II. Graphentheoretische und statistische Eigenschaften zellulärer Netzwerkarchitekturen

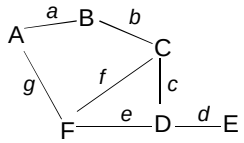
II. 1 Grundbegriffe der Graphentheorie

- Graph: Tupel bestehend aus Menge **V** von Knoten (engl: vertices) und Menge **E** von Kanten (engl: edges): **G**=(**V,E**)

- Kante $u \in$ **E** ist Knotenpaar (A,B) mit $A,B \in$ **V**

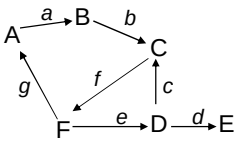
- ungerichteter Graph: Reihenfolge der 2 Knoten in den Kanten ohne Bedeutung
- gerichteter Graph (*digraph*): Kanten (*arc*) sind *geordnete* Paare von Knoten und für Kante u= (A,B) ist A der Anfangs- und B der Endknoten. (jeder gerichtete Graph hat zugrunde liegenden ungerichteten Graphen)

Bsp: Ungerichteter Graph (G1)



V={*A,B,C,D,E,F*}
E={*a,b,c,d,e,f,g*}
={(*A,B*),(*B,C*),(*C,D*),(*D,E*),(*D,F*),(*C,F*),(*F,A*)}

Bsp: Gerichteter Graph (G2)



V={*A,B,C,D,E,F*}
E={*a,b,c,d,e,f,g*}
={(*A,B*),(*B,C*),(*D,C*),(*D,E*),(*F,D*),(*C,F*),(*F,A*)}

• selfloop: Kante $u \in \mathbf{E}$ wo Anfangs- und Endknoten gleich sind: *u=(A,A)*

• den Kanten (und Knoten) können numerische Gewichte, Vorzeichen oder andere Labels zugeordnet sein

• Graphen können z.B. über Adjazenzmatrix (**A**), Inzidenzmatrix (**I**) oder Adjazenzliste (**L**) gespeichert werden

• Bsp G1: (ungerichtet)

$$\mathbf{A}=\begin{array}{c}\begin{matrix}A&B&C&D&E&F\end{matrix}\\\begin{pmatrix}0&1&0&0&0&1\\1&0&1&0&0&0\\0&1&0&1&0&1\\0&0&1&0&1&1\\0&0&0&1&0&0\\1&0&1&1&0&0\end{pmatrix}\begin{matrix}A\\B\\C\\D\\E\\F\end{matrix}\end{array},\quad \mathbf{I}=\begin{array}{c}\begin{matrix}a&b&c&d&e&f&g\end{matrix}\\\begin{pmatrix}1&0&0&0&0&0&1\\1&1&0&0&0&0&0\\0&1&1&0&0&1&0\\0&0&1&1&1&0&0\\0&0&0&1&0&0&0\\0&0&0&0&1&1&1\end{pmatrix}\begin{matrix}A\\B\\C\\D\\E\\F\end{matrix}\end{array},\quad \mathbf{L}=\begin{array}{c}\begin{pmatrix}B,F\\A,C\\B,D,F\\C,E,F\\D\\A,C,D\end{pmatrix}\begin{matrix}A\\B\\C\\D\\E\\F\end{matrix}\end{array}$$

→ **A** ist symmetrisch

• Bsp G2: (gerichtet)

$$\mathbf{A}=\begin{array}{c}\begin{matrix}A&B&C&D&E&F\end{matrix}\\\begin{pmatrix}0&1&0&0&0&0\\0&0&1&0&0&0\\0&0&0&0&0&1\\0&0&1&0&1&0\\0&0&0&0&0&0\\1&0&0&1&0&0\end{pmatrix}\begin{matrix}A\\B\\C\\D\\E\\F\end{matrix}\end{array},\quad \mathbf{I}=\begin{array}{c}\begin{matrix}a&b&c&d&e&f&g\end{matrix}\\\begin{pmatrix}-1&0&0&0&0&0&1\\1&-1&0&0&0&0&0\\0&1&1&0&0&-1&0\\0&0&-1&-1&1&0&0\\0&0&0&1&0&0&0\\0&0&0&0&-1&1&-1\end{pmatrix}\begin{matrix}A\\B\\C\\D\\E\\F\end{matrix}\end{array},\quad \mathbf{L}=\begin{array}{c}\begin{pmatrix}B\\C\\F\\C,E\\-\\A,D\end{pmatrix}\begin{matrix}A\\B\\C\\D\\E\\F\end{matrix}\end{array}$$

→ **A** nicht symmetrisch

• Grad eines Knotens (*deg*): Anzahl der Kanten, an denen ein Knoten „hängt“
Bsp: G1: *deg*(C)=3

in gerichteten Graphen:
Eingangsgrad (*indeg*) eines Knotens: Anzahl eingehender Kanten
Ausgangsgrad (*outdeg*) eines Knotens: Anzahl ausgehender Kanten
Bsp. G2: *indeg*(*C*)=2, *outdeg*(*C*)=1

Im Folgenden werden nur gerichtete Graphen betrachtet, alle Definitionen lassen sich aber sehr leicht auf ungerichtete Graphen übertragen. Man unterscheidet dann z.B. zwischen gerichtetem und ungerichtetem Pfad, Weg, ...

• Kette (*walk*): Folge von Kanten $\mathbf{K}=(k_1,k_2,\ldots,k_n)$, wo jeder Anfangsknoten von k_{i+1} gleich Endknoten von k_i ist: $\mathbf{K}=((S,L_1),(L_1,L_2),(L_2,L_3),\ldots(L_{n-2},L_{n-1}),(L_{n-1},E))$
Bsp G2: **K**=(*a,b,f,e,c*) ist (*A,C*)-Kette: führt von A über B,C,F,D nach C

• Pfad (*trail*): ist eine Kette, wo keine Kante mehr als einmal durchlaufen wird
Bsp G2: **P**=(*a,b,f,e,c*) ist ein (*A,C*)-Pfad.

• Weg (*path*): ist ein Pfad, wo weder Kanten noch Knoten doppelt durchlaufen werden
Bsp G2: **W**=(*a,b,f,e*) ist ein (*A,D*)-Weg.

Begriffe Pfad und Weg oft uneinheitlich definiert.

- Zyklus (*cycle, circuit*): Pfad, bei dem Anfangs- und Endknoten identisch sind (Knoten dürfen doppelt auftreten; Kanten nicht)

- Kreis (*simple cycle* oder *circuit*): Zyklus, bei dem alle Knoten (außer Start- und Endknoten) nur einmal durchlaufen werden
  Bsp G2: C1=(*a,b,f,g*), C2=(*f,e,c*);
  in G1 (ungerichtet) existiert zusätztlich C3=(*a,b,c,e,g*)

- gerichtete azyklische Graphen (*DAG*): ohne gerichtete Kreise
  Bsp: G2 ist nicht azyklisch; wäre f entgegengerichtet, dann ja

- ein Knoten *A* ist verbunden/erreichbar (*connected*) mit Knoten *B* wenn es einen (*A,B*)-Weg gibt.
  Bsp: *E* ist mit *F* verbunden in Graph G1 aber nicht in G2

- ein ungerichteter Graph ist zusammenhängend (*connected*), wenn alle Paare von Knoten verbunden sind; eine Zusammenhangskomponente (*component*) ist ein maximal zusammenhängender Teilgraph
  Bsp: G1 ist zusammenhängend und hat deshalb genau eine Zusammenhangskomponente (G1 selber)

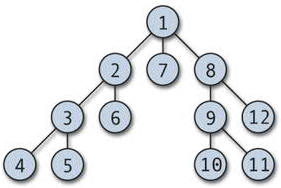
- ein gerichteter Graph ist stark zusammenhängend (*strongly connected*) wenn für zwei Knoten *A* und *B* gilt, dass *A* mit *B* und *B* mit *A* verbunden ist
- eine starke Zusammenhangskomponente *(strong component)* ist maximaler stark zusammenhängender Teilgraph
- ein gerichteter Graph ist schwach zusammenhängend (*weakly connected*) wenn der zugrunde liegende ungerichtete Graph zusammenhängend ist
  Bsp: G2 ist schwach zusammenhängend; seine starken Zusammenhangskomponenten sind {*A,B,C,D,F*} und {*E*}
  (ganz G2 ist eine schwache Zusammenhangskomponente)

- zusammenhängende ungerichtete Graphen bzw. schwach zusammenhängende gerichtete Graphen ohne Kreise heißen Baum *(tree)*
  Bsp: G1 ist kein Baum (ohne Kanten f und e aber schon)
  G2 ist kein Baum (ohne Kante f aber schon)

Suchalgorithmen in Graphen

• zwei fundamentale Suchalgorithmen in Graphen:

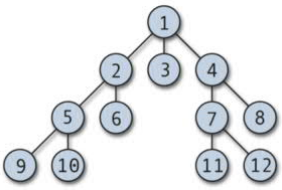
Tiefensuche (Depth-First)



• rekursiv

• verwendet Stacks (LIFO)

Breitensuche (Breadth-First)



• sequentiell

• verwendet Warteschlange (FIFO)

Falls Zyklen existieren (keine Baumstruktur), ist Markierung („besucht“) in den Knoten notwendig – sonst können Endlosschleifen entstehen.

Kürzeste Wege

• wir betrachten *gerichtete, gewichtete Graphen*: **G**=(**E,V**,*w*)
w ist Gewichtsfunktion (oder ‚Länge“) : *w*: **E** → ℜ ; für Gewicht der Kante *u*=(*A,B*) schreiben wir *w*(A,B); hier: nur positive Gewichte

• Aufgabe: finde kürzesten Weg (*shortest path;* dist(*A,B*)) **W** zwischen zwei Knoten *A* und *B*, d.h. Summe der Gewichte aller Kanten in **W** soll minimal sein

• verschiedene Algorithmen; bekanntester: Dijkstra Algorithmus:

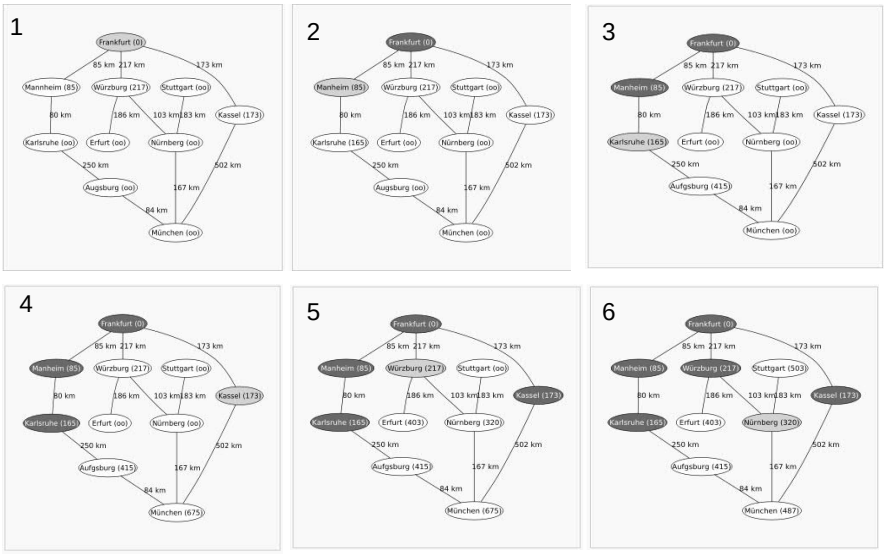
Gesucht sind die kürzesten Verbindungen von einem Knoten **s** zu allen anderen Knoten **v**. Die Idee des Verfahrens besteht darin, schrittweise eine Menge **M** von Knoten zu vergrößern, für deren Elemente **v** wir bereits einen kürzesten Weg von **s** nach **v** gefunden haben. Allen anderen Knoten **v**, die nicht in **M** liegen, ordnen wir die Länge des bisher gefundenen kürzesten Weges zu. Wir bezeichnen dazu mit **Dist(v)** die Länge des (bisher gefundenen) kürzesten Weges und mit **Vor(v)** den Vorgänger auf einem solchen Weg. **w**(**a**,**b**) sind die Gewichte für die Kante von **a** nach **b**. **E/V** = Menge der Kanten/Knoten.

Setze **Dist(s) := 0, Vor(s) := s, M := {s}**
Für **v aus V** mit **(s, v) aus E** setze **Dist(v) := w(s, v)** und **Vor(v) := s**
Für **v aus V** mit **(s, v) nicht aus E** setze **Dist(v) := unendlich** und **Vor(v) := leer**

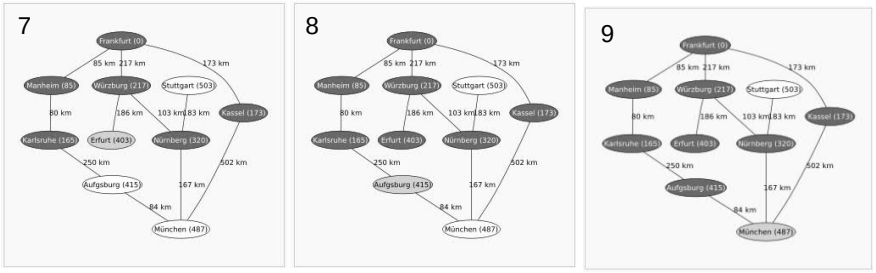
*Bestimme **u nicht aus M** mit **Dist(u) = min {Dist(v): v nicht aus M}**
Falls **Dist(u) = unendlich** , dann **STOP**. Andernfalls setze **M := M ∪ {u}**
Für alle **v aus V** mit **(u, v) aus E**:
Falls **Dist(v)>Dist(u)+w(u,v)** , dann setze:
Dist(v) := Dist(u)+w(u,v)
Vor(v) := u
Falls **M ≠ V** dann **STOP**, sonst gehe zu *.

Beispiel Dijkstra: Kürzester Weg von Frankfurt nach München

(von http://de.wikipedia.org/wiki/Dijkstra-Algorithmus)



... Fortsetzung: Kürzester Weg von Frankfurt nach München



- der Algorithmus wird (noch) einfacher, wenn man die Weglänge über Anzahl der durchlaufenden Kanten quantifiziert (setze alle Gewichte=1): nach n-tem Schritt alle Knoten identifiziert, die vom Startknoten aus in n Schritten erreichbar sind (entspricht dann Breitensuche)
- auch: alle kürzesten Wege von einem/allen Startknoten zu allen anderen

- polynomiale Laufzeit (sehr effizient; polynomialer Algorithmus)
- viel schwieriger: längster Weg zw. *A* und *B* (NP-vollständig)

- Alternative Kürzeste-Wege-Algorithmen:
  - Bellman-Ford: auch mit negativen Kanten (aber keine negativen Kreise)
  - Floyd-Warshall: für sämtliche n*n kürzesten Pfade (n: Anzahl Knoten)

II.2 Globale Architektur von zellulären Netzwerken

Globale/Statistische Kennzahlen von Graphen

- Konnektivität: ausgedrückt über P(k) = Wahrscheinlichkeit, dass ein Knoten Grad k hat (möglicherweise Ein/Ausgangsgrad unterscheiden)

- durchschnittlicher Knotengrad: <k>=Σdeg(k) / n

- Netzwerk-Durchmesser: Maximum aller kürzesten Weglängen: $\max_{A,B\in \mathbf{V}}(\mathrm{dist}(A,B))$
- mittlere Weglänge: Mittelwert über alle dist(*A*,*B*))

- Clusterkoeffizient C_A für einen Knoten *A*: Anzahl L_A der Kanten zwischen allen Nachbarn von *A* geteilt durch die theoretisch mögliche Anzahl

$$C_A = \frac{L_A}{\deg(A)\cdot(\deg(A)-1)/2}$$

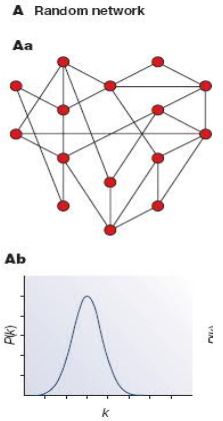
Bsp: in G1: C_F = 1/3

Netzwerk-Clusterkoeffizient C ist der Mittlerwert über alle Knoten-Clusterkoeffizienten; Cliquen (Teilgraphen wo zwischen jedem Paar von Knoten eine Kante ist): C=1

Zwei wichtige Klassen von Netzwerkarchitekturen

Random-Netzwerke (Erdös-Renyi-Modell)

• nimm N Knoten und verbinde jedes Paar von Knoten mit Wahrscheinlichkeit *p (approximativ p*N*(N-1)/2 Kanten entstehen)*



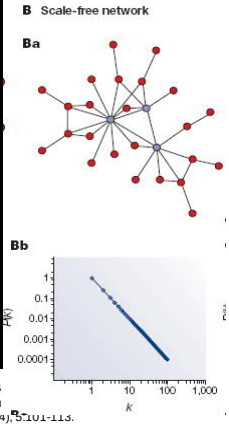
P(k)=Poissonverteil. $P(k)\approx e^{-k}$ für große k (k=Knotengrad)

„Hubs" extrem unwahrscheinlich

Kürzeste Pfade: ≈ log N

Scale-free (skalenfreie Netzwerke) (Barabasi-Albert-Modell)

• nimm N Knoten und füge Kanten ein: Knoten mit hoher Konnektivität k haben höhere Wahrscheinlichkeit neue Kante zu bekommen; für Knoten L: $p_L=k_L/\Sigma_J k_J$



$P(k)= k^{-\gamma}$ Potenzgesetzverteilung (Power-law-distribution): Linie in log-log-Plot

Viele Knoten schwach verknüpft, aber auch relativ hohe Wahrsch. von „Hubs" (Knoten mit hoher Konnektivität)

Kürzeste Pfade ($2<\gamma<3$): ≈ log log N

Barabási, A. & Oltvai, Z. N. Network Biology: Understanding the cell's fu organisation. Nature Reviews (2004).5:101-113.

Viele zelluläre Netzwerke sind skalenfrei

- viele reale Netze sind skalenfrei (z.B. soziale Netze, Internet, Verkehrssysteme)
- nachgewiesen auch in biologischen Netzwerken: z.B. Stoffwechselnetze (dargestellt als Graph!; $2<\gamma_{in},\gamma_{out}<3$):), genregulatorische Netze und Protein-Interaktionsnetze
- biologisch relevante Eigenschaften von skalenfreien Netzen:
  - Skalenfreiheit in zellulären Netzen evolutionär interpretierbar
  - viele Knoten schwach verknüpft und einige wenige („Hubs“ - z.B. ATP, NADH Precursor in Stoffwechselnetzen) sehr stark → robust gege zufällige Störungen (geringe Wahrscheinlichkeit, dass Hubs getroffen werden) – aber fragil gegen gezielte Störungen gegen Hubs
  - kleiner Durchmesser, kürzeste Wege klein = *small-world property:* Störungen und Signale werden schnell weitergeleitet

Beispiele für skalenfreie zelluläre Netzwerke

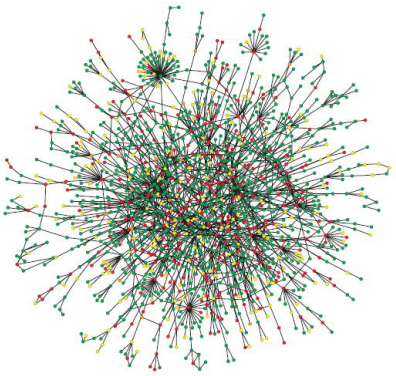


Figure 2 | **Yeast protein interaction network.** A map of protein–protein interactions[18] in *Saccharomyces cerevisiae*, which is based on early yeast two-hybrid measurements[23], illustrates that a few highly connected nodes (which are also known as hubs) hold the network together. The largest cluster, which contains ~78% of all proteins, is shown. The colour of a node indicates the phenotypic effect of removing the corresponding protein (red = lethal, green = non-lethal, orange = slow growth, yellow = unknown). Reproduced with permission from REF. 18 © Macmillan Magazines Ltd.

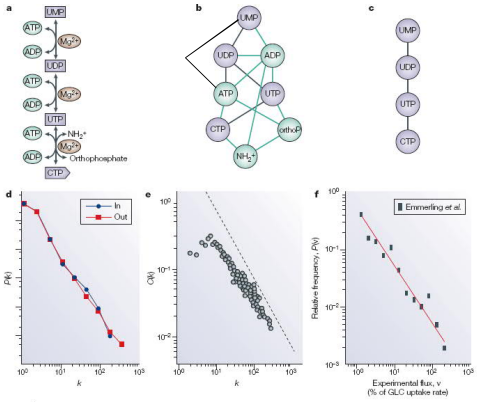


Figure 1 | **Characterizing metabolic networks.** To study the network characteristics of the metabolism a graph theoretic description needs to be established. Here, the graph theoretic description for a simple pathway (catalysed by Mg^{2+}-dependant enzymes) is illustrated (**a**). In the most abstract approach (**b**) all interacting metabolites are considered equally. The links between nodes represent reactions that interconvert one substrate into another. For many biological applications it is useful to ignore co-factors, such as the high-energy-phosphate donor ATP, which results in a second type of mapping (**c**) that connects only the main source metabolites to the main products. **d** | The degree distribution, $P(k)$, of the metabolic network illustrates its scale-free topology[30]. **e** | The scaling of the clustering coefficient $C(k)$ with the degree k illustrates the hierarchical architecture of metabolism[31] (The data shown in **d** and **e** represent an average over 43 organisms[30,31]). **f** | The flux distribution in the central metabolism of *Escherichia coli* follows a power law, which indicates that most reactions have small metabolic flux, whereas a few reactions, with high fluxes, carry most of the metabolic activity[32]. This plot is based on data that was collected by Emmerling *et al.*[33]. It should be noted that on all three plots the axis is logarithmic and a straight line on such log–log plots indicates a power-law scaling. CTP, cytidine triphosphate; GLC, aldo-hexose glucose; UDP, uridine diphosphate; UMP, uridine monophosphate; UTP, uridine triphosphate.

Barabási, A. & Oltvai, Z. N. Network Biology: Understanding the cell's functional organisation. Nature Reviews, (2004), 5:101-113.

Eine andere Perspektive: Metabolische Netzwerke zeigen Bowtie-Struktur

Doyle J & Csete M (2004) Bowties, Metabolism and Disease. Trends in Biotechnology, Volume 22, 446-450.

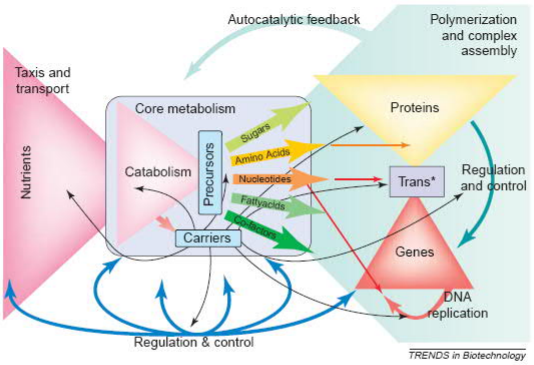


Figure 1. The nested bow-tie architectures of metabolism input a wide range of nutrients and produce a large variety of products and complex macromolecules using a relatively few intermediate common currencies. The common currencies and their enzymes form the knot of the bow tie. The overall bow tie can be decomposed into three principal subsidiary bow ties. One produces the activated carriers, such as ATP, NAD and NADP, that globally supply the cell with energy, reducing power and small moieties. In parallel, catabolism produces a standard group of 12 precursor metabolites, among them glucose 6-phosphate (G6P), fructose 6-phosphate (F6P), phosphoenolpyruvate (PEP), pyruvate (PYR), α-ketoglutarate (AKG) and acetyl-coenzyme A (ACCOA), which are the starting points for the biosynthesis of amino acids, nucleotides, carbohydrates, fatty acids and cofactor building blocks. These building blocks are then used by general-purpose polymerases, particularly in the transcription and translation (trans*) bow tie, to assemble complex macromolecules. This architecture uses selective homogeneity at the knot to facilitate control, organization and management of the enormous heterogeneity in enzyme specificity, action and regulation, and in substrate size, flux and concentration. All modern technologies, from manufacturing to the power grid to the Internet, are organized with bow ties.

- hohe Diversität in Inputs (Substrate) und Outputs (Proteine, Biomasse, ...)
- Zentralstoffwechsel: einheitliche „Protokolle“ und „Währungen“(ATP, NAD(P)H, Precursor) → hohe Flexibilität (plug and play); leicht erweiterbar (evolvierbar); aber: leicht anfällig gegen „Geiselnahme“ dieser Protokolle (z.B. Viren)

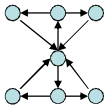
II.3 Netzwerkmotive

- Netzwerkmotive (*network motifs*): sich wiederholende Strukturmuster in Graphen, mit signifikant erhöhter Auftrittshäufigkeit im Vergleich zu randomisierten Netzen

- Motive wurden erstmals in genregulatorischen Netzwerken (Knoten: Gene/ Transkriptionsfaktoren) von Prokaryoten (*E. coli*) identifiziert; mittlerweile wurden Motive auch in anderen Netzwerktypen (z.B. Signaltransduktion; aber immer: Signalflussnetze) nachgewiesen

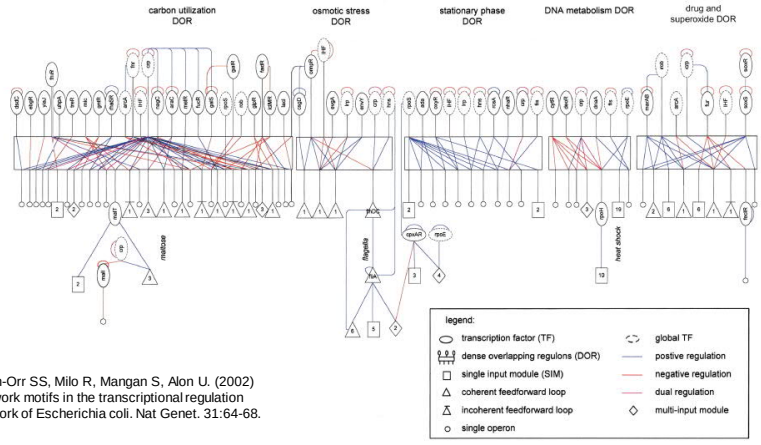
- Netzwerkmodell ist meistens Vorzeichengraph (gerichtete Graphen mit +/- Kanten; siehe Kapitel 4)

- Motive können als „Netzwerk-Bausteine“ verstanden werden



Motiv mit signifikant erhöhter Auftrittshäufigkeit

Transkriptions- (genregulatorisches) Netzwerk in *E.coli*



Shen-Orr SS, Milo R, Mangan S, Alon U. (2002) Network motifs in the transcriptional regulation network of Escherichia coli. Nat Genet. 31:64-68.

Fig. 3 Part of the network of direct transcriptional interactions in the *E. coli* data set, represented using network motifs. Nodes represent operons, and lines represent transcriptional regulation, directed so that the regulating transcription factor is above the regulated operons. Network motifs are represented by their corresponding symbols (Fig. 1). The DORs are named according to the common function of their output operons. Each transcription factor appears in only a single subgraph, except for transcription factors regulating more than ten operons ('global transcription factors'), which can appear in several subgraphs. For an image of the entire network, see Web Fig. A online.

- 4 signifikant häufige Motive:

Die 4 wichtigsten Netzwerkmotive im E.coli Genregulationsnetzwerk

1) Negative Autoregulation

- Transkriptionsfaktor inhibiert eigene Expression
- Funktion: Robustheit gegen Störungen

2) Feed-Forward Loop

X → Y → Z

Bsp: crp → araC → araABD

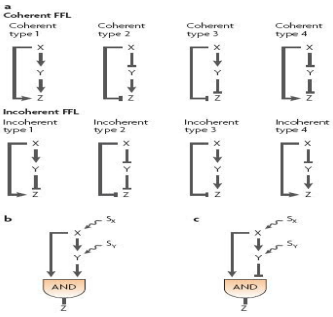


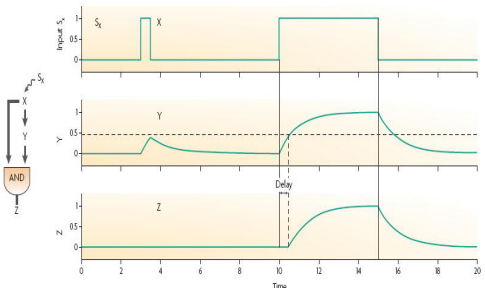
Figure 2 | **Feedforward loops (FFLs). a** | The eight types of feedforward loops (FFLs) are shown. In coherent FFLs, the sign of the direct path from transcription factor X to output Z is the same as the overall sign of the indirect path through transcription factor Y. Incoherent FFLs have opposite signs for the two paths. **b** | The coherent type-1 FFL with an AND input function at the Z promoter. **c** | The incoherent type-1 FFL with an AND input function at the Z promoter. S_x and S_y are input signals for X and Y.

- durch +/- Interaktionen theoretisch 2^3=8 Feedforward Loops möglich: 4 sind kohärent und 4 inkohärent; dazu unterschiedliche AND/OR/SUM Kombinationen am Knoten Z
- am häufigsten sind Kohärent Typ1 (CFFL1) und Inkohärent Typ1 (IFFL1, siehe links)
- Funktion:
  CFFL1: Filtern von persistenten Signalen (kurze ON Signale (AND Logik) bzw. kurze OFF Signale (OR) werden gefiltert)

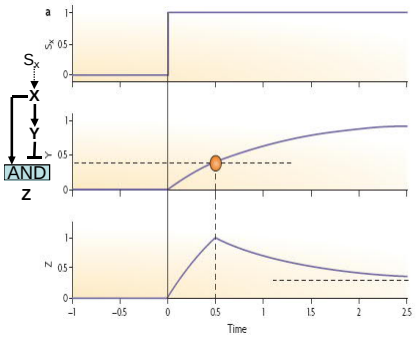
  IFFL1: z.B. Pulsgenerator

Alon U. Network motifs: theory and experimental approaches. Nat Rev Genet. 2007 Jun;8(6):450-61.

Kohärenter Feedforward-Loop mit AND gate: Persistenz-Detektor

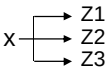


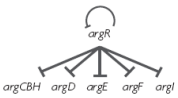
Inkohärenter Feedforward-Loop mit AND gate: Pulsgenerator



Alon U. Network motifs: theory and experimental approaches. Nat Rev Genet. 2007 Jun;8(6):450-61.

3) Einfaches Inputmodul





- ein Transkriptionsfaktor aktiviert viele andere
- Funktion: simultane Aktivierung/Deaktivierung mehrerer Gene (Regulon/Modulon);

4) Dicht überlappende Regulons

- kombinatorische Verschaltung von mehreren Inputs auf mehrere Outputs
- Verrechnung der Signale oft nicht bekannt

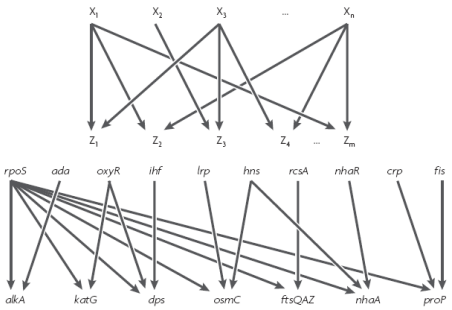
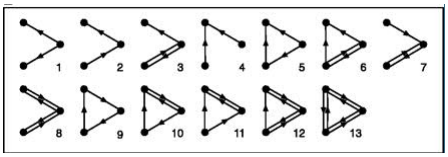


Figure 6 | **The dense overlapping regulon (DOR) network motif.** In this motif, many inputs regulate many outputs (top panel). The bottom panel shows an example from the stress-response system of *Escherichia coli*.

Detektion von Netzwerkmotiven

- Aufspüren signifikant überrepräsentierter Muster im Vergleich zu Zufallsnetzen
- hochgradig kombinatorisches Problem: z.B. 13 mögliche Motive bei 3 Knoten und 199 Motive bei 4 Knoten
- deshalb nur Suche nach relativ kleinen Motiven möglich; meist über wiederholtes Random-Sampling



Alle möglichen Netzwerkmotive mit 3 Knoten

R Milo, S Shen-Orr, S Itzkovitz, N Kashtan, D Chklovskii, U Alon. (2002) Network Motifs: Simple Building Block of Complex Networks. *Science* 298: 824 – 827

Referenzen

- Barabási, A. & Oltvai, Z. N. Network Biology: Understanding the cell's functional organisation. Nature Reviews, (2004), 5:101-113.
- Shen-Orr SS, Milo R, Mangan S, Alon U. Network motifs in the transcriptional regulation network of Escherichia coli. Nat Genet. (2002) 31:64-68.
- Alon, U: An Introduction to Systems Biology. Chapman & Hall, 2005.
- Jungnickel D: Graphs, Networks and Algorithms. Springer Verlag Heidelberg, 2007.
- Junker BH and Schreiber F. Analysis of Biological Networks. Wiley, 2008.

Tools

- Cytoscape: Darstellen großer (zellulärer) Graphen: www.cytoscape.org
- Pajek: Darstellung + Analyse großer Graphen: http://mrvar.fdv.uni-lj.si/pajek/

Übungen

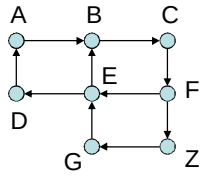
MATLAB weiter festigen ...

Wiederholung: Beispiel für eine Funktion *inci2adja*, die eine Inzidenmatrix eines gerichteten Graphen in eine Adjazenzmatrix umwandelt (Syntaxregeln bzw. MATLAB Kommandos sind hervorgehoben)

```
function adja = inci2adja(inci)     % Deklaration der Funktion inci2adja, welche als Input die Inzidenzmatrix
                                    % bekommt und als Output die dazugehörige Adjazenzmatrix liefert
specs=size(inci,1);                 % Anzahl Knoten = Anzahl Zeilen in inci
edges=size(inci,2);                 % Anzahl Kanten = Anzahl Spalten in inci
adja=zeros(specs,specs);            % Initialisiere eine Adjazenzmatrix mit Nullen
for i=1:edges                       % Von i=1 bis Anzahl Spalten
   s=find(inci(:,i)<0);             % Finde für die i-te Kante (Spalte) den Startknoten (-1)
   z=find(inci(:,i)>0);             % Finde für die i-te Kante (Spalte) den Endknoten (1)
   adja(s,z)=adja(s,z)+1;           % Zähle Anzahl Kanten S->Z eins hoch
end                                 % Ende von for
```

Übungen

1)



Betrachte jeweils ungerichteten/gerichteten Graphen:

- wieviele Kreise gibt es?
- wieviele Wege von E nach Z gibt es
- wieviele Pfade von E nach Z gibt es
- Knotenreihenfolge bei Tiefensuche (Start in F) (nur für gerichteten Graphen; nimm an, dass alphabetische Ordnung existiert (z.B. A vor B))
- Knotenreihenfolge bei Breitensuche (Start in F) (nur für gerichteten Graphen; nimm an, dass alphabetische Ordnung existiert (z.B. A vor B))

2) Wie kann man einer Adjazenzmatrix entnehmen, ob ein gerichteter Graph stark/schwach zusammenhängend ist? Schreiben Sie eine MATLAB-Funktion, die die Adjazenzmatrix eines gerichteten Graphen übergeben bekommt und ausgibt, ob der Graph stark zusammenhängend ist!

3) Schreiben Sie in MATLAB eine Funktion, der eine Adjazenz-Matrix eines gerichteten Graphen und ein Startknoten übergeben wird und die die Knotenreihenfolge bei Breitensuche ausgibt.- Testen Sie damit Ihre Ergebnisse aus Aufgabe (1)!

Übungen

4) Passen Sie den Breitensuchalgorithmus von (3) so an, dass die kürzesten Weglängen von einem Knoten zu allen anderen Knoten berechnet werden! Überprüfen Sie damit Ergebnisse von Aufgabe 1.

5) Erweitern/Ändern sie die Funktion von (4) dahingehend, dass alle kürzesten Wege (für jedes Paar von Knoten) für eine übergegebene Adjazenzmatrix berechnet werden. Rückgabe sollte eine nxn-Matrix sein (n=Anzahl Knoten) in der im Element (i,j) die Weglänge von *i* nach *j* steht.

6) Schreiben Sie in MATLAB eine Funktion, der eine Adjazenz-Matrix eines gerichteten Graphen und ein Startknoten übergeben wird und die die Knotenreihenfolge bei Tiefensuche ausgibt.
- Testen Sie damit Ihre Ergebnisse aus Aufgabe (1)!