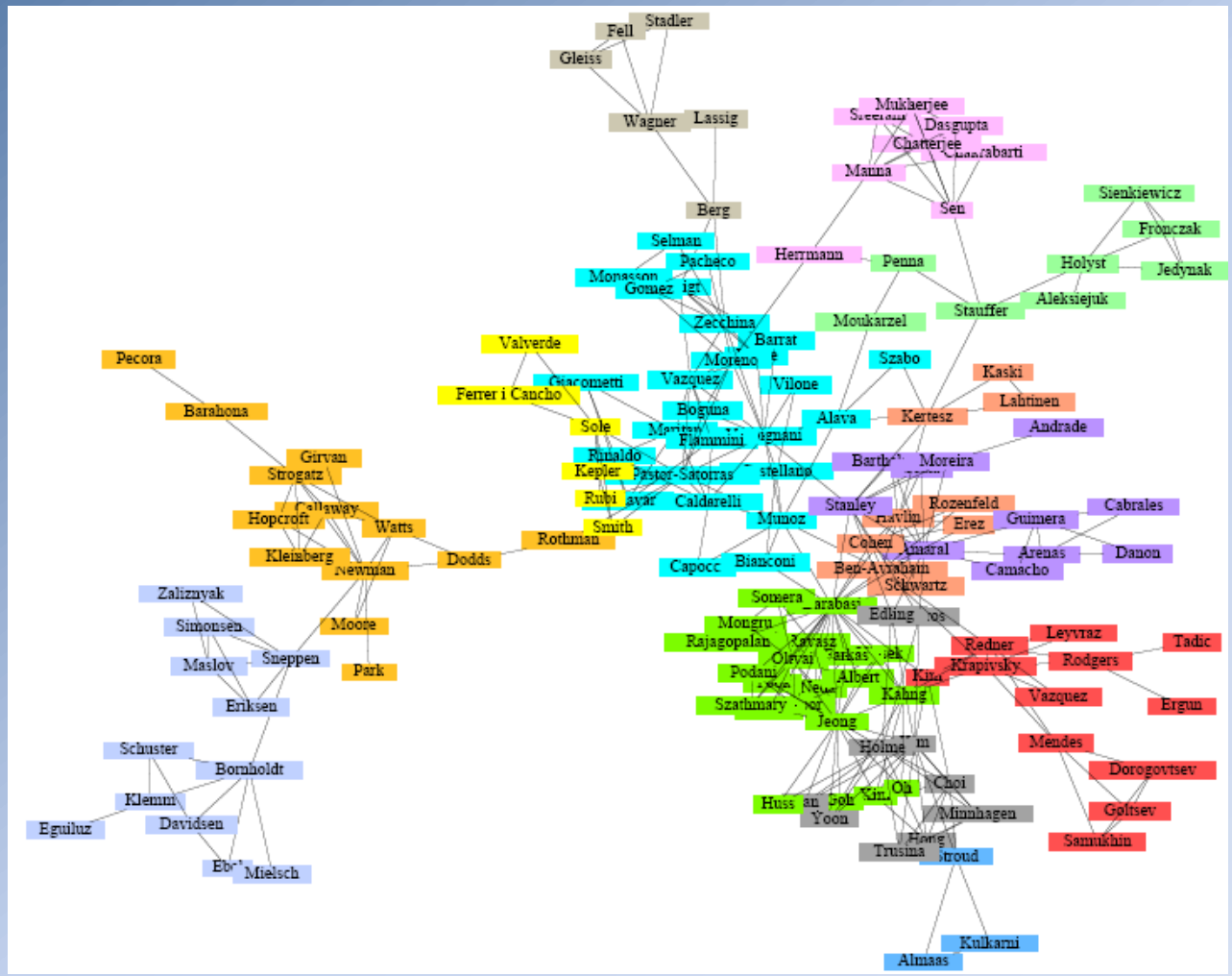
Community structures in networks

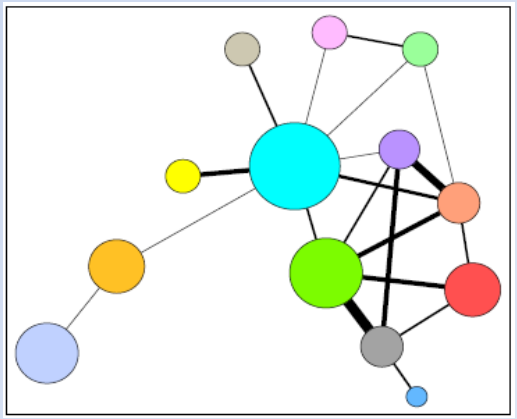
- **Necessity for definition:** Really complex biological graphs- a typical human metabolic reaction graphs have about 88K edges!



Community structures in networks

- **Advantages:** Such a simple 'moduled' representation of any graph would be easy to analyze and handle.





Hierarchical Clustering

"These techniques are aimed at discovering natural divisions of networks into groups, based on various metrics of similarity or strength of connection between vertices"

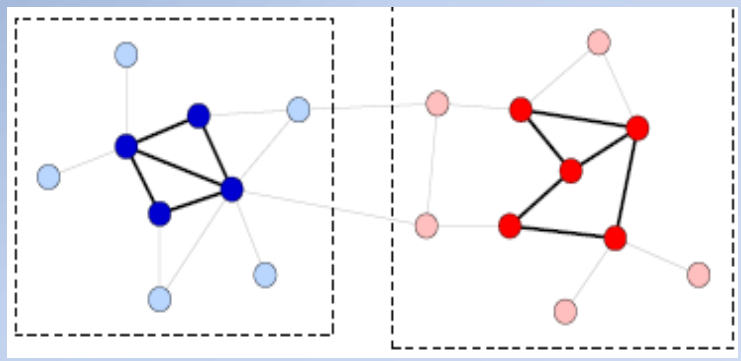
HC

Agglomerative

-Addition of edges

-Similarity calculated between vertex pairs

-Disadvantage: Peripheral nodes are neglected- core nodes retained due to strong similarity



Divisive

-Removal of edges

-Least similar connected pair of vertices removed

-Authors approach is different: edges that are most "*between*" other vertices removed!

3- Existing *betweenness* algorithms

UCSD

[1] Random-walk betweenness:

- A random walk from Source node to Destination node is considered.

- Expected net number of times a random walk between a particular pair of vertices will pass down a particular edge is calculated
- sum over all vertex pairs will give the RW betweenness value for that edge.

Existing *betweenness* algorithms

[2] Current-flow betweenness:

- Circuit created by placing a unit resistance on each edge, unit current source and sink at a particular pair of vertices.
- The resulting current chooses the least resistance path.
- The current-flow betweenness for an edge = the absolute value of the current along the edge summed over all source/sink pairs (calculated using Kirchoff's laws)

The current-flow betweenness is exactly the same as the random walk betweenness!

4- Insight into *Edge-betweenness* algorithm

UCSD

Existing betweenness algorithms

[3] Edge-betweenness:

- All paths between inter-community vertices must pass through the relatively fewer edges that connect the 2 communities.

- Betweenness is some measure that favors edges that lie between communities and disfavors those that lie inside communities.

- **Method used:** the shortest paths between all pairs of vertices is found and the count of how many run along each edge betweenness measure 'rush' 'edge-betweenness' 'shortest path betweenness'!

Work Performed

(Girvan and Newman)

Two step algorithm for modularity

(1) Iterative removal of edges to split into communities (driven by betweenness values)

(2) Crucial step: *The betweenness measures recalculated after each removal!*

5- Modularity factor

UCSD

- Note that never-ending iterative removal of edges would lead to a stage where all edges are removed and we have just the nodes- which are after all individual modules themselves.
- Do we want such a division of a given network?
- *'Modularity'* is a factor **Q** defined to control the limit up to which the algorithm should run so as to give a logically sound and informative output.

Q FACTOR:

1. Consider a particular division of a network into k communities.
2. k×k symmetric matrix e: with element eij -> fraction of edges that link vertices in community i to vertices in community j.

3. **Tr e = Σi eii** : fraction of edges that connect vertices in same community
4. A good division into communities should have a high value of this trace.
5. **row (or column) sums:** ai = Σj eij -> represent fraction of edges that connect to vertices in community i.
6. eij = aiaj -> when edges fall between vertices without regard for the communities they belong to

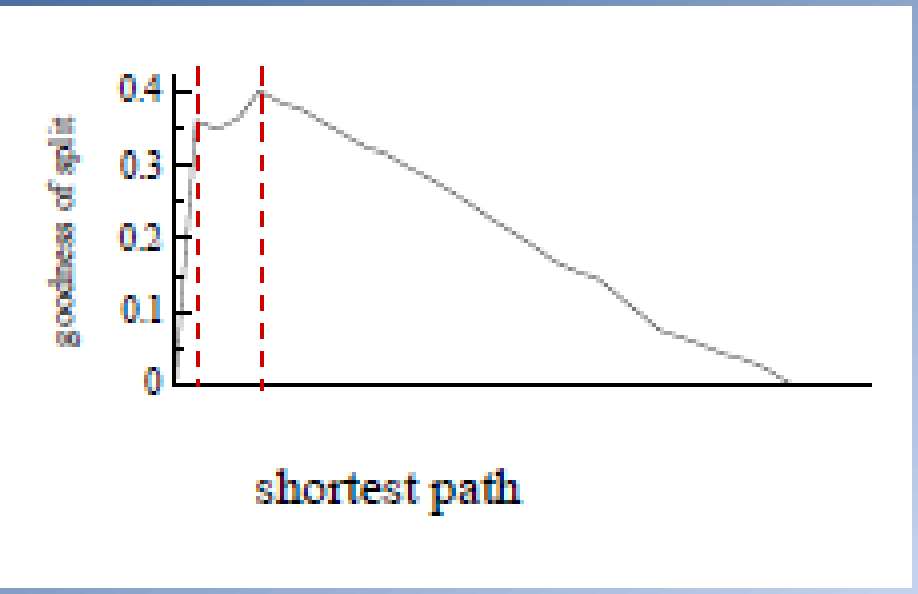
modularity measure defined as:

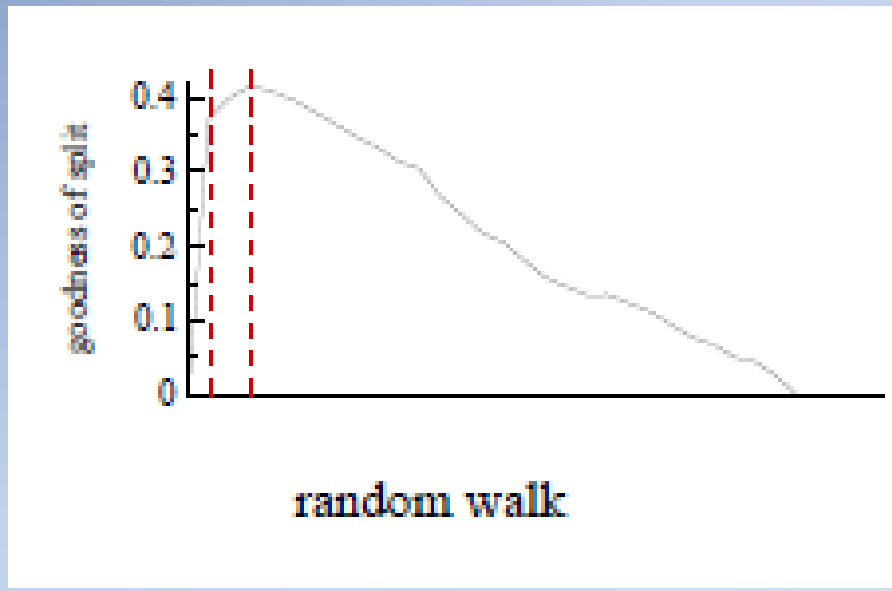
$$Q = \sum_i (e_{ii} - a_i^2) = \mathrm{Tr}\,\mathbf{e} - \left\| \mathbf{e}^2 \right\|,$$

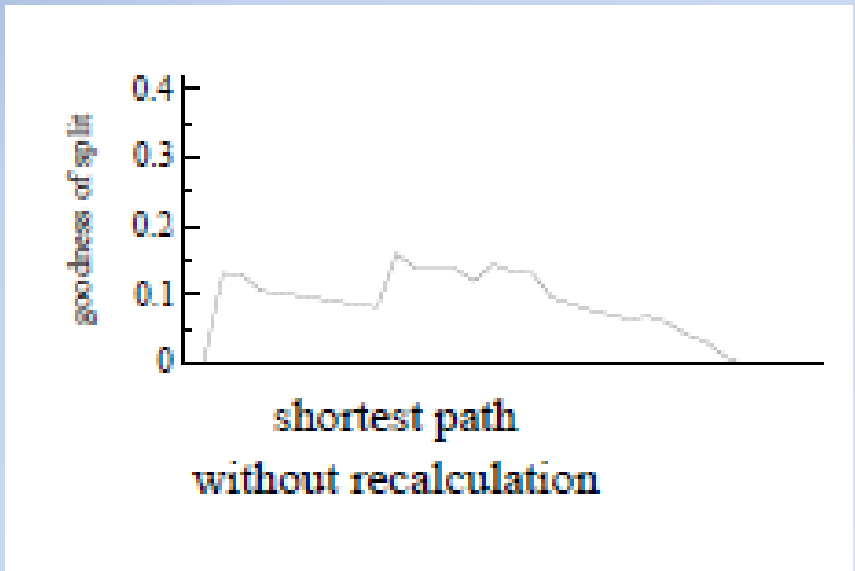
||x|| = the sum of elements of matrix x.

- Q = fraction of within-community edges – E[same quantity in a network with the same community divisions], but random connections between the vertices.
- typically values: 0.3 < Q < 0.7

Advantages


goodness of split
0.4
0.3
0.2
0.1
0
shortest path


goodness of split
0.4
0.3
0.2
0.1
0
random walk


goodness of split
0.4
0.3
0.2
0.1
0
shortest path
without recalculation

Parallel Implementation

(Yang and Lonardi)

UCSD

- ✓ Girvan and Newman's edge betweenness algorithm – computationally very intensive [time and memory issues]
- ✓ Yang and Lonardi came up with parallelized version: decreases time complexity. [s/w publicly available]

PROCEDURE:

1. Vertices evenly assigned to all processors (each processor has its own copy of the graph).
2. Procedure initiated by host processor; each processor performs BFS from all the vertices assigned to it and sums up partial pair-dependencies obtained from each BFS.
3. Partial pair-dependencies are sent to host processor, which is responsible for summing them up, thus obtaining the global betweenness values for every edge.

4. Edge with the highest betweenness value is then broadcast by the host processor to all the processors in the communication world.
5. All processors delete the edge received in their own graph copy
6. Next iteration is started. Process is continued until no edges are left in the graph.
7. The output is the order of removal of edges, which implicitly defines a hierarchical tree on the nodes of the graph.
8. The graph is then reconstructed by these when Q factor is calculated for every edge removal.
9. Clusters with maximum Q factor is declared as final answer.

Gmean Proposal

- Computational time issues addresses by parallel implementation
- Our proposal addresses memory issues by introducing an earlier stopping point. [Gmean algo in parallel thus saves time and memory]
- Gives only one set of final clusters unlike original algorithm which gives >1 possible sets of clusters (corresponding to Qmax) as final output.

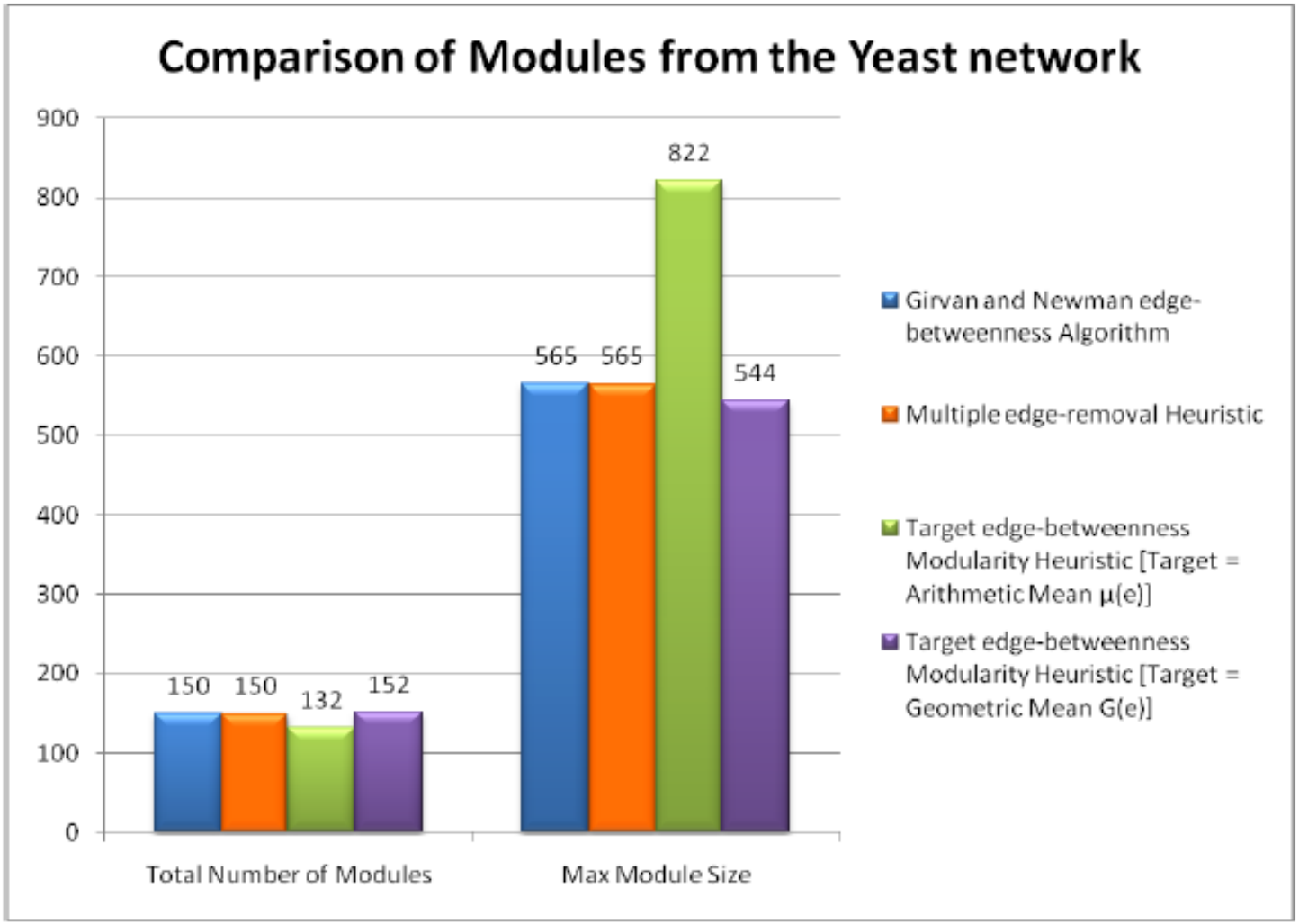
OBJECTIVE: To find an earlier stopping point

STEPS:

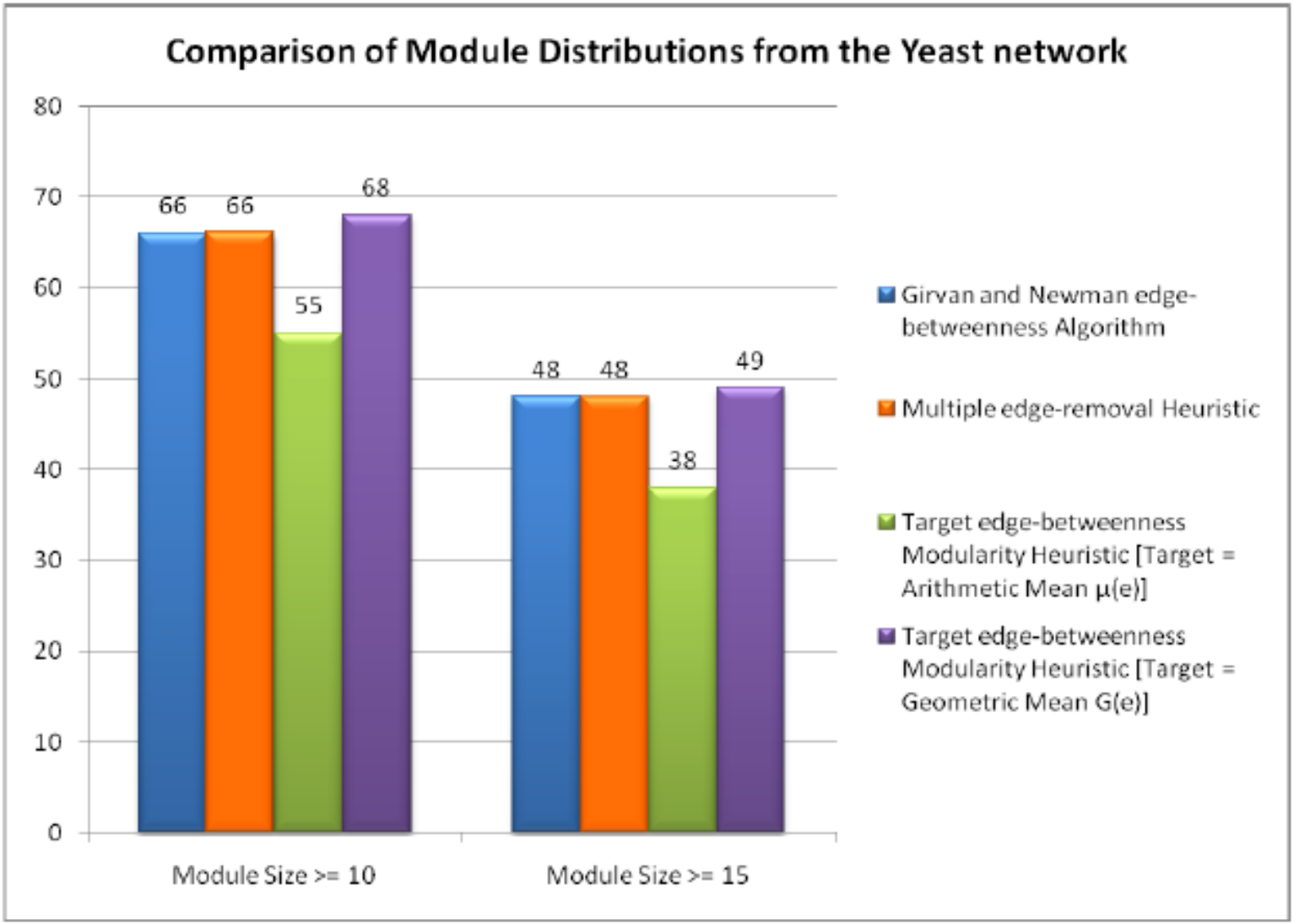
1. Calculate the betweenness for all edges in the original network
2. Calculate the mean (gmean) after only the first iteration
3. Remove the edge with the highest betweenness
4. Recalculate the betweenness for all edges
5. Repeat steps 3->4 till the value of edgebetweenness of edge to be removed is < the calculated gmean.

Results

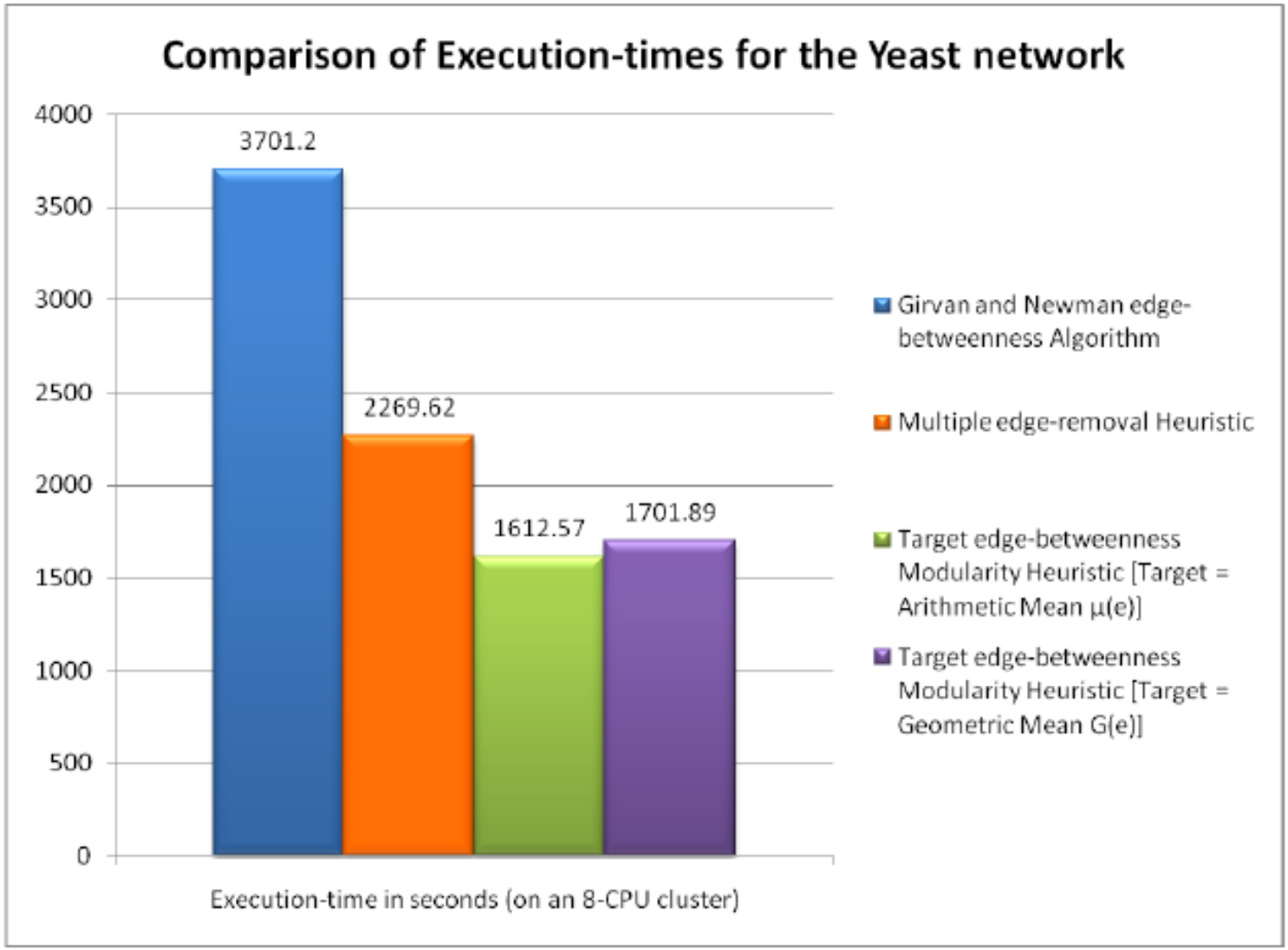
The following are the results for the Yeast network:



Hist 1: Comparison of overall resuts (in terms of modules) over all heuristics

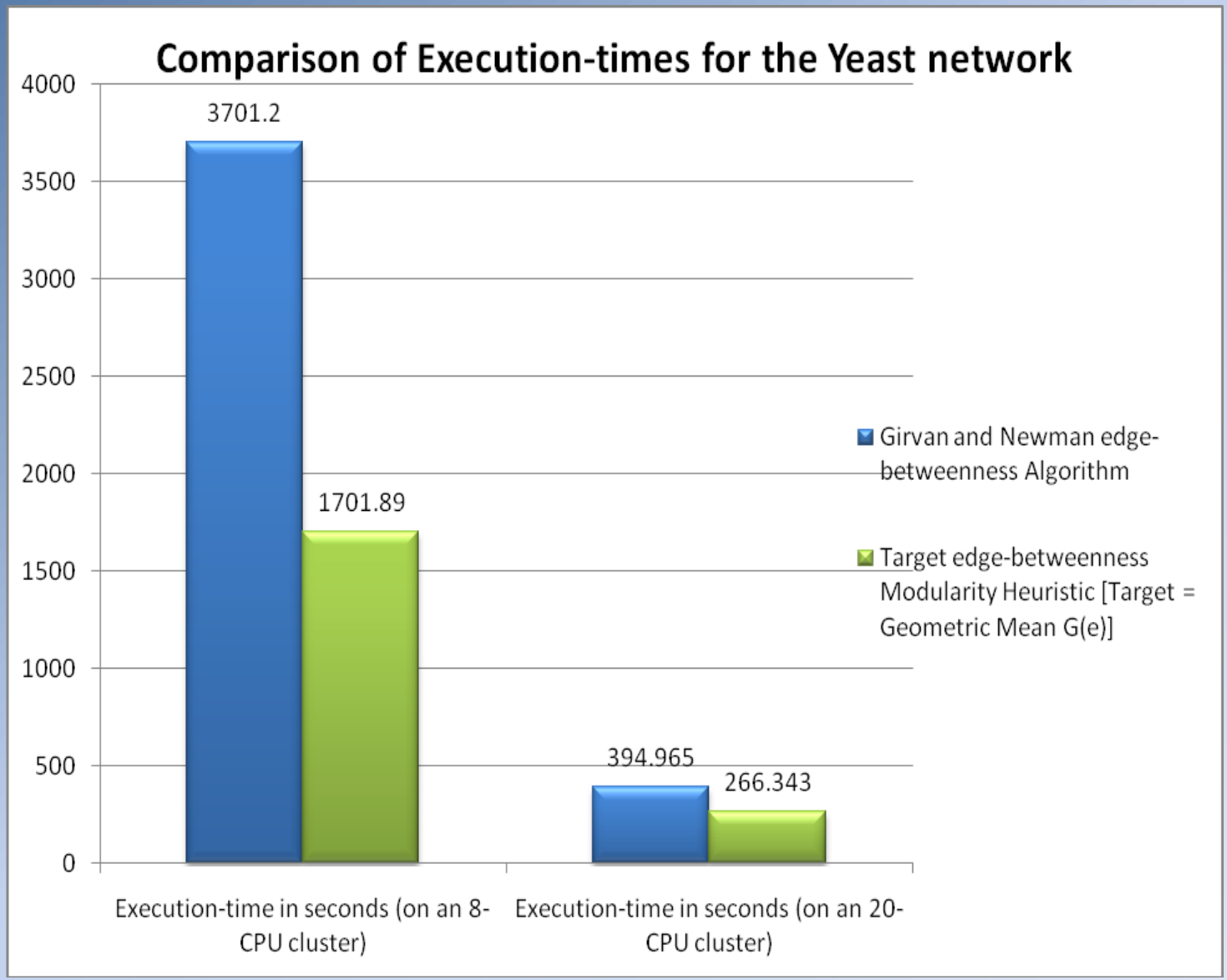


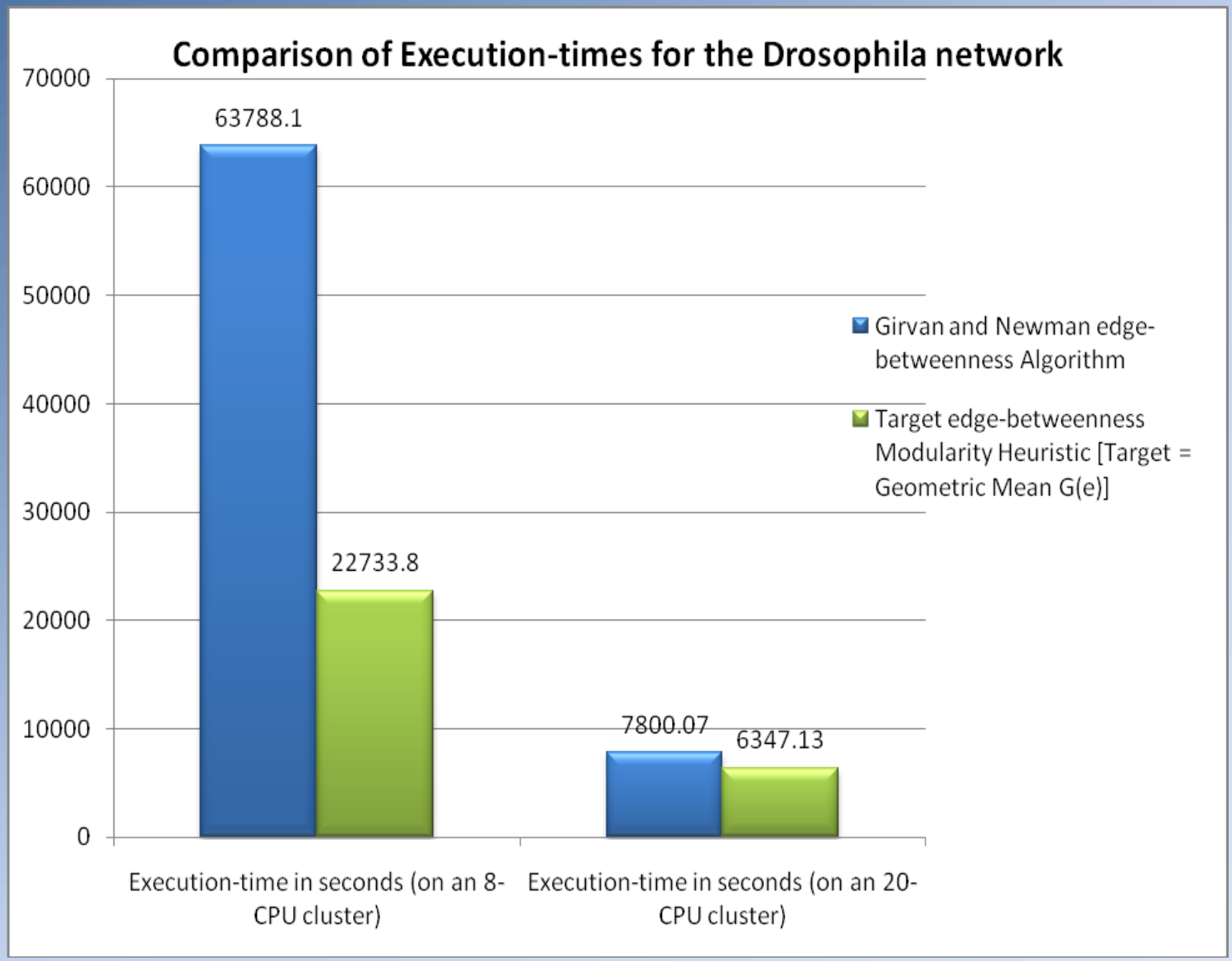
Hist 2: Comparison of module distribution over all heuristics

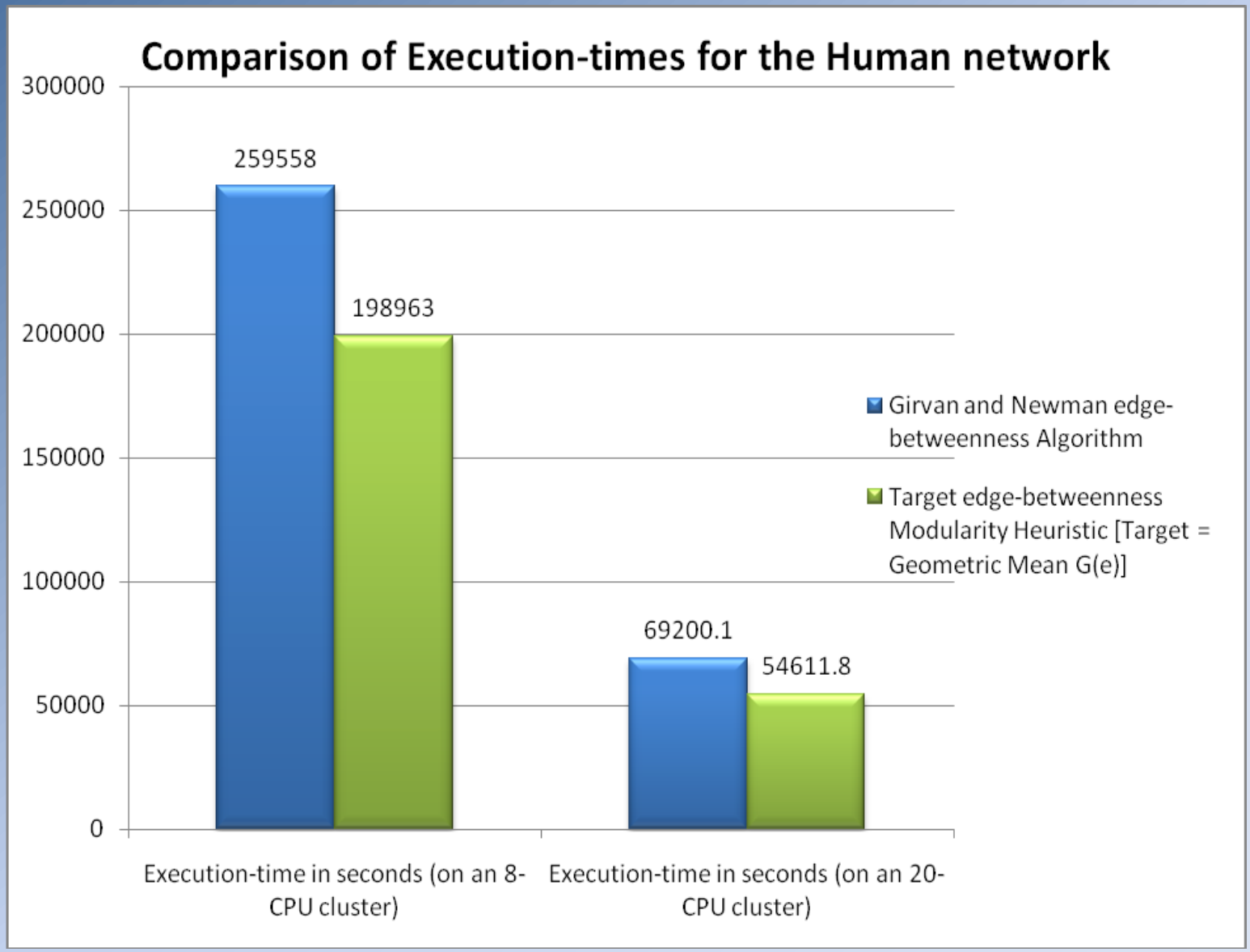


Hist 3: Comparison of time of execution over all heuristics

Comparison with number of processors




Comparison of Execution-times for the Drosophila network
70000
60000
50000
40000
30000
20000
10000
0
63788.1
22733.8
7800.07
6347.13
Girvan and Newman edge-betweenness Algorithm
Target edge-betweenness Modularity Heuristic [Target = Geometric Mean G(e)]
Execution-time in seconds (on an 8-CPU cluster)
Execution-time in seconds (on an 20-CPU cluster)


Comparison of Execution-times for the Human network
300000
250000
200000
150000
100000
50000
0
259558
198963
69200.1
54611.8
Girvan and Newman edge-betweenness Algorithm
Target edge-betweenness Modularity Heuristic [Target = Geometric Mean G(e)]
Execution-time in seconds (on an 8-CPU cluster)
Execution-time in seconds (on an 20-CPU cluster)

Execution times

[20 processors]

	Girvan and Newman edge-betweenness Algorithm	Target edge-betweenness modularity heuristic *Target = G(e)*
Yeast	394.96 sec	266.34 sec $\tau = 32.56\ \%$
Drosophila	7800.07 sec	6347.13 sec $\tau = 18.63\ \%$
H. Sapiens	69200.10 sec	54611.80 sec $\tau = 21.08\ \%$

Q factor comparison

	Qmax	*Gmean betweenness*	*ebtarget*	*Q target*
Yeast	*0.625419*	1780.33	1792.52	*0.605768*
Drosophila	*0.355212*	3687.32	3687.79	*0.336284*
Human	0.438003	1121.65	1121.77	0.371623

Though we do not calculate Q factor, the comparison of Q factors for the modules produced by gmean algorithm, with that of the original algorithms, proved consistency.

Applications

The definition and identification of modular graphs from really large complex real life networks help largely in the analysis of:

- Protein interaction n/w
- Metabolic reactions n/w
- Neuronal connectivity n/w

[and many such complex *biological networks*]

- Visual Segmentation
- Data analysis

[and other applications in the field of *Machine Learning*!]

- Social n/w [and other networks of interest to *statisticians*]