V7 – Biological PPI Networks

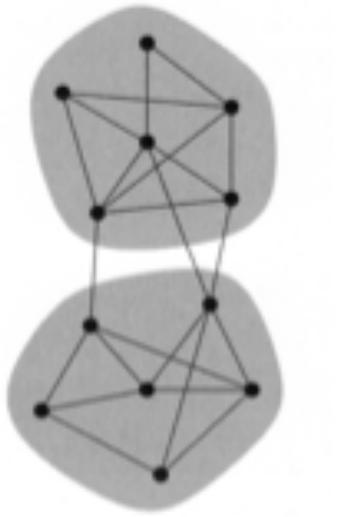
- graph bisection (-> communities)
- are biological networks really scale-free?
- network growth
- functional annotation in the network

Mon, Nov 14, 2016

Modularity: an example of graph partitioning

The simplest graph partitioning problem is the division of a network into just 2 parts. This is called **graph bisection**.

If we can divide a network into 2 parts, we can also divide it further by dividing one or both of these parts …



graph bisection problem: divide the vertices of a network into 2 non-overlapping groups of given sizes such that the **number of edges** running **between** vertices in **different groups is minimized**.

The number of edges between groups is called the **cut size**.

In principle, one could simply look through all possible divisions of the network into 2 parts and choose the one with smallest cut size.

Algorithms for graph partitioning

But this exhaustive search is prohibitively expensive!

Given a network of n vertices. There are $\frac{n!}{n_1!n_2!}$ different ways of dividing it into 2 groups of n_1 and n_2 vertices.

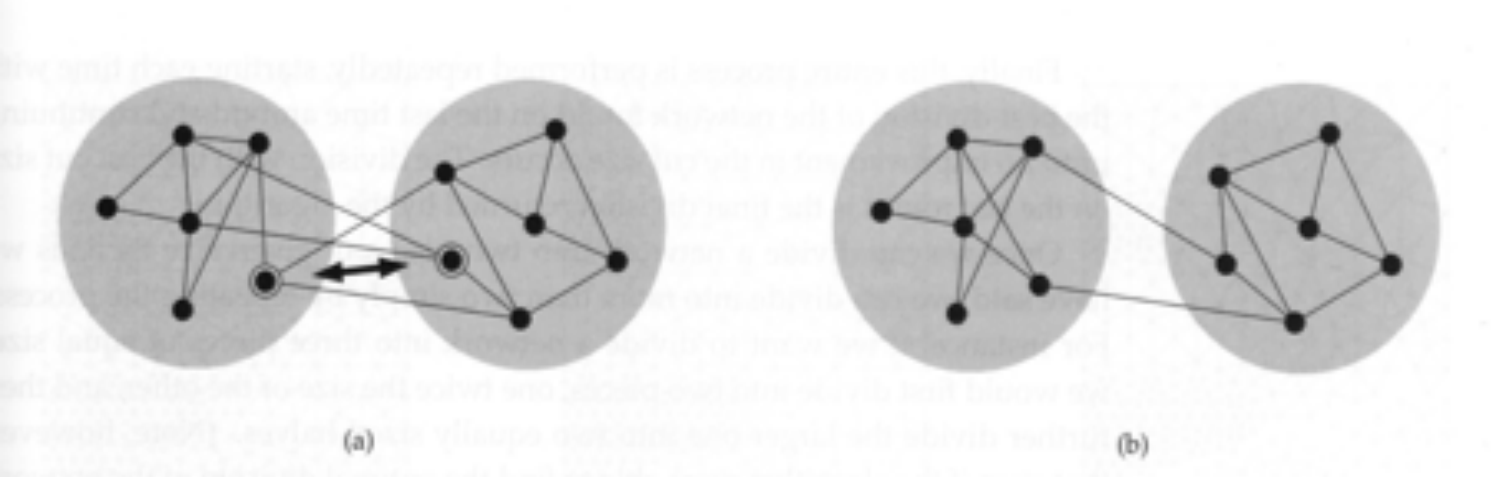
The amount of time to look through all these divisions will go up roughly exponentially with the size of the system.

Only values of up to n = 30 are feasible with today‘s computers.

In computer science, either an algorithm can be clever and run quickly, but will fail to provide the optimal answer in some (or perhaps in many) cases, or it will always find the optimal answer, but takes an impractical length of time to do so.

The Kernighan-Lin algorithm

This algorithm proposed by Brian Kernighan and Shen Lin in 1970 is one of the simplest and best known **heuristic** algorithms for the graph bisection problem.
(Kernighan is also one of the developers of the C language).



(a) The algorithm starts with any division of the vertices of a network into two groups (shaded) and then searches for pairs of vertices, such as the pair highlighted here, whose **interchange** would **reduce** the **cut size** between the groups.
(b) The same network after interchange of the 2 vertices.

The Kernighan-Lin algorithm

(1) Divide the vertices of a given network into 2 groups (e.g. randomly).

(2) For each pair *(i,j)* of vertices, where *i* belongs to the first group and *j* to the second group, calculate **how much** the **cut size** between the groups would **change** if *i* and *j* were interchanged between the groups.

(3) Find the pair that reduces the cut size by the largest amount and swap the vertices.

If no pair reduces it, find the pair that increases it by the smallest amount.

Repeat this process, but with the important restriction that **each vertex** in the network can **only** be **moved once**.

Stop when there is no pair of vertices left that can be swapped.

The Kernighan-Lin algorithm (II)

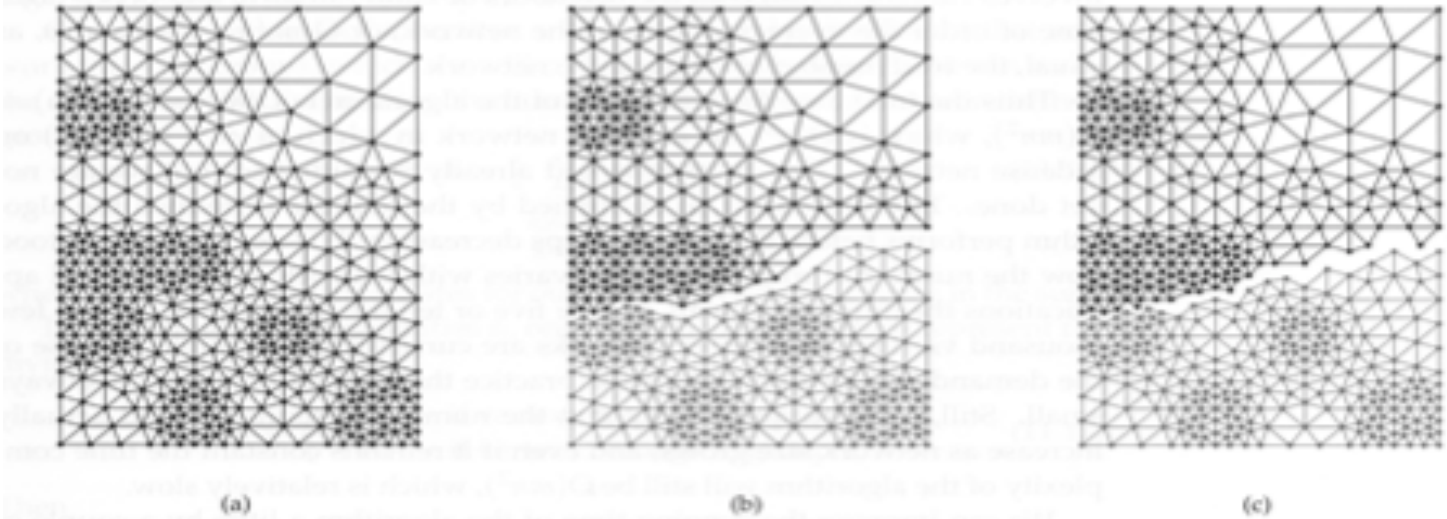
(3) Go back through every state that the network passed through during the swapping procedure and choose among them the state in which the cut size takes its smallest value.

(4) Perform this entire process repeatedly, starting each time with the best division of the network found in the last round.

(5) Stop when no improvement on the cut size occurs.

Note that if the initial assignment of vertices to groups is done randomly, the Kernighan-Lin algorithm may give (slightly) different answers when it is run twice on the same network.

The Kernighan-Lin algorithm (II)



(a) A mesh network of 547 vertices of the kind commonly used in finite element analysis.

(b) The best division found by the Kernighan-Lin algorithm when the task is to split the network into 2 groups of almost equal size.

This division involves cutting 40 edges in this mesh network and gives parts of 273 and 274 vertices.

(c) The best division found by spectral partitioning (alternative method).

Runtime of the Kernighan-Lin algorithm

The number of swaps performed during one round of the algorithm is equal to the smaller of the sizes of the two groups $\in [0, n / 2]$.

$\rightarrow$ in the worst case, there are $O(n)$ swaps.

For each swap, we have to examine all pairs of vertices in different groups to determine how the cut size would be affected if the pair was swapped.

At most (if both groups have the same size),
there are $n / 2 \times n / 2 = n^2 / 4$ such pairs, which is $O(n^2)$.

Runtime of the Kernighan-Lin algorithm (ii)

When a vertex *i* moves from one group to the other group, any edges connecting it to vertices in its current group become edges between groups after the swap.

Let us suppose that there are k_i^{same} such edges.

Similarly, any edges that *i* has to vertices in the other group, (say k_i^{other} ones) become within-group edges after the swap.

There is one exception. If *i* is being swapped with vertex *j* and they are connected by an edge, then the edge is still between the groups after the swap

→ the change in the cut size due to the movement of *i* is $-(k_i^{other} - k_i^{same} - A_{ij})$

A similar expression applies for vertex *j*.

→ the total change in cut size due to the swap is

$$-(k_i^{other} - k_i^{same} + k_j^{other} - k_j^{same} - 2A_{ij})$$

Runtime of the Kernighan-Lin algorithm (iii)

For a network stored in adjacency list form, the evaluation of this expression involves running through all the neighbors of *i* and *j* in turn, and hence takes time on the order of the average degree in the network, or *O (m/n)* with *m* edges in the network.

→ the total running time is $O (n \times n^2 \times m/n) = O(mn^2)$.

For a sparse network with $m \propto n$, this is $O(n^3)$.

For a dense network (with $m \to \frac{n(n-1)}{2}$) , this is $O(n^4)$.

This time still needs to be **multiplied** by the **number** of **rounds** the algorithm is run before the cut size stops decreasing.
For networks up to a few 1000 of vertices, this number may be between 5 and 10.

Network Growth Mechanisms

Given: an observed PPI network → how did it grow (evolve)?

Inferring network mechanisms: The *Drosophila melanogaster* protein interaction network

Manuel Middendorf[†], Etay Ziv[‡], and Chris H. Wiggins[§¶]

[†]Department of Physics, [‡]College of Physicians and Surgeons, [§]Department of Applied Physics and Applied Mathematics, and [¶]Center for Computational Biology and Bioinformatics, Columbia University, New York, NY 10027

Communicated by Barry H. Honig, Columbia University, New York, NY, December 20, 2004 (received for review September 7, 2004)

PNAS **102** (2005) 3192

Look at **network motifs** (local connectivity):
compare motif distributions from various network prototypes to fly network

Idea: each growth **mechanism** leads to a typical motif **distribution**,
even if global measures are comparable

The Fly Network

Y2H PPI network for *D. melanogaster* from Giot et al. [*Science* **302** (2003) 1727]

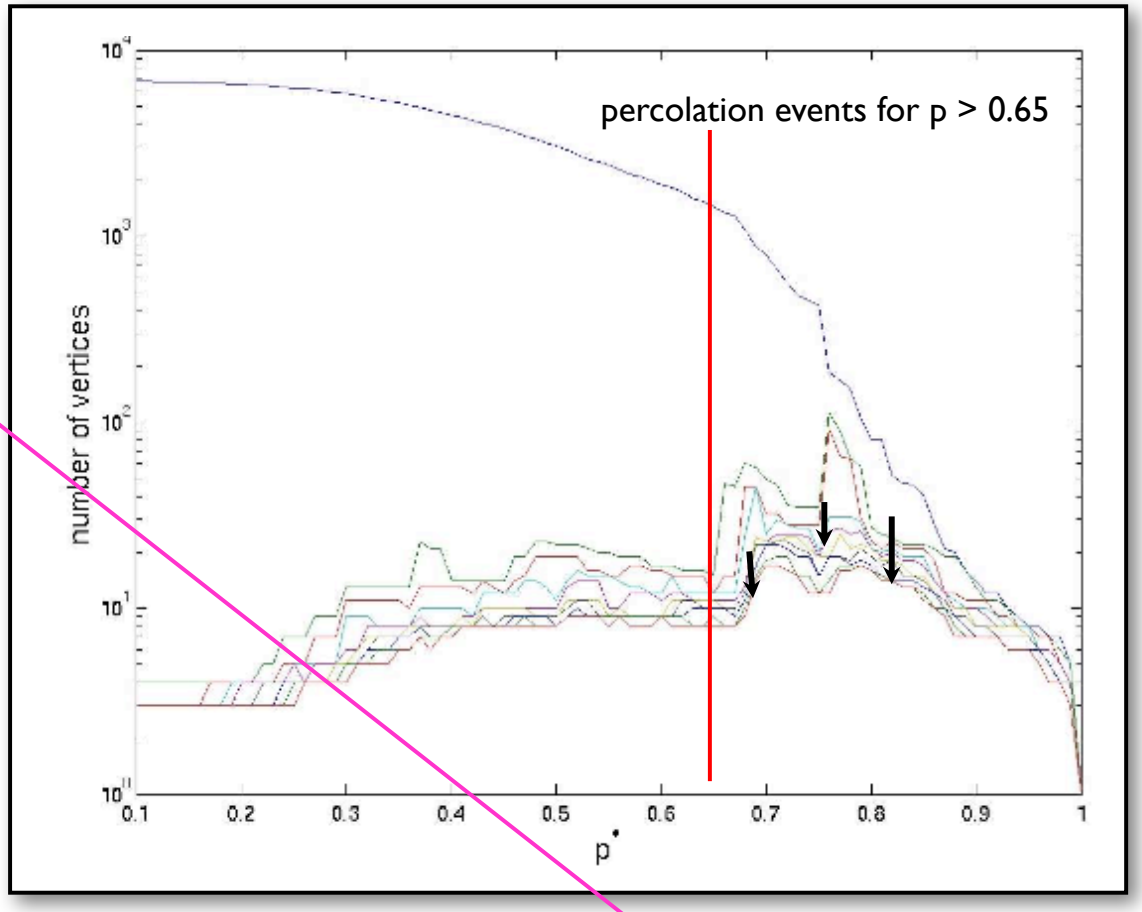
Giot *et al.* assigned a confidence score [0, 1] for every observed interaction.

→ use only data with $p > 0.65$ (0.5) because …

→ remove self-interactions and isolated nodes

High confidence network with 3359 (4625) nodes and 2795 (4683) edges.

Use prototype networks of same size for training.



Size of largest components. At p = 0.65, there is one large component with 1433 nodes and the other 703 components contain at most 15 nodes.

Middendorf et al, *PNAS* **102** (2005) 3192

Network subgraphs -> motives

All non-isomorphic subgraphs that can be generated with a walk of length 8

Middendorf et al, *PNAS* **102** (2005) 3192

Growth Mechanisms

Generate 1000 networks, each, of the following 7 types
(same size as fly network, undefined parameters were scanned)

DMC	Duplication-mutation, preserving complementarity
DMR	Duplication with random mutations
RDS	Random static networks
RDG	Random growing network
LPA	Linear preferential attachment network (Albert-Barabasi)
AGV	Aging vertices network
SMW	Small world network

Growth Type 1: DMC

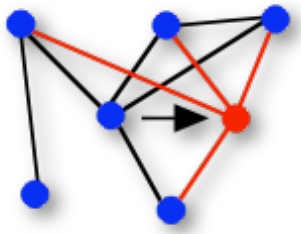
"Duplication – mutation with preserved complementarity"

Evolutionary idea: gene **duplication**, followed by a partial **loss** of function of one of the copies, making the other copy essential

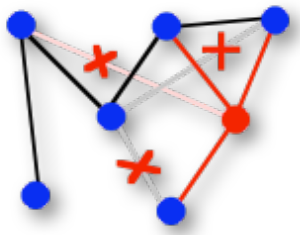
Algorithm:

Start from two connected nodes

- duplicate existing node with all interactions



- for all neighbors: delete with probability q_{del} either link from original node **or** from copy



Repeat these steps many (e.g. $N - 2$) times

Growth Type 2: DMR

"Duplication with random mutations"

Gene duplication, but no correlation between original and copy
(original unaffected by copy)

Algorithm:

Start growth from five-vertex cycle,
repeat *N* - 5 times:

- duplicate existing node with all interactions
- for all neighbors: delete with probability q_{del} link from copy
- add **new links** to non-neighbors with probability q_{new}/n

Growth Types 3–5: RDS, RDG, and LPA

RDS = static random network

Start from *N* nodes, add *L* links randomly

RDG = growing random network

Start from small random network, add nodes,
then edges between all existing nodes

LPA = linear preferential attachment

Add new nodes similar to Barabási-Albert algorithm,
but with preference according to $(k_i + \alpha)$, $\alpha = 0 \ldots 5$
(BA for $\alpha = 0$)

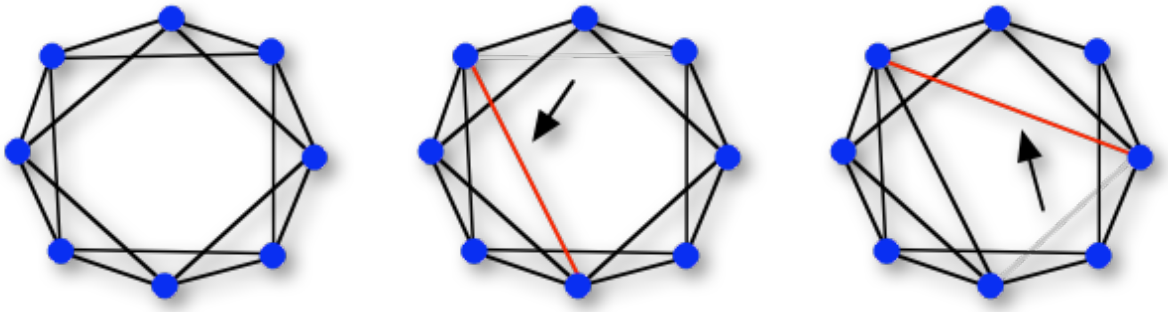
Growth Types 6-7: AGV and SMW

AGV = aging vertices network

Like growing random network,
but preference decreases with age of the node
→ citation network: more recent publications are cited more likely

SMW = small world networks, see Watts, Strogatz, *Nature* **363**, 202 (1998)

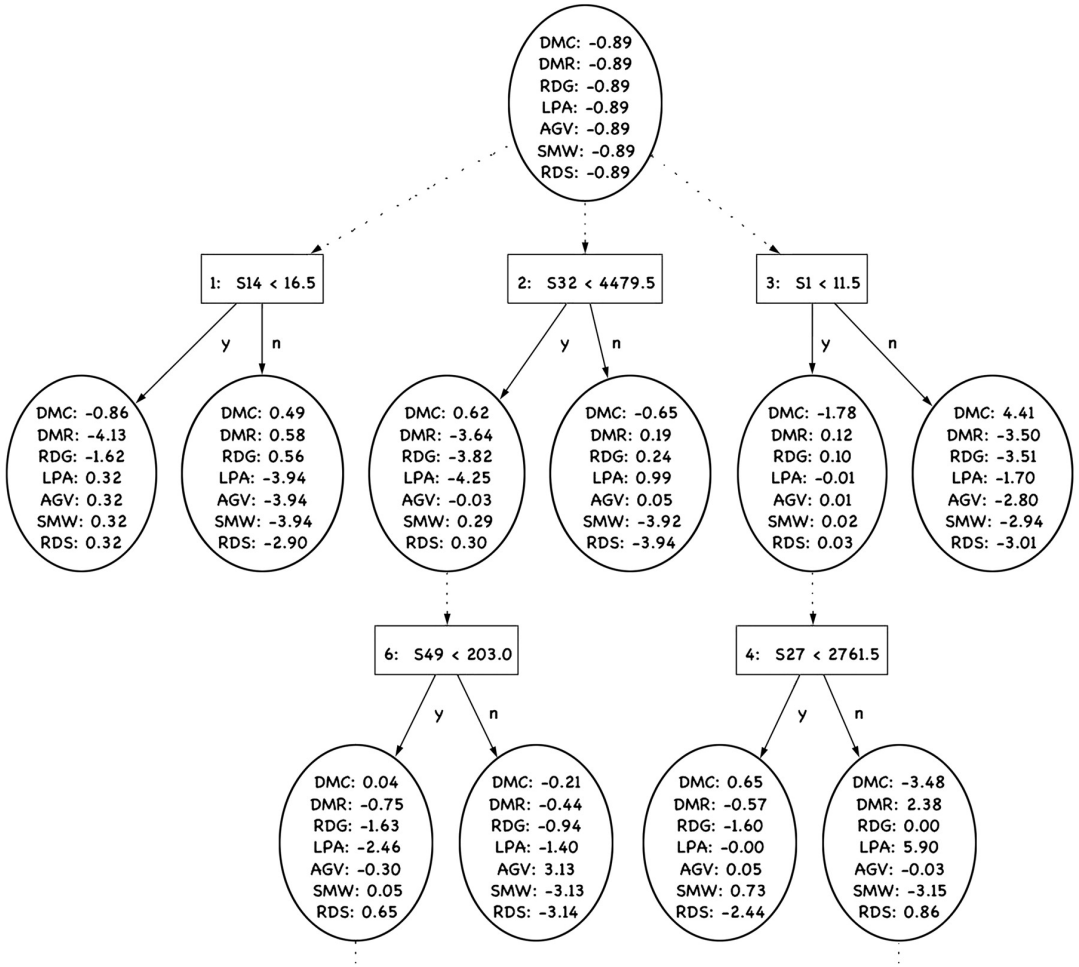
Randomly rewire regular ring lattice



Alternating Decision Tree Classifier

Trained with the motif counts from 1000 networks of each of the 7 types

→ prototypes are well separated and can be reliably classified



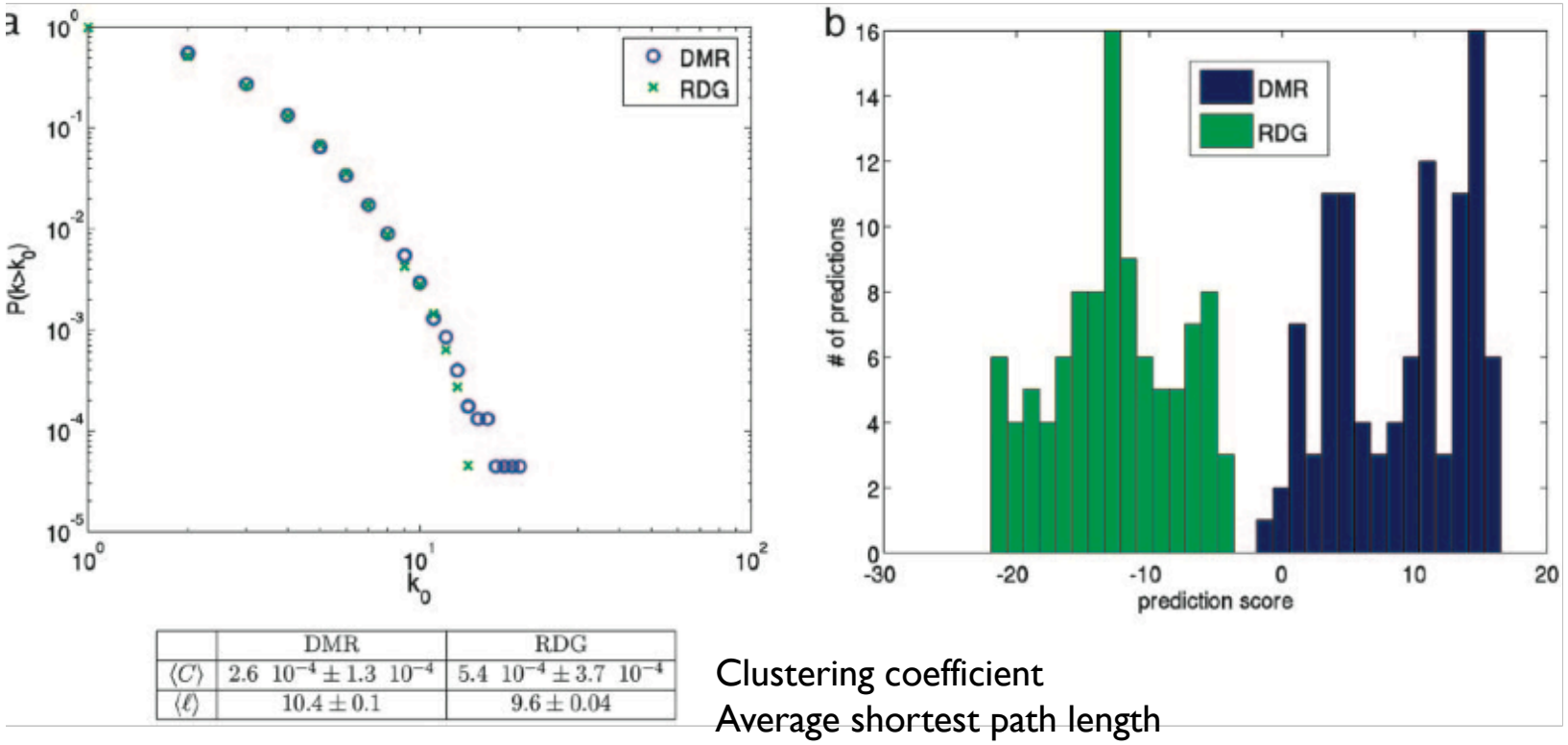
Decision nodes count occurrence of subgraphs

Prediction accuracy for networks similar to fly network with $p = 0.5$:

	Prediction						
Truth	DMR	DMC	AGV	LPA	SMW	RDS	RDG
DMR	99.3	0.0	0.0	0.0	0.0	0.1	0.6
DMC	0.0	99.7	0.0	0.0	0.3	0.0	0.0
AGV	0.0	0.1	84.7	13.5	1.2	0.5	0.0
LPA	0.0	0.0	10.3	89.6	0.0	0.0	0.1
SMW	0.0	0.0	0.6	0.0	99.0	0.4	0.0
RDS	0.0	0.0	0.2	0.0	0.8	99.0	0.0
RDG	0.9	0.0	0.0	0.1	0.0	0.0	99.0

Middendorf et al, *PNAS* **102** (2005) 3192

Are the generated networks different?



	DMR	RDG
$\langle C \rangle$	$2.6\ 10^{-4} \pm 1.3\ 10^{-4}$	$5.4\ 10^{-4} \pm 3.7\ 10^{-4}$
$\langle \ell \rangle$	10.4 ± 0.1	9.6 ± 0.04

Clustering coefficient
Average shortest path length

Example: DMR vs. RDG: Similar global parameters <C> and <l> (left),
but different counts of the network motifs (right)

-> networks can (only) be perfectly separated by motif-based classifier

Middendorf et al, *PNAS* **102** (2005) 3192

How Did the Fly Evolve?

	Eight-step subgraphs ($p^* = 0.65$)		Subgraphs with up to seven edges ($p^* = 0.65$)		Eight-step subgraphs ($p^* = 0.5$)	
Rank	Class	Score	Class	Score	Class	Score
1	DMC	8.2 ± 1.0	DMC	8.6 ± 1.1	DMC	0.8 ± 2.9
2	DMR	−6.8 ± 0.9	DMR	−6.1 ± 1.7	DMR	−2.1 ± 2.0
3	RDG	−9.5 ± 2.3	RDG	−9.3 ± 1.6	AGV	−3.1 ± 2.2
4	AGV	−10.6 ± 4.2	AGV	−11.5 ± 4.1	LPA	−10.1 ± 3.1
5	LPA	−16.5 ± 3.4	LPA	−14.3 ± 3.2	SMW	−20.6 ± 1.9
6	SMW	−18.9 ± 0.7	SMW	−18.3 ± 1.9	RDS	−22.3 ± 1.7
7	RDS	−19.1 ± 2.3	RDS	−19.9 ± 1.5	RDG	−22.5 ± 4.7

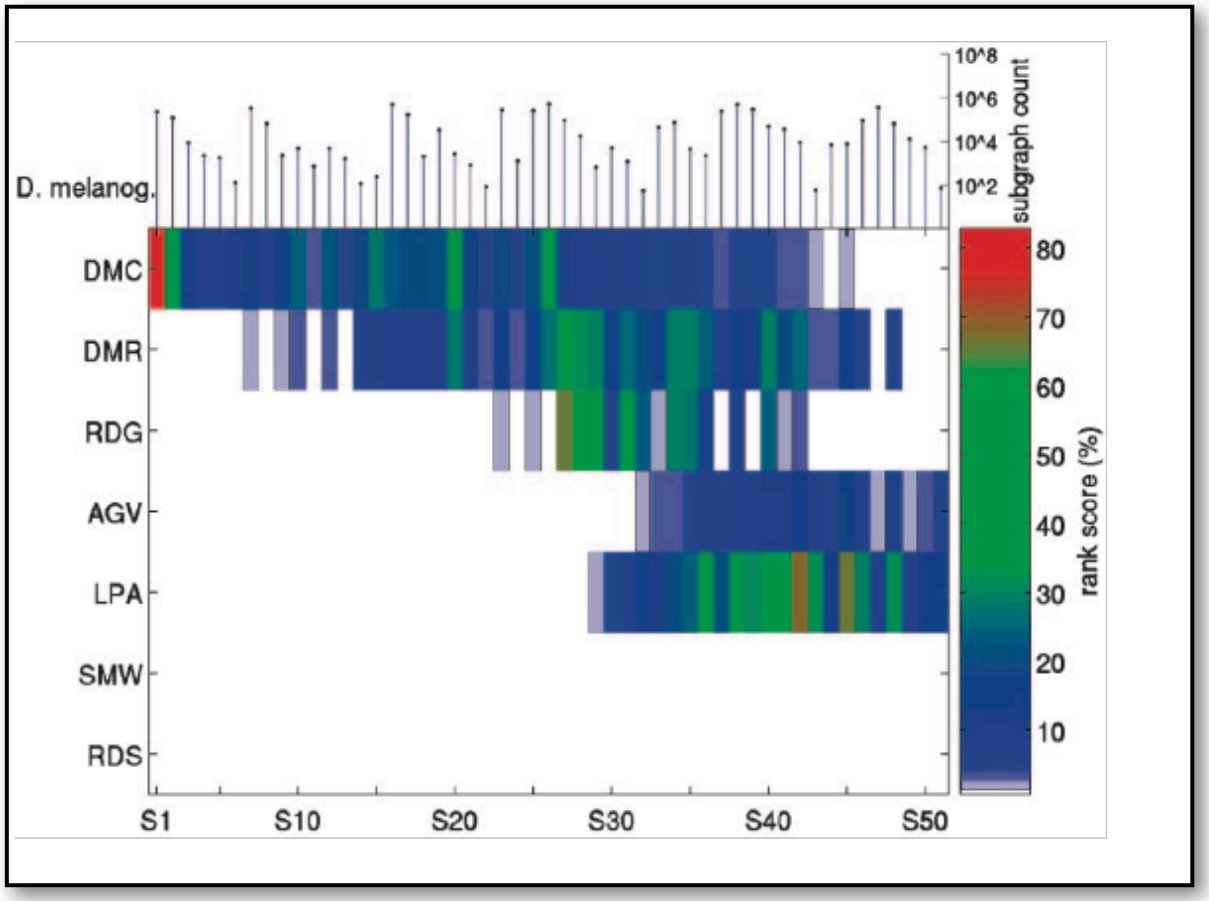
Drosophila is consistently (independently of the cut-off in subgraph size) classified as a DMC network, with an especially strong prediction for a confidence threshold of $p^* = 0.65$.

→ Best overlap with DMC (Duplication-mutation, preserved complementarity)

→ Scale-free (LPA) or random networks (RDS/RDG) are very unlikely

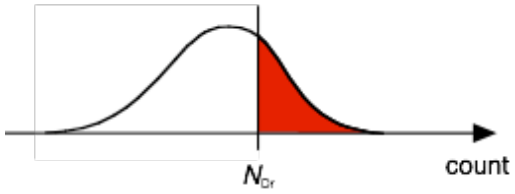
Middendorf et al, *PNAS* **102** (2005) 3192

Motif Count Frequencies



-> DMC and DMR networks contain most subgraphs in similar amount as fly network (top).

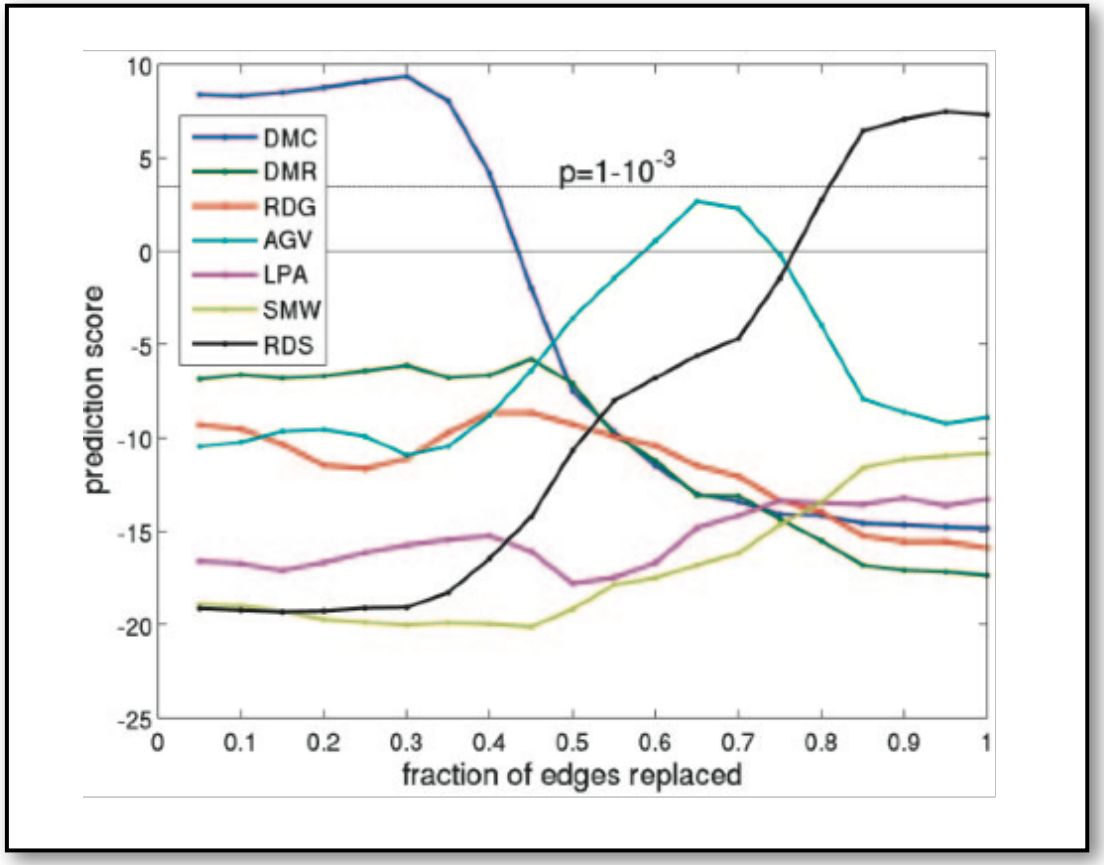
rank score: fraction of test networks with a higher count than Drosophila
(50% = same count as fly on avg.)



Middendorf et al, *PNAS* **102** (2005) 3192

Experimental Errors?

Randomly replace edges in **fly** network and **classify** again:



→ Classification **unchanged** for ≤ **30%** incorrect edges,
at higher values RDS takes over (as to be expected)

Summary (I)

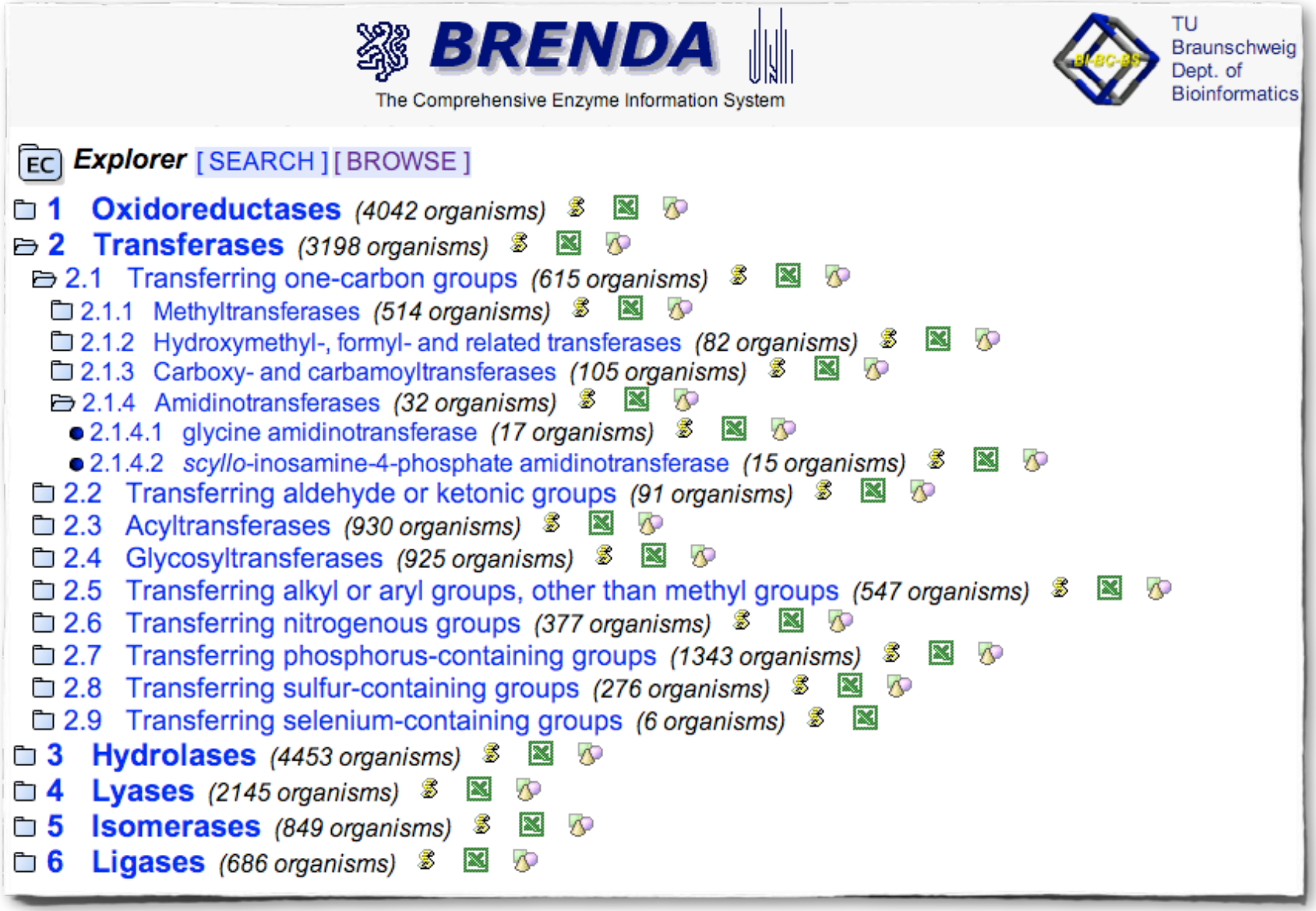
Sampling matters!

→ "Scale-free" *P(k)* is obtained by sparse sampling from many network types

Test different **hypotheses** for

- **global** features
  → depends on unknown parameters and sampling
    → no clear statement possible
- **local** features (motifs)
  → are better preserved
    → DMC best among tested prototypes

What Does a Protein Do?



Enzyme **C**lassification scheme
(from http://www.brenda-enzymes.org/)

What about Un-Classified Proteins?

BIOINFORMATICS

Vol. 21 Suppl. 1 2005, pages i302–i310
doi:10.1093/bioinformatics/bti1054

Whole-proteome prediction of protein function via graph-theoretic analysis of interaction maps

Elena Nabieva[1,2], Kam Jim[2], Amit Agarwal[1], Bernard Chazelle[1] and Mona Singh[1,2,]*

[1]Computer Science Department and [2]Lewis-Sigler Institute for Integrative Genomics, Princeton University, Princeton, NJ 08544, USA

Received on January 15, 2005; accepted on March 27, 2005

Many **unclassified proteins**:

→ estimate: ~1/3 of the yeast proteome not annotated functionally

→ BioGRID: 4495 proteins in the largest cluster of the yeast physical interaction map.

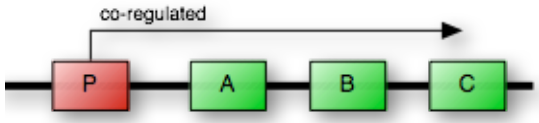
only 2946 have a MIPS functional annotation

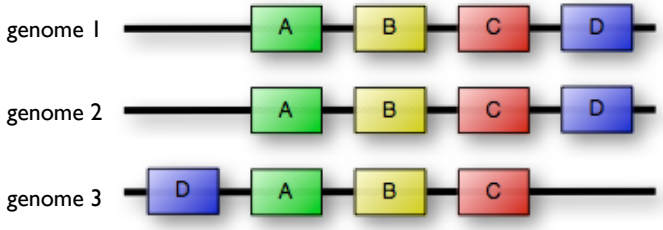
Partition the Graph

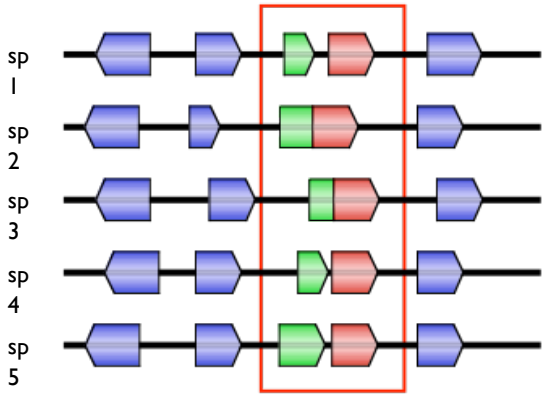
Large **PPI networks** can be built from (see V3, V4, V5):

- HT experiments (Y2H, TAP, synthetic lethality, coexpression, coregulation, …)
- predictions (gene profiling, gene neighborhood, phylogenetic profiles, …)

→ proteins that are functionally linked





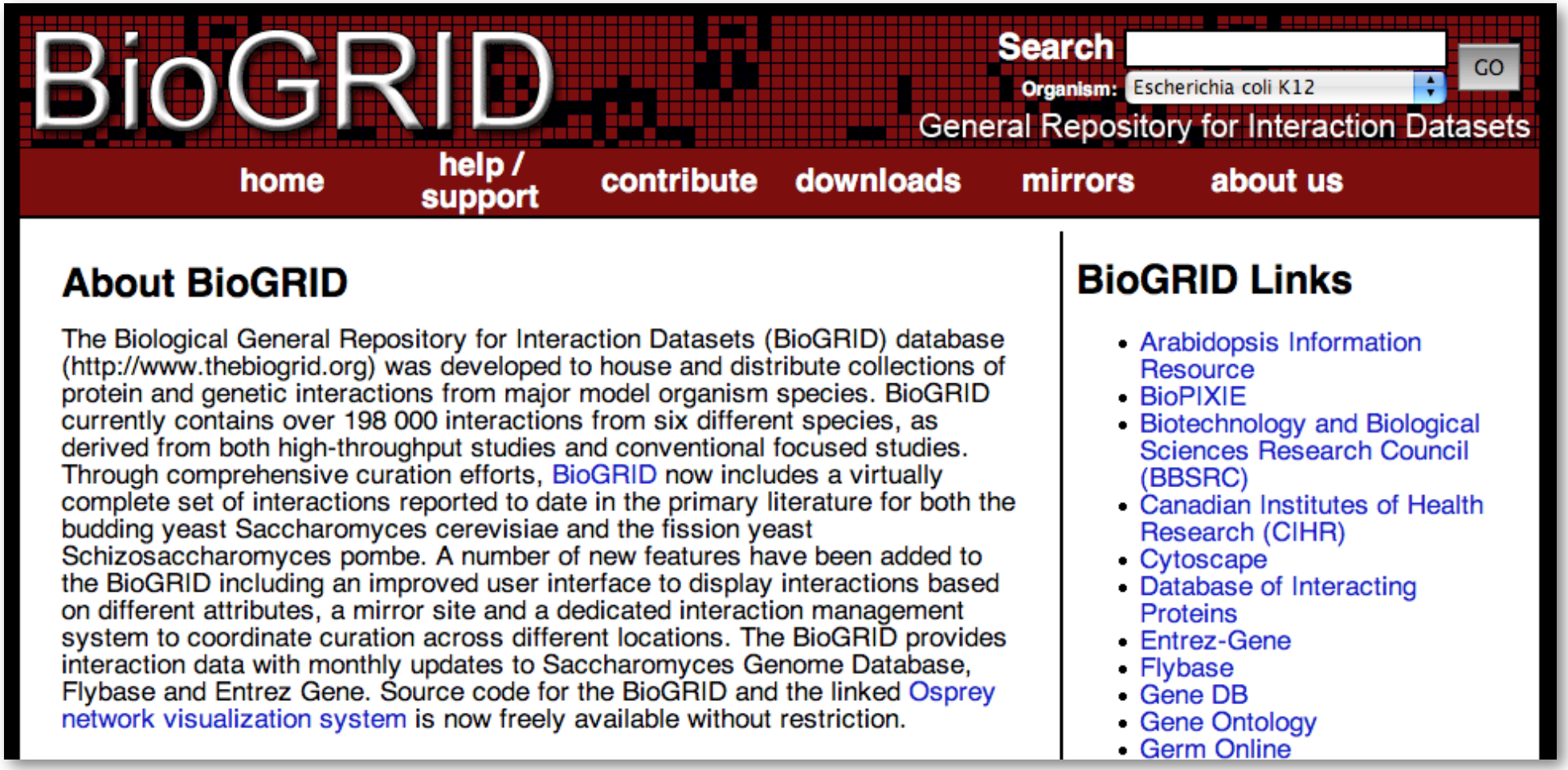


Identify **unknown functions** from **clustering** of these networks by, e.g.:

- shared interactions (similar neighborhood)
- membership in a community
- similarity of shortest path vectors to all other proteins (= similar path into the rest of the network)

Protein Interactions

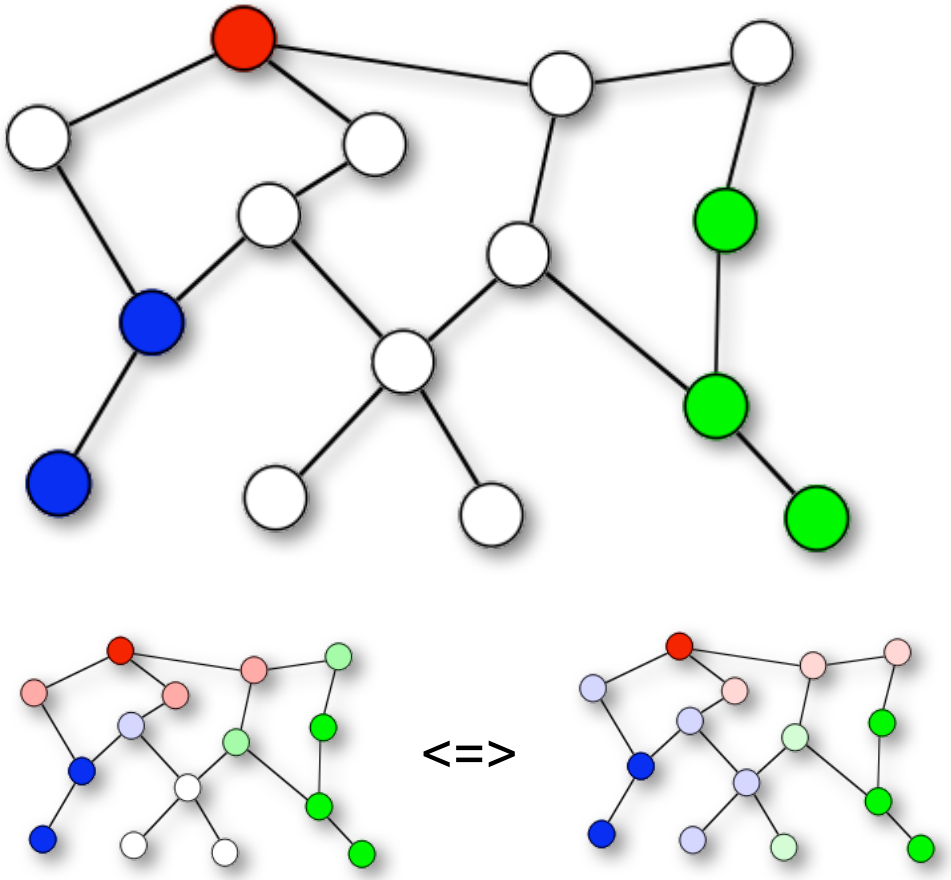
Nabieva et al used the *S. cerevisiae* dataset from GRID of 2005 (now BioGRID)
→ 4495 proteins and 12 531 physical interactions in the largest cluster



http://www.thebiogrid.org/about.php

Function Annotation

Task: **predict** function (= functional annotation) for an unlabeled protein from the **available** annotations of other proteins in the network



Similar task:
How to **assign colors** to the white nodes?

Use information on:

- distance to colored nodes
- local connectivity
- reliability of the links
- …

Algorithm I: Majority

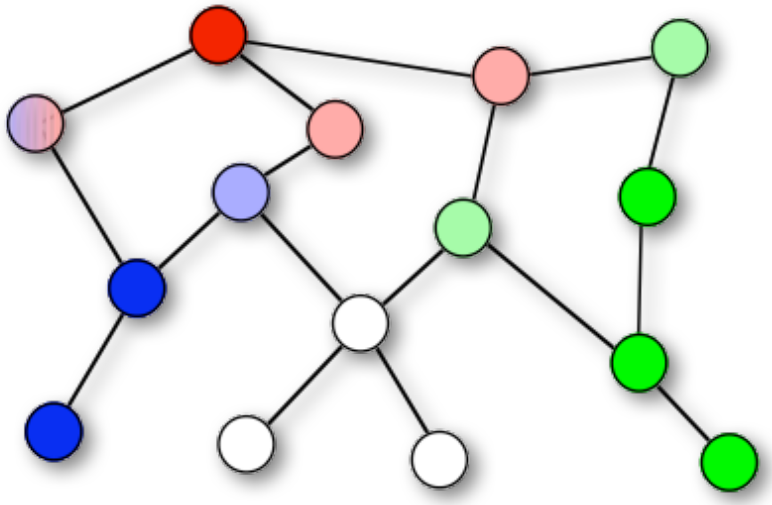
This concept was presented in

Schwikowski, Uetz, and Fields, " A network of protein–protein interactions in yeast" *Nat. Biotechnol.* **18** (2000) 1257

Consider all direct neighbors and **sum up** how often a certain **annotation occurs**

→ score for an annotation = count among the direct neighbors

→ take the 3 most frequent functions



Majority makes only limited use of the local connectivity

→ cannot assign function to next-neighbors

For weighted graphs:

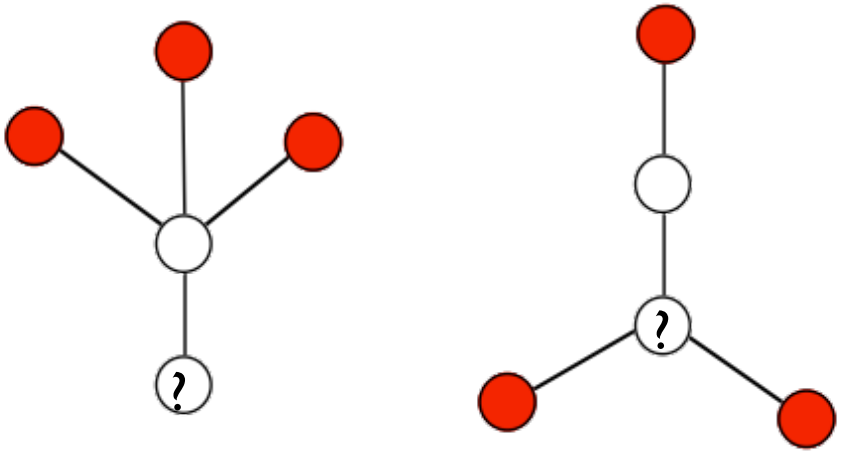
→ use weighted sum

Extended Majority: Neighborhood

This concept was presented in

Hishigaki, Nakai, Ono, Tanigami, and Takagi, "Assessment of prediction accuracy of protein function from protein–protein interaction data",
Yeast **18** (2001) 523

Look for **overrepresented** functions within a given **radius** of 1, 2, or 3 links
→ use as function score the value of a χ^2–test



Neighborhood algorithm does not consider local network topology

Both examples (left) are treated **identically** with r = 2
although the right situation feels more certain (2 direct neighbors of ? are labeled)

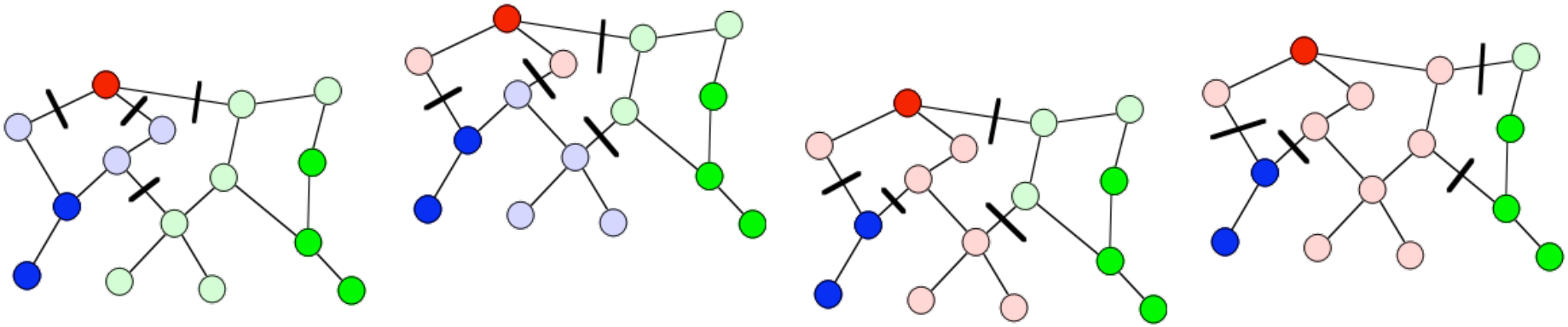
Minimize Changes: GenMultiCut

This concept was presented in

Karaoz, Murali, Letovsky, Zheng, Ding, Cantor, and Kasif, "Whole-genome annotation by using evidence integration in functional-linkage networks"
PNAS **101** (2004) 2888

"Annotate proteins so as to **minimize** the number of times that **different** functions are associated to **neighboring** (i.e. interacting) proteins"

→ generalization of the multiway *k*-cut problem for weighted edges,
can be stated as an integer linear program (ILP)



Multiple possible solutions → scores from **frequency** of annotations

Nabieva *et al*: FunctionalFlow

Extend the idea of **"guilty by association"**

→ each annotated protein is considered as a source of "function"-flow

→ propagate/simulate for a few time steps

→ choose the annotation *a* with the highest accumulated flow

Each node *u* has a reservoir $R_t(u)$, each edge a capacity constraint (weight) $w_{u,v}$

Initially: $R_0^a(u) = \begin{cases} \infty, & \text{if } u \text{ is annotated with } a, \\ 0, & \text{otherwise.} \end{cases}$ and $g_0^a(u,v) = 0$

Then: **downhill flow** from node *u* to neighbor node *v*:

$$g_t^a(u,v) = \begin{cases} 0, & \text{if } R_{t-1}^a(u) < R_{t-1}^a(v) \\ \min\left(w_{u,v}, \frac{w_{u,v}}{\sum_{(u,y)\in E} w_{u,y}}\right), & \text{otherwise.} \end{cases}$$

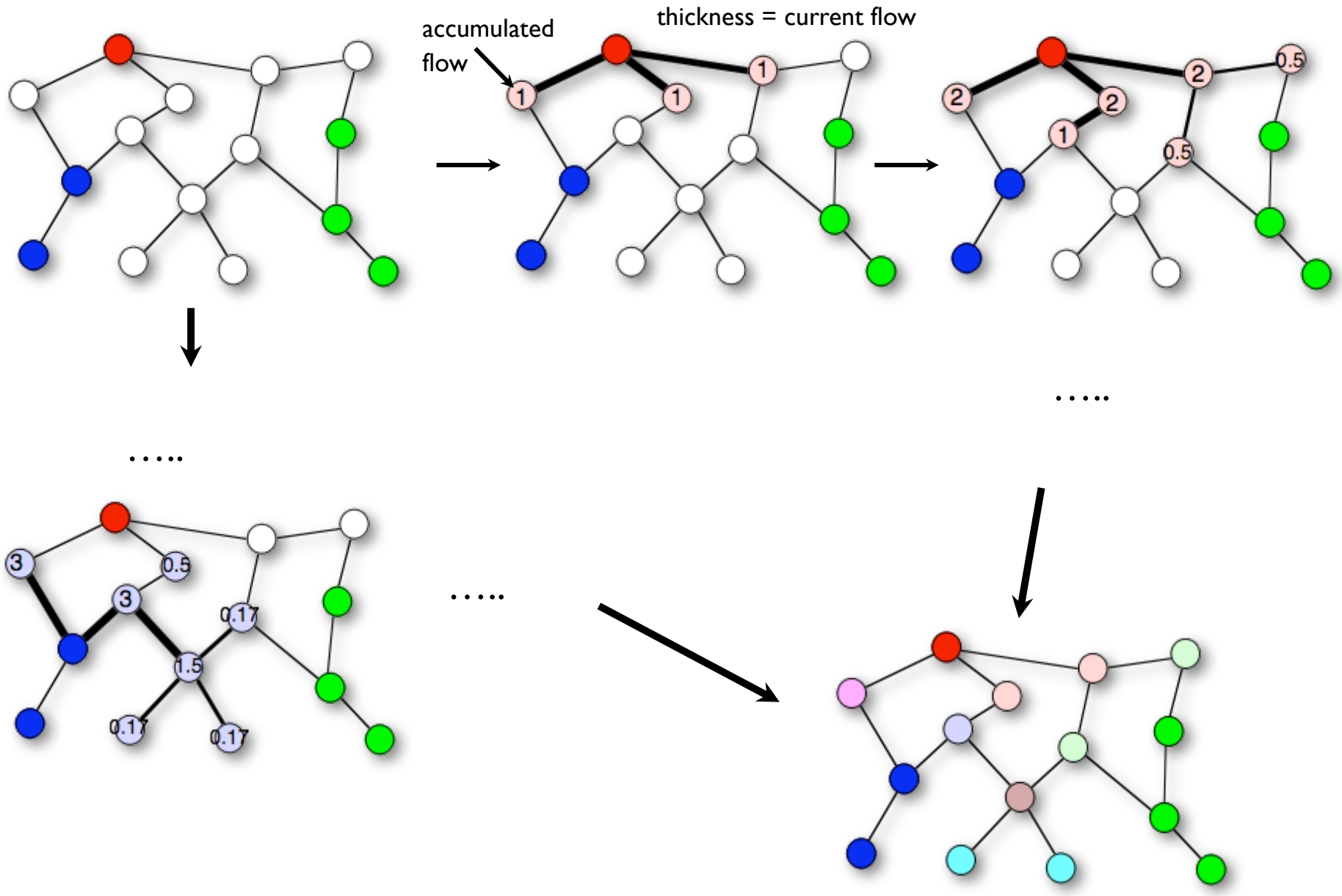
Idea: Node v has already „more function" than node u → no flow uphill

Score from accumulated in-flow:

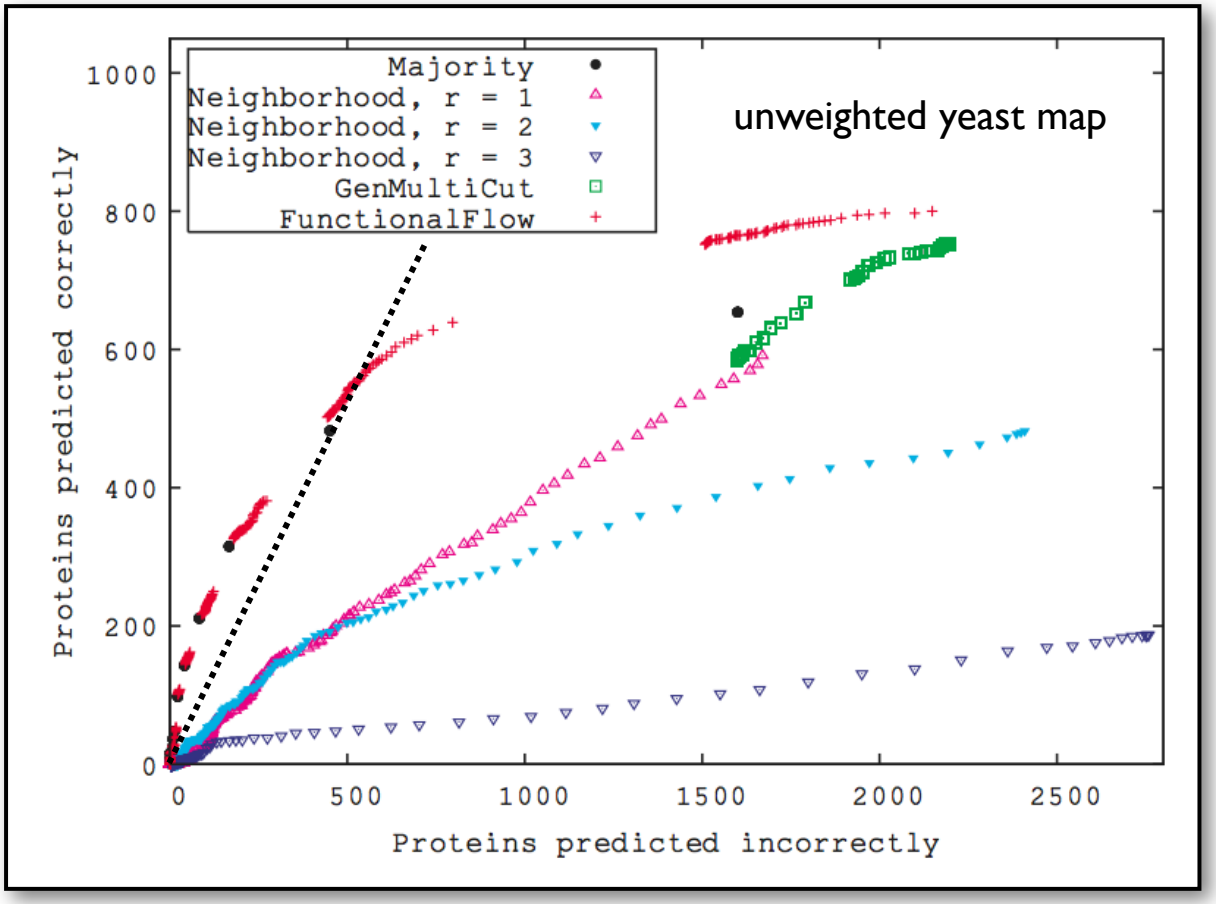
$$f_a(u) = \sum_{t=1}^{d} \sum_{v:(u,v)\in E} g_t^a(v,u)$$

Nabieva et al, Bioinformatics 21 (2005) i302

An Example


accumulated
flow
thickness = current flow
1
1
1
2
2
2
1
0.5
0.5
3
3
0.5
0.17
1.5
0.17
0.17

Comparison



For FunctionalFlow:
six propagation steps were simulated; this is comparable to the diameter of the yeast network ≈ 12

Majority results are initially very good, but has limited coverage.

Results with neighborhood get more imprecise for larger radii r

Change **score threshold** for accepting annotations → ratio **TP/FP**
→ **FunctionalFlow** performs **best** in the high-confidence region
→ but generates still many false predictions!!!

Nabieva et al, Bioinformatics 21 (2005) i302