Bioinformatics 3

V 5 – Robustness and Modularity

Fri, Nov 11, 2016

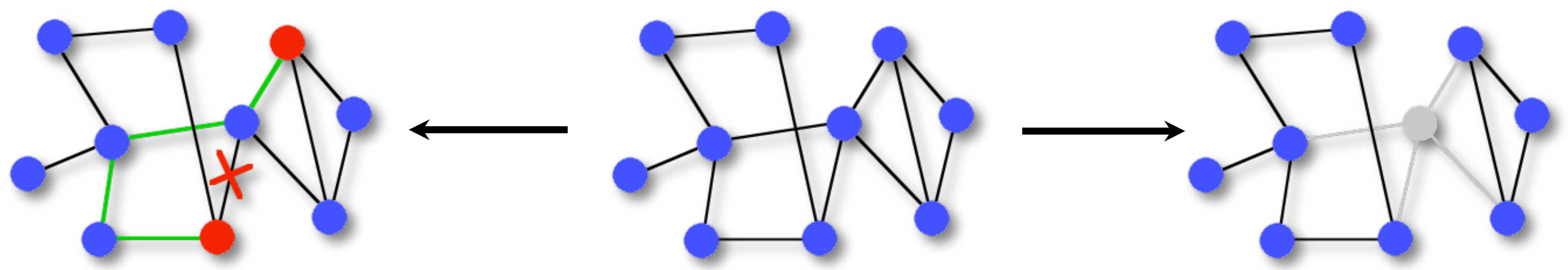
Network Robustness

Network = set of connections

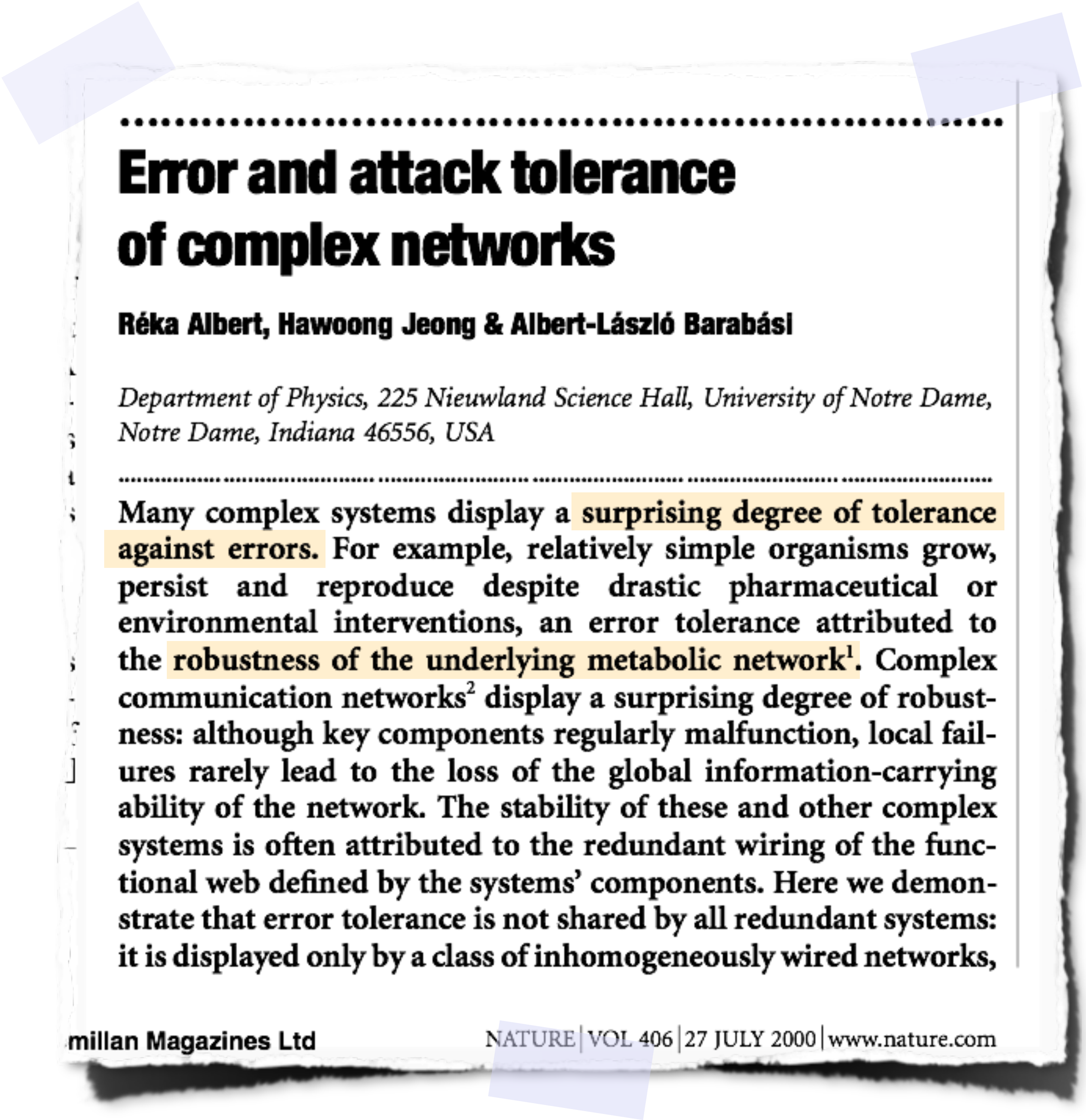
Failure events:
- loss of edges
- loss of nodes (together with their edges)

→ loss of connectivity
- paths become longer (detours required)
- connected components break apart

→ network characteristics change



→ **Robustness** = how much does the network (not) change when edges/nodes are removed



Error and attack tolerance of complex networks

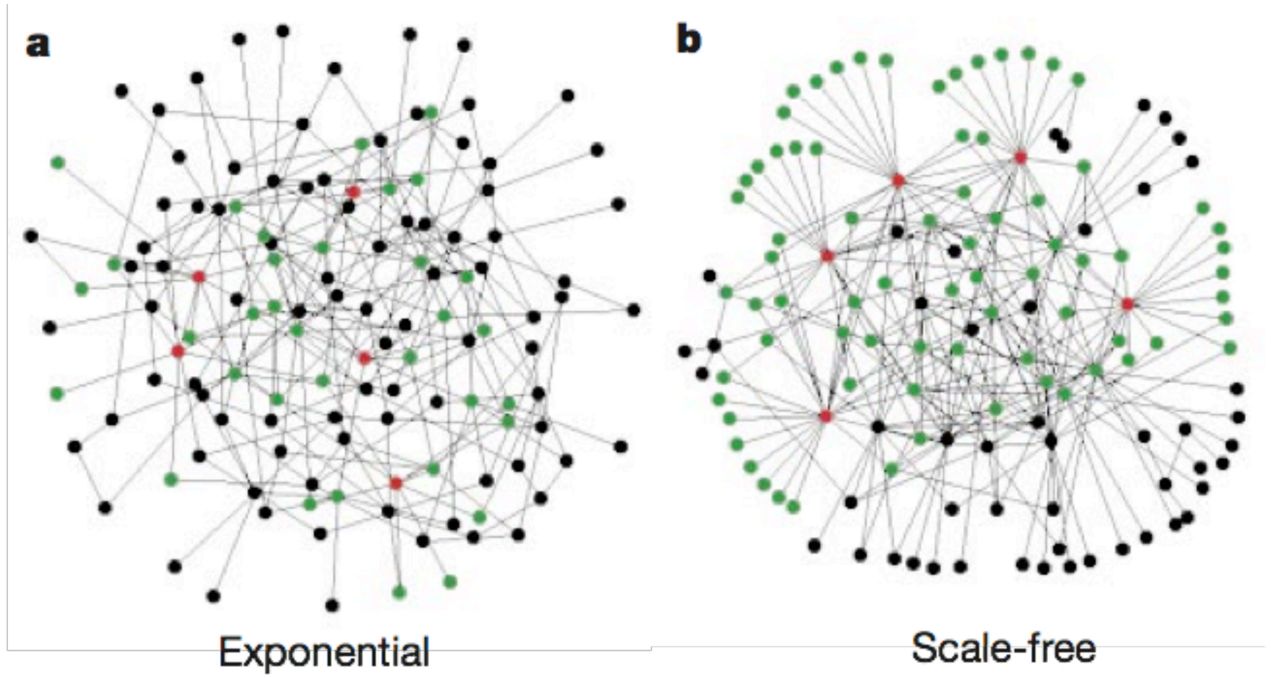
Réka Albert, Hawoong Jeong & Albert-László Barabási

Department of Physics, 225 Nieuwland Science Hall, University of Notre Dame, Notre Dame, Indiana 46556, USA

Many complex systems display a surprising degree of tolerance against errors. For example, relatively simple organisms grow, persist and reproduce despite drastic pharmaceutical or environmental interventions, an error tolerance attributed to the robustness of the underlying metabolic network[1]. Complex communication networks[2] display a surprising degree of robustness: although key components regularly malfunction, local failures rarely lead to the loss of the global information-carrying ability of the network. The stability of these and other complex systems is often attributed to the redundant wiring of the functional web defined by the systems' components. Here we demonstrate that error tolerance is not shared by all redundant systems: it is displayed only by a class of inhomogeneously wired networks,

millan Magazines Ltd NATURE | VOL 406 | 27 JULY 2000 | www.nature.com

Random vs. Scale-Free



130 nodes, 215 edges

The **top 5** nodes with the highest *k* **connect** to…

… 27% of the network

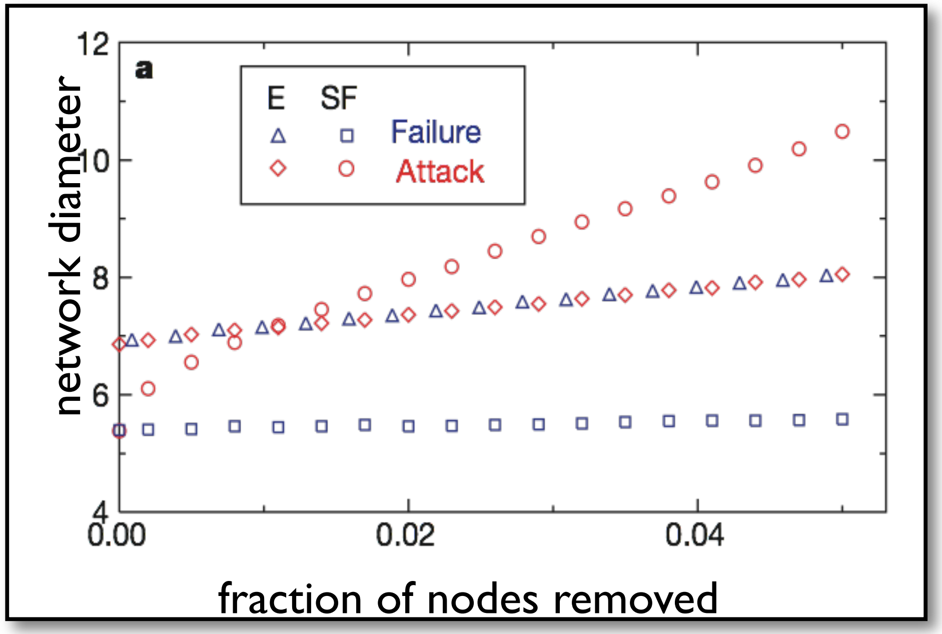
… 60% of the network

Albert, Jeong, Barabási, *Nature* **406** (2000) 378

Failure vs. Attack

Failure: remove **randomly** selected nodes

Attack: remove nodes with highest **degrees**



SF: scale-free network -> attack

E: exponential (random) network
-> failure / attack

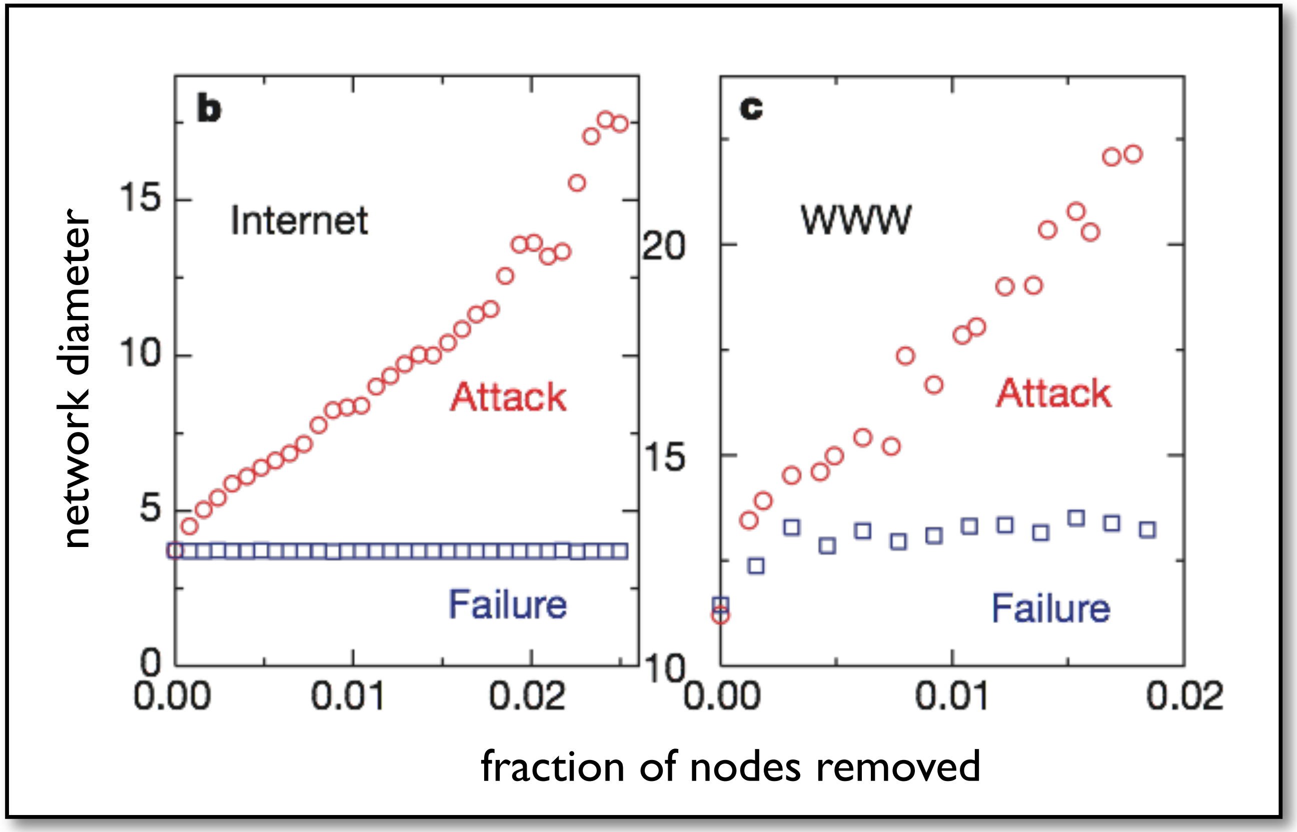
SF: failure

N = 10000, L = 20000, but effect is size-independent;

Interpretation:
SF network diameter increases strongly when network is attacked
but not when nodes fail randomly

Two real-world networks

Scale-free:
- very **stable** against random **failure** ("packet re-rooting")
- very **vulnerable** against dedicated **attacks** ("9/11")



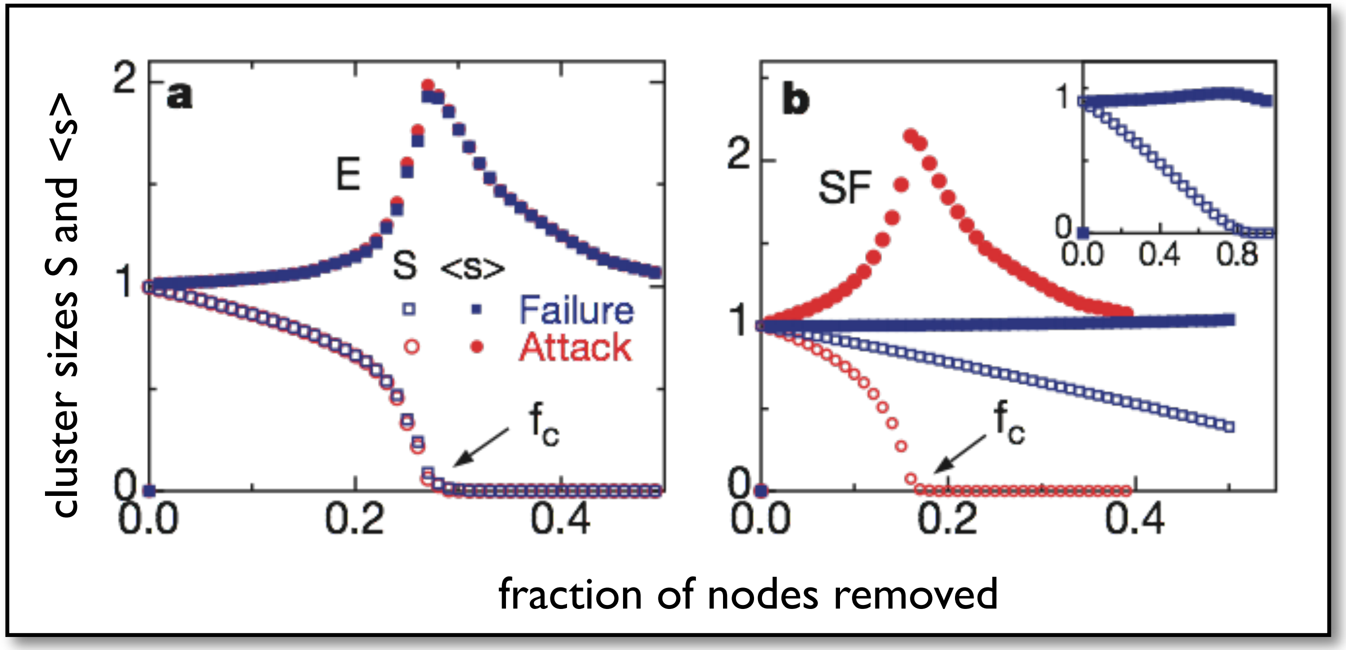
http://moat.nlanr.net/Routing/rawdata/ : 6209 nodes and 12200 links (2000)

WWW-sample containing 325729 nodes and 1498353 links

Network Fragmentation

<s>: average size of the isolated clusters (except the largest one)

S: relative size of the largest cluster S; this is defined as the fraction of nodes contained in the largest cluster (that is, *S = 1 for f = 0)*



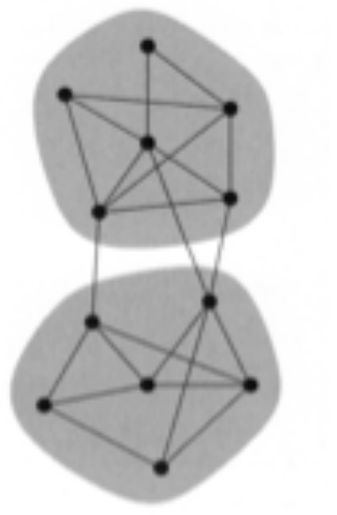
Random network: • **no difference** between attack and failure (homogeneity)
• fragmentation threshold at $f_c \gtrsim 0.28$ $(S \approx 0)$

Scale-free network: • **delayed fragmentation** and isolated nodes for failure
• critical breakdown under attack at $f_c \approx 0.18$

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.

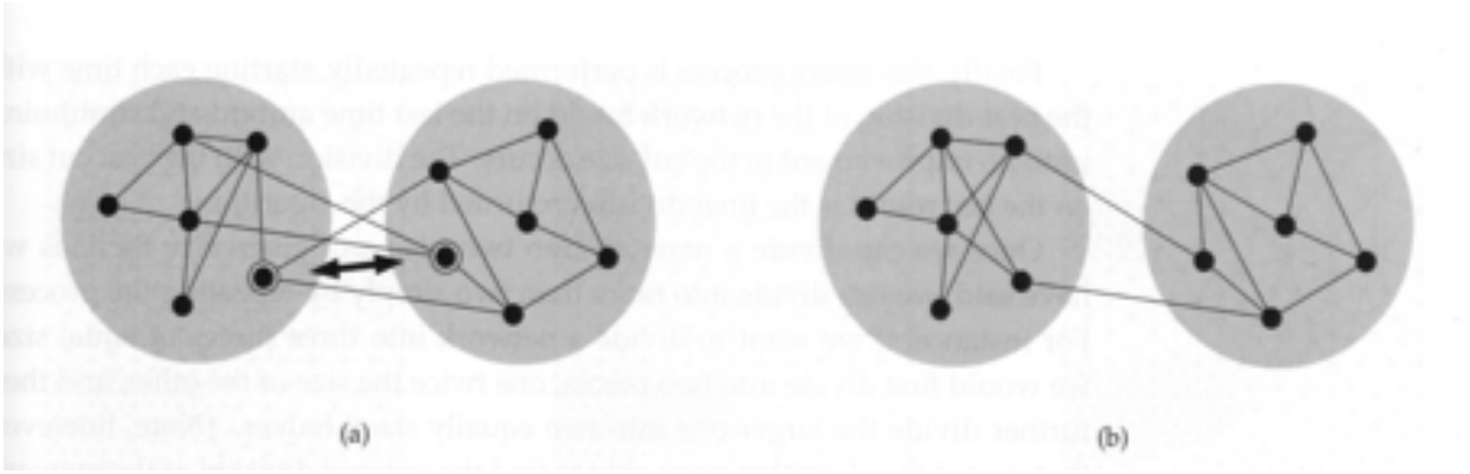
The amount of time to look through all these divisions will go up roughly exponentially with the size of the system.

Only values of up to $n = 30$ are feasible with current computers.

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

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.

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)

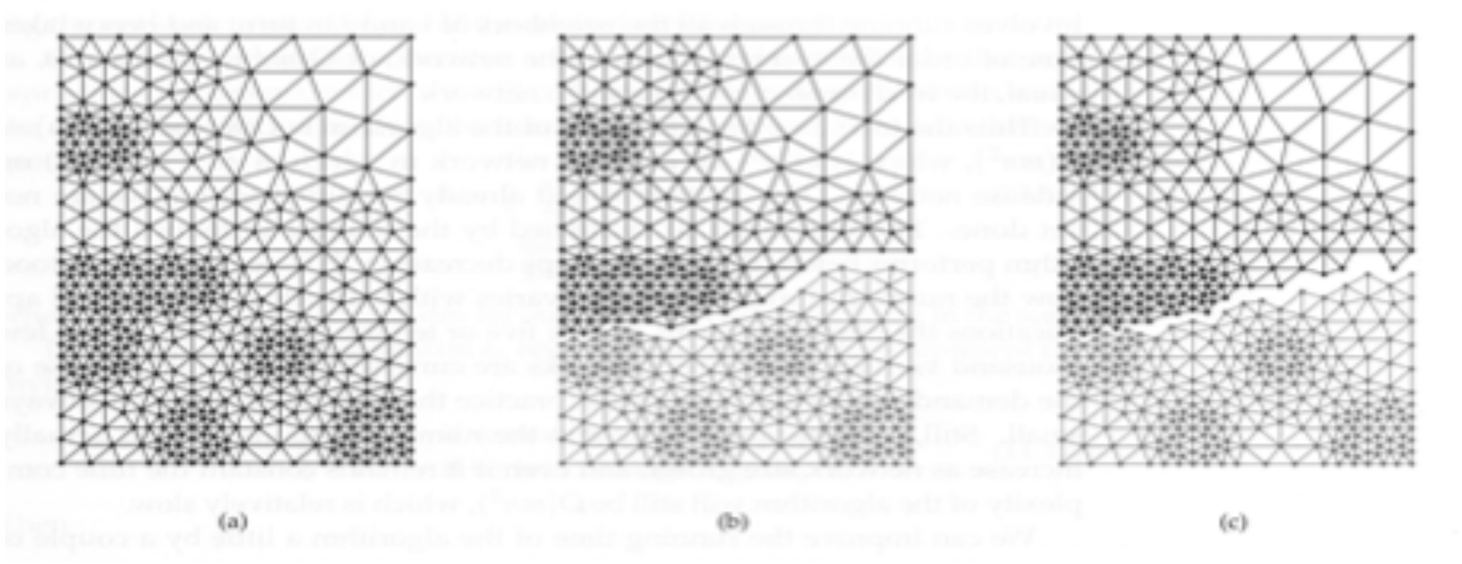
(3) Go back through every state that the network passed through during the swapping procedure and choose among them the state in which the cut size takes its smallest value.

(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 group 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.

In the worst case, 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 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.

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

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

On a dense network (with $m \rightarrow \frac{n(n-1)}{2}$) , 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.

Mesoscale properties of networks - identify cliques and highly connected clusters

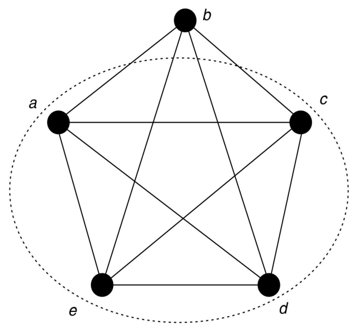
Most relevant processes in biological networks correspond to the mesoscale (5-25 genes or proteins) not to the entire network.

However, it is computationally enormously expensive to study mesoscale properties of biological networks.
e.g. a network of 1000 nodes contains 1×10^{23} possible 10-node sets.

Spirin & Mirny analyzed combined network of protein interactions in *S. cereviseae* with data from CELLZOME, MIPS, BIND: 6500 interactions.

Identify connected subgraphs

The network of protein interactions is typically presented as an undirected graph with proteins as nodes and protein interactions as undirected edges.



First aim: identify **fully connected subgraphs** (cliques)
A clique is a set of nodes that are all neighbors of each other.

The „maximum clique problem“ – finding the largest clique in a given graph is known be NP-hard.

In this example, the whole graph is a clique and consequently any subset of it is also a clique, for example *{a,c,d,e}* or *{b,e}*.

A **maximal clique** is a clique that is not contained in any larger clique. Here only *{a,b,c,d,e}* is a maximal clique.

In general, protein complexes need not to be fully connected.

Spirin, Mirny,
PNAS 100, 12123 (2003)

Identify all fully connected subgraphs (cliques)

Although the general problem - finding all cliques of a graph - is very hard, this can be done relatively quickly for the given network because the protein interaction graph is quite sparse (the number of interactions (edges) is similar to the number of proteins (nodes).

To find cliques of size *n* one needs to enumerate only the cliques of size *n-1*.

The search for cliques starts with *n* = 4, pick all (known) pairs of edges (6500 × 6500 protein interactions) successively.
For every pair *A-B* and *C-D* check whether there are edges between *A* and *C*, *A* and *D*, *B* and *C*, and *B* and *D*. If these edges are present, *ABCD* is a clique.

For every clique identified, *ABCD*, pick all known proteins successively.
For every picked protein *E*, if all of the interactions *E-A*, *E-B*, *E-C*, and *E-D* exist, then *ABCDE* is a clique with size 5.

Continue for *n* = *6, 7, ...*
The largest clique found in the protein-interaction network has size 14.

Spirin, Mirny, PNAS 100, 12123 (2003)

Identify all fully connected subgraphs (cliques)

These results include, however, many redundant cliques.
For example, the clique with size 14 contains 14 cliques with size 13.

To find all nonredundant subgraphs, mark all proteins comprising the clique of size 14, and out of all subgraphs of size 13 pick those that have at least one protein other than marked.

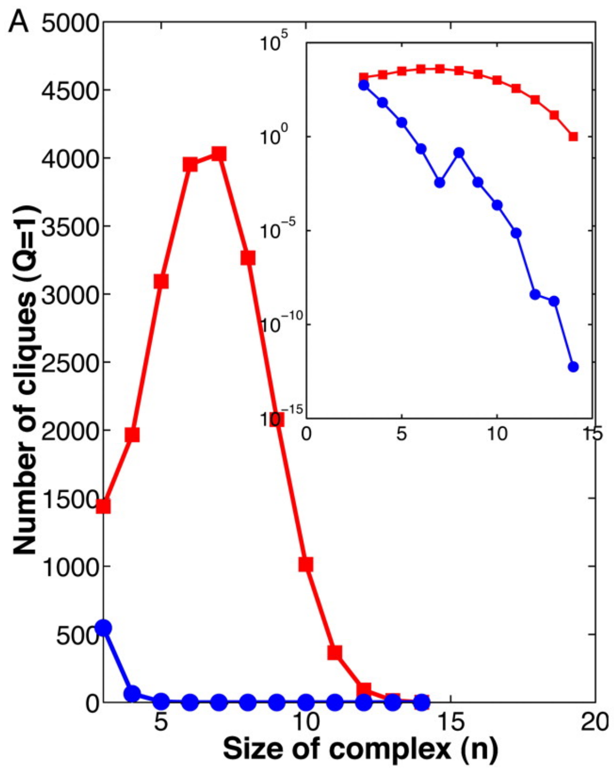
After all redundant cliques of size 13 are removed, proceed to remove redundant twelves etc.

In total, only 41 nonredundant cliques with sizes 4 - 14 were found by Spirin & Mirny.

Spirin, Mirny, PNAS 100, 12123 (2003)

Statistical significance of cliques

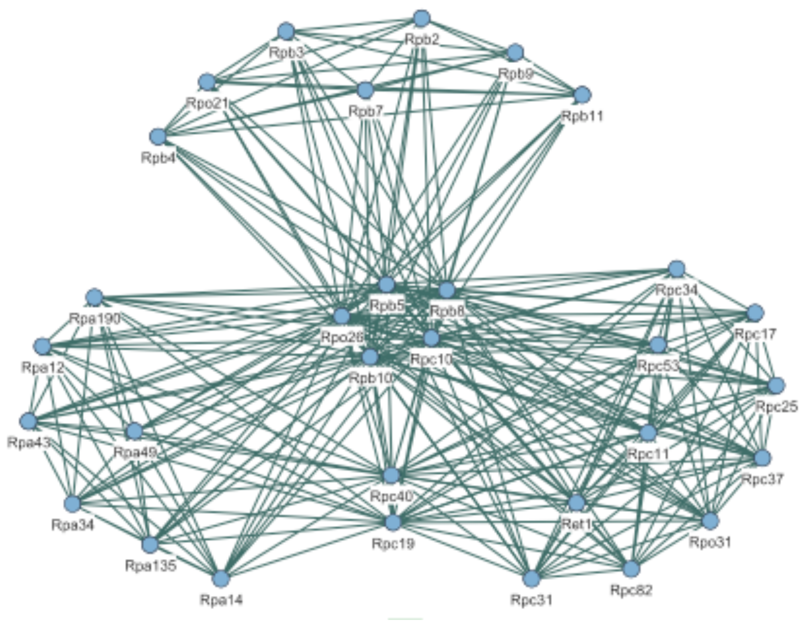
Number of complete cliques as a function of clique size enumerated in the network of protein interactions (red) and in randomly rewired graphs (blue, averaged over >1,000 graphs where the number of interactions for each protein is preserved).

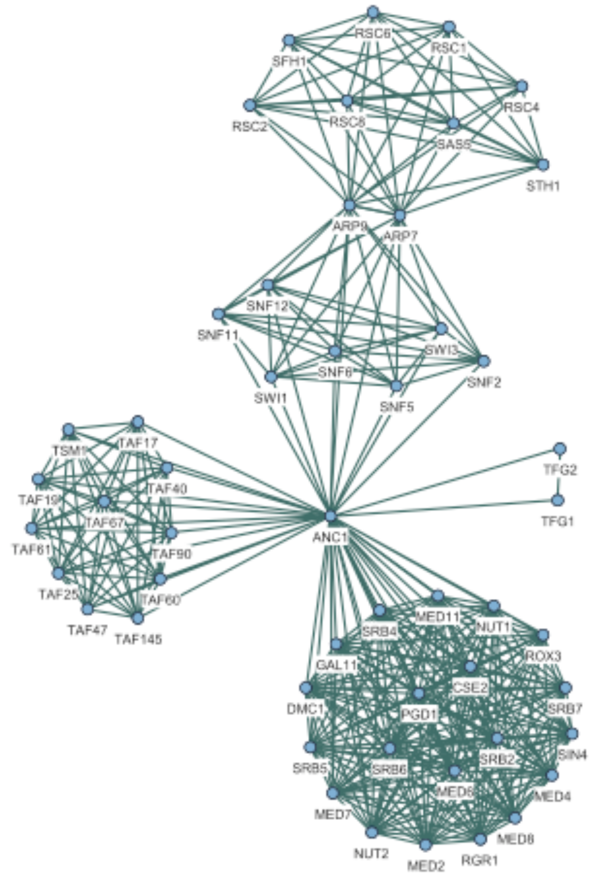


Inset shows the same plot on a log-normal scale. Note the dramatic enrichment in the number of cliques in the protein-interaction graph compared with the random graphs. Most of these cliques are parts of bigger complexes and modules.

Spirin, Mirny, PNAS 100, 12123 (2003)

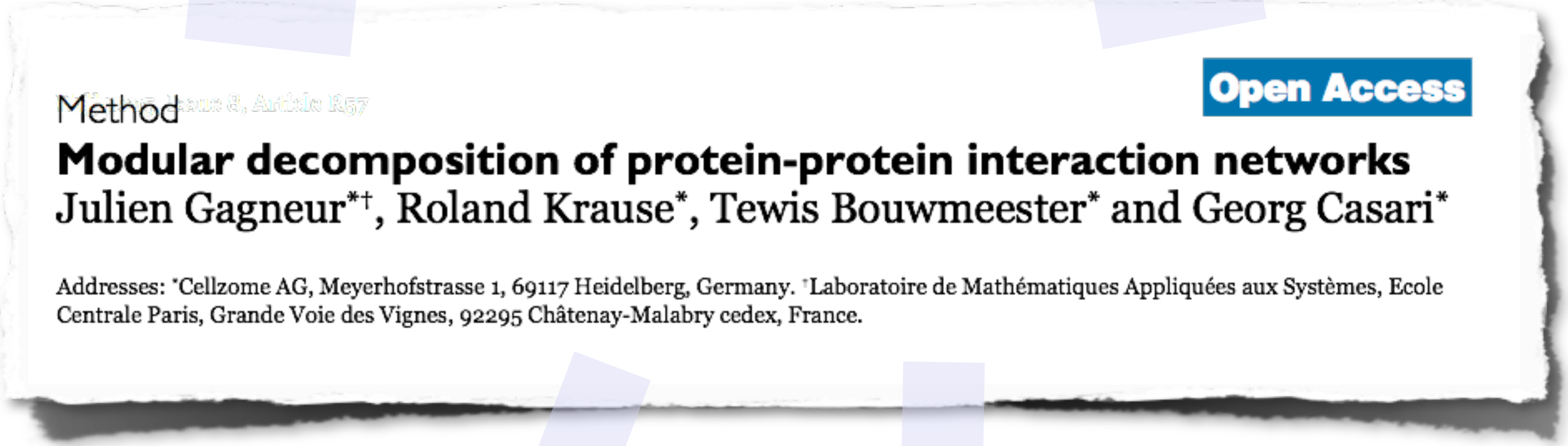
Reducing Network Complexity?





Is there a **representation** that highlights the **structure** of these networks???

- Modular Decomposition (Gagneur, …, Casari, 2004)
- Network Compression (Royer, …, Schröder, 2008)



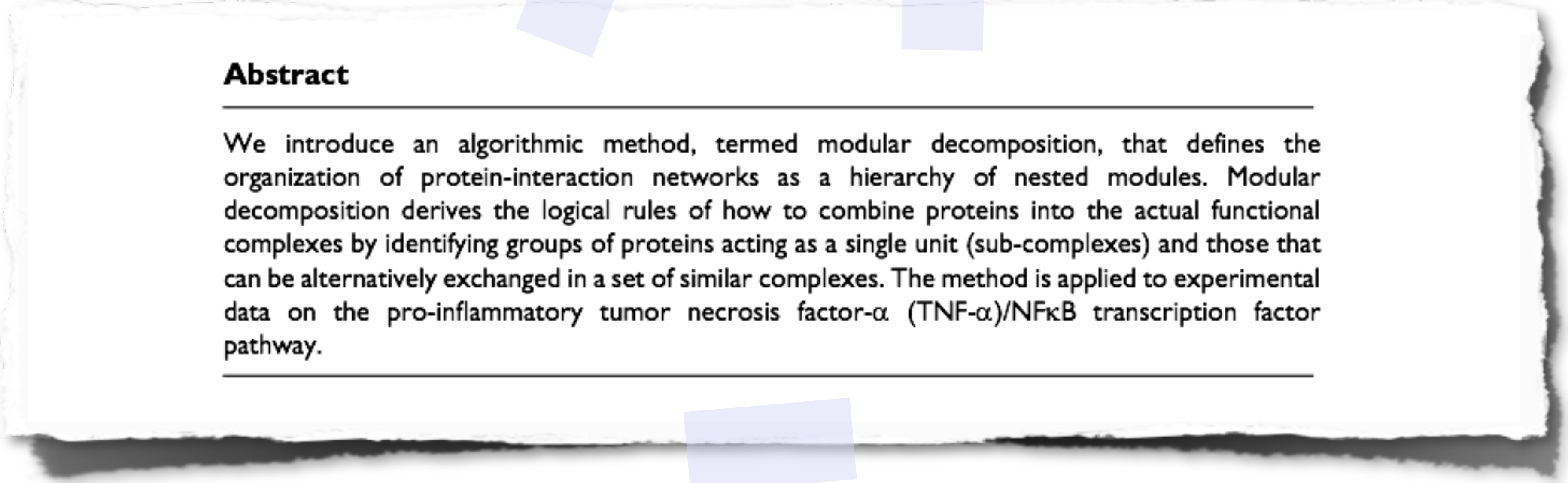
Method [illegible] Issue 8, Article R57

Open Access

Modular decomposition of protein-protein interaction networks

Julien Gagneur[*†], Roland Krause[*], Tewis Bouwmeester[*] and Georg Casari[*]

Addresses: [*]Cellzome AG, Meyerhofstrasse 1, 69117 Heidelberg, Germany. [†]Laboratoire de Mathématiques Appliquées aux Systèmes, Ecole Centrale Paris, Grande Voie des Vignes, 92295 Châtenay-Malabry cedex, France.



Abstract

We introduce an algorithmic method, termed modular decomposition, that defines the organization of protein-interaction networks as a hierarchy of nested modules. Modular decomposition derives the logical rules of how to combine proteins into the actual functional complexes by identifying groups of proteins acting as a single unit (sub-complexes) and those that can be alternatively exchanged in a set of similar complexes. The method is applied to experimental data on the pro-inflammatory tumor necrosis factor-α (TNF-α)/NFκB transcription factor pathway.

Genome Biology **5** (2004) R57

Shared Components

Shared components = proteins or groups of proteins occurring in different complexes are fairly common. A shared component may be a small part of many complexes, acting as a **unit** that is constantly **reused** for its function.

Also, it may be the **main part** of the complex e.g. in a family of variant complexes that differ from each other by distinct proteins that provide functional specificity.

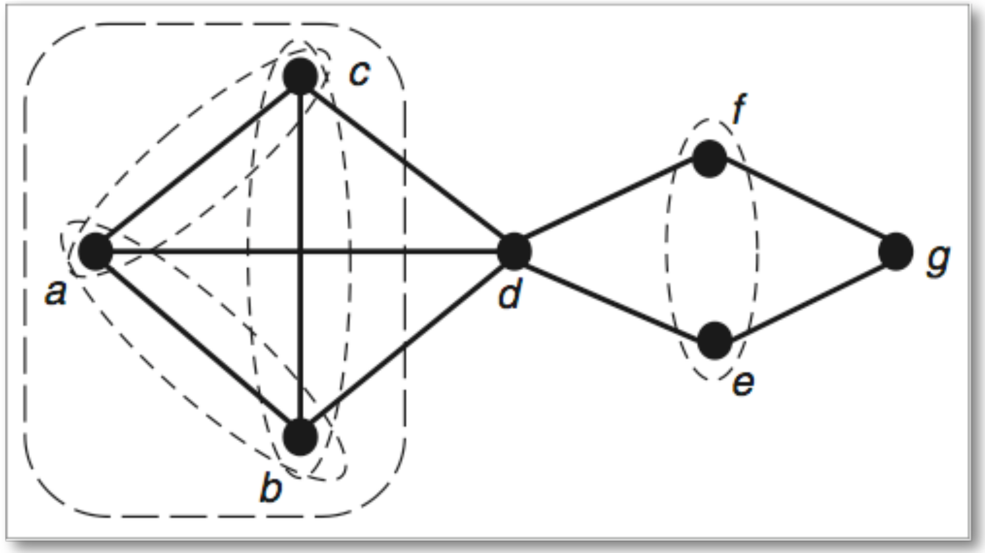
Aim: **identify** and properly **represent** the modularity of protein-protein interaction networks by identifying the **shared components** and the way they are arranged to generate **complexes**.

Georg Casari, Cellzome (Heidelberg)

Gagneur et al. Genome Biology 5, R57 (2004)

Modular Decomposition of a Graph

Module := set of **nodes** that have the **same neighbors** outside of the module



trivial modules:

{a}, {b}, …, {g}
{a, b, …, g}

non-trivial modules:

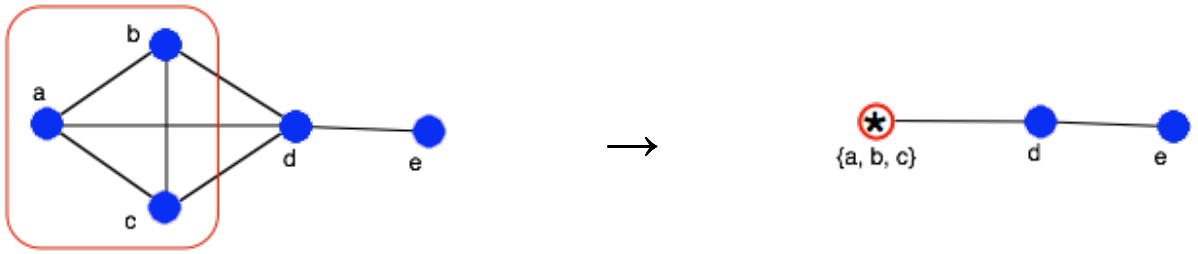
{a, b}, {a, c}, {b, c}
{a, b, c}
{e, f}

Quotient: representative node for a module

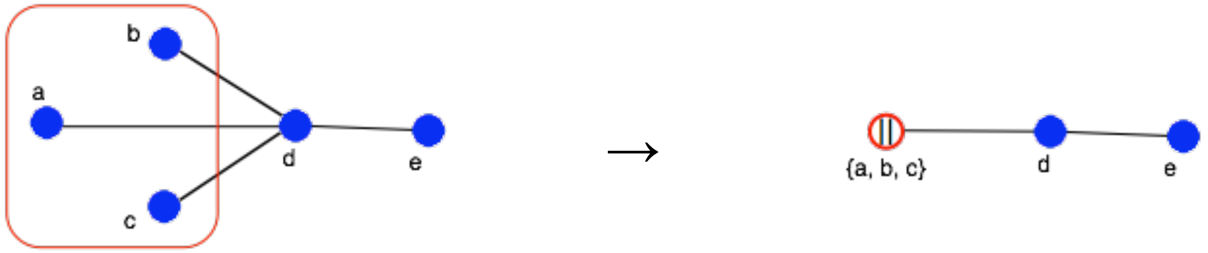
Iterated quotients → labeled tree representing the original network
→ "**modular decomposition**"

Quotients

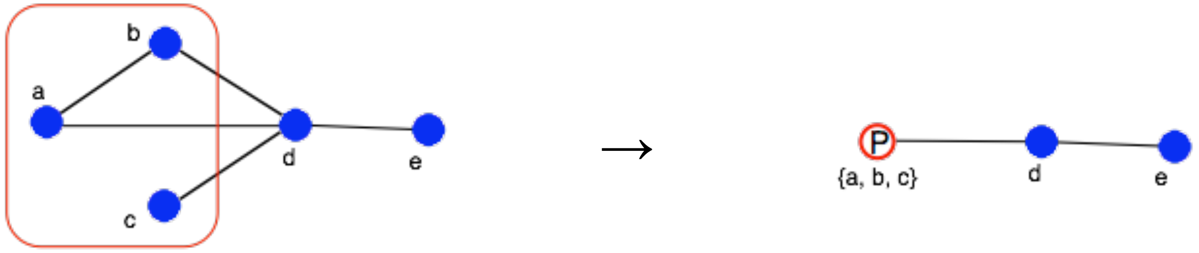
Series: all included nodes are direct **neighbors** (= **clique**)



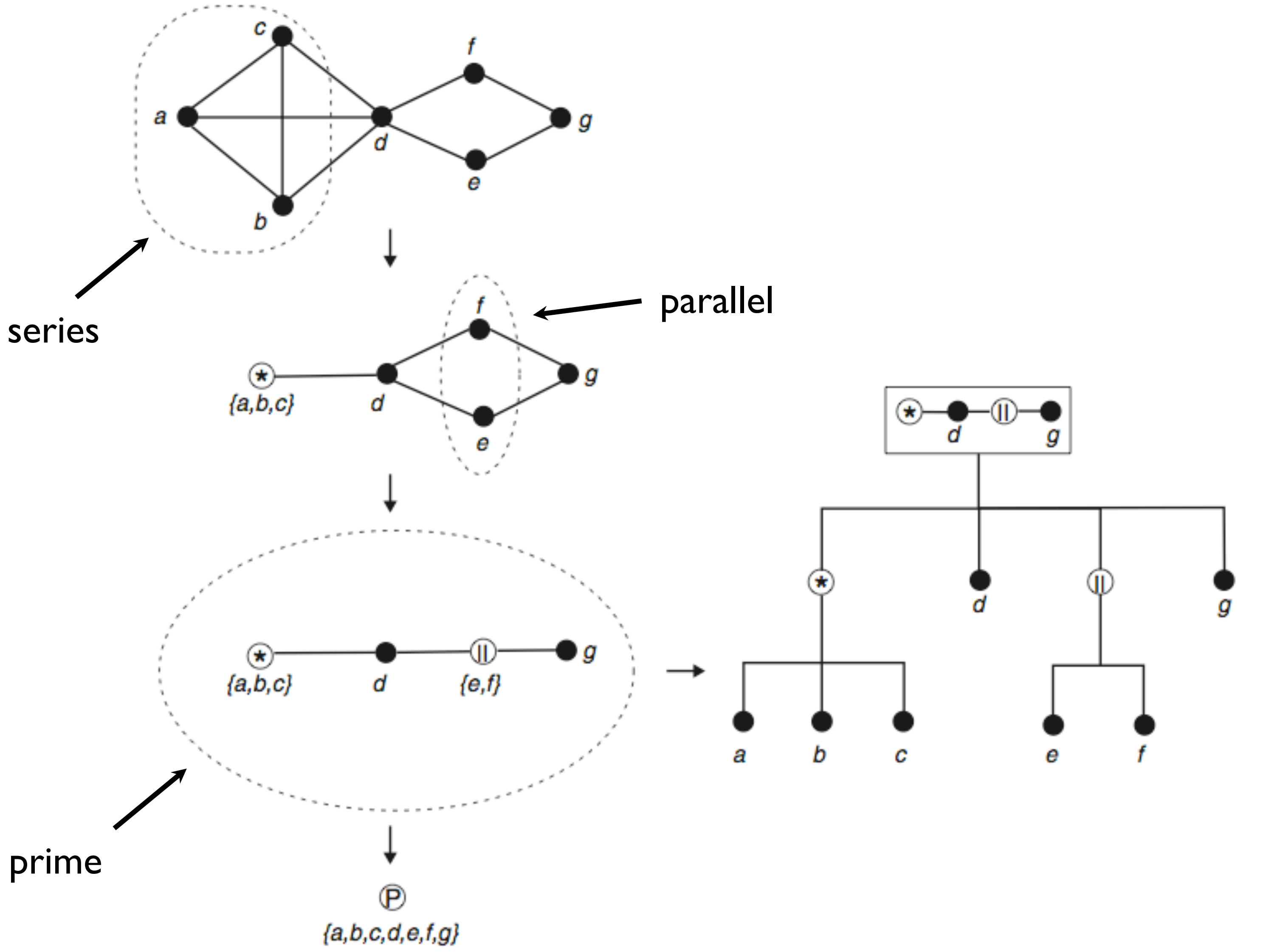
Parallel: all included nodes are **non-neighbors**



Prime: "anything else" (best labeled with the actual structure)



A Simple Recursive Example



Gagneur et al, *Genome Biology* **5** (2004) R57

Using data from protein complex purifications e.g. by TAP

Different types of data:

• Y2H: detects direct physical interactions between proteins

• PCP by tandem affinity purification with mass-spectrometric identification of the protein components identifies multi-protein complexes

→ Molecular decomposition will have a **different meaning** due to different **semantics** of such graphs.

Here, we focus analysis on **PCP content** from TAP-MS data.

PCP experiment: select bait protein where TAP-label is attached → Co-purify protein with those proteins that co-occur in at least one complex with the bait protein.

Gagneur et al. Genome Biology 5, R57 (2004)

Data from Protein Complex Purification

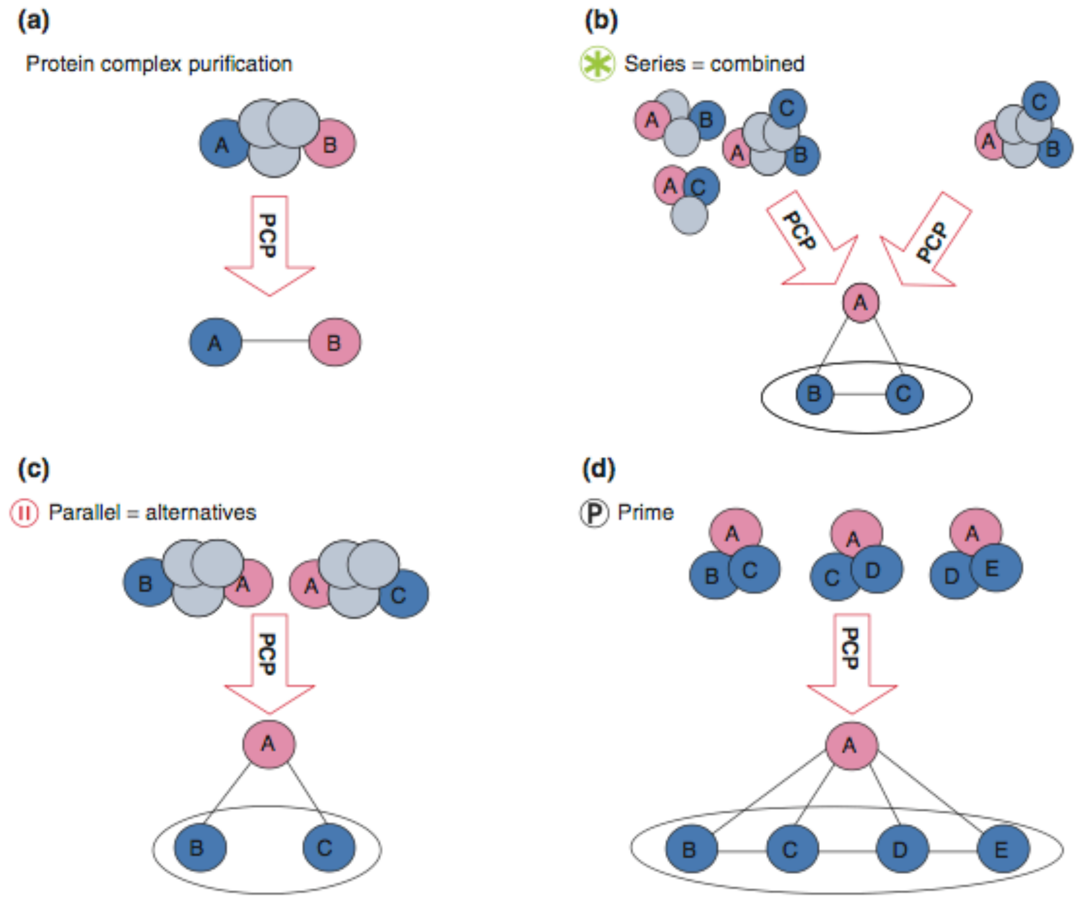
Graphs and module labels from systematic PCP experiments:

(a) Two neighbors in the network are proteins occurring in a same complex.

(b) Several potential sets of complexes can be the origin of the same observed network. Restricting interpretation to the simplest model (top right), the **series** module reads as a logical AND between its members.

(c) A module labeled ´**parallel**´ corresponds to proteins or modules working as strict alternatives with respect to their common neighbors.

(d) The ´**prime**´ case is a structure where none of the two previous cases occurs.



Gagneur et al. Genome Biology 5, R57 (2004)

Real World Examples

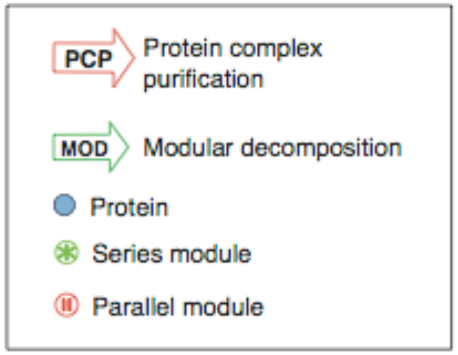
Two examples of modular decompositions of protein-protein interaction networks.

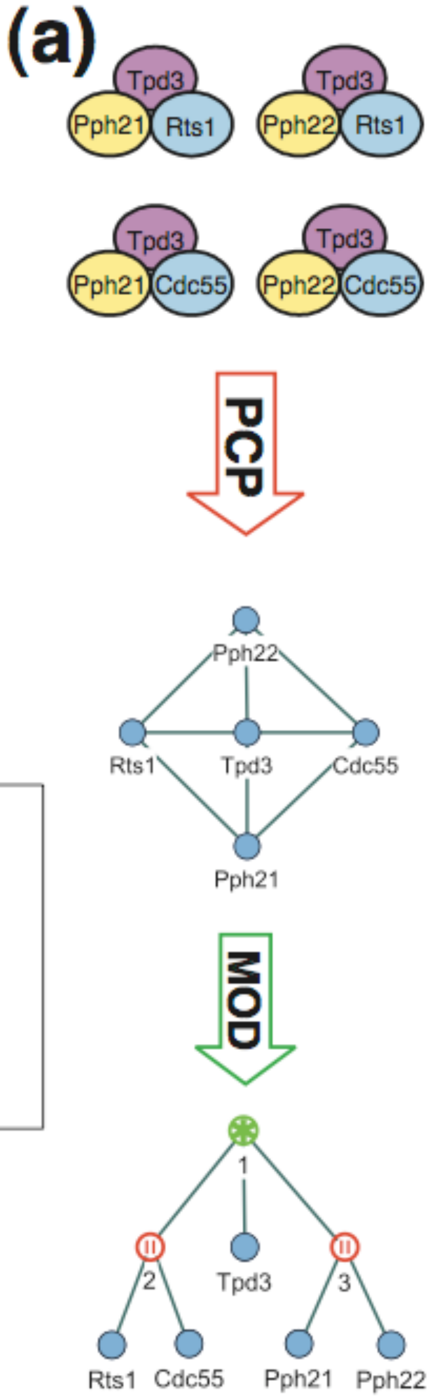
In each case from top to bottom: schemata of the complexes, the corresponding protein-protein interaction network as determined from PCP experiments, and its modular decomposition (MOD).

(a) Protein phosphatase 2A.

Parallel modules group proteins that do not interact but are functionally equivalent.

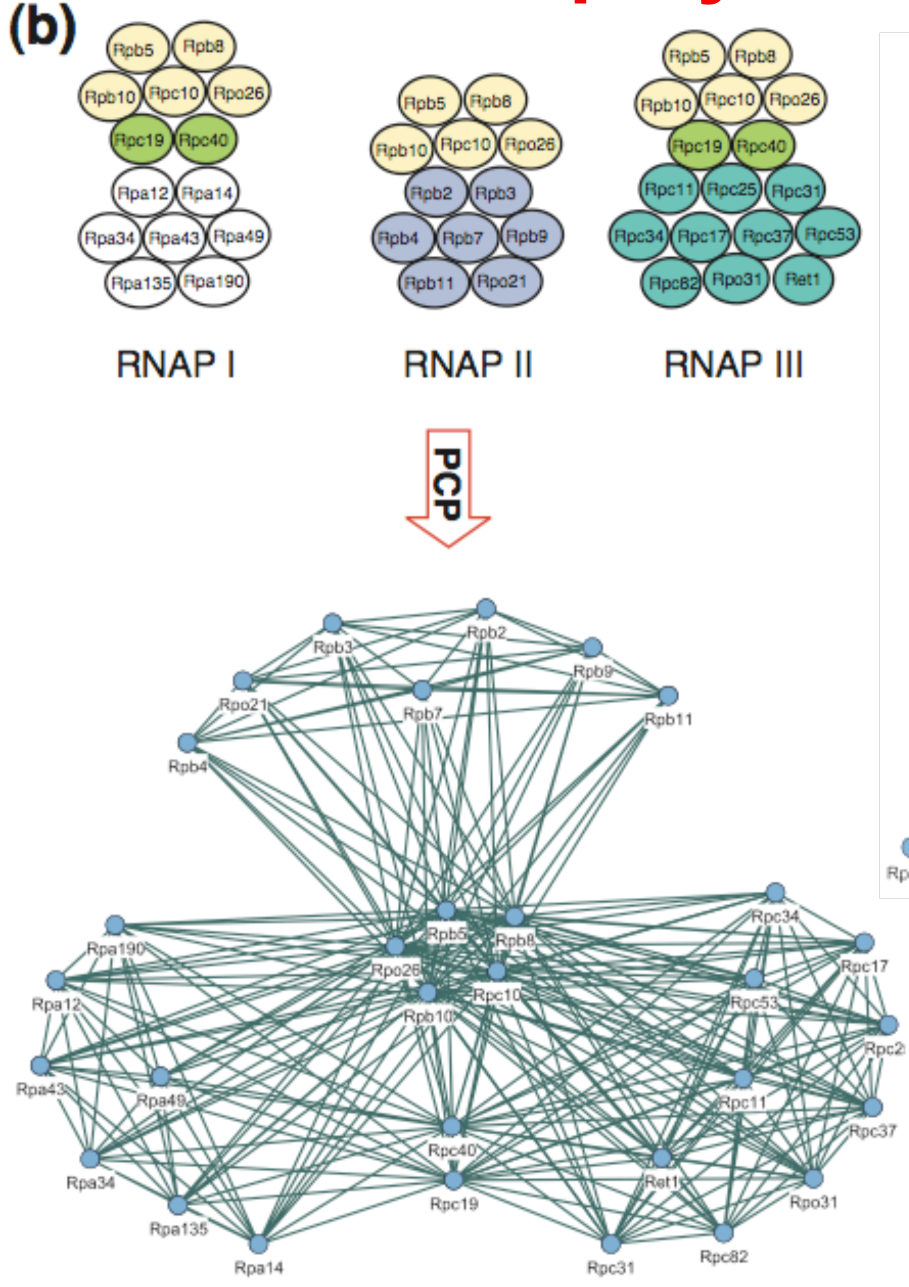
Here these are the catalytic proteins Pph21 and Pph22 (module 2) and the regulatory proteins Cdc55 and Rts1 (module 3), connected by the Tpd3 „backbone“.

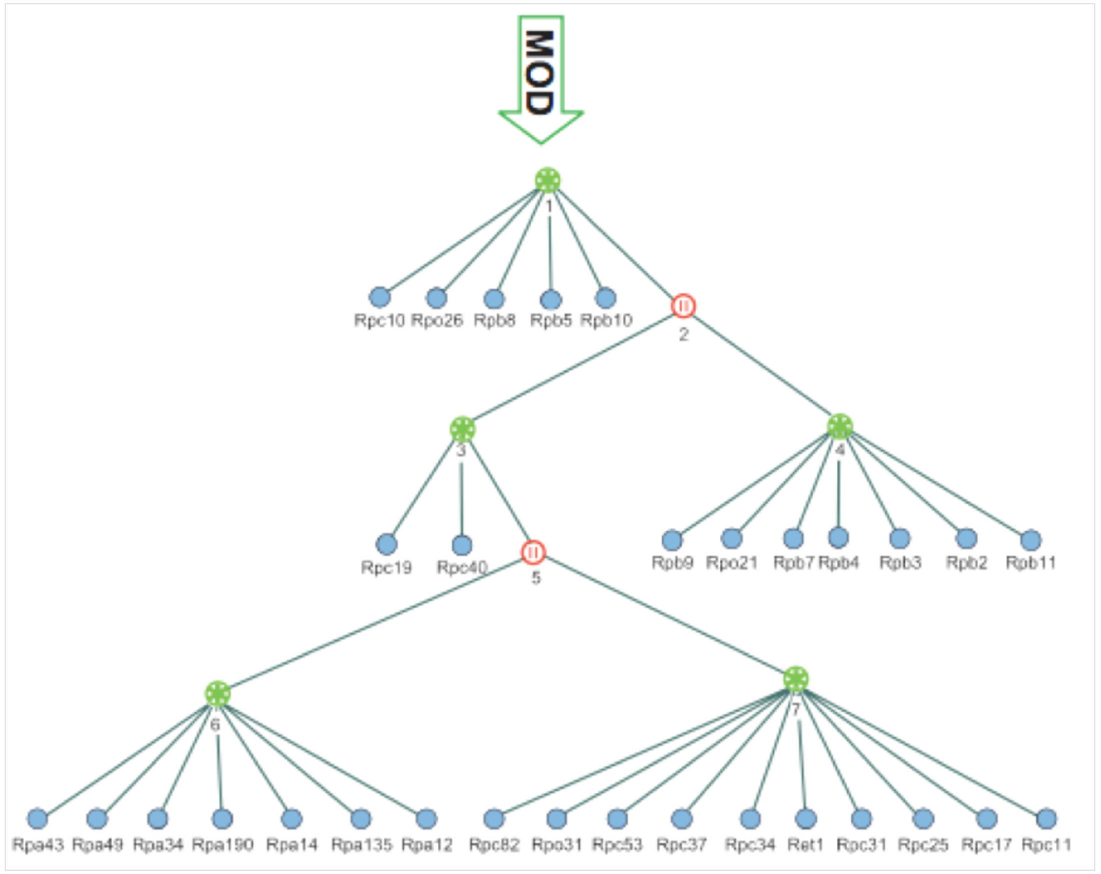




Notes: • Graph does not show functional alternatives!!!
• other decompositions also possible

RNA polymerases I, II and III





MOD
1
Rpc10 Rpo26 Rpb8 Rpb5 Rpb10
2
3
4
Rpc19 Rpc40
5
Rpb9 Rpo21 Rpb7 Rpb4 Rpb3 Rpb2 Rpb11
6
7
Rpa43 Rpa49 Rpa34 Rpa190 Rpa14 Rpa135 Rpa12 Rpc82 Rpo31 Rpc53 Rpc37 Rpc34 Ret1 Rpc31 Rpc25 Rpc17 Rpc11

Again: modular decomposition is much easier to understand than the connectivity graph

Gagneur et al. Genome Biology 5, R57 (2004)

Summary

Modular decomposition of graphs is a **well-defined concept**.

• One can proof thoroughly for which graphs a modular decomposition exists.

• Efficient *O(m + n)* algorithms exist to compute the decomposition.

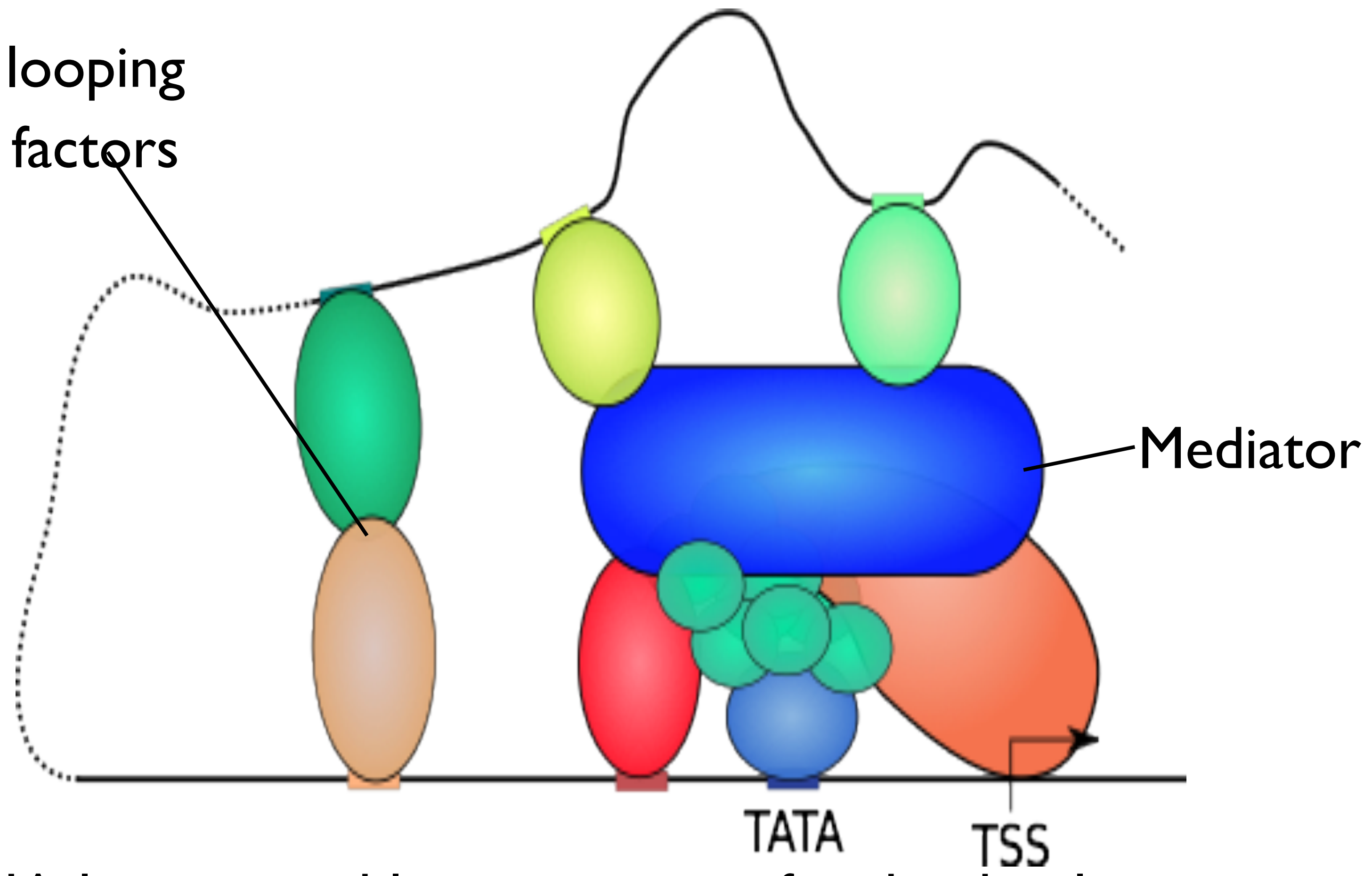
However, experiments have shown that **biological** complexes are **not strictly disjoint**. They often share components

→ separate complexes do not always fulfill the strict requirements of modular graph decomposition.

Also, there exists a „danger“ of false-positive or false-negative interactions.

→ **other methods**, e.g., for detecting communities (Girven & Newman) or densely connected clusters are **more suitable** for identification of **complexes** because they are more sensitive.

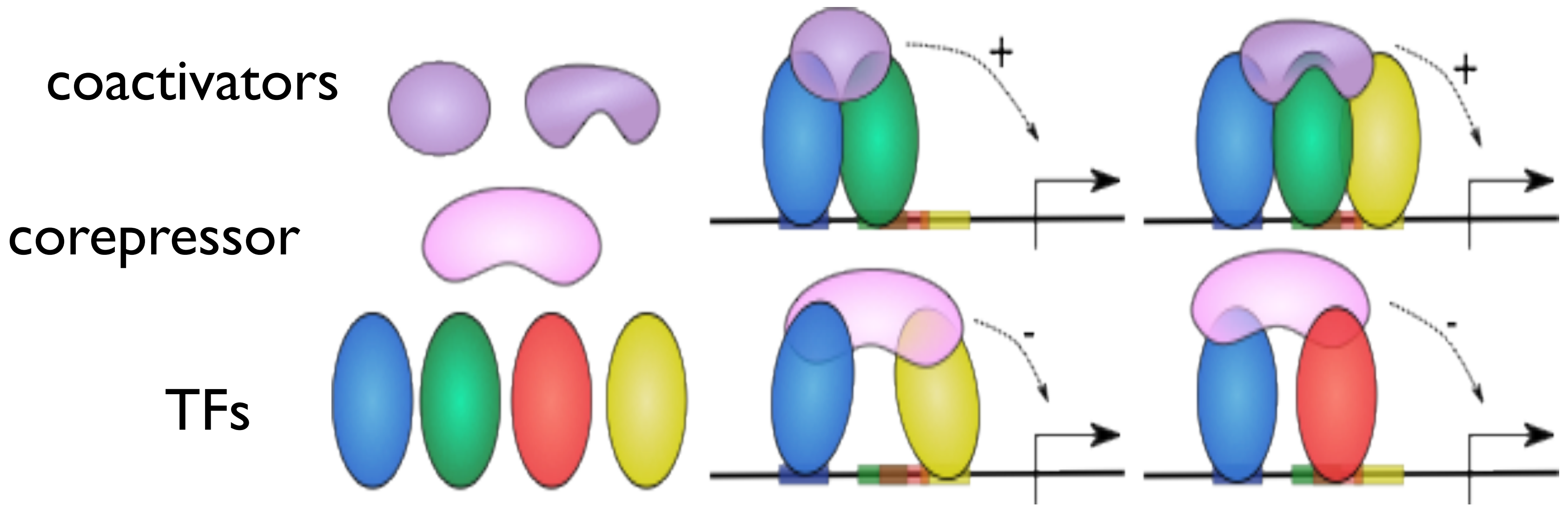
Transcriptional activation



DNA-looping enables interactions for the distal promotor regions,

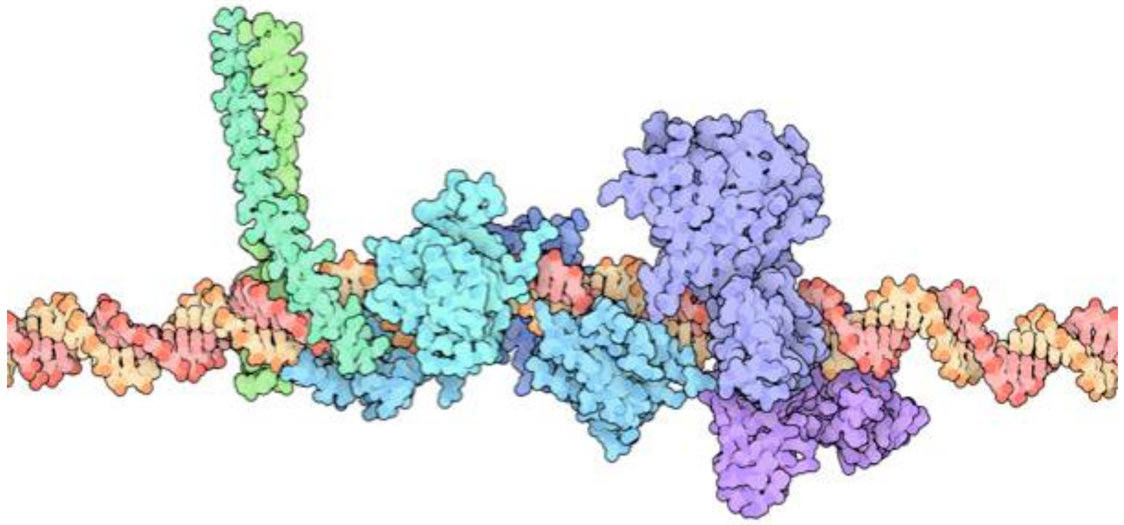
Mediator cofactor-complex serves as a huge linker

cis-regulatory modules



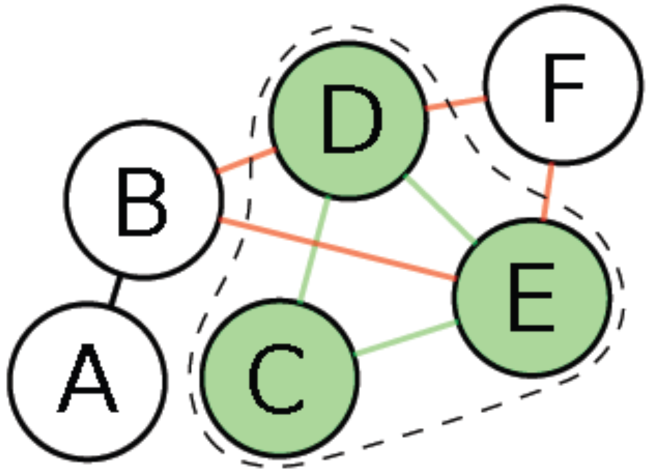
TFs are not dedicated activators or respressors!

It‘s the assembly that is crucial.



IFN-enhanceosome from RCSB Protein Data Bank, 2010

Protein complexes involving multiple transcription factors



Borrow idea from ClusterOne method:

Identify candidates of TF complexes

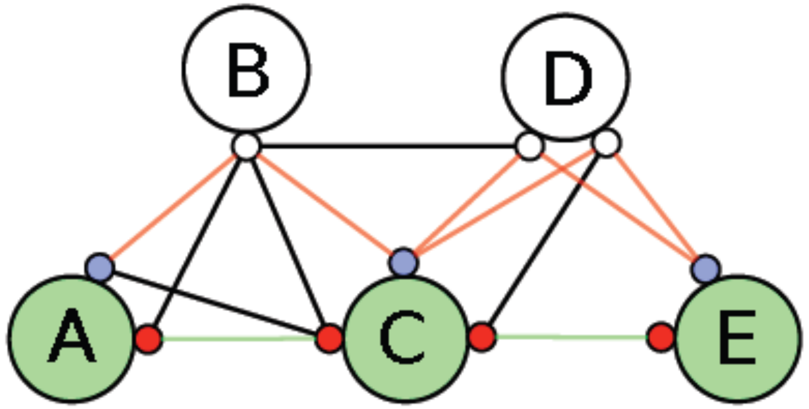
in protein-protein interaction graph

by **optimizing the cohesiveness**

$$f(V) = \frac{w^{in}(V)}{w^{in}(V) + w^{bound}(V)}$$

underlying domain-domain representation of PPIs

Assumption: every domain supports only one interaction.

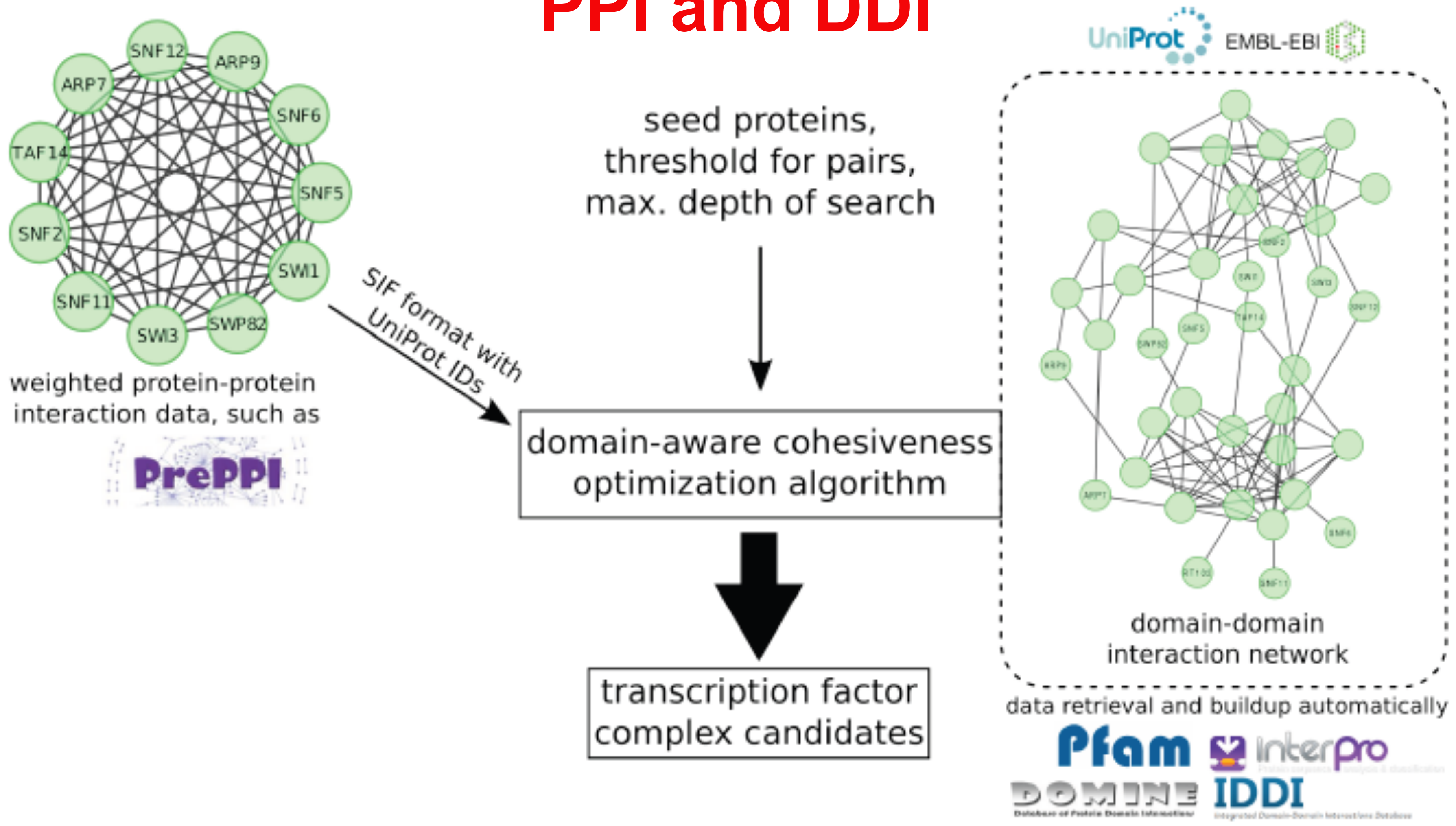


Green proteins A, C, E form actual complex.

Their red domains are connected by the two green edges.

B and D are incident proteins. They could form new interactions (red edges) with unused domains (blue) of A, C, E

data source used: Yeast Promoter Atlas, PPI and DDI

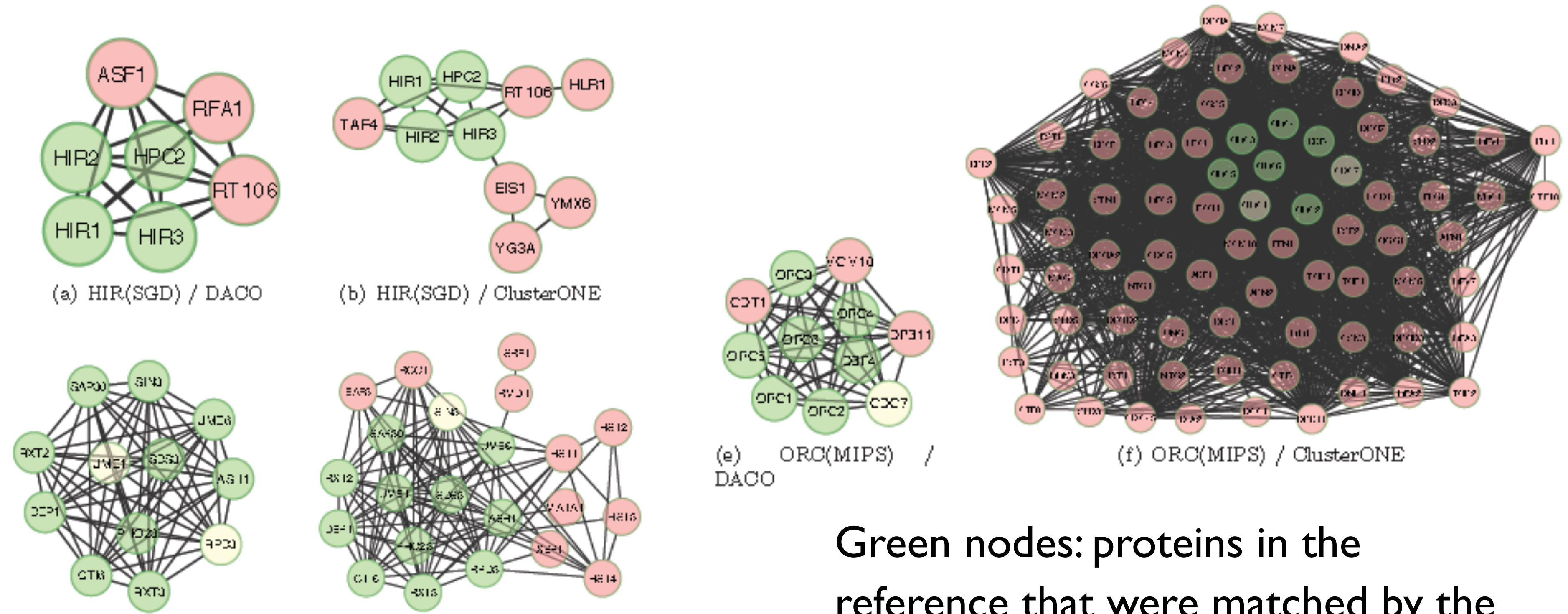

UniProt
EMBL-EBI
seed proteins, threshold for pairs, max. depth of search
SNF12
ARP9
ARP7
SNF6
TAF14
SNF5
SNF2
SWI1
SNF11
SWP82
SWI3
weighted protein-protein interaction data, such as
PrePPI
SIF format with UniProt IDs
domain-aware cohesiveness optimization algorithm
transcription factor complex candidates
domain-domain interaction network
data retrieval and buildup automatically
Pfam
interpro
DOMINE
IDDI

Will, T. and Helms, V. (2014)
Bioinformatics, 30, i415-i421

Daco identifies far more TF complexes than other methods

	DACO	Cl1ps	Cl1s	Cl1	MCD	MCL
TF complexes	1375	175/176	61/63	106/106	16/38	75/79
TF variants	412	134/138	59/61	80/80	16/38	75/79

Examples of TF complexes – comparison with ClusterONE



(a) HIR(SGD) / DACO

(b) HIR(SGD) / ClusterONE

(c) RPD3L(CYC2008) / DACO

(d) RPD3L(CYC2008) / ClusterONE

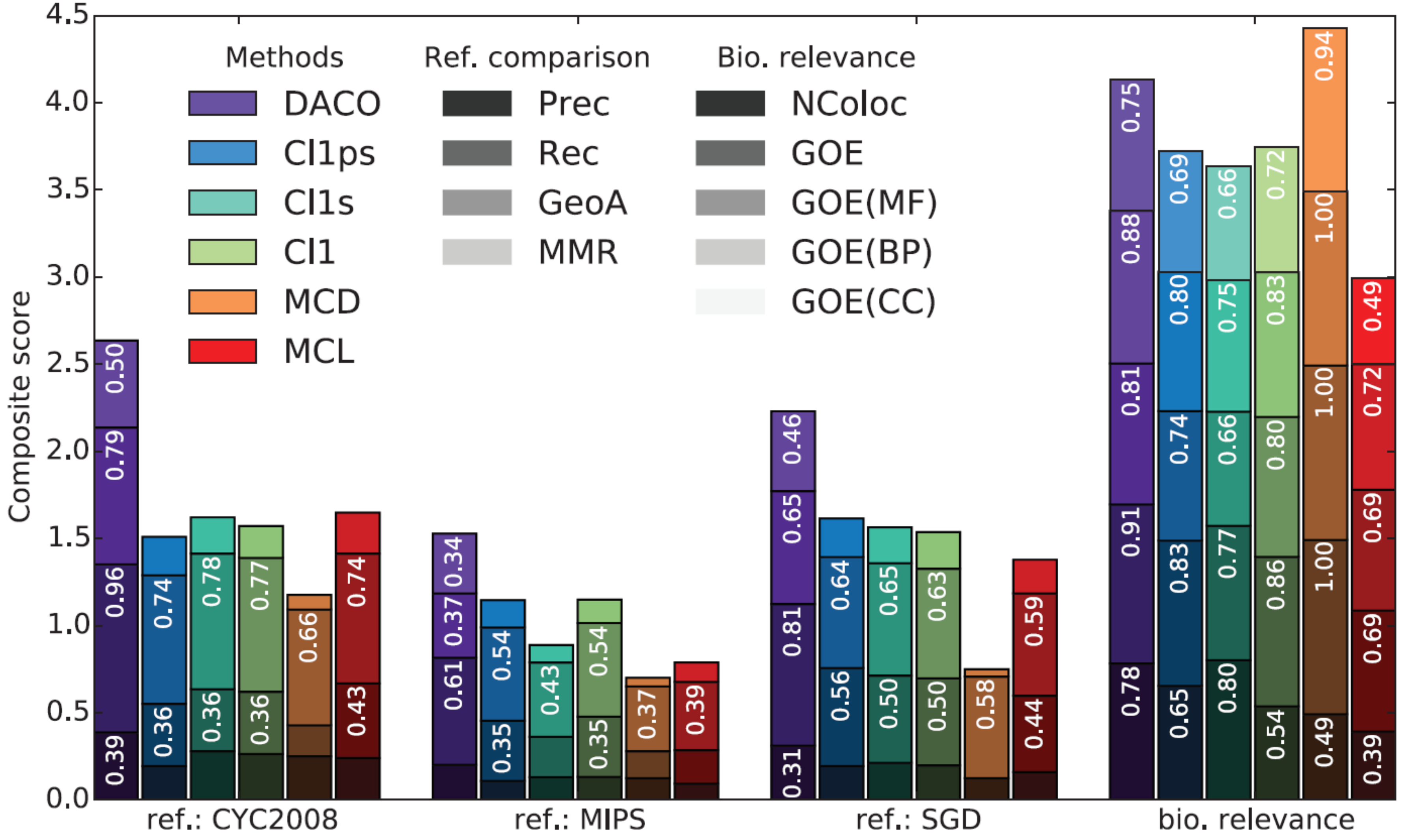
(e) ORC(MIPS) / DACO

(f) ORC(MIPS) / ClusterONE

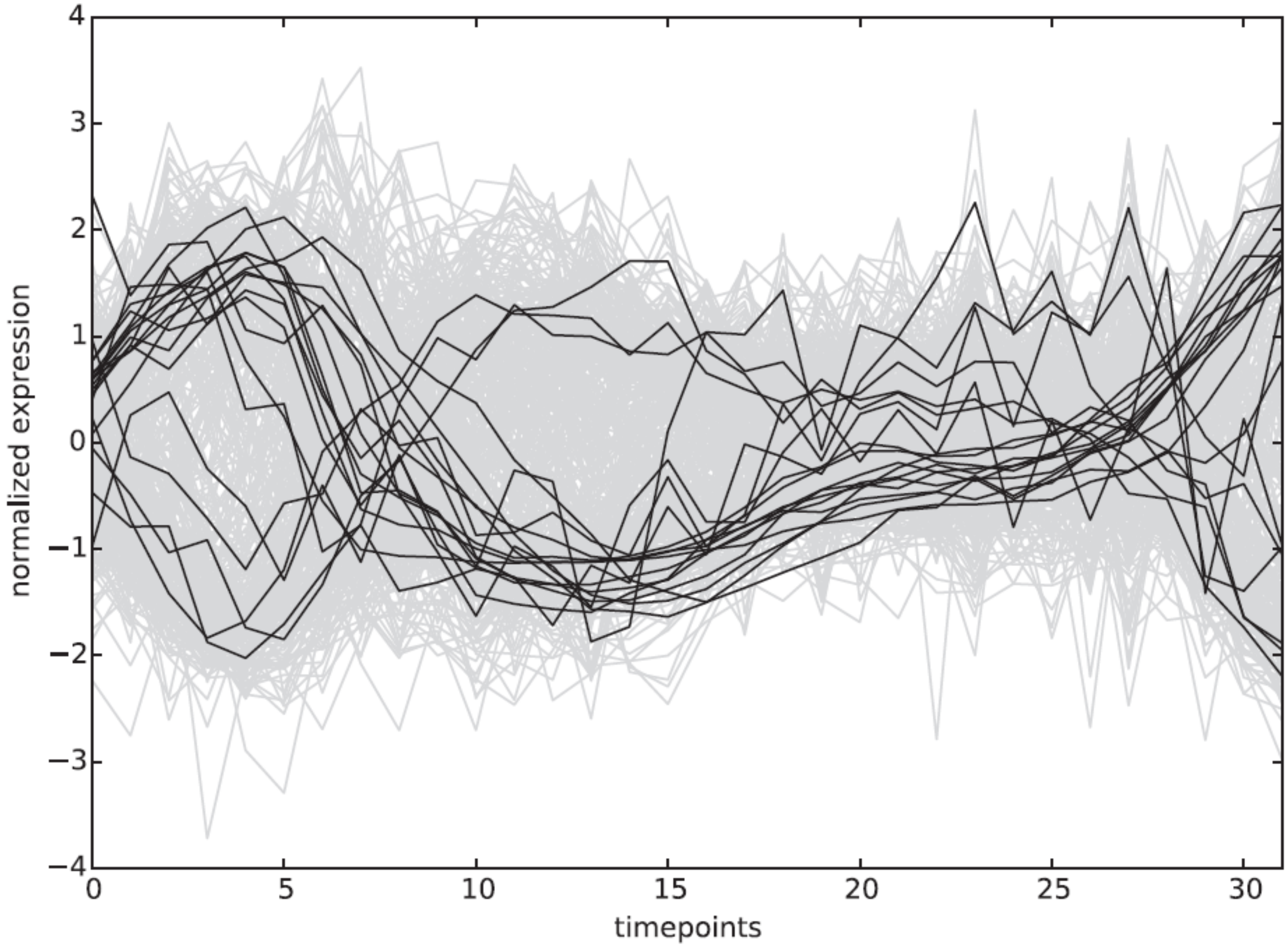
Green nodes: proteins in the reference that were matched by the prediction

red nodes: proteins that are in the predicted complex, but not part of the reference.

Performance evaluation



Co-expressed target genes of MET4/MET32 TF complex during yeast cell cycle



Functional role of TF complexes

TFs	P_{dECS}	Binding mode	Targets	Regulatory influence	GO process enrichment (P <0.05, Bonferroni corrected) in targets
MET4/MET32	0.0010	coloc.	19	+	Methionine metabolic process
TBP/HAP5	0.0335	med.	47	+	/
GLN3/DAL80	0.0009	med.	28	/	Allantoin catabolic process
DIG1/STE12/SWI6	0.0369	all	15	/	Fungal-type cell wall organization
FHL1/RAP1	0.0001	coloc.	116	+	rRNA transport
RPH1/GIS1	0.0001	med.	100	–	Hexose catabolic process
CBF1/MET32	0.0002	coloc.	33	o	Sulfate assimilation
DIG1/STE12	0.0003	med.	34	–	Response to pheromone
GCN4/RAP1	0.033	med.	62	+	/
MSN4/MSN2	0.0021	med.	105	+	Oligosaccharide biosynthetic process
DAL80/GZF3	0.0044	med.	20	–	Purine nucleobase metabolic process
SWI6/SWI4	0.0039	med.	53	+	Regulation of cyclin-dependent protein serine/threonine kinase activity
STB1/SWI6	0.0275	all	47	+	/
TBP/SWI6	0.0159	med.	14	+	/
GLN3/GZF3	0.0120	adj.	31	/	Allantoin catabolic process
MBP1/SWI6/SWI4	0.0307	med.	18	+	Regulation of cyclin-dependent protein serine/threonine kinase activity
MBP1/SWI6	0.0124	adj.	25	/	Cell cycle process

Note: Owing to the number of permutations of the test, the lowest possible value is $P_{dECS} = 10^{-4}$. The calculations were conducted for different conceivable modes of targeting (all shared target proteins, direct adjacency, mediated adjacency and colocalization) to have a detailed picture of the possible target–gene sets. Only the most enriched GO process term is shown for each target set. The inferred regulatory influence on the rate of transcription is abbreviated as follows: + (increase), – (decrease), o (no statement possible), / (conflicting annotations).

Summary

What you learned **today**:

- Network **robustness**
  scale-free networks are failure-tolerant, but fragile to attacks
  <=> the few **hubs** are important
  => immunize hubs!

- **Modules** in networks
  => modular decomposition
  => optimization of cohesiveness (DACO)

Next lecture:

- Are biological networks scale-free? (other models?)
- Network growth mechanisms