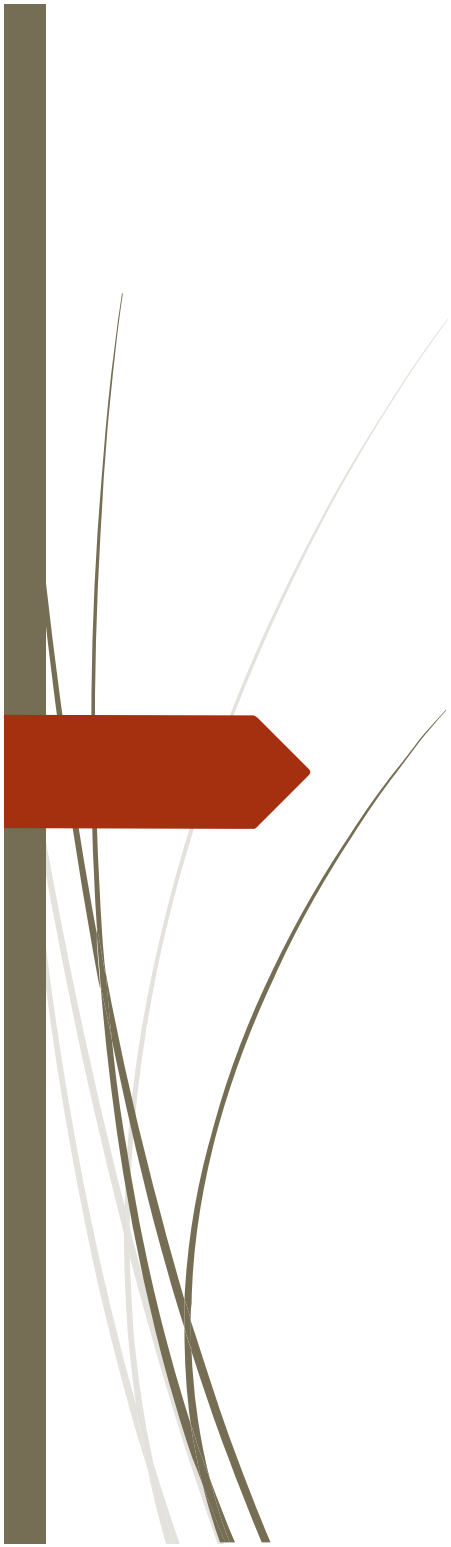
Programmation par ensembles-réponses, application à l'étude de systèmes biologiques à grande-échelle

Anne Siegel

CNRS, Univ Rennes

IRISA



Short team presentation: Dyliss @Irisa/Inria Rennes

Dynamics, **L**ogics and **I**nference
for biological **S**ystems and **S**equences

Team presentation

IRISA

- Computer science lab of Rennes
- 750 employees, 40 teams
- Univ Rennes, CNRS, Inria.

Bioinformatics

- GenOuest: ressource center
- Genscale: NGS sequence analysis
- Dyliss: integration of heterogeneous data

Dyliss team for short

Interdisciplinary team:

- Computer science & Math
- Biologists (environment, health)
- Multiple applications: microbiology, marine biology, health…
- Collaborations with Germany and Chile

Bioinfomatics:

Decipher
key genomic actors
from multi-scale observations
of a system response

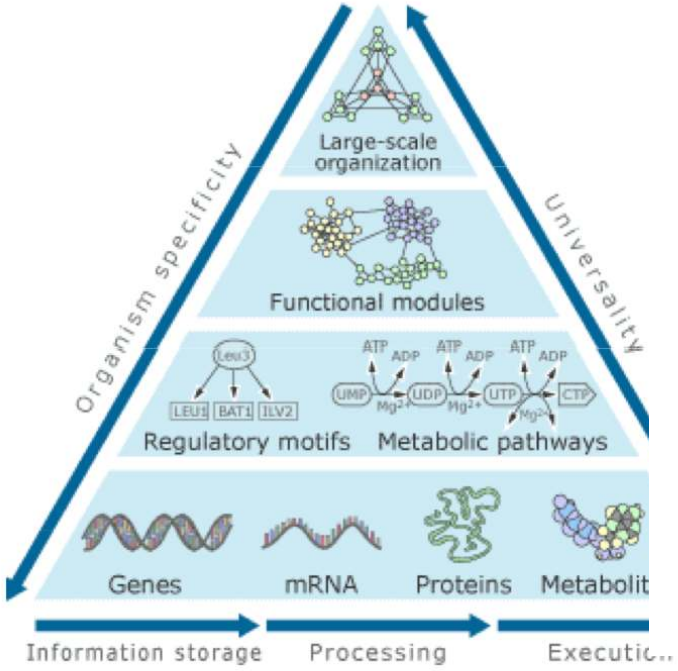
Computer Science:

Formalizing, integrating & reasoning with heterogeneous large-scale biological data

Motivation: systems biology… in concrete applications ?

Statement : **biology is a complex (dynamical) system**

- *« Requires to examine the structure and dynamics of a cellular function rather than the characteristics of isolated parts of a cell »* *(Kitano, 2002)*

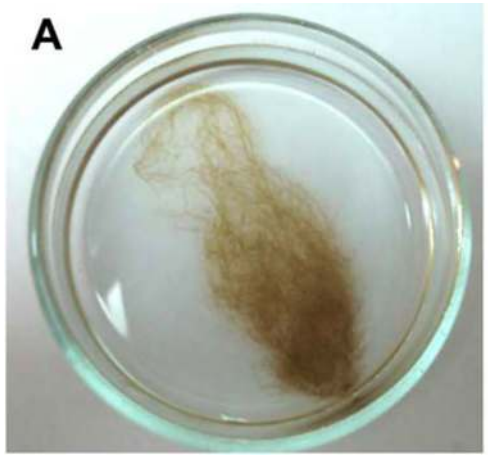


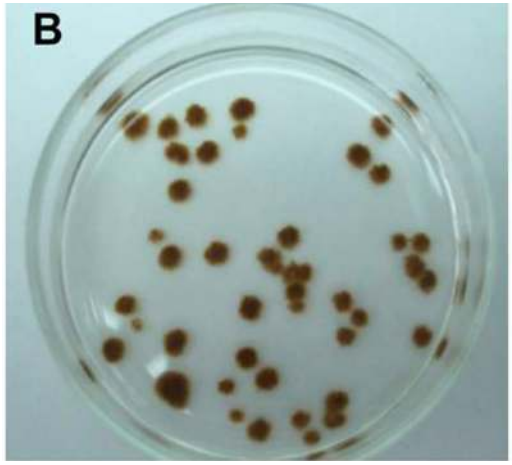
Systems biology: **Interpreting multi-layer data and graphs**

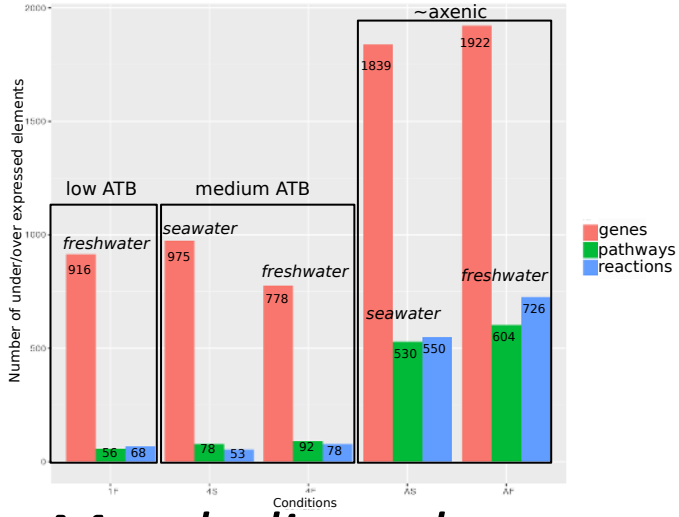
- **Produce predictive statements** that can be experimentally validated

Case-study: algal metabolism

CNRS • SORBONNE UNIVERSITÉ
Station Biologique de Roscoff







E. siliculosus

In axenic condition…

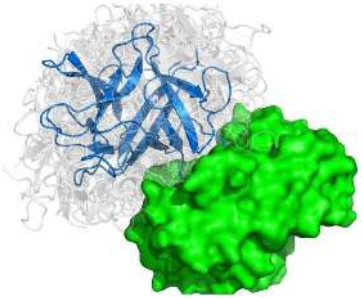
Metabolic scale

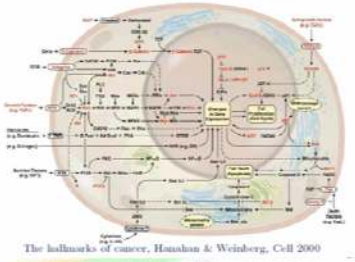
[Dittami2014, Tapia2016]

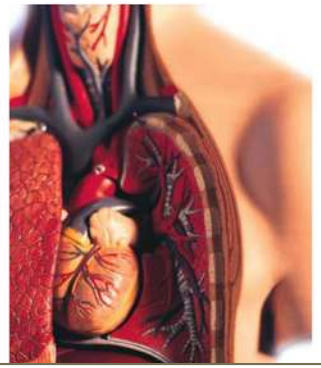
Fremy, Frioux, unpublished

What is the role of environmental bacteria at metabolic scale ?

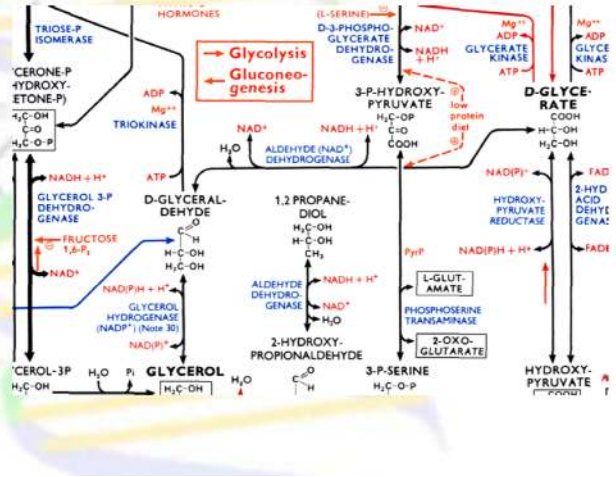
Complex systems are everywhere



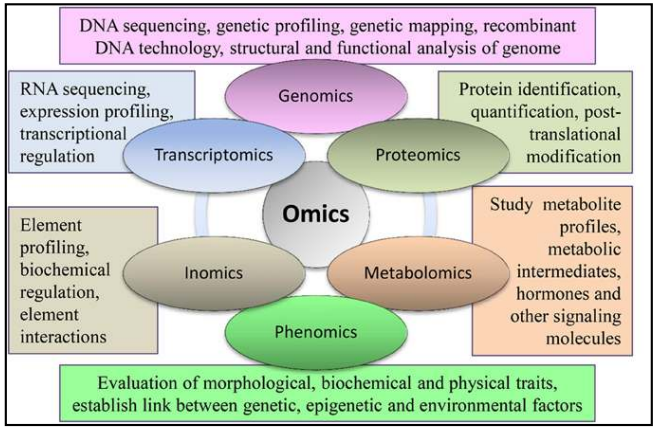




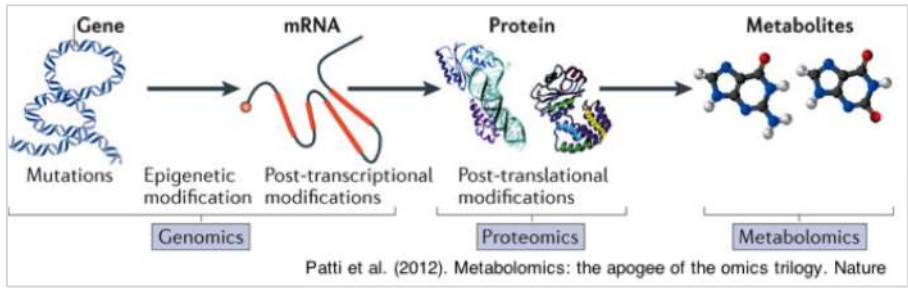
Are molecular/cellular complex systems different than others ?



Life science data nightmare



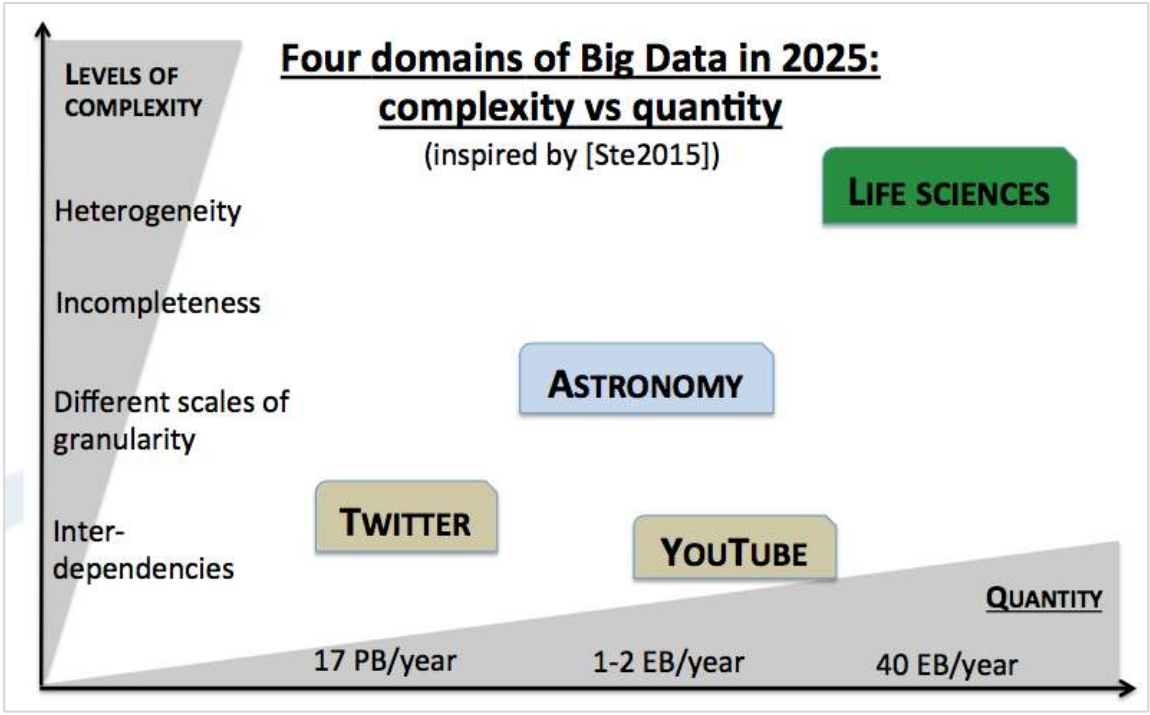
Data deluge (?)



Interdependy processes

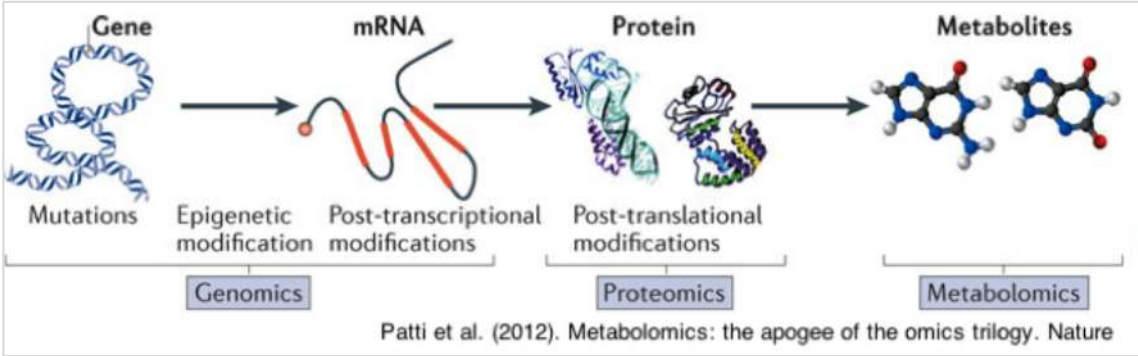
Data characteristics

- Large-scale
- Incomplete
- Interdependent
- **Heterogeneous**
- **Multi-layer**



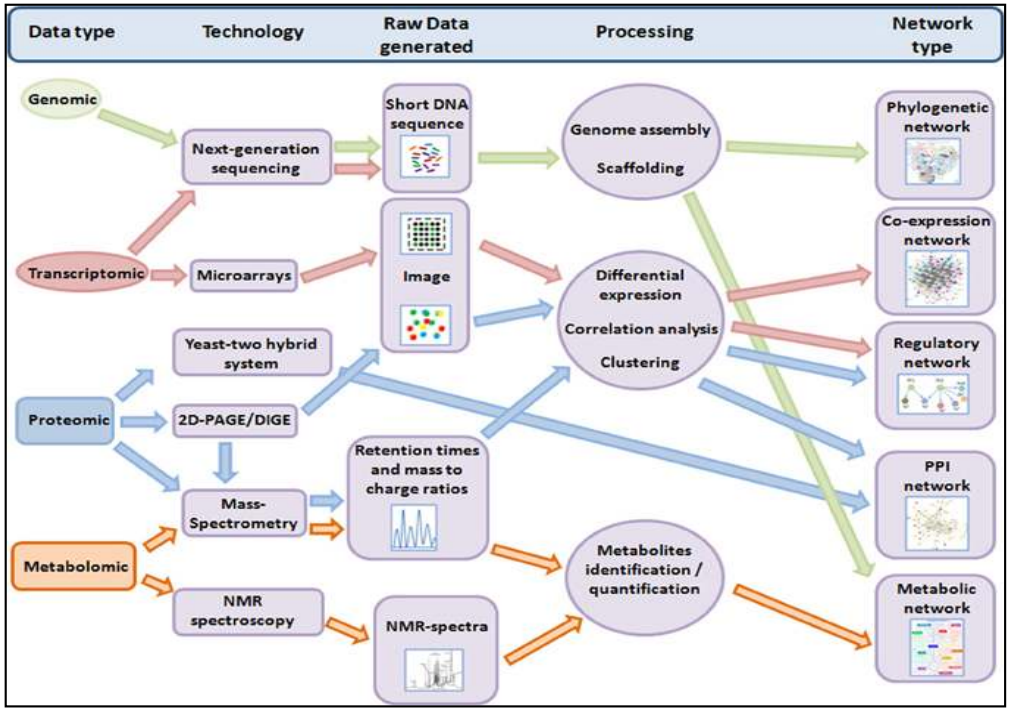
Multi-layer is quite specific to biology

Integrative biology: setting all together



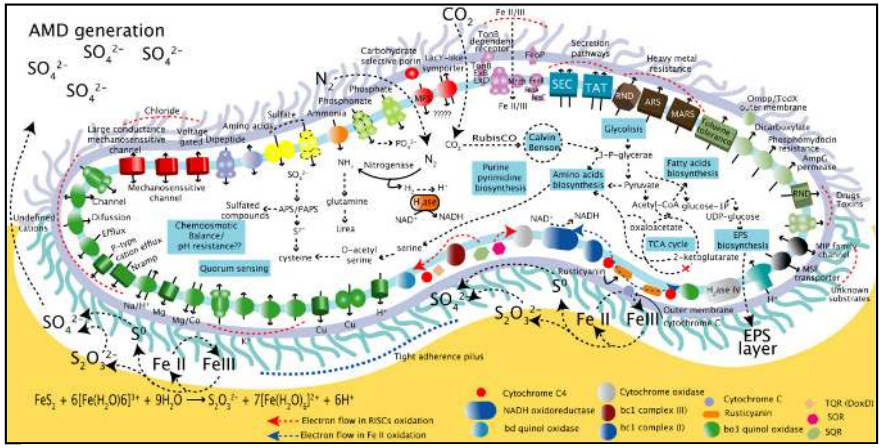
Gene function = regulation of a intra-cellular transformation procedure

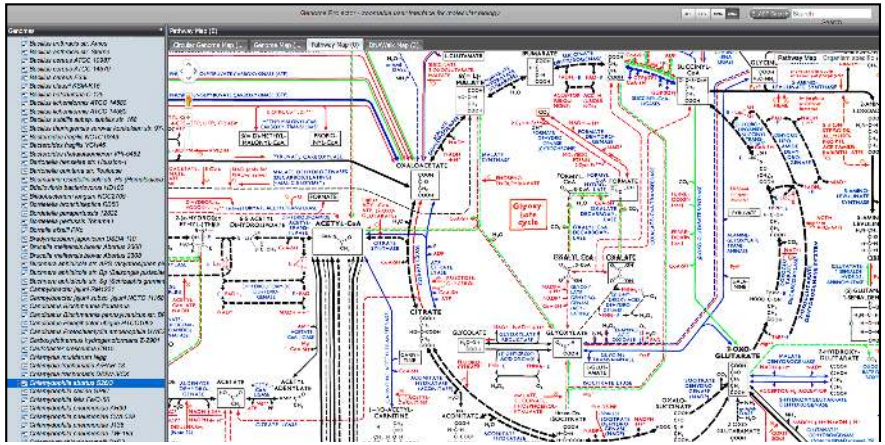
- Biological interactions !
- Graphs / networks



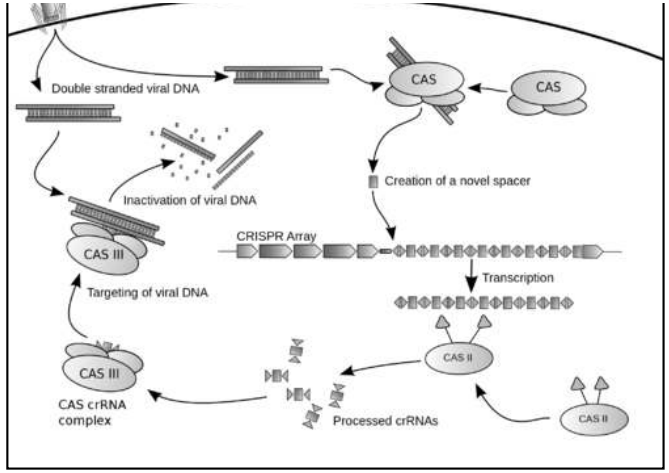
What we get…

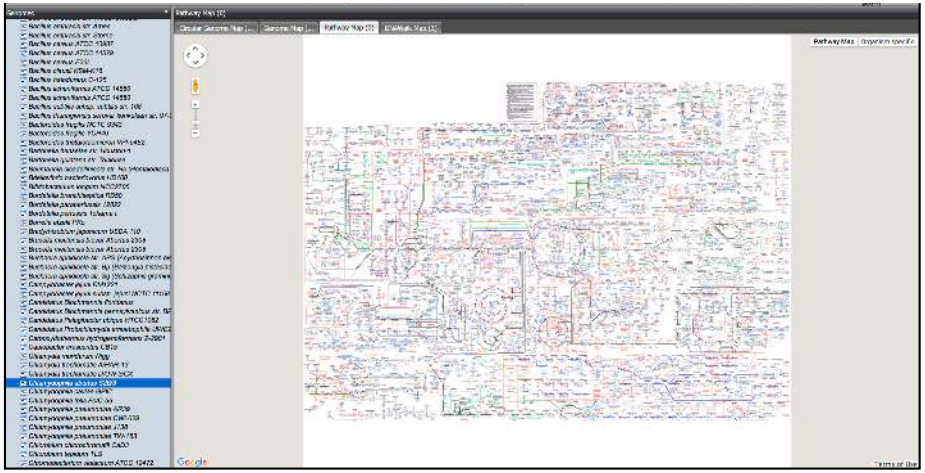
Large-scale graph description of interactions between compounds

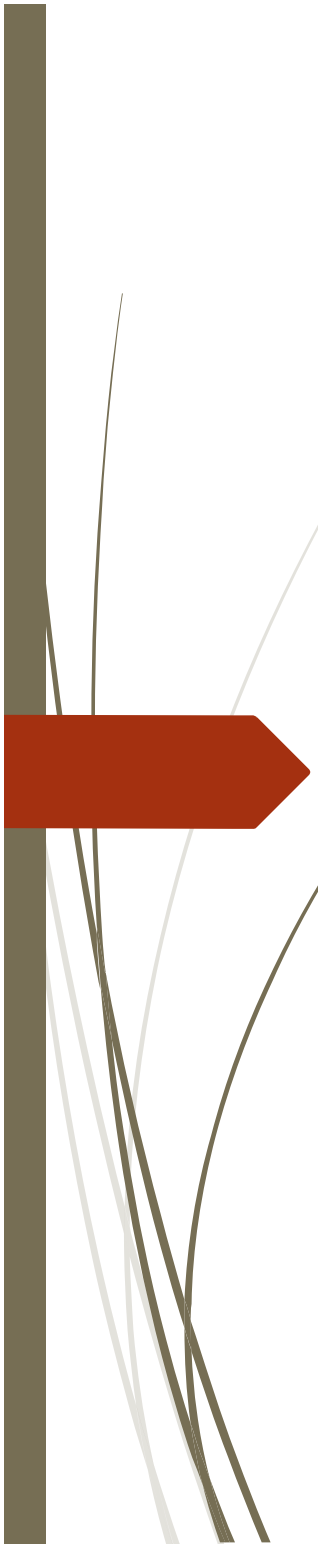




Major issue: how to turn these graphs/networks into predictions ?







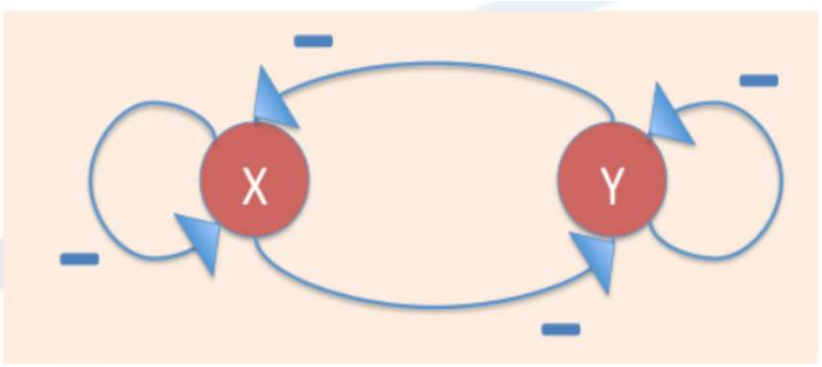
Dynamical systems… in molecular biology

Dynamical systems

Historical motivation

Model the evolution of the set of components in a system according to time.

$$F : \underbrace{\mathbb{T}}_{(\text{time}} \times \underbrace{\mathbb{S}}_{\text{state})} \rightarrow \mathbb{S} \quad (t, \mathbf{z}) \mapsto \underbrace{F(t, \mathbf{z})}_{\text{new state at time } t}$$



$$\frac{dX}{dt} = \frac{k}{K + Y^n} - aX$$
$$\frac{dY}{dt} = \frac{l}{L + X^n} - bY$$

Parameterized numerical system

$$f(X) \leftarrow 1 - Y$$
$$f(Y) \leftarrow 1 - X$$

Boolean model with asynchronous update scheme

Identification of a dynamical system

Find the **best function F** which parcimounously explains and describes the observed responses of a system.

Model identification

What has always allowed a model identification

- **A priori knowledge about the laws governing the system**
  - Predetermined shape for the function F
- **Limited number of components**
  - Reduction of the search space
- **Wide panel of sensors and perturbations**
  - Discriminate parameters

$$
\begin{array}{rccccc}
 & \mathbb{T} & \times & \mathbb{S} & \to & \mathbb{S} \\
F: & (t & , & \mathbf{z}) & \mapsto & F(t,\mathbf{z}) \\
 & \text{(time} & , & \text{state)} & & \text{new state at time } t
\end{array}
$$

Where is the complexity ?

- The search space grows exponentially with the number of measured compounds

The more compounds we measure, the less identifiable a system is.

Differences between application domains

Physical sciences

- **Knowledge.**
  Fundamental laws of physics.
- **Sensors.**
  Numerous.
- **Perturbations.**
  Various protocoles in controled frameworks.
- **System description.**
  Independent components

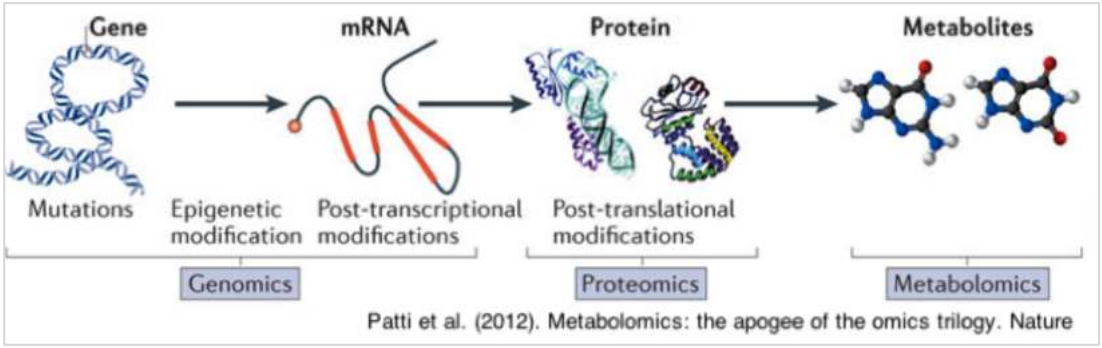
Biological sciences

- **Knowledge.**
  Empirical laws
- **Sensors.**
  Low quality although numerous.
- **Perturbations.**
  Quite few according to sensors
- **System description.**
  Hidden dependencies

Today's biological systems

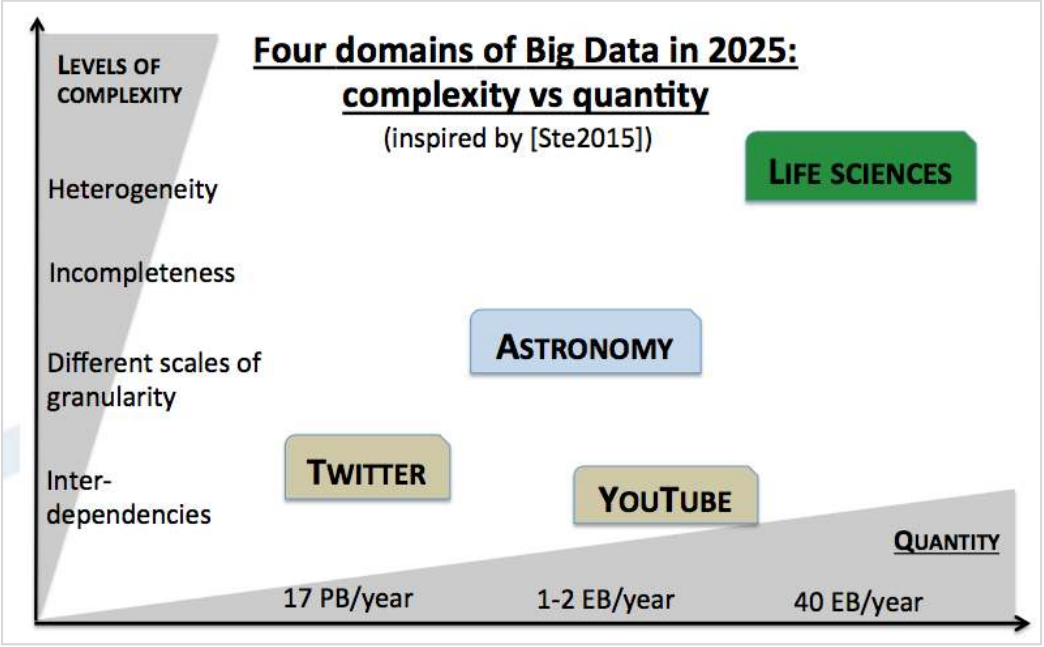
Omics data.

- Large-scale
- Noisy
- Heterogeneous.



Systems characteristics

- Large-scale
- Empirical laws
- Few data wrt the search space size



Biological systems observed with omics data are not uniquely identifiable

Strategy: combine dynamical systems and constraints programming

Describe a system by a family of abstract models

- Reason over a family of models
- instead of selecting a single one

(Logical) knowledge representation

- Search space **description**
- Structured knowledge (link open data)

Discrete dynamical systems

- L**inks** between multi-scale observations.
- **Invariants** of model families.

Solving optimisation problems

- **Replace laws by constraints**
- Extract robust information

Regulatory networks

- Siegel et al, Biosystems 2006.
- Guziolowski et al, BMC Genomics 2008,
- Baumuratova et al, BMC Systems Biology, 2010,
- Guziolowski et al, IEEE TCBB, 2010
- Thiele et al, BMC Systems Biology, 2014

Equipe MeForBio

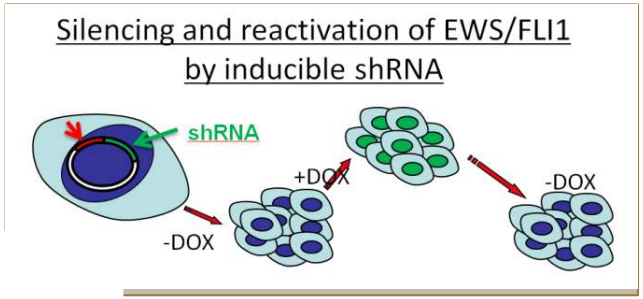
Universität Potsdam

institutCurie

Integration procedure: an example

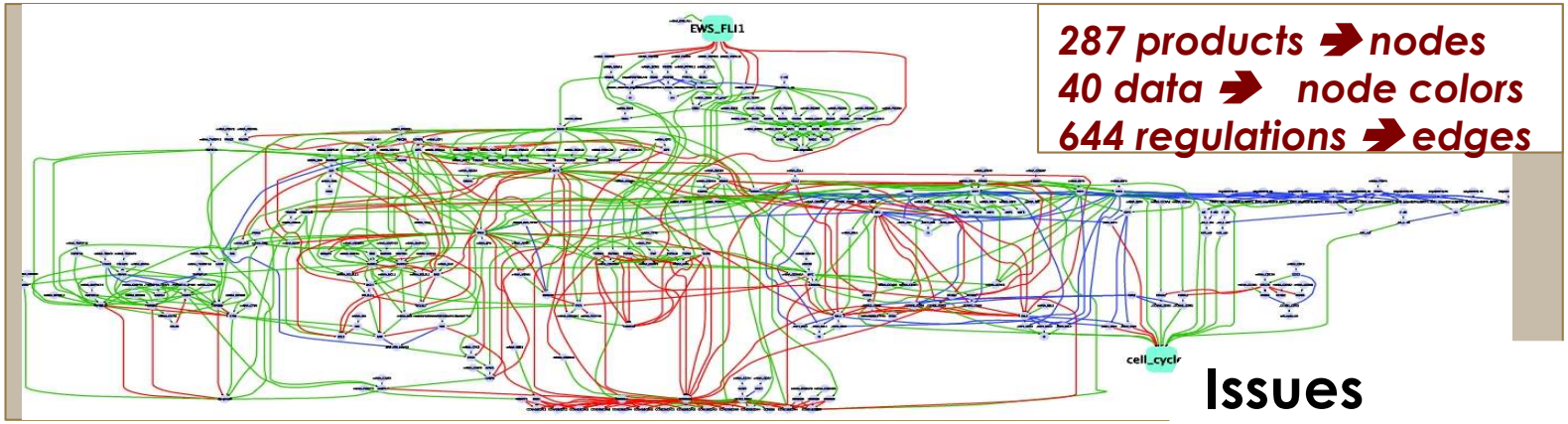
Ewing sarcoma (Curie Institute)

- Chimeric protein
- Experimental model: silencing



Available information

- Knowledge (regulation): large-scale network
- Measure (compounds): significant variations

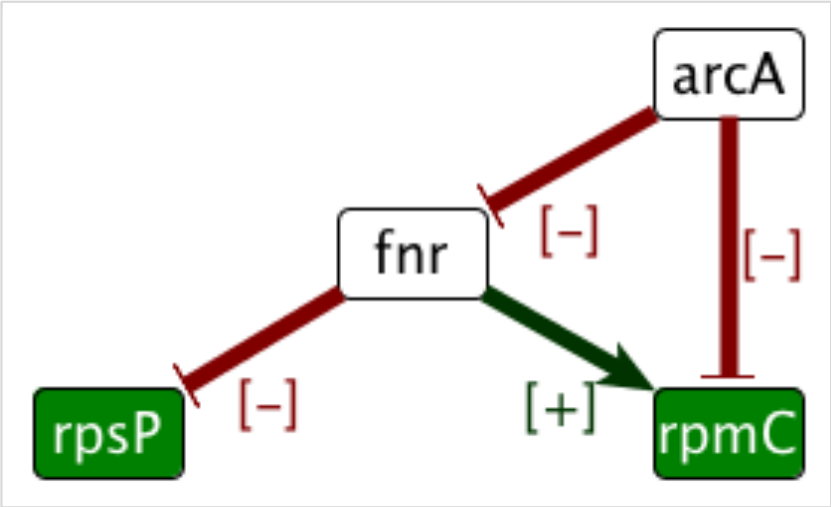


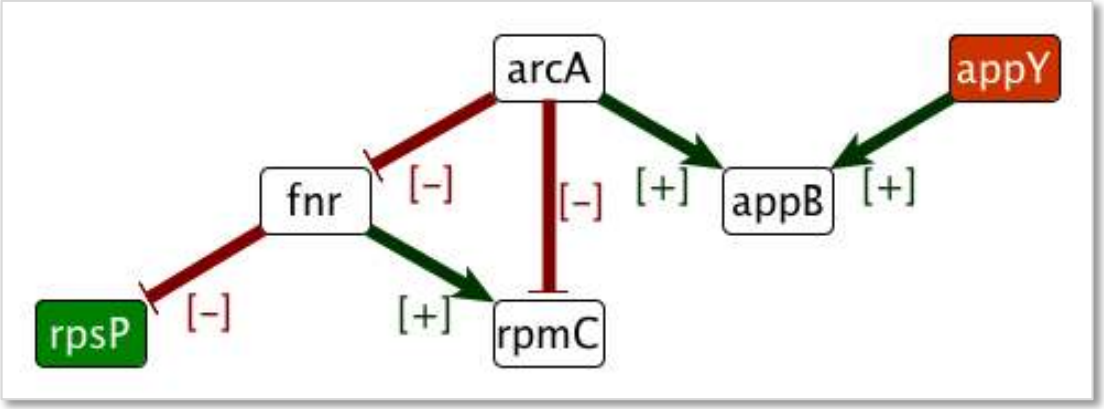
287 products ➔ nodes
40 data ➔ node colors
644 regulations ➔ edges

Issues

- Compatibility
- Predictions
- Key regulation pathways

(Implicit) reasoning…





Inconsistency

Partial predictions

What would you like to say ?

Problem reformulation with dynamical systems

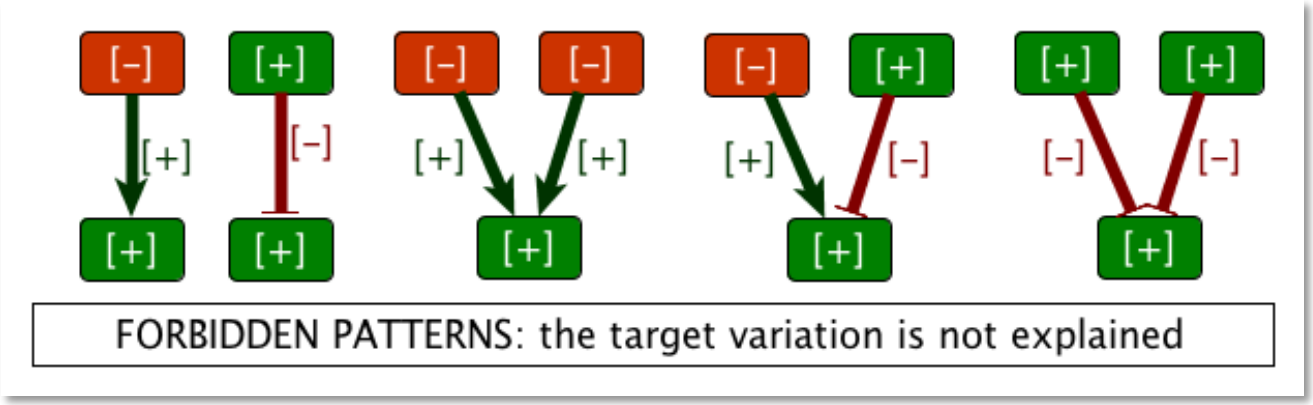
Theorem. *Assume that*

- *Specy i autoregulates itself negatively:* $\frac{\partial \mathbf{F}_{X(i)}}{\partial X(i)} < -C, \quad C > 0.$
- *There is no direct influence from* $\mathbf{P}$ *on* $X(i)$
- *When i is absent, the system produces it* $\mathbf{F}_{X(i)}(\{\mathbf{X},\ X(i) = 0\}) > 0$
- *For every predecessor $k \to i$, the sign of the action* $\frac{\partial \mathbf{F}_{X(i)}}{\partial X(k)}$ *is constant during the experimentation*

Then the variations of the species between two steady states (for different parameters) satisfy the following relationship:

$$sign(\Delta X(i)) \simeq \sum_{k \neq i,\, k \to i} sign\left(\frac{\partial \mathbf{F}_{X(i)}}{\partial X(k)}\right) \times sign(\Delta X(k)).$$

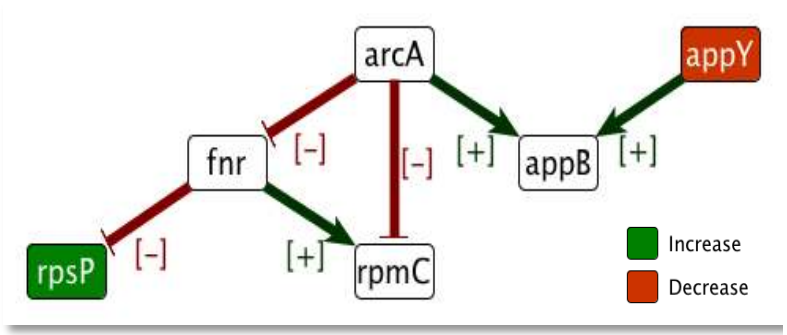
Interpretation. For each gene which does not self-induces itself, the gene expression variation must be explained by a consistent regulation from at least one predecessor.



Computational issues

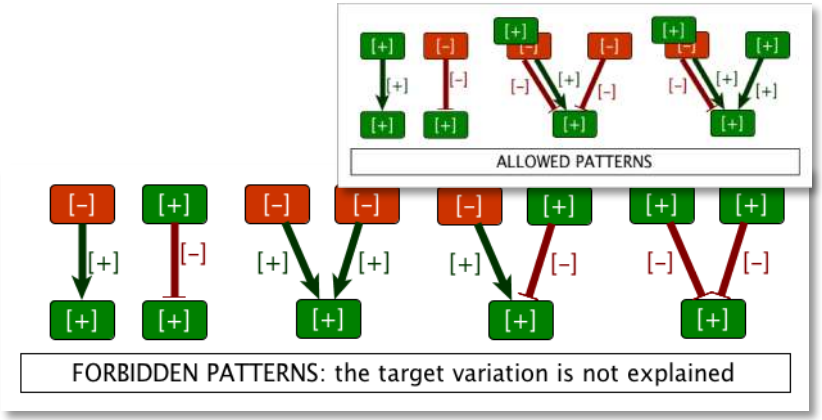
Modelling

- Interactions → oriented signed graph
- Data → (partial) colors on nodes



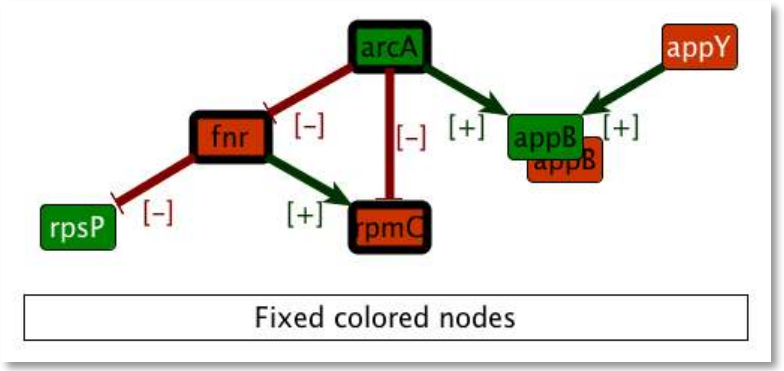
Mathematics

- Formalize forbidden patterns
- Validity of the rules



Predicting

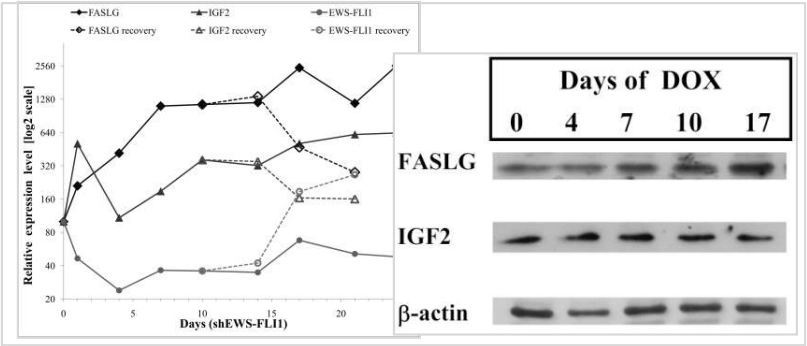
- Fill white nodes with one or two colors
- Intersect colorations
- Guide corrections



Example of application

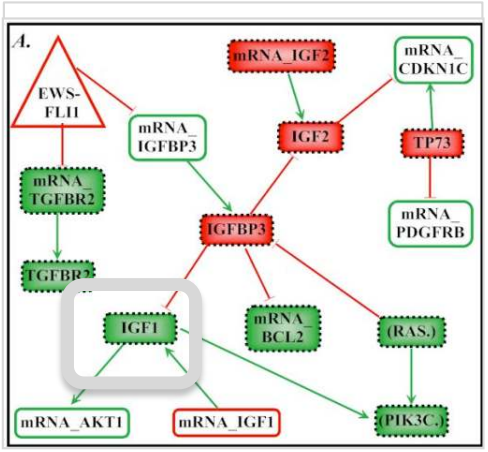
Explain and predict

- Competing effects
- Significant predictions



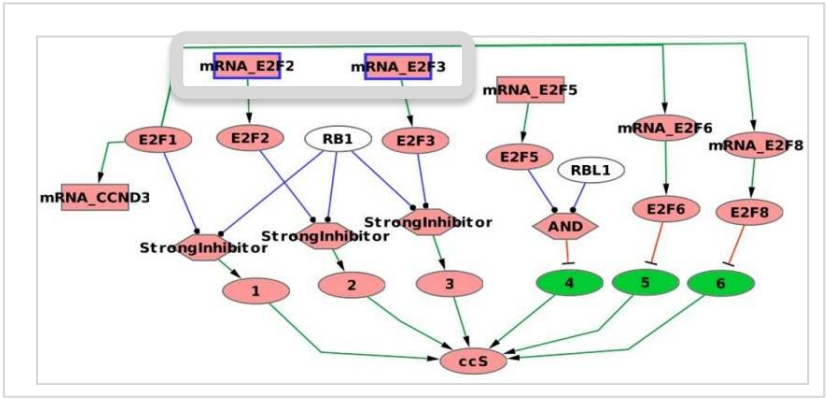
Key processes

- Incomplete IGF1- pathway



Better understanding of regulation

- Prediction of two new potential targets
- Confirmed by si-RNA



Applications

OPEN ACCESS Freely available online

PLOS | COMPUTATIONAL BIOLOGY

Detecting and Removing Inconsistencies between Experimental Data and Signaling Network Topologies Using Integer Linear Programming on Interaction Graphs

Ioannis N. Melas[1☯], Regina Samaga[2☯], Leonidas G. Alexopoulos[1], Steffen Klamt[2*]

1 National Technical University of Athens, Athens, Greece, 2 Max Planck Institute for Dynamics of Complex Technical Systems, Magdeburg, Germany

Thiele *et al. BMC Bioinformatics* (2015) 16:345
DOI 10.1186/s12859-015-0733-7

BMC Bioinformatics

METHODOLOGY ARTICLE Open Access

Extended notions of sign consistency to relate experimental data to signaling and regulatory network topologies

SCIENTIFIC REPORTS

OPEN

Logic programming reveals alteration of key transcription factors in multiple myeloma

Received: 29 November 2016
Accepted: 25 July 2017
Published online: 23 August 2017

Bertrand Miannay[1,2], Stéphane Minvielle[2,3], Olivier Roux[1], Pierre Drouin[1], Hervé Avet-Loiseau[4], Catherine Guérin-Charbonnel[2,5], Wilfried Gouraud[2,5], Michel Attal[6], Thierry Facon[7], Nikhil C Munshi[8,9], Philippe Moreau[2,3], Loïc Campion[2,5], Florence Magrangeas[2,3] & Carito Guziolowski[1]

Klamt's group & Guziolowski's group

→ Discriminate patient responses from a large-scale interaction graph

Link with the general scheme

Describe a system by a family of abstract models

Discrete dynamical systems

- Deduce contraints for propagations

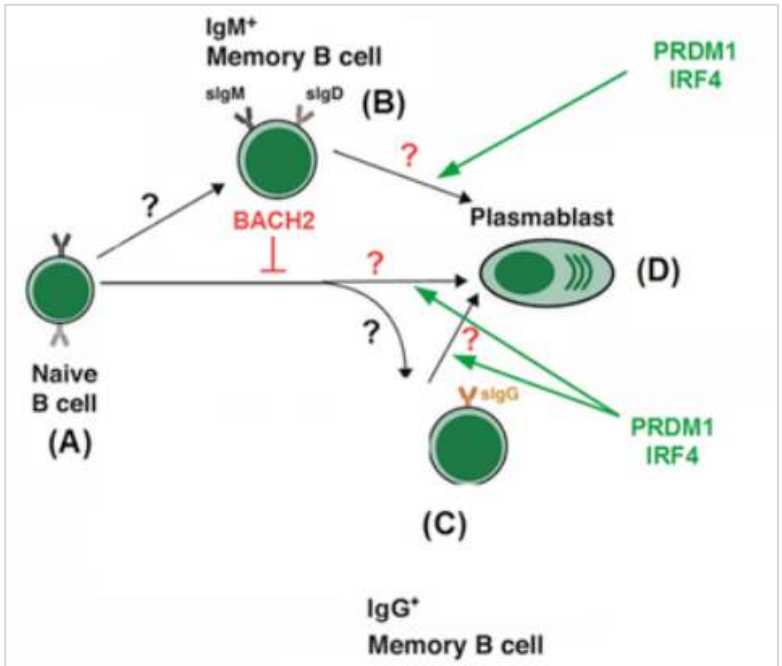
(Logical) knowledge representation

- Introduce several reasoning rules according to biological reactions.

Solving optimisation problems

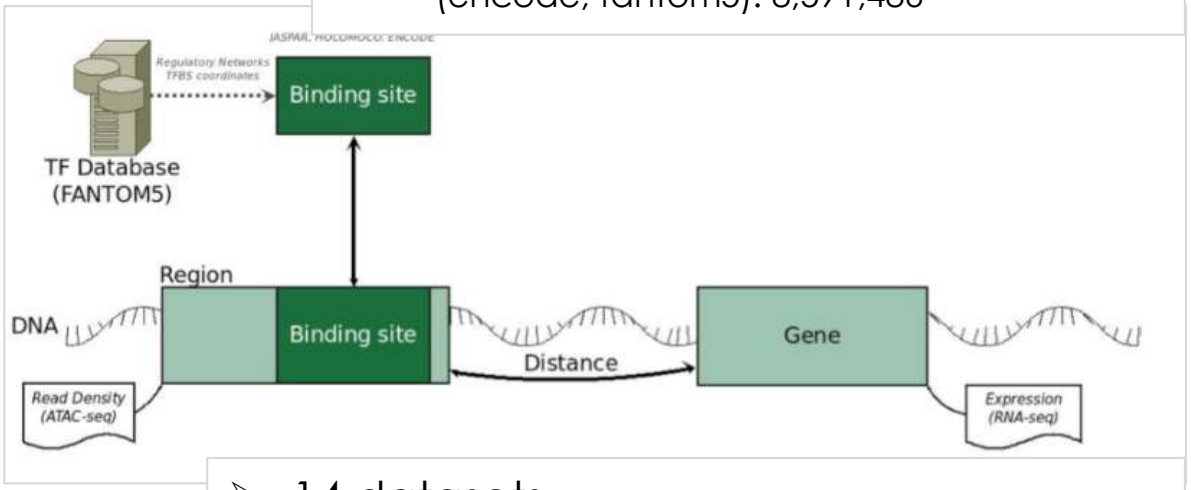
- Use relevant searching methods
- Make the union of all solutions !

Work in progress: consistency rules for epignetics



- B-cells differentiate into 4 cell types
- Memory B cells are useful for vaccines

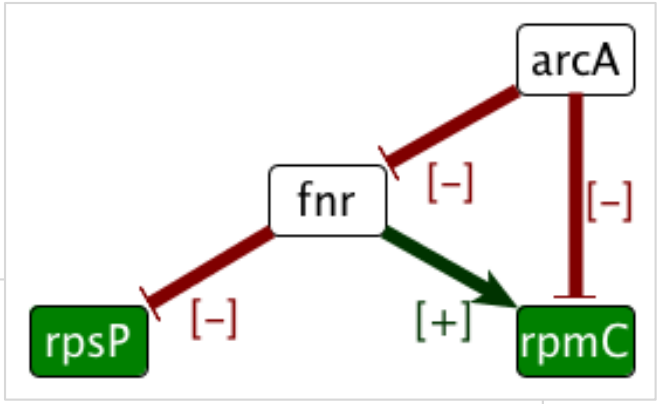
- **Genetics** (RNA-seq): 29,261
- **Epigenetics**
  - ATAC-seq: 35,078
  - 5hmC: 88,446
- **Background knowledge**
  - TF databases (Hocomoco, Jaspar): 593 TFBS
  - TF location and regulation network (encode, fantom5): 8,591,486



- 14 datasets
- 9,116,795 entities
- **58,140,252 relations (triples in rdf)**

Issue: identify regulatory candidates for pattern response consistently with RNA-seq and epigenetics ?

Magic behind...

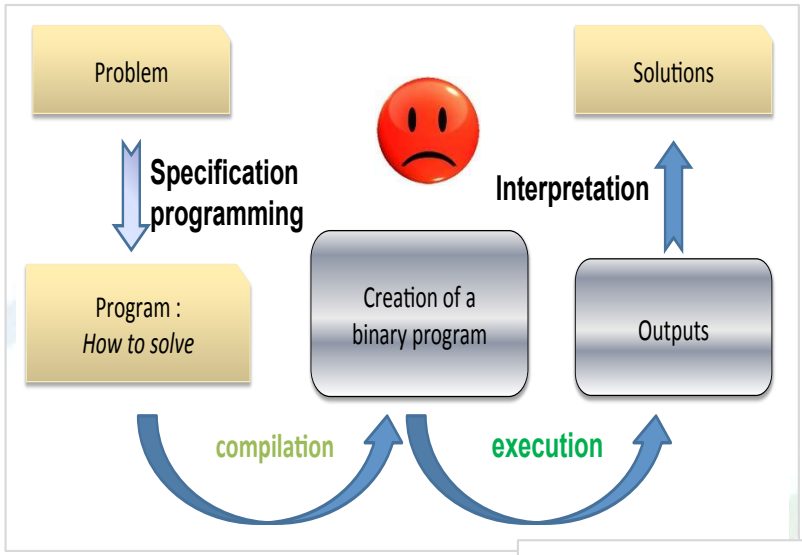



```
vertex(rpsP). observedV(rpsP,-).
vertex(rpmC). observedV(rpmC,-).
vertex(fnr).
vertex (arcA).
edge(fnr,rpsP). observedE(fnr,rpsP,-).
edge(fnr,rpmC). observedE(fnr,rpmC,+).
edge(arcA,fnr). observedE(arcA,fnr,-).
edge(arcA,rpmC). observedE(arcA,rpmC,-).


1{labelV(I,+) ; labelV (I,-)}1 :- vertex(I).
labelV(I,S) :- observedV(I,S).
1{labelE(J,I,+) ; labelE (J,I,-)}1 :- edge(J,I).
labelE(J,I,S) :- observedE(J,I,S).

receive(I,+) :- labelE(J,I,S), labelV(J,S).
receive(I,-) :- labelE(J,I,S), labelV(J,T), S!=T.

:- labelV (I,S), not receive(I,S).
```

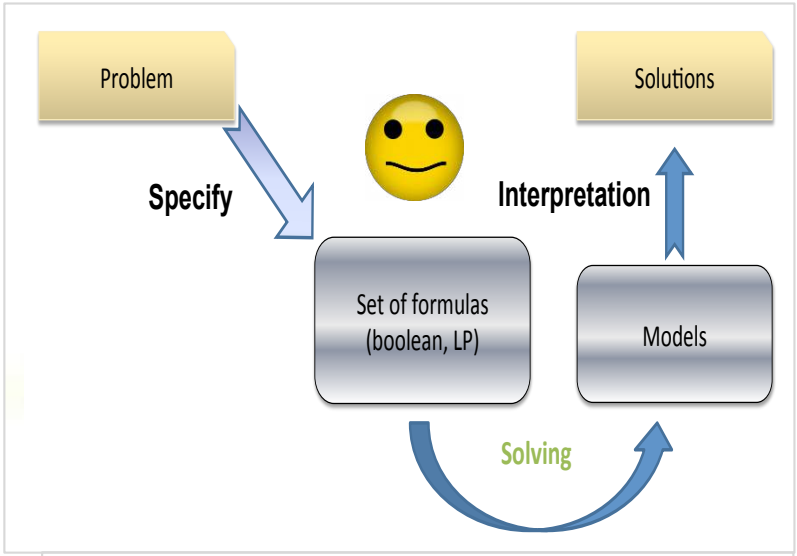
Knowledge representation

```
1{murderer(ms_Scarlet); murderer(colonel_Mustard)}1.
1{weapon_of_crime(revolver); weapon_of_crime(candlestick)}1.
1{place_of_crime(kitchen); place_of_crime(hall);
                                place_of_crime(dining_room)}1.

crim_record(ms_Scarlet,7). crim_record(colonel_Mustard,4).

weapon_of_crime(candlestick).
:- place_of_crime(kitchen).
place_of_crime(hall) :- murderer(colonel_Mustard), not
                                        weapon_of_crime(revolver).

sol(X,Y,Z) :- murderer(X),weapon_of_crime(Y),place_of_crime(Z).

#maximize{W , sol : sol(X,Y,Z) , crim_record(X,W) , murdered(W)}.

#show sol/3.
```

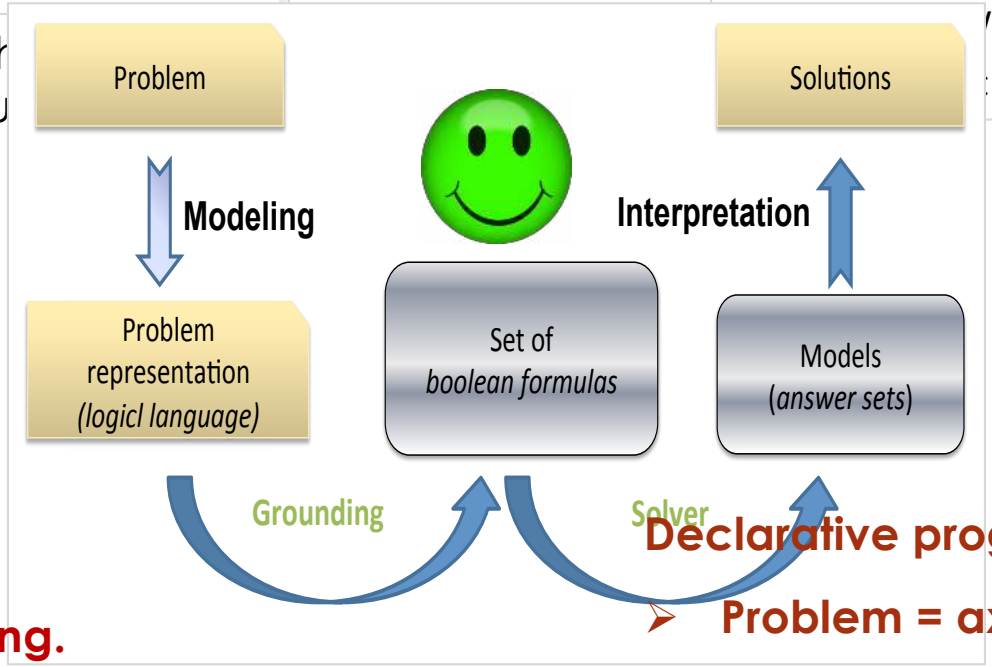
Solving combinatorial problems



Write a program which
how the problem shou



rite (boolean, linear)
onstraints (*SAT, ILP,...*)

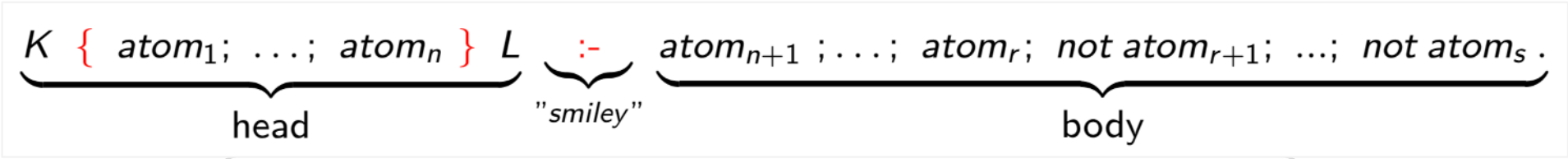


Answer set programming.
Describe what you want to solve

Declarative programing

- **Problem = axioms & rules**
- **No need of algorithm**

ASP logical rules : declarative programming



If **all terms** on the **right side** are true,
then **at least K and at most L** terms are true on the **left side**.

If **nothing** on the **left side**,
then **always false.**

```
:- K{atom1, .. atomN}L.
```

If **nothing** on the **right side**,
then **always true.**

```
K{atom1, .. atomN}L.
```

Optimisation rule

```
#maximize{W,atom(X): condition(X),W}.
```

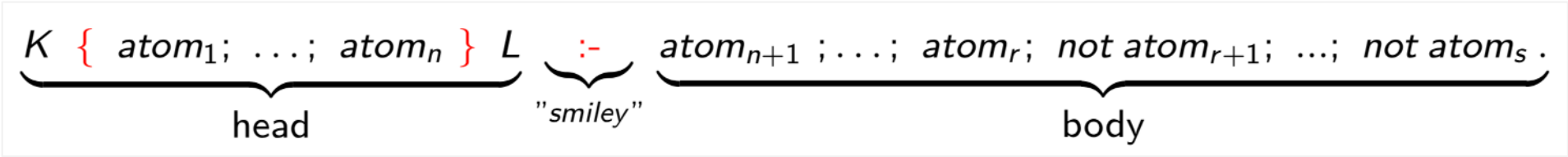
High-level model language

- Propositional logics
- Model for negation

Highly performant solving technics

- SAT-based and deductive-DB technics
- Decidable: no infinite loop

ASP logical rules




```
1{murderer(ms_Scarlet); murderer(colonel_Mustard)}1.
1{weapon_of_crime(revolver); weapon_of_crime(candlestick)}1.
1{place_of_crime(kitchen); place_of_crime(hall);
                           place_of_crime(dining_room)}1.

crim_record(ms_Scarlet,7). crim_record(colonel_Mustard,4).

weapon_of_crime(candlestick).
:- place_of_crime(kitchen).
place_of_crime(hall) :- murderer(colonel_Mustard), not
                                   weapon_of_crime(revolver).

sol(X,Y,Z) :- murderer(X),weapon_of_crime(Y),place_of_crime(Z).


#show sol/3.
```

How many solutions ?

ASP logical rules

$$\underbrace{K\ \{\ atom_1;\ \ldots;\ atom_n\ \}\ L}_{\text{head}}\ \underbrace{:-}_{\text{"smiley"}}\ \underbrace{atom_{n+1}\ ;\ldots;\ atom_r;\ not\ atom_{r+1};\ \ldots;\ not\ atom_s\ .}_{\text{body}}$$

Declare what you know

```
1{murderer(ms_Scarlet); murderer(colonel_Mustard)}1.
1{weapon_of_crime(revolver); weapon_of_crime(candlestick)}1.
1{place_of_crime(kitchen); place_of_crime(hall);
                           place_of_crime(dining_room)}1.

crim_record(ms_Scarlet,7). crim_record(colonel_Mustard,4).
```

What you have deduced up to know

```
weapon_of_crime(candlestick).
:- place_of_crime(kitchen).
place_of_crime(hall) :- murderer(colonel_Mustard), not
                                    weapon_of_crime(revolver).
```

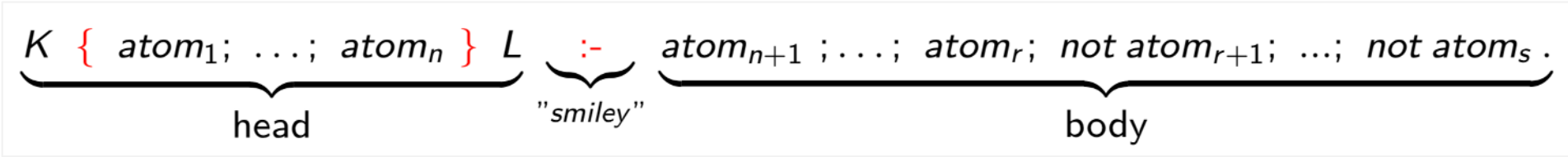
```
sol(X,Y,Z) :- murderer(X),weapon_of_crime(Y),place_of_crime(Z).
```

What look for

```
#show sol/3.
```

How many solutions ?

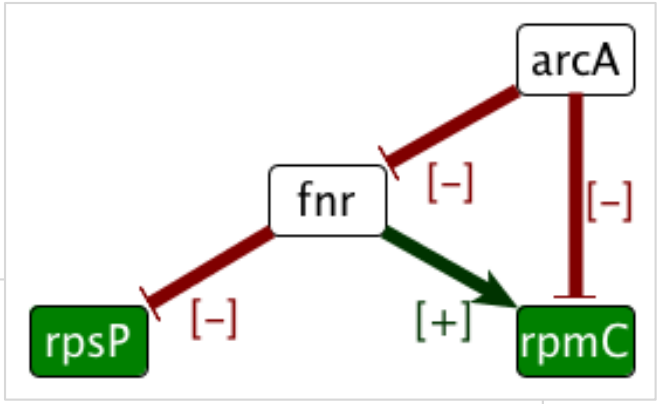
ASP logical rules




```
1{murderer(ms_Scarlet); murderer(colonel_Mustard)}1.
1{weapon_of_crime(revolver); weapon_of_crime(candlestick)}1.
1{place_of_crime(kitchen); place_of_crime(hall);
                           place_of_crime(dining_room)}1.

crim_record(ms_Scarlet,7). crim_record(colonel_Mustard,4).

weapon_of_crime(candlestick).
:- place_of_crime(kitchen).
place_of_crime(hall) :- murderer(colonel_Mustard), not
                                   weapon_of_crime(revolver).

sol(X,Y,Z) :- murderer(X),weapon_of_crime(Y),place_of_crime(Z).

#maximize{W , sol : sol(X,Y,Z) , crim_record(X,W) , murdered(W)}

#show sol/3.
```

Introduce optimization rules

Magic behind…




```
vertex(rpsP). observedV(rpsP,-).
vertex(rpmC). observedV(rpmC,-).
vertex(fnr).
vertex (arcA).
edge(fnr,rpsP). observedE(fnr,rpsP,-).
edge(fnr,rpmC). observedE(fnr,rpmC,+).
edge(arcA,fnr). observedE(arcA,fnr,-).
edge(arcA,rpmC). observedE(arcA,rpmC,-).
```

(1) Declare nodes, edges and signs

```
1{labelV(I,+) ; labelV (I,-)}1 :- vertex(I).
labelV(I,S) :- observedV(I,S).
1{labelE(J,I,+) ; labelE (J,I,-)}1 :- edge(J,I).
labelE(J,I,S) :- observedE(J,I,S).
```

(2) Associate a predicted sign to each node and edge

```
receive(I,+) :- labelE(J,I,S), labelV(J,S).
receive(I,-) :- labelE(J,I,S), labelV(J,T), S!=T.
```

(3) Propagate information

```
:- labelV (I,S), not receive(I,S).
```

(4) Ensure that propagation is consistent with prediction

(….) Add optimization predicate to repair the network

Link with systems biology ?

Integrative/systems biology is a very relevant field to challenge ASP technologies

- Repair large-scale interaction graph with **branch and bound** solving heuristics (KR 2010)
- Scale metabolic network completion problem with **unsatisfiable core** solving strategy (LPNMR 2013)
- Design experiments with **incremental solving** (Frontiers 2015)
- Implement and benchmark **constraints propagators** (TPLP 2018)

Problem statement & modelling

Solving heuristics & problem reformulation

Linear constrains atoms

```
&sum{a1*x1;...;al*xl} <= k
```

What is ASP: langage synthax and principles

- Written with the help of J. Nicolas, INRIA Rennes

Description rapide

Règles disjonctives

$$\underbrace{K\ \{\ atom_1;\ \ldots;\ atom_n\ \}\ L}_{\text{tête}}\ \underbrace{\text{:-}}_{\text{"smiley"}}\ \underbrace{atom_{n+1}\ ;\ldots;\ atom_r;\ not\ atom_{r+1};\ \ldots;\ not\ atom_s\ .}_{\text{corps}}$$

$$\underbrace{K\ \{\ atom_1;\ \ldots;\ atom_n\ \}\ L}_{\text{head}}\ \underbrace{\text{:-}}_{\text{"smiley"}}\ \underbrace{atom_{n+1}\ ;\ldots;\ atom_r;\ not\ atom_{r+1};\ \ldots;\ not\ atom_s\ .}_{\text{body}}$$

- $fact_1, \ldots, fact_n$ sont des *atomes*
- Règle de déduction

  Si tous les faits apparaissant dans le corps sont vrais,
  alors la tête contient entre K et L faits qui sont vérifiés.

- Contrainte d'intégrité. S'il n'y a pas de tête: le corps est *toujours faux*

  :- $fact_0$.

- Faits vrais. S'il n'y a pas de corps, la tête est *toujours vraie*

  $fact_0$.

Les *Ensembles-réponses* sont tous les modèles valides

- Ce sont des ensembles d'atomes qui vérifient toutes les règles.
- Tout atome d'un modèle valide apparait dans la tête d'au moins une règle.

Who killed Mr Boddy ?

Program

```
1{murderer(ms_Scarlet);murderer(colonel_Mustard)}1.
1{weapon_of_crime(revolver);weapon_of_crime(candlestick)}1.
1{place_of_crime(kitchen);place_of_crime(hall);place_of_crime(dining-room)}1.

:- place_of_crime(kitchen).
place_of_crime(hall) :- murderer(colonel_Mustard);
                                        not weapon_of_crime(revolver).
weapon_of_crime(candlestick).

#show murderer/1.
#show place_of_crime/1.
#show weapon_of_crime/1.
```

Answer Sets? **Exercice**

Who killed Mr Boddy ?

Program

```
% Specify that there is only one murder
1{murderer(ms_Scarlet);murderer(colonel_Mustard)}1.
1{weapon_of_crime(revolver);weapon_of_crime(candlestick)}1.
1{place_of_crime(kitchen);place_of_crime(hall);place_of_crime(dining-room)}1.

% Declare what you can deduce from your cards
:- place_of_crime(kitchen).
place_of_crime(hall) :- murderer(colonel_Mustard);
                                        not weapon_of_crime(revolver).
weapon_of_crime(candlestick).

% Enumerate the solutions
#show murderer/1.
#show place_of_crime/1.
#show weapon_of_crime/1.
```

Answer Sets? **Exercice**

Who killed Mr Boddy ?

Program

```
% Specify that there is only one murder
1{murderer(ms_Scarlet);murderer(colonel_Mustard)}1.
1{weapon_of_crime(revolver);weapon_of_crime(candlestick)}1.
1{place_of_crime(kitchen);place_of_crime(hall);place_of_crime(dining-room)}1.
% Declare what you can deduce from your cards
:- place_of_crime(kitchen).
place_of_crime(hall) :- murderer(colonel_Mustard);
                                        not weapon_of_crime(revolver).
weapon_of_crime(candlestick).
% Enumerate the solutions
#show murderer/1.
#show place_of_crime/1.
#show weapon_of_crime/1.
```

Answer Sets? **Exercice**

```
Solving...
Answer:  1
weapon_of_crime(candlestick) murderer(colonel_Mustard) place_of_crime(hall)
Answer:  2
weapon_of_crime(candlestick) murderer(ms_Scarlet) place_of_crime(dining_room)
Answer:  3
weapon_of_crime(candlestick) murderer(ms_Scarlet) place_of_crime(hall)
SATISFIABLE
```

ASP: Principe généraux à retenir

Approche déclarative pour la résolution de problèmes combinatoires de complexité NP

"*What* instead of *How* ?"

Approche déclarative

Modélisation du problème à résoudre sous formes d'axiomes et de contraintes exprimées dans un langage logique.

Pourquoi ce nom ?

Les modèles logiques solutions de l'ensemble de formules, les ensembles réponses, sont les résultats du programme.

Principe

Des solveurs associés effectuent la recherche d'une, de plusieurs, ou de l'ensemble des solutions.

Un peu plus précisément

Langage de modélisation à haut-niveau

expressivité: ASP $\simeq$ Prolog

- Logique propositionnelle

  $\rightarrow$ *On ne peut travailler que sur un ensemble fini d'atomes.*

- Négation : formalisme élégant (monde clos).

  $\rightarrow$ *Un prédicat est faux tant qu'aucune indication ne permet de dire qu'il est vrai.*

Capacité de résolution de problèmes très élevée

ASP $\simeq$ SAT, ILP

- Décidable techniques de résolution SAT & BDD déductives

  $\rightarrow$ *Pas de réécriture de programmes*

  $\rightarrow$ L'ordre des clauses n'a pas d'impact

  $\rightarrow$ Pas de boucle infinie dans la résolution

- Optimisation, raisonnement flexible (préférences), solveurs hybrides.

Une spécificité très utile en biologie moléculaire et bioinformatique

Différents modes de raisonnements

Déduire différents types d'information biologique

- Enumération des solutions à un problème (`./clingo cluedo.lp -n 0`)

  ???

- Intersection (cautious reasoning) des atomes présents dans toutes les solutions (`./clingo cluedo.lp --enum-mode=cautious`)

  ???

- Union (brave reasoning) des atomes présents dans toutes les solutions (`./clingo cluedo.lp --enum-mode=brave`)

  ???

Une spécificité très utile en biologie moléculaire et bioinformatique

Différents modes de raisonnements

Déduire différents types d'information biologique

- Enumération des solutions à un problème (`./clingo cluedo.lp -n 0`)

  → Variabilité et taille de l'espace des modèles possibles

- Intersection (cautious reasoning) des atomes présents dans toutes les solutions (`./clingo cluedo.lp --enum-mode=cautious`)

  → Déductions robustes des données et connaissances

- Union (brave reasoning) des atomes présents dans toutes les solutions (`./clingo cluedo.lp --enum-mode=brave`)

  → Espace des déductions possibles

Syntaxe: Termes

CONSTANTE : entier, mot de {a-z, A-Z, 0-9, _ }*

→ débute par une MINUSCULE. `porquerolles` `jean-Paul_Comet`

VARIABLE : mot de {a-z, A-Z, 0-9, _ }*

→ débute par une MAJUSCULE. `Lieu` `Organisateur`

FONCTION

- de la forme *constante(terme, ..., terme)*
- inclut les opérateurs : `a+b`
- inclut la notation de liste. `L=[a,b,c]=[a|U], U =[b,c]`

Question: `Cout(Repas(Participants,Jour),nuits(participants,jour))*5` est-il valide ?

Réponse: `cout(repas(Participants,Jour),nuits(Participants,Jour))*5`

LITTERAL

- Négation `not q`
- Calculs `X<Y` `U!=V` `X+Y<5`

Syntaxe: Termes

Condition

TailleMin { *atome*1 : *domaine_validite*1 ; *atome*2 : *domaine_validite*2 } *TailleMax*

- On construit l'ensemble des atomes1 et atomes 2 qui vérifient toutes les conditions de validité.
- On isole tous les sous-ensembles de taille qui contiennent entre *TailleMin* et *TailleMax* atomes valides.

```
25{chambreSimple(X):particip(X) ; chambreDouble(Y,Z):particip(Y),particip(Z)}30
```

→ Sous-ensembles de 25 à 30 chambres simples ou doubles qui hébergent des participants.

```
5 { chambreSimple(X) : participant(X), dureeSejour(X)=DureeMax } 8
```

→ Sous-ensembles de 5 à 8 chambres simples qui hébergent des participants dont la durée du séjour est égale à la constante DureeMax.

Exemple

Variables

```
% Declare what you are playing with

place(kitchen; hall; diningroom).
weapon(revolver; candlestick).
character(colonel_Mustard;ms_Scarlet).


% Specify that there is only one murder
1{murderer(X):character(X)}1.


1{weapon_of_crime(X):weapon(X)}1.


1{place_of_crime(X):place(X)}1.
```

Exemple

Variables

```
% Declare what you are playing with

place(kitchen; hall; diningroom).
weapon(revolver; candlestick).
character(colonel_Mustard;ms_Scarlet).


% Specify that there is only one murder
1{murderer(X):character(X)}1.
```

→ Le grounder (`gringo cluedo.lp -t`) va instancier cette clause en `1{murderer(ms_Scarlet) ; murderer(colonel_Mustard)}1.`

`1{weapon_of_crime(X):weapon(X)}1.`

→ grounding: `1{weapon_of_crime(revolver) ; weapon_of_crime(candlestick)}1.`

`1{place_of_crime(X):place(X)}1.`

→ grounding:

`1{place_of_crime(kitchen);place_of_crime(hall);place_of_crime(dining-room)}1.`

Syntaxe: Termes

```
1{bouchanourrirs(X):perso(X)}2
```

???????

```
2{r(U):q(U)=U ; s(V):t(V)=2}
```

???????

Syntaxe: Termes

```
1{bouchanourrirs(X):perso(X)}2
```

$\rightarrow$ sous-ensemble de taille 1 à 2 de l'ensemble des atomes *bouchanourrirs*(X) qui sont vrais, où X est une variable dans le domaine défini par le prédicat *perso*.

```
2{r(U):q(U)=U ; s(V):t(V)=2}
```

Syntaxe: Termes

```
1{bouchanourrirs(X):perso(X)}2
```

$\rightarrow$ sous-ensemble de taille 1 à 2 de l'ensemble des atomes *bouchanourrirs*(X) qui sont vrais, où X est une variable dans le domaine défini par le prédicat *perso*.

Sans indication supplémentaire sur les prédicats `bouchanourrirs` et `perso`, ce terme est strictement équivalent à

```
1{r(X):q(X)}2
```

```
2{r(U):q(U)=U ; s(V):t(V)=2}
```

$\rightarrow$ Sous-ensemble de taille au moins 2 de l'ensemble composé

- des $r(U)$ vrais pour lesquels U est tel que $q(U) = U$,
- ET de l'ensemble des $s(V)$ vrais pour lesquels V est tel que $t(V) = 2$.

Syntaxe: Clauses

FAITS `p(2,a).`

→ p(2,a) est vrai.

Syntaxe: Clauses

FAITS `p(2,a).`

→ p(2,a) est vrai.

RÈGLE `q(X) :- p(X,Y) ; q(Y) .`

→ Si p(X,Y) est vrai et q(Y) est vrai, alors q(X) est vrai aussi.

Syntaxe: Clauses

Faits `p(2,a).`

→ p(2,a) est vrai.

Règle `q(X) :- p(X,Y) ; q(Y) .`

→ Si p(X,Y) est vrai et q(Y) est vrai, alors q(X) est vrai aussi.

Contrainte `:- q(2) .`

→ q(2) est faux.

`:- p(X) ; q(X) .`

→ Le prédicat "p(X) et q(X)" est faux.

Syntaxe: Clauses

Faits `p(2,a).`

→ p(2,a) est vrai.

Règle `q(X) :- p(X,Y) ; q(Y) .`

→ Si p(X,Y) est vrai et q(Y) est vrai, alors q(X) est vrai aussi.

Contrainte `:- q(2) .`

→ q(2) est faux.

`:- p(X) ; q(X) .`

→ Le prédicat "p(X) et q(X)" est faux.

Exercice

Question : que veut dire ?

```
:- {proprieteLambda(U+1):conditionQuelconque(U)} M ; domaineValidite(M).
```

Exercice

Question : que veut dire ?

```
:- {proprieteLambda(U+1):conditionQuelconque(U)} M ; domaineValidite(M).
```

- **Il est faux d'avoir** :
  si M qui vérifie le prédicat `domaineValidite`,
  il existe un ensemble contenant au plus M atomes vrais de la forme `proprieteLambda(U+1)`,
  où U vérifie le prédicat `conditionQuelconque`.

Exercice

Question : que veut dire ?

```
:- {proprieteLambda(U+1):conditionQuelconque(U)} M ; domaineValidite(M).
```

- **Il est faux d'avoir** :
  si M qui vérifie le prédicat `domaineValidite`,
  il existe un ensemble contenant au plus M atomes vrais de la forme `proprieteLambda(U+1)`,
  où U vérifie le prédicat `conditionQuelconque`.

- **Autre formulation**:
  Pour tout M qui vérifie `domaineValidite`,
  le prédicat `proprieteLambda(U+1)` est vrai au moins $M + 1$ fois
  lorsqu'on parcours les U qui vérifient `conditionQuelconque`.

Exercice

Question : que veut dire ?

```
:- {proprieteLambda(U+1):conditionQuelconque(U)} M ; domaineValidite(M).
```

- **Il est faux d'avoir** :
  si M qui vérifie le prédicat `domaineValidite`,
  il existe un ensemble contenant au plus M atomes vrais de la forme `proprieteLambda(U+1)`,
  où U vérifie le prédicat `conditionQuelconque`.

- **Autre formulation**:
  Pour tout M qui vérifie `domaineValidite`,
  le prédicat `proprieteLambda(U+1)` est vrai au moins $M + 1$ fois
  lorsqu'on parcours les U qui vérifient `conditionQuelconque`.

- **Encore une autre formulation**:
  Pour tout M qui vérifie `domaineValidite`,
  il y a au moins $M + 1$ valeurs de U qui vérifie `conditionQuelconque` et qui sont telles que `proprieteLambda(U+1)` est vraie.

Exercice

Question : que veut dire ?

```
:- {proprieteLambda(U+1):conditionQuelconque(U)} M ; domaineValidite(M).
```

- **Il est faux d'avoir** :
  si M qui vérifie le prédicat `domaineValidite`,
  il existe un ensemble contenant au plus M atomes vrais de la forme `proprieteLambda(U+1)`,
  où U vérifie le prédicat `conditionQuelconque`.

- **Autre formulation**:
  Pour tout M qui vérifie `domaineValidite`,
  le prédicat `proprieteLambda(U+1)` est vrai au moins $M + 1$ fois
  lorsqu'on parcours les U qui vérifient `conditionQuelconque`.

- **Encore une autre formulation**:
  Pour tout M qui vérifie `domaineValidite`,
  il y a au moins $M + 1$ valeurs de U qui vérifie `conditionQuelconque` et qui sont telles que `proprieteLambda(U+1)` est vraie.

Nouvelle question : que veut dire ?

```
:- {r(U+1):q(U)}M ; p(M).
```

Exercice 2

Question : que veut dire ?

```
:- not {proprieteLambda(U+1):conditionQuelconque(U)}M ; domaineValidite(M).
```

Exercice 2

Question : que veut dire ?

```
:- not {proprieteLambda(U+1):conditionQuelconque(U)}M ; domaineValidite(M).
```

- **Il est faux d'avoir** :
  si M vérifie le prédicat `domaineValidite`,
  il n'existe pas d'ensemble contenant au plus M atomes vrais de la forme `proprieteLambda(U+1)`,
  où U vérifie le prédicat `conditionQuelconque`,

Exercice 2

Question : que veut dire ?

```
:- not {proprieteLambda(U+1):conditionQuelconque(U)}M ; domaineValidite(M).
```

- **Il est faux d'avoir** :
  si M vérifie le prédicat `domaineValidite`,
  il n'existe pas d'ensemble contenant au plus M atomes vrais de la forme `proprieteLambda(U+1)`,
  où U vérifie le prédicat `conditionQuelconque`,

- **Autre formulation**:
  si M vérifie `domaineValidite`,
  le prédicat `proprieteLambda(U+1)` est vrai au plus M fois
  lorsqu'on parcours les U qui vérifient `conditionQuelconque`.

Exercice 2

Question : que veut dire ?

```
:- not {proprieteLambda(U+1):conditionQuelconque(U)}M ; domaineValidite(M).
```

- **Il est faux d'avoir** :
  si M vérifie le prédicat `domaineValidite`,
  il n'existe pas d'ensemble contenant au plus M atomes vrais de la forme `proprieteLambda(U+1)`,
  où U vérifie le prédicat `conditionQuelconque`,

- **Autre formulation**:
  si M vérifie `domaineValidite`,
  le prédicat `proprieteLambda(U+1)` est vrai au plus M fois
  lorsqu'on parcours les U qui vérifient `conditionQuelconque`.

- **Encore une autre formulation**:
  Pour tout M qui vérifie `domaineValidite`,
  il y a au plus M valeurs de U qui vérifie `conditionQuelconque` et qui sont telles que `proprieteLambda(U+1)` est vraie.

Exemple optimisation

```
place(kitchen; hall; diningroom).
weapon(revolver; candlestick).
character(colonel_Mustard;ms_Scarlet).
1{murderer(X):character(X)}1.
1{weapon_of_crime(X):weapon(X)}1.
1{place_of_crime(X):place(X)}1.
:- place_of_crime(kitchen).
place_of_crime(hall) :- murderer(colonel_Mustard) ;
                        not weapon_of_crime(revolver).
weapon_of_crime(candlestick).

sol(X,Y,Z) :- murderer(X) ; weapon_of_crime(Y) ; place_of_crime(Z).

criminal_record(colonel_Mustard,20).  criminal_record(ms_Scarlet,18).

number_previous_crime(hall,8).  number_previous_crime(dining_room,6).
number_previous_crime(kitchen,2).

#show sol/3.  #show murderer/1.  #show place_of_crime/1.  #show weapon_of_crime/1.
```

Exemple optimisation

```
place(kitchen; hall; diningroom).
weapon(revolver; candlestick).
character(colonel_Mustard;ms_Scarlet).
1{murderer(X):character(X)}1.
1{weapon_of_crime(X):weapon(X)}1.
1{place_of_crime(X):place(X)}1.
:- place_of_crime(kitchen).
place_of_crime(hall) :- murderer(colonel_Mustard) ;
                        not weapon_of_crime(revolver).
weapon_of_crime(candlestick).

sol(X,Y,Z) :- murderer(X) ; weapon_of_crime(Y) ; place_of_crime(Z).

criminal_record(colonel_Mustard,20).  criminal_record(ms_Scarlet,18).

number_previous_crime(hall,8).  number_previous_crime(dining_room,6).
number_previous_crime(kitchen,2).

#show sol/3.  #show murderer/1.  #show place_of_crime/1.  #show weapon_of_crime/1.
```

```
Answer:  1
weapon_of_crime(candlestick) murderer(ms_Scarlet) place_of_crime(dining_room)
sol(ms_Scarlet,candlestick,dining_room)
Answer:  2
weapon_of_crime(candlestick) place_of_crime(hall) murderer(ms_Scarlet)
sol(ms_Scarlet,candlestick,hall)
Answer:  3
weapon_of_crime(candlestick) place_of_crime(hall) murderer(colonel_Mustard)
sol(colonel_Mustard,candlestick,hall)
```

Question: écrire une règle qui permet de caractériser les solutions de poids minimal?

Synthaxe

```
#minimize{W , sol :  sol(X,Y,Z) , criminal_record(X,W) ,
                          character(X) , place(Z) , weapon(Y)}.
```

```
(./clingo cluedo.lp --opt-mode=optN)
```

```
Answer:  1
weapon_of_crime(candlestick) place_of_crime(hall)
murderer(ms_Scarlet)
sol(ms_Scarlet,candlestick,hall)
Optimization:  3
Answer:  2
weapon_of_crime(candlestick) murderer(ms_Scarlet)
sol(ms_Scarlet,candlestick,dining_room)
place_of_crime(dining_room)
Optimization:  3
OPTIMUM FOUND
```

Syntaxe: Clauses pour l'optimisation

```
#minimize { N , ensembleProjete:  ensembleProjete(X,Y,Z) , DomaineValidite(X),
            DomaineValidite(Y), DomaineValidite(Z), poids(X,Y,Z,N) }.
```

On construit l'ensemble qui instancie tous les atomes `ensembleProjete(X,Y,Z)` où les variables sont définies par leurs domaines de validité.

Pour chaque solution du programme, on filtre cet ensemble avec les atomes appartenant à la solution.

Chaque ensemble d'atomes instancié `ensembleProjete(x,y,z)` pondéré par N déterminé par le prédicat `poids(x,y,z,N)`.

Les sorties de l'optimisation sont les solutions de poids total (somme) minimal

Syntaxe: Clauses pour l'optimisation

```
#minimize { N , ensembleProjete: ensembleProjete(X,Y,Z) , DomaineValidite(X),
            DomaineValidite(Y), DomaineValidite(Z), poids(X,Y,Z,N) }.
minimize { W , sol : sol(X,Y,Z),criminal_record(X,W),character(X),place(Z),weapon(Y)
           T, place : place(Z), place_of_crime(Z), number_previous_crime(Z,T)}.
```

On construit l'ensemble qui instancie tous les atomes `ensembleProjete(X,Y,Z)` où les variables sont définies par leurs domaines de validité.

Pour chaque solution du programme, on filtre cet ensemble avec les atomes appartenant à la solution.

Chaque ensemble d'atomes instancié `ensembleProjete(x,y,z)` pondéré par N déterminé par le prédicat `poids(x,y,z,N)`.

Les sorties de l'optimisation sont les solutions de poids total (somme) minimal

Syntaxe: Clauses pour l'optimisation

```
#minimize { N , ensembleProjete: ensembleProjete(X,Y,Z) , DomaineValidite(X),
            DomaineValidite(Y), DomaineValidite(Z), poids(X,Y,Z,N) }.
minimize { W , sol : sol(X,Y,Z),criminal_record(X,W),character(X),place(Z),weapon(Y)
           T, place : place(Z), place_of_crime(Z), number_previous_crime(Z,T)}.
```

On construit l'ensemble qui instancie tous les atomes `ensembleProjete(X,Y,Z)` où les variables sont définies par leurs domaines de validité.

{ sol(ms_Scarlet,revolver,kitchen) ,sol(ms_Scarlet,candlestick,hall), sol(ms_Scarlet,revolver,hall), sol(colonel Mustard,revolver,dining_room) , ... , place(hall), place(dining_room), place(kitchen) }

Pour chaque solution du programme, on filtre cet ensemble avec les atomes appartenant à la solution.

Chaque ensemble d'atomes instancié `ensembleProjete(x,y,z)` pondéré par N déterminé par le prédicat `poids(x,y,z,N)`.

Les sorties de l'optimisation sont les solutions de poids total (somme) minimal

Syntaxe: Clauses pour l'optimisation

```
#minimize { N , ensembleProjete: ensembleProjete(X,Y,Z) , DomaineValidite(X),
            DomaineValidite(Y), DomaineValidite(Z), poids(X,Y,Z,N) }.
minimize { W , sol : sol(X,Y,Z),criminal_record(X,W),character(X),place(Z),weapon(Y)
           T, place : place(Z), place_of_crime(Z), number_previous_crime(Z,T)}.
```

On construit l'ensemble qui instancie tous les atomes `ensembleProjete(X,Y,Z)` où les variables sont définies par leurs domaines de validité.

{ sol(ms_Scarlet,revolver,kitchen) ,sol(ms_Scarlet,candlestick,hall), sol(ms_Scarlet,revolver,hall), sol(colonel Mustard,revolver,dining_room) , ... , place(hall), place(dining_room), place(kitchen) }

Pour chaque solution du programme, on filtre cet ensemble avec les atomes appartenant à la solution.

```
Answer: 1. weapon_of_crime(candlestick) murderer(ms_Scarlet) place_of_crime(dining_room)
sol(ms_Scarlet,candlestick,dining_room) → {sol(ms_Scarlet,candlestick,dining_room) , place(dining_room)}
Answer: 2. weapon_of_crime(candlestick) place_of_crime(hall) murderer(ms_Scarlet)
sol(ms_Scarlet,candlestick,hall) → { sol(ms_Scarlet,candlestick,hall) , place(hall) }
Answer: 3. weapon_of_crime(candlestick) place_of_crime(hall) murderer(colonel_Mustard)
sol(colonel_Mustard,candlestick,hall) → { sol(colonel_Mustard,candlestick,hall) , place(hall) }
```

Chaque ensemble d'atomes instancié `ensembleProjete(x,y,z)` pondéré par N déterminé par le prédicat `poids(x,y,z,N)`.

Les sorties de l'optimisation sont les solutions de poids total (somme) minimal

Syntaxe: Clauses pour l'optimisation

```
#minimize { N , ensembleProjete: ensembleProjete(X,Y,Z) , DomaineValidite(X),
            DomaineValidite(Y), DomaineValidite(Z), poids(X,Y,Z,N) }.
minimize { W , sol : sol(X,Y,Z),criminal_record(X,W),character(X),place(Z),weapon(Y)
           T, place : place(Z), place_of_crime(Z), number_previous_crime(Z,T)}.
```

On construit l'ensemble qui instancie tous les atomes `ensembleProjete(X,Y,Z)` où les variables sont définies par leurs domaines de validité.

{ sol(ms_Scarlet,revolver,kitchen) ,sol(ms_Scarlet,candlestick,hall), sol(ms_Scarlet,revolver,hall), sol(colonel Mustard,revolver,dining_room) , ... , place(hall), place(dining_room), place(kitchen) }

Pour chaque solution du programme, on filtre cet ensemble avec les atomes appartenant à la solution.

`Answer: 1. weapon_of_crime(candlestick) murderer(ms_Scarlet) place_of_crime(dining_room)`
`sol(ms_Scarlet,candlestick,dining_room)` → {sol(ms_Scarlet,candlestick,dining_room) , place(dining_room)}
`Answer: 2. weapon_of_crime(candlestick) place_of_crime(hall) murderer(ms_Scarlet)`
`sol(ms_Scarlet,candlestick,hall)` → { sol(ms_Scarlet,candlestick,hall) , place(hall) }
`Answer: 3. weapon_of_crime(candlestick) place_of_crime(hall) murderer(colonel_Mustard)`
`sol(colonel_Mustard,candlestick,hall)` → { sol(colonel_Mustard,candlestick,hall) , place(hall) }

Chaque ensemble d'atomes instancié `ensembleProjete(x,y,z)` pondéré par N déterminé par le prédicat `poids(x,y,z,N)`.

{sol(ms_Scarlet,candlestick,dining_room),place(dining_room)} → N=4, T=6
{sol(ms_Scarlet,candlestick,hall) } → N=4, T=8
{sol(colonel_Mustard,candlestick,hall)} → N=7, T=8

Les sorties de l'optimisation sont les solutions de poids total (somme) minimal

Syntaxe: Clauses pour l'optimisation

```
#minimize { N , ensembleProjete: ensembleProjete(X,Y,Z) , DomaineValidite(X),
            DomaineValidite(Y), DomaineValidite(Z), poids(X,Y,Z,N) }.
minimize { W , sol : sol(X,Y,Z),criminal_record(X,W),character(X),place(Z),weapon(Y)
           T, place : place(Z), place_of_crime(Z), number_previous_crime(Z,T)}.
```

On construit l'ensemble qui instancie tous les atomes `ensembleProjete(X,Y,Z)` où les variables sont définies par leurs domaines de validité.

{ sol(ms_Scarlet,revolver,kitchen) ,sol(ms_Scarlet,candlestick,hall), sol(ms_Scarlet,revolver,hall), sol(colonel Mustard,revolver,dining_room) , ... , place(hall), place(dining_room), place(kitchen) }

Pour chaque solution du programme, on filtre cet ensemble avec les atomes appartenant à la solution.

`Answer: 1. weapon_of_crime(candlestick) murderer(ms_Scarlet) place_of_crime(dining_room)`
`sol(ms_Scarlet,candlestick,dining_room)` → {sol(ms_Scarlet,candlestick,dining_room) , place(dining_room)}
`Answer: 2. weapon_of_crime(candlestick) place_of_crime(hall) murderer(ms_Scarlet)`
`sol(ms_Scarlet,candlestick,hall)` → { sol(ms_Scarlet,candlestick,hall) , place(hall) }
`Answer: 3. weapon_of_crime(candlestick) place_of_crime(hall) murderer(colonel_Mustard)`
`sol(colonel_Mustard,candlestick,hall)` → { sol(colonel_Mustard,candlestick,hall) , place(hall) }

Chaque ensemble d'atomes instancié `ensembleProjete(x,y,z)` pondéré par N déterminé par le prédicat `poids(x,y,z,N)`.

{sol(ms_Scarlet,candlestick,dining_room),place(dining_room)} → N=4, T=6
{sol(ms_Scarlet,candlestick,hall) } → N=4, T=8
{sol(colonel_Mustard,candlestick,hall)} → N=7, T=8

Les sorties de l'optimisation sont les solutions de poids total (somme) minimal `Answer:1.`

Exemple

Exercice: que veut dire cette clause ?

```
#maximize { 1 :  p  ; 2 , q :  q(X,Y), X>0 , Y<0 }
```

Exemple

Exercice: que veut dire cette clause ?

```
#maximize { 1 :  p  ; 2 , q :  q(X,Y), X>0 , Y<0 }
```

Maximiser au sens de leur poids global
l'ensemble d'atomes constitué

de l'atome `p` de poids 1, (*pas de projection indiquée puisqu'il s'agit d'une constante*)

et de tous les `q(X,Y)` (chacun avec un poids 2)

où `X` et `Y` sont définis par ailleurs dans les clauses définissant `q`
mais avec la restriction supplémentaire
que `X` soit strictement positif et `X` strictement négatif.

Exemple optimisation

Quelle est la différence entre les deux règles suivantes:

```
#minimize{W , sol :  sol(X,Y,Z) , weight(X,W) , character(X) , place(Z) , weapon(Y)}.
```

et

```
#minimize { 1 :  place(kitchen) ; W, sol :  sol(X,Y,Z), weight(X,W),
                              character(X) , place(Z) , weapon(Y) }
```

Exemple optimisation

Quelle est la différence entre les deux règles suivantes:

```
#minimize{W , sol :  sol(X,Y,Z) , weight(X,W) , character(X) , place(Z) , weapon(Y)}.
```

et

```
#minimize { 1 :  place(kitchen) ; W, sol :  sol(X,Y,Z), weight(X,W),
                          character(X) , place(Z) , weapon(Y) }
```

```
Answer:  1
weapon_of_crime(candlestick) place_of_crime(hall) murderer(ms_Scarlet)
sol(ms_Scarlet,candlestick,hall)
Optimization:  3
Answer:  2
weapon_of_crime(candlestick) murderer(ms_Scarlet) sol(ms_Scarlet,candlestick,dining_room)
place_of_crime(dining_room)
Optimization:  3
OPTIMUM FOUND

Answer:  1
weapon_of_crime(candlestick) place_of_crime(hall) murderer(ms_Scarlet)
sol(ms_Scarlet,candlestick,hall)
Optimization:  4
Answer:  2
weapon_of_crime(candlestick) murderer(ms_Scarlet) sol(ms_Scarlet,candlestick,dining_room)
place_of_crime(dining_room)
Optimization:  4
OPTIMUM FOUND
```

Syntaxe: Clauses

Mutli-optimisation

```
#maximize { 1@2 :  p ; 4@2 , q :  q(X,Y), X>0,Y<0 }.
#minimize { X-1@1, t :  t(X), s(X) } .
```

On cherche d'abord à maximiser (priorité @2) avant de minimiser (priorité @1 $<$ @2).

Sorties

```
#show p(X). #show p/1.  #show s(X):r(X).
```

Le `/1` désigne l'arité de `p`. `p/1` est équivalent à `p(X)`.

Exercice

Proposer un programme ASP pour modéliser le système suivant

Le métabolisme du lactose chez Escherichia coli nécessite la présence de 3 enzymes: la β-galactosidase LacZ, la pérméase LacY et la transacetylase LacA. La perméase LacY transporte le lactose de l'extérieur de la cellule à l'intérieur du milieu cellulaire. LacZ hydrolyse le lactose interne en glucose et en allolactose. Les protéines LacY et LacZ sont sous contrôle direct d'un inhibiteur appelé LacI. Inversement, un complexe formé d'AMP cyclique et de la protéine CRP (cAMP − CRP) agit sur l'activité des ARN-polymérases et permet d'activer fortement la transcription de LacY et LacZ. Le glucose inhibe la production d'AMP cyclique. Enfin, l'allolactose a une action inhibitrice sur LacI via la formation d'un complexe.

(Début de solution de l') Exercice

```
%enzyme(lacI).
enzyme(lacZ).
%enzyme(lacY).
%enzyme(lacA).
metabolite(lactoseExterne).
metabolite(lactoseInterne) :- enzyme(lacY) ; metabolite(lactoseExterne).
metabolite(allolactose) :- enzyme(lacZ) ; metabolite(lactoseInterne).
metabolite(glucose) :- enzyme(lacZ) ; metabolite(lactoseInterne).
```

```
%:- enzyme(lacI), enzyme(lacY).
```

(Intégrité: si lacI et lacY sont présents, le système est incompatible)

```
enzyme(lacY) :- not enzyme(lacI).
```

(règle: si lacI est absent, alors lacY est présent)

```
:- enzyme(lacI), enzyme(lacZ).
enzyme(lacZ):- coumpound(ampcyc) ; proteine(crp).
enzyme(lacY):- coumpound(ampcyc) ; proteine(crp).
:-coumpound(ampcyc) ; metabolite(glucose).
:-metabolite(allolactose) ; enzyme(lacI).
#show enzyme/1.
#show metabolite/1.
#show coumpound/1.
```

Le codage des règles est très sensible à ce que nous voulons exprimer et peut changer tous les résultats !

Exercice

Données : nombre de minutes que doit mettre un taxi pour joindre un client

```
cout(taxi1,client1,10).   cout(taxi2,client1,8).
cout(taxi3,client1,12).   cout(taxi1,client2,11).
cout(taxi2,client2,15).   cout(taxi3,client2,13).
cout(taxi1,client3 ,7).   cout(taxi2,client3 ,7).
cout(taxi3,client3,10).
```

Dérivation d'un graphe : comment identifier les taxis et les clients ?

```
taxi(T):- cout(T,_,_).   client(T):- cout(_,T,_).
```

Caractéristiques d'une solution

Exercice: introduire des clauses pour définir le prédicat `sol(Taxi, Client)` qui affecte un taxi par client et un client par taxi

Exercice

Données : nombre de minutes que doit mettre un taxi pour joindre un client

```
cout(taxi1,client1,10).  cout(taxi2,client1,8).
cout(taxi3,client1,12).  cout(taxi1,client2,11).
cout(taxi2,client2,15).  cout(taxi3,client2,13).
cout(taxi1,client3 ,7).  cout(taxi2,client3 ,7).
cout(taxi3,client3,10).
```

Dérivation d'un graphe : comment identifier les taxis et les clients ?

```
taxi(T):- cout(T,_,_).  client(T):- cout(_,T,_).
```

Caractéristiques d'une solution

Exercice: introduire des clauses pour définir le prédicat `sol(Taxi, Client)` qui affecte un taxi par client et un client par taxi

```
1{sol(X,Y):taxi(X)}1:- client(Y).
1{sol(X,Y):client(Y)}1:- taxi(X).
```

Optimisation

Exercice: minimiser le coût d'affectation ?

Exercice

Données : nombre de minutes que doit mettre un taxi pour joindre un client

```
cout(taxi1,client1,10).  cout(taxi2,client1,8).
cout(taxi3,client1,12).  cout(taxi1,client2,11).
cout(taxi2,client2,15).  cout(taxi3,client2,13).
cout(taxi1,client3 ,7).  cout(taxi2,client3 ,7).
cout(taxi3,client3,10).
```

Dérivation d'un graphe : comment identifier les taxis et les clients ?

```
taxi(T):- cout(T,_,_).  client(T):- cout(_,T,_).
```

Caractéristiques d'une solution

Exercice: introduire des clauses pour définir le prédicat `sol(Taxi, Client)` qui affecte un taxi par client et un client par taxi

```
1{sol(X,Y):taxi(X)}1:- client(Y).
1{sol(X,Y):client(Y)}1:- taxi(X).
```

Optimisation

Exercice: minimiser le coût d'affectation ?

```
#minimize { Temps, sol:  sol(Taxi, Client), cout(Taxi, Client, Temps) } .
```

Exercice

Données : nombre de minutes que doit mettre un taxi pour joindre un client

```
cout(taxi1,client1,10).  cout(taxi2,client1,8).
cout(taxi3,client1,12).  cout(taxi1,client2,11).
cout(taxi2,client2,15).  cout(taxi3,client2,13).
cout(taxi1,client3 ,7).  cout(taxi2,client3 ,7).
cout(taxi3,client3,10).
```

Dérivation d'un graphe : comment identifier les taxis et les clients ?

```
taxi(T):- cout(T,_,_).  client(T):- cout(_,T,_).
```

Caractéristiques d'une solution

Exercice: introduire des clauses pour définir le prédicat `sol(Taxi, Client)` qui affecte un taxi par client et un client par taxi

```
1{sol(X,Y):taxi(X)}1:- client(Y).
1{sol(X,Y):client(Y)}1:- taxi(X).
```

Optimisation

Exercice: minimiser le coût d'affectation ?

```
#minimize { Temps, sol:  sol(Taxi, Client), cout(Taxi, Client, Temps) } .
```

.... généralisable à de nombreux problèmes de graphes.

ASP est utile dans quel cas ?

Résolution des problèmes combinatoires: problèmes NP-complets et NP-difficiles (graphes, raisonnement, ...)

- Cadre unifié où sont intégrés des aspects bases de données, bases de connaissances, résolution de contraintes et programmation logique.
- Déclarativité Exprimer facilement des propriétés mathématiques sur un espace de recherche : plus il y a de propriétés, mieux ça marchera !
- Optimisation
- Différents modes de raisonnement: une solution, toutes, les meilleures, leur intersection ou leur union.
- Facilité pour mettre au point et tester différents modèles et différentes heuristiques.

Limites

Limite

- On n'échappe pas à la complexité mais on tire parti des particularités d'une instance de problème à résoudre.
- Difficile de tracer un programme ASP pour savoir où se cache la partie complexe.
  $\rightarrow$ Procéder par étapes : découpage modulaire en clauses indépendantes.
  $\rightarrow$ Prototype de profileur de l'exécution d'un programme.

A ne pas utiliser...

- Pour remplacer des algorithmes classiques maîtrisés (tri, calcul traitement de séquences)
  $\rightarrow$ Python est intégré dans les dernières versions de clingo.
- travailler dans des espaces peu structurés et peu contraints
- travailler dans des espaces difficiles à discrétiser.
- résoudre un problème qui nécessiterait de nombreuses étapes successives qui communiquent
  $\rightarrow$ un programme ASP ne permet pas d'écrire des procédures.

Le piège...

Explosion lors de l'instanciation

- ASP peut gérer des programmes avec quelques millions d'atomes.
- MAIS : ce nombre est vite atteint !
- Limiter au maximum le nombre de variables différentes dans une clause !

Compromis grounding/solving

- Gringo (instancieur) dépense de la mémoire. Précalcule (tabule) tout ce qui peut être instancié à l'avance de manière déterministe.
- Clasp (solveur) dépense du CPU, il effectue les choix et élague dynamiquement l'espace de recherche.
- Il faut faire en sorte que le solveur apprenne de nouvelles contraintes en cours de résolution.

Le choix des heuristiques

Dépend du problème considéré !

clasp: Calcul des ensemble-réponses minimaux en taille.

unsatisfiable core: Autre heuristique pour calculer des ensembles-réponses minimaux en taille
$\rightarrow$ petits sous-ensembles d'un espace de grande taille.

Disjunctive utilise des metaprogrammes additionnels pour calculer des ensembles-réponses minimaux pour l'inclusion ensembliste.

heuristique utilise des prédicats supplémentaires pour guider la rechercher et réaliser des "mémorisations de solutions" pour calculer des ensembles minimaux pour l'inclusion ensembliste.

incremental solver Solveur incrémental : (base + paramètre incrémentant d'un pas de résolution à l'autre)

La plupart de ces heuristiques sont maintenant intégrées dans les dernières versions de clasp sous la forme d'options

Programmation par ensembles-réponses: un peu de sémantique

Ensemble-réponse dans le cas de programmes positifs

Programme positif

Un programme positif est constitué d'un ensemble de *règles* (clauses définies)

$$\underbrace{A}_{tete} \text{ :- } \underbrace{B_1; B_2; \ldots B_m}_{corps}.$$

où A (la *tête* de la règle) et les B (le *corps*) sont des *atomes* (variables booléennes)

Ensemble-réponse dans le cas de programmes positifs

Programme positif

Un programme positif est constitué d'un ensemble de *règles* (clauses définies)

$$\underbrace{A}_{tete} \text{ :- } \underbrace{B_1; B_2; \ldots B_m}_{corps}.$$

où A (la *tête* de la règle) et les B (le *corps*) sont des *atomes* (variables booléennes)

A :- $B_1; B_2; \ldots B_m.$ correspond à la formule $\neg B_1 \vee \neg B_2 \vee \neg \vee \neg B_m \vee A$

Les programmes positifs sont des clauses disjonctives avec exactement un atome positif.

Ensemble-réponse dans le cas de programmes positifs

Programme positif

Un programme positif est constitué d'un ensemble de *règles* (clauses définies)

$$\underbrace{A}_{tete} \text{ :- } \underbrace{B_1; B_2; \ldots B_m}_{corps}.$$

où A (la *tête* de la règle) et les B (le *corps*) sont des *atomes* (variables booléennes)

A :- $B_1; B_2; \ldots B_m.$ correspond à la formule $\neg B_1 \vee \neg B_2 \vee \neg \vee \neg B_m \vee A$

Les programmes positifs sont des clauses disjonctives avec exactement un atome positif.

Ensemble clos d'un programme positif

Un ensemble d'atomes X est *clos pour un programme positif P* si

$$\textbf{pour toute règle } r \textbf{ du programme } P, \quad \textbf{corps}(r) \subset X \implies \textbf{tête}(r) \subset X$$

C'est un *modèle du programme* vu comme une formule.

Ensemble-réponse dans le cas de programmes positifs

Programme positif

Un programme positif est constitué d'un ensemble de *règles* (clauses définies)

$$\underbrace{A}_{tete} \text{ :- } \underbrace{B_1; B_2; \ldots B_m}_{corps}.$$

où A (la *tête* de la règle) et les B (le *corps*) sont des *atomes* (variables booléennes)

A :- $B_1; B_2; \ldots B_m.$ correspond à la formule $\neg B_1 \vee \neg B_2 \vee \neg \vee \neg B_m \vee A$

Les programmes positifs sont des clauses disjonctives avec exactement un atome positif.

Ensemble clos d'un programme positif

Un ensemble d'atomes X est *clos pour un programme positif P* si

$$\textbf{pour toute règle } r \textbf{ du programme } P, \quad \textbf{corps}(r) \subset X \implies \textbf{tête}(r) \subset X$$

C'est un *modèle du programme* vu comme une formule.

Ensemble-réponse

L'ensemble réponse $Cn(P)$ (*answer set*) d'un programme positif P est le plus petit ensemble d'atomes qui soit clos pour P. Il existe et est unique.

Exemple 1

Vérifier que pour toute règle r, on a bien $\text{corps}(r) \subset X \implies \text{tête}(r) \subset X$

Programme

```
a :- b;p;q.
b :- q.
```

Liste des sous-ensembles d'atomes

{a,b,p,q}, {a,b,p}, {a,b,q}, {a,p,q}, {b,p,q}, {a,b}, {a,p}, {a,q}, {b,p}, {b,q}, {p,q}, {a}, {b}, {p}, {q}, $\emptyset$.

Exemple 1

Vérifier que pour toute règle r, on a bien $\text{corps}(r) \subset X \implies \text{tête}(r) \subset X$

Programme

```
a :- b;p;q.
b :- q.
```

Liste des sous-ensembles d'atomes

{a,b,p,q}, {a,b,p}, {a,b,q}, {a,p,q}, {b,p,q}, {a,b}, {a,p}, {a,q}, {b,p}, {b,q}, {p,q}, {a}, {b}, {p}, {q}, ∅.

Sous-ensembles clos

Question: quels sont-ils ?

Exemple 1

Vérifier que pour toute règle r, on a bien $\text{corps}(r) \subset X \implies \text{tête}(r) \subset X$

Programme

```
a :- b;p;q.
b :- q.
```

Liste des sous-ensembles d'atomes

{a,b,p,q}, {a,b,p}, {a,b,q}, {a,p,q}, {b,p,q}, {a,b}, {a,p}, {a,q}, {b,p}, {b,q}, {p,q}, {a}, {b}, {p}, {q}, ∅.

Sous-ensembles clos

Question: quels sont-ils ?

{a,b,p,q}, {a,b,p}, {a,b,q}, NONCLOS{a,p,q}, NONCLOS{b,p,q},
{a,b}, {a,p}, NONCLOS{a,q}, NONCLOS{b,p},
{b,q}, NONCLOS{p,q}, {a}, {b}, {p}, NONCLOS{q}, ∅.

Exemple 1

Vérifier que pour toute règle r, on a bien $\text{corps}(r) \subset X \implies \text{tête}(r) \subset X$

Programme

```
a :- b;p;q.
b :- q.
```

Liste des sous-ensembles d'atomes

{a,b,p,q}, {a,b,p}, {a,b,q}, {a,p,q}, {b,p,q}, {a,b}, {a,p}, {a,q}, {b,p}, {b,q}, {p,q}, {a}, {b}, {p}, {q}, ∅.

Sous-ensembles clos **Question: quels sont-ils ?**

{a,b,p,q}, {a,b,p}, {a,b,q}, NONCLOS{a,p,q}, NONCLOS{b,p,q},
{a,b}, {a,p}, NONCLOS{a,q}, NONCLOS{b,p},
{b,q}, NONCLOS{p,q}, {a}, {b}, {p}, NONCLOS{q}, ∅.

Quel est l'ensemble-réponse du programme ?

Exemple 1

Vérifier que pour toute règle r, on a bien $\text{corps}(r) \subset X \implies \text{tête}(r) \subset X$

Programme

```
a :- b;p;q.
b :- q.
```

Liste des sous-ensembles d'atomes

{a,b,p,q}, {a,b,p}, {a,b,q}, {a,p,q}, {b,p,q}, {a,b}, {a,p}, {a,q}, {b,p}, {b,q}, {p,q}, {a}, {b}, {p}, {q}, ∅.

Sous-ensembles clos **Question: quels sont-ils ?**

{a,b,p,q}, {a,b,p}, {a,b,q}, NONCLOS{a,p,q}, NONCLOS{b,p,q},
{a,b}, {a,p}, NONCLOS{a,q}, NONCLOS{b,p},
{b,q}, NONCLOS{p,q}, {a}, {b}, {p}, NONCLOS{q}, ∅.

Quel est l'ensemble-réponse du programme ?

L'ensemble-réponse du programme est l'ensemble vide ∅

→ Le programme ne contient que des implications, et rien n'est "vrai".

Tout ce qui n'est pas prouvé par un fait est considéré comme faux!

Exemple 2

Vérifier que pour toute règle r, on a bien $\text{corps}(r) \subset X \implies \text{tête}(r) \subset X$

Programme

```
a :- b;p;q.
b :- q.
q.
```

Liste des sous-ensembles d'atomes

{a,b,p,q}, {a,b,p}, {a,b,q}, {a,p,q}, {b,p,q}, {a,b}, {a,p}, {a,q}, {b,p}, {b,q}, {p,q}, {a}, {b}, {p}, {q}, $\emptyset$.

Exemple 2

Vérifier que pour toute règle r, on a bien $\text{corps}(r) \subset X \implies \text{tête}(r) \subset X$

Programme

```
a :- b;p;q.
b :- q.
q.
```

Liste des sous-ensembles d'atomes

{a,b,p,q}, {a,b,p}, {a,b,q}, {a,p,q}, {b,p,q}, {a,b}, {a,p}, {a,q}, {b,p}, {b,q}, {p,q}, {a}, {b}, {p}, {q}, $\emptyset$.

Liste des sous-ensembles clos ?

Exemple 2

Vérifier que pour toute règle r, on a bien $\text{corps}(r) \subset X \implies \text{tête}(r) \subset X$

Programme

```
a :- b;p;q.
b :- q.
q.
```

Liste des sous-ensembles d'atomes

{a,b,p,q}, {a,b,p}, {a,b,q}, {a,p,q}, {b,p,q}, {a,b}, {a,p}, {a,q}, {b,p}, {b,q}, {p,q}, {a}, {b}, {p}, {q}, $\emptyset$.

Liste des sous-ensembles clos ?

{a,b,p,q}, NONCLOS{a,b,p}, {a,b,q}, NONCLOS{a,p,q}, NONCLOS{b,p,q}, NONCLOS{a,b}, NONCLOS{a,p}, NONCLOS{a,q}, NONCLOS{b,p}, {b,q}, NONCLOS{p,q}, NONCLOS{a}, NONCLOS{b}, NONCLOS{p}, NONCLOS{q}, NONCLOS $\emptyset$.

$\rightarrow$ Comme $\emptyset$ est contenu dans un corps, q doit être contenu dans tous les sous-ensembles clos.

Exemple 2

Vérifier que pour toute règle r, on a bien $\text{corps}(r) \subset X \implies \text{tête}(r) \subset X$

Programme

```
a :- b;p;q.
b :- q.
q.
```

Liste des sous-ensembles d'atomes

{a,b,p,q}, {a,b,p}, {a,b,q}, {a,p,q}, {b,p,q}, {a,b}, {a,p}, {a,q}, {b,p}, {b,q}, {p,q}, {a}, {b}, {p}, {q}, $\emptyset$.

Liste des sous-ensembles clos ?

{a,b,p,q}, NONCLOS{a,b,p}, {a,b,q}, NONCLOS{a,p,q}, NONCLOS{b,p,q}, NONCLOS{a,b}, NONCLOS{a,p}, NONCLOS{a,q}, NONCLOS{b,p}, {b,q}, NONCLOS{p,q}, NONCLOS{a}, NONCLOS{b}, NONCLOS{p}, NONCLOS{q}, NONCLOS $\emptyset$.

$\rightarrow$ Comme $\emptyset$ est contenu dans un corps, q doit être contenu dans tous les sous-ensembles clos.

Quel est l'ensemble-réponse?

Exemple 2

Vérifier que pour toute règle r, on a bien $\text{corps}(r) \subset X \implies \text{tête}(r) \subset X$

Programme

```
a :- b;p;q.
b :- q.
q.
```

Liste des sous-ensembles d'atomes

{a,b,p,q}, {a,b,p}, {a,b,q}, {a,p,q}, {b,p,q}, {a,b}, {a,p}, {a,q}, {b,p}, {b,q}, {p,q}, {a}, {b}, {p}, {q}, ∅.

Liste des sous-ensembles clos ?

{a,b,p,q}, NONCLOS{a,b,p}, {a,b,q}, NONCLOS{a,p,q}, NONCLOS{b,p,q}, NONCLOS{a,b}, NONCLOS{a,p}, NONCLOS{a,q}, NONCLOS{b,p}, {b,q}, NONCLOS{p,q}, NONCLOS{a}, NONCLOS{b}, NONCLOS{p}, NONCLOS{q}, NONCLOS ∅.

→ Comme ∅ est contenu dans un corps, q doit être contenu dans tous les sous-ensembles clos.

Quel est l'ensemble-réponse?

L'ensemble réponse du programme est **{b,q}**

→ On n'a pas gardé p puisqu'on n'a aucune preuve de p.

Les atomes d'un ensemble-réponse d'un programme positif apparaissent dans la tête d'au moins une règle

Programmes avec une négation ?

Programme "quelconque"

Un *programme* est constitué de manière plus générale d'un ensemble de règles

$$\texttt{r:}\quad \underbrace{A}_{tete} \texttt{:-} \underbrace{B_1; B_2; \ldots B_m}_{corps^+(r)} \underbrace{not\ B_{m+1};\ not\ B_{m+2}; \ldots not\ B_{m+n}}_{corps^-(r)} .$$

Programmes avec une négation ?

Programme "quelconque"

Un *programme* est constitué de manière plus générale d'un ensemble de règles

$$\texttt{r:} \quad \underbrace{A}_{tete} \texttt{ :- } \underbrace{B_1; B_2; \ldots B_m}_{corps^+(r)} \underbrace{not\ B_{m+1};\ not\ B_{m+2}; \ldots not\ B_{m+n}}_{corps^-(r)} .$$

Réduit par rapport à un ensemble

Le réduit d'un programme par rapport à un ensemble d'atomes X est l'ensemble des règles de la forme

`r:` `tête(r) :- corps`$^+$`(r).` pour tout règle r telle que $corps^-(r) \cap X = \emptyset$.

Identification

- On choisit un sous-ensemble d'atomes X
- On ne garde que les règles r : A `:-` $B_1; B_2; \ldots B_m$ pour lesquelles aucun des $B_{m+1}, B_{m+2}, \ldots B_{m+n}$ n'appartiennent à X.

Programmes avec une négation ?

Programme "quelconque"

Un *programme* est constitué de manière plus générale d'un ensemble de règles

$$\texttt{r:}\quad \underbrace{A}_{tete}\ \texttt{:-}\ \underbrace{B_1; B_2; \ldots B_m}_{corps^+(r)}\ \underbrace{not\ B_{m+1};\ not\ B_{m+2}; \ldots not\ B_{m+n}}_{corps^-(r)}.$$

Réduit par rapport à un ensemble

Le réduit d'un programme par rapport à un ensemble d'atomes X est l'ensemble des règles de la forme

`r:` `tête(r) :- corps`$^+$`(r).` pour tout règle r telle que $corps^-(r) \cap X = \emptyset$.

Identification

- On choisit un sous-ensemble d'atomes X
- On ne garde que les règles r : A `:-` $B_1; B_2; \ldots B_m$ pour lesquelles aucun des $B_{m+1}, B_{m+2}, \ldots B_{m+n}$ n'appartiennent à X.

Exemple

```
a :- b; not q.      b:- p; q.      p.
```

- **Exercice: calculer le réduit du programme par rapport à** { `q` }

  ```
  b:- p; q.
  p.
  ```

- **Exercice: calculer le réduit du programme par rapport à** { `a` }

Programmes avec une négation ?

Programme "quelconque"

Un *programme* est constitué de manière plus générale d'un ensemble de règles

$$\texttt{r:}\quad \underbrace{A}_{tete}\ \texttt{:-}\ \underbrace{B_1; B_2; \ldots B_m}_{corps^+(r)}\ \underbrace{not\ B_{m+1};\ not\ B_{m+2}; \ldots not\ B_{m+n}}_{corps^-(r)}.$$

Réduit par rapport à un ensemble

Le réduit d'un programme par rapport à un ensemble d'atomes X est l'ensemble des règles de la forme

`r:` `tête(r) :- corps`$^+$`(r).` pour tout règle r telle que $corps^-(r) \cap X = \emptyset$.

Identification

- On choisit un sous-ensemble d'atomes X
- On ne garde que les règles r : A `:-` $B_1; B_2; \ldots B_m$ pour lesquelles aucun des $B_{m+1}, B_{m+2}, \ldots B_{m+n}$ n'appartiennent à X.

Exemple `a :- b; not q.` `b:- p; q.` `p.`

- **Exercice: calculer le réduit du programme par rapport à** { `q` }

  ```
  b:- p; q.
  p.
  ```

- **Exercice: calculer le réduit du programme par rapport à** { `a` }

  ```
  a :- b.
  b:- p; q.
  p.
  ```

"Not": ensemble-réponse

Ensemble-réponse

Un *ensemble réponse* (*modèle stable*) d'un programme P est un sous-ensemble X d'atomes tel que

- X est clos pour le programme obtenu comme réduit de P par rapport à X.
- X est **stable**: X **est égal à l'ensemble-réponse de son programme réduit**.
- X est minimal pour l'inclusion parmi tous les ensembles clos et stables.

"Not": ensemble-réponse

Ensemble-réponse

Un *ensemble réponse* (*modèle stable*) d'un programme P est un sous-ensemble X d'atomes tel que

- X est clos pour le programme obtenu comme réduit de P par rapport à X.
- X est **stable**: X **est égal à l'ensemble-réponse de son programme réduit**.
- X est minimal pour l'inclusion parmi tous les ensembles clos et stables.

Remarque

- Il faut d'abord réduire le programme par rapport à X et ensuite vérifier que ce qu'on obtient est clos vis à vis de X.
- Il peut y en avoir 0, 1 ou plusieurs.

"Not": ensemble-réponse

Ensemble-réponse

Un *ensemble réponse* (*modèle stable*) d'un programme P est un sous-ensemble X d'atomes tel que

- X est clos pour le programme obtenu comme réduit de P par rapport à X.
- X est **stable**: X **est égal à l'ensemble-réponse de son programme réduit**.
- X est minimal pour l'inclusion parmi tous les ensembles clos et stables.

Remarque

- Il faut d'abord réduire le programme par rapport à X et ensuite vérifier que ce qu'on obtient est clos vis à vis de X.
- Il peut y en avoir 0, 1 ou plusieurs.

Contenu d'un ensemble-réponse

Un modèle contient l'atome d'une tête de règle

- si la règle est un fait (règle réduite à une tête) après en avoir enlevé les littéraux négatifs qui ne sont pas dans le modèle
- ou si tous les littéraux positifs du corps sont dans le modèle et aucun des littéraux négatifs.

Exemple 1

```
p:- p.  q:- not p.
```

Enumérer les sous-ensembles X d'atomes.
{p,q}, {p}, {q}, $\emptyset$.

Exemple 1

```
p:- p.  q:- not p.
```

Enumérer les sous-ensembles X d'atomes.
{p,q}, {p}, {q}, ∅.

Pour chacun, calculer le réduit.

- {p,q}. Réduit: ?
- {p}. Réduit: ?
- {q}. Réduit: ?
- ∅. Réduit: ?

Exemple 1

```
p:- p.  q:- not p.
```

Enumérer les sous-ensembles X d'atomes.
{p,q}, {p}, {q}, ∅.

Pour chacun, calculer le réduit.

- {p,q}. Réduit: ? `p:-p.`
- {p}. Réduit: ?
- {q}. Réduit: ?
- ∅. Réduit: ?

Exemple 1

```
p:- p.  q:- not p.
```

Enumérer les sous-ensembles X d'atomes.
{p,q}, {p}, {q}, ∅.

Pour chacun, calculer le réduit.

- {p,q}. Réduit: ? `p:-p.`
- {p}. Réduit: ? `p:-p.`
- {q}. Réduit: ?
- ∅. Réduit: ?

Exemple 1

```
p:- p.  q:- not p.
```

Enumérer les sous-ensembles X d'atomes.
{p,q}, {p}, {q}, ∅.

Pour chacun, calculer le réduit.

- {p,q}. Réduit: ? `p:-p.`
- {p}. Réduit: ? `p:-p.`
- {q}. Réduit: ? `p:-p.  q.`
- ∅. Réduit: ?

Exemple 1

```
p:- p.   q:- not p.
```

Enumérer les sous-ensembles X d'atomes.
{p,q}, {p}, {q}, ∅.

Pour chacun, calculer le réduit.

- {p,q}. Réduit: ? `p:-p.`
- {p}. Réduit: ? `p:-p.`
- {q}. Réduit: ? `p:-p.  q.`
- ∅. Réduit: ? `p:-p.  q.`

Exemple 1

```
p:- p.   q:- not p.
```

Enumérer les sous-ensembles X d'atomes.
{p,q}, {p}, {q}, ∅.

Pour chacun, calculer le réduit.

- {p,q}. Réduit: ? `p:-p.`
- {p}. Réduit: ? `p:-p.`
- {q}. Réduit: ? `p:-p.` `q.`
- ∅. Réduit: ? `p:-p.` `q.`

Pour chaque réduit, calculer l'ensemble clos minimal

- {p,q}. Réduit: `p:-p.` . Ensemble clos minimal?
- {p}. Réduit: `p:-p.` Ensemble clos minimal?
- {q}. Réduit: `p:-p.` `q.` Ensemble clos minimal?
- ∅. Réduit: `p:-p.` `q.` Ensemble clos minimal?

Exemple 1

```
p:- p.   q:- not p.
```

Enumérer les sous-ensembles X d'atomes.
{p,q}, {p}, {q}, ∅.

Pour chacun, calculer le réduit.

- {p,q}. Réduit: ? `p:-p.`
- {p}. Réduit: ? `p:-p.`
- {q}. Réduit: ? `p:-p.` `q.`
- ∅. Réduit: ? `p:-p.` `q.`

Pour chaque réduit, calculer l'ensemble clos minimal

- {p,q}. Réduit: `p:-p.` . Ensemble clos minimal? ∅
- {p}. Réduit: `p:-p.` Ensemble clos minimal?
- {q}. Réduit: `p:-p.` `q.` Ensemble clos minimal?
- ∅. Réduit: `p:-p.` `q.` Ensemble clos minimal?

Exemple 1

```
p:- p.   q:- not p.
```

Enumérer les sous-ensembles X d'atomes.
{p,q}, {p}, {q}, ∅.

Pour chacun, calculer le réduit.

- {p,q}. Réduit: ? `p:-p.`
- {p}. Réduit: ? `p:-p.`
- {q}. Réduit: ? `p:-p.` `q.`
- ∅. Réduit: ? `p:-p.` `q.`

Pour chaque réduit, calculer l'ensemble clos minimal

- {p,q}. Réduit: `p:-p.` . Ensemble clos minimal? ∅
- {p}. Réduit: `p:-p.` Ensemble clos minimal? ∅
- {q}. Réduit: `p:-p.` `q.` Ensemble clos minimal?
- ∅. Réduit: `p:-p.` `q.` Ensemble clos minimal?

Exemple 1

```
p:- p.   q:- not p.
```

Enumérer les sous-ensembles X d'atomes.
{p,q}, {p}, {q}, ∅.

Pour chacun, calculer le réduit.

- {p,q}. Réduit: ? `p:-p.`
- {p}. Réduit: ? `p:-p.`
- {q}. Réduit: ? `p:-p.` `q.`
- ∅. Réduit: ? `p:-p.` `q.`

Pour chaque réduit, calculer l'ensemble clos minimal

- {p,q}. Réduit: `p:-p.` . Ensemble clos minimal? ∅
- {p}. Réduit: `p:-p.` Ensemble clos minimal? ∅
- {q}. Réduit: `p:-p.` `q.` Ensemble clos minimal? {q}
- ∅. Réduit: `p:-p.` `q.` Ensemble clos minimal?

Exemple 1

```
p:- p.   q:- not p.
```

Enumérer les sous-ensembles X d'atomes.
{p,q}, {p}, {q}, ∅.

Pour chacun, calculer le réduit.

- {p,q}. Réduit: ? `p:-p.`
- {p}. Réduit: ? `p:-p.`
- {q}. Réduit: ? `p:-p.` `q.`
- ∅. Réduit: ? `p:-p.` `q.`

Pour chaque réduit, calculer l'ensemble clos minimal

- {p,q}. Réduit: `p:-p.` . Ensemble clos minimal? ∅
- {p}. Réduit: `p:-p.` Ensemble clos minimal? ∅
- {q}. Réduit: `p:-p.` `q.` Ensemble clos minimal? {q}
- ∅. Réduit: `p:-p.` `q.` Ensemble clos minimal? {q}

Exemple 1

```
p:- p.   q:- not p.
```

Enumérer les sous-ensembles X d'atomes.
{p,q}, {p}, {q}, ∅.

Pour chacun, calculer le réduit.

- {p,q}. Réduit: ? `p:-p.`
- {p}. Réduit: ? `p:-p.`
- {q}. Réduit: ? `p:-p.  q.`
- ∅. Réduit: ? `p:-p.  q.`

Pour chaque réduit, calculer l'ensemble clos minimal

- {p,q}. Réduit: `p:-p.` . Ensemble clos minimal? ∅
- {p}. Réduit: `p:-p.` Ensemble clos minimal? ∅
- {q}. Réduit: `p:-p.  q.` Ensemble clos minimal? {q}
- ∅. Réduit: `p:-p.  q.` Ensemble clos minimal? {q}

Garder les clos stables minimaux
Ensembles-réponses ?

Exemple 1

```
p:- p.  q:- not p.
```

Enumérer les sous-ensembles X d'atomes.
{p,q}, {p}, {q}, ∅.

Pour chacun, calculer le réduit.

- {p,q}. Réduit: ? `p:-p.`
- {p}. Réduit: ? `p:-p.`
- {q}. Réduit: ? `p:-p.  q.`
- ∅. Réduit: ? `p:-p.  q.`

Pour chaque réduit, calculer l'ensemble clos minimal

- {p,q}. Réduit: `p:-p.` . Ensemble clos minimal? ∅
- {p}. Réduit: `p:-p.` Ensemble clos minimal? ∅
- {q}. Réduit: `p:-p.  q.` Ensemble clos minimal? {q}
- ∅. Réduit: `p:-p.  q.` Ensemble clos minimal? {q}

Garder les clos stables minimaux

Ensembles-réponses ? {q}

(c'est le seul ensemble qui est égal à l'ensemble-réponse de son réduit.)

Exemple 2

```
p:- not q.  q:- not p.
```

Enumérer les sous-ensembles X d'atomes.
{p,q}, {p}, {q}, ∅.

Pour chacun, calculer le réduit.

- {p,q}. Réduit: ?→
- {p}. Réduit: ? →
- {q}. Réduit: ? →
- ∅. Réduit: ? →

Pour chaque réduit, calculer l'ensemble clos minimal

- {p,q}. Réduit: . Ensemble clos minimal?
- {p}. Réduit: Ensemble clos minimal?
- {q}. Réduit: Ensemble clos minimal?
- ∅. Réduit: Ensemble clos minimal?

Garder les clos minimaux

Ensembles-réponses ?

Exemple 2

```
p:- not q.   q:- not p.
```

Enumérer les sous-ensembles X d'atomes.
{p,q}, {p}, {q}, ∅.

Pour chacun, calculer le réduit.

- {p,q}. Réduit: ?→ ∅
- {p}. Réduit: ? → p.
- {q}. Réduit: ? → q.
- ∅. Réduit: ? → p. q.

Pour chaque réduit, calculer l'ensemble clos minimal

- {p,q}. Réduit: programme vide . Ensemble clos minimal?
- {p}. Réduit: p. Ensemble clos minimal? {p}
- {q}. Réduit: q. Ensemble clos minimal?
- ∅. Réduit: p. q. Ensemble clos minimal?

Garder les clos minimaux

Ensembles-réponses ?

Exemple 2

```
p:- not q.   q:- not p.
```

Enumérer les sous-ensembles X d'atomes.
{p,q}, {p}, {q}, ∅.

Pour chacun, calculer le réduit.

- {p,q}. Réduit: ?→ ∅
- {p}. Réduit: ? → p.
- {q}. Réduit: ? → q.
- ∅. Réduit: ? → p. q.

Pour chaque réduit, calculer l'ensemble clos minimal

- {p,q}. Réduit: programme vide . Ensemble clos minimal? rien
- {p}. Réduit: p. Ensemble clos minimal? {p}
- {q}. Réduit: q. Ensemble clos minimal? {q}
- ∅. Réduit: p. q. Ensemble clos minimal? {p,q}

Garder les clos minimaux

Ensembles-réponses ? {p}, {q}

Exemple 3

```
p:- not p.
```

Enumérer les sous-ensembles X d'atomes.

{p}, ∅.

Pour chacun, calculer le réduit.

- {p,q}. →
- ∅. →

Pour chaque réduit, calculer l'ensemble clos minimal

- {p,q}. Réduit: . Ensemble clos minimal?
- ∅. Réduit: Ensemble clos minimal?

Garder les clos minimaux

Ensembles-réponses ?

Exemple 3

```
p:- not p.
```

Enumérer les sous-ensembles X d'atomes.

{p}, ∅.

Pour chacun, calculer le réduit.

- {p,q}. → ∅
- ∅. → p.

Pour chaque réduit, calculer l'ensemble clos minimal

- {p,q}. Réduit: programme vide . Ensemble clos minimal?
- ∅. Réduit: p. Ensemble clos minimal?

Garder les clos minimaux

Ensembles-réponses ?

Exemple 3

```
p:- not p.
```

Enumérer les sous-ensembles X d'atomes.

{p}, ∅.

Pour chacun, calculer le réduit.

- {p,q}. → ∅
- ∅. → p.

Pour chaque réduit, calculer l'ensemble clos minimal

- {p,q}. Réduit: programme vide . Ensemble clos minimal? rien
- ∅. Réduit: p. Ensemble clos minimal? {p}

Garder les clos minimaux

Ensembles-réponses ?

Exemple 3

```
p:- not p.
```

Enumérer les sous-ensembles X d'atomes.

{p}, ∅.

Pour chacun, calculer le réduit.

- {p,q}. → ∅
- ∅. → p.

Pour chaque réduit, calculer l'ensemble clos minimal

- {p,q}. Réduit: programme vide . Ensemble clos minimal? rien
- ∅. Réduit: p. Ensemble clos minimal? {p}

Garder les clos minimaux

Ensembles-réponses ? Aucun: il n'y a pas d'ensemble stable !

A retenir

Un ensemble réponse contient des atomes se trouvant dans une tête de règle du programme. Tout élément d'un ensemble réponse est supporté par une règle.

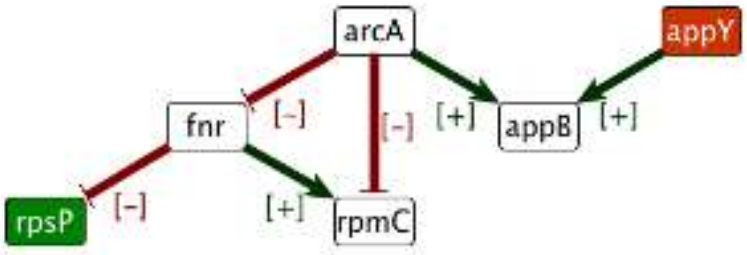
Logique non monotone Rajouter des faits à une théorie peut faire réduire l'ensemble de conclusions.

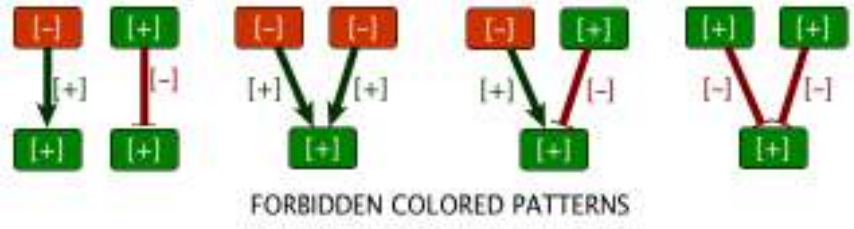
Jouons un peu...

Application réelle

Contrainte sur la coloration de graphes

Expliquer l'expression de chaque gène cible par la régulation consistante d'au moins une source.





Exercice

Ecrire un programme qui vérifie que le modèle est inconsistant ou, s'il est consistant, qui donne les prédictions existantes

Suite

Déclarer les signes, sommets, observations des sommets, aretes avec leurs observations.

Chaque sommet a exactement une prédiction `labelV`, qui est compatible avec les observations si elles existent.

Les observations permettent de spécifier des signes reçus (prédicat "receive(I,S)")

une prediction doit être expliquée par au moins un signe reçu

Suite

Déclarer les signes, sommets, observations des sommets, aretes avec leurs observations.

```
signe(down;up).
vertex(rpsP;fnr;arcA;rpmC;appY;appB).
observedV(rpsP,up).
observedV(appY,down).
observedE(fnr,rpsP,down).  observedE(fnr,rpmC,up).
observedE(arcA,fnr,down).  observedE(arcA,rpmC,down).
observedE(arcA,appB,up).  observedE(appY,appB,up).
```

Chaque sommet a exactement une prédiction `labelV`, qui est compatible avec les observations si elles existent.

```
1{labelV(I,S):signe(S)}1 :- vertex(I).
labelV(I,S) :- observedV(I,S).
```

Les observations permettent de spécifier des signes reçus (prédicat "receive(I,S)")

```
receive(I,up) :- observedE(J,I,S); labelV(J,S).
receive(I,down) :- observedE(J,I,S); labelV(J,T); S!=T.
receive(I,T) :- labelV(I,T); not observedE(J,I,S): vertex(J) , signe(S).
```

une prediction doit être expliquée par au moins un signe reçu

```
:- labelV(I,S); not receive(I,S).
```

Application

Enumeration

./clingo consistance.lp -n 0

Intersection des solutions : ce qui est certainement déduit

./clingo consistance.lp −−enum-mode=cautious

Union des solutions: grouper tout ce qui peut être déduit

./clinfo consistance.lp −−enum-mode=brave

Application

Enumeration

./clingo consistance.lp -n 0

```
Answer:  1
labelV(appY,down) labelV(rpsP,up) labelV(appB,up) labelV(rpmC,down)
labelV(arcA,up) labelV(fnr,down)
Answer:  2
labelV(appY,down) labelV(rpsP,up) labelV(appB,down) labelV(rpmC,down)
labelV(arcA,up) labelV(fnr,down)
SATISFIABLE
```

Intersection des solutions : ce qui est certainement déduit

./clingo consistance.lp −−enum-mode=cautious

Union des solutions: grouper tout ce qui peut être déduit

./clinfo consistance.lp −−enum-mode=brave

Application

Enumeration

./clingo consistance.lp -n 0

```
Answer:  1
labelV(appY,down) labelV(rpsP,up) labelV(appB,up) labelV(rpmC,down)
labelV(arcA,up) labelV(fnr,down)
Answer:  2
labelV(appY,down) labelV(rpsP,up) labelV(appB,down) labelV(rpmC,down)
labelV(arcA,up) labelV(fnr,down)
SATISFIABLE
```

Intersection des solutions : ce qui est certainement déduit

./clingo consistance.lp −−enum-mode=cautious

```
Cautious consequences:
labelV(appY,down) labelV(rpsP,up) labelV(rpmC,down) labelV(arcA,up)
labelV(fnr,down)
SATISFIABLE
```

Union des solutions: grouper tout ce qui peut être déduit

./clinfo consistance.lp −−enum-mode=brave

Application

Enumeration

./clingo consistance.lp -n 0

```
Answer:  1
labelV(appY,down) labelV(rpsP,up) labelV(appB,up) labelV(rpmC,down)
labelV(arcA,up) labelV(fnr,down)
Answer:  2
labelV(appY,down) labelV(rpsP,up) labelV(appB,down) labelV(rpmC,down)
labelV(arcA,up) labelV(fnr,down)
SATISFIABLE
```

Intersection des solutions : ce qui est certainement déduit

./clingo consistance.lp −−enum-mode=cautious

```
Cautious consequences:
labelV(appY,down) labelV(rpsP,up) labelV(rpmC,down) labelV(arcA,up)
labelV(fnr,down)
SATISFIABLE
```

Union des solutions: grouper tout ce qui peut être déduit

./clinfo consistance.lp −−enum-mode=brave

```
Brave consequences:
labelV(appY,down) labelV(rpsP,up) labelV(appB,up) labelV(appB,down)
labelV(rpmC,down) labelV(arcA,up) labelV(fnr,down)
SATISFIABLE
```

Extensions

Programme pour trouver une coloration qui contient le plus de signe "-" possibles ?

Extensions

Programme pour trouver une coloration qui contient le plus de signe "-" possibles ?

```
poids(up,1).
poids(down,0).
#minimize { T, poids:  poids(S,T), labelV(I,S), vertex(I) } .
```

Concrètement.

- On recherche l'optimum avec *./clingo programme.lp -n 0*
- On recherche tous les modèles optimaux avec
  ./clingo programme.lp −−opt-mode=optN

AuReMe

Integrative framework for metabolic networks

- Collet et al, LPNMR 2013.
- Bourdon et al, LPNMR 2013.
- Prigent et al, Plant journal 2014.
- Bourdon et al, Environm, 2016.
- Prigent et al, PLOS computational Biology, 2017.
- Frioux et al, LPNMR, 2017.
- Aite et al, submitted.

CNRS UPMC
Station Biologique
Roscoff

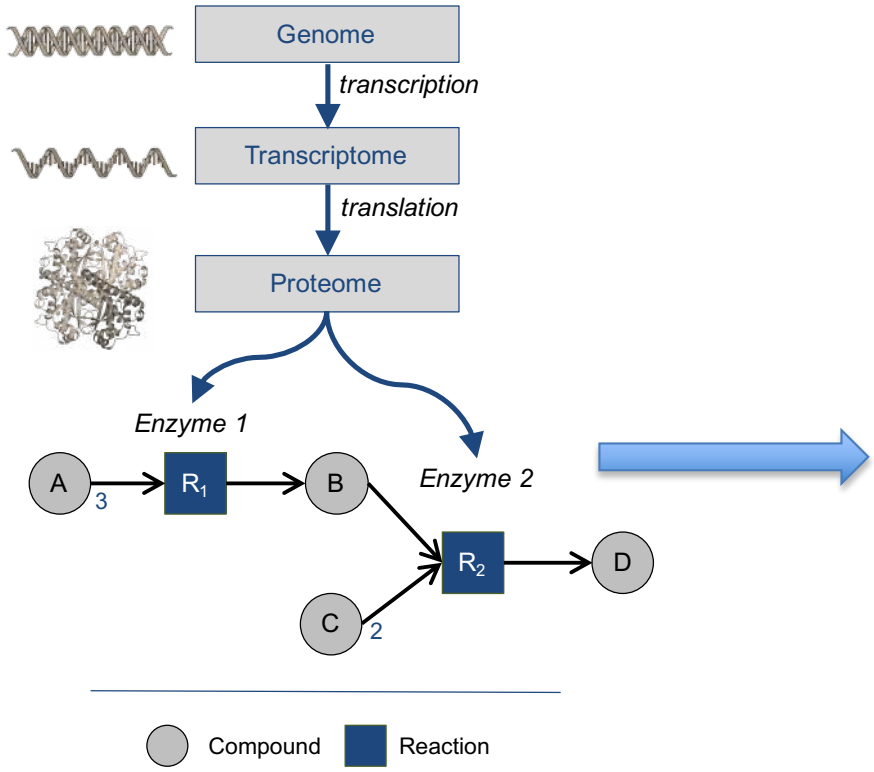
Universität Potsdam

CMM
Center for
Mathematical
Modeling

COMBI

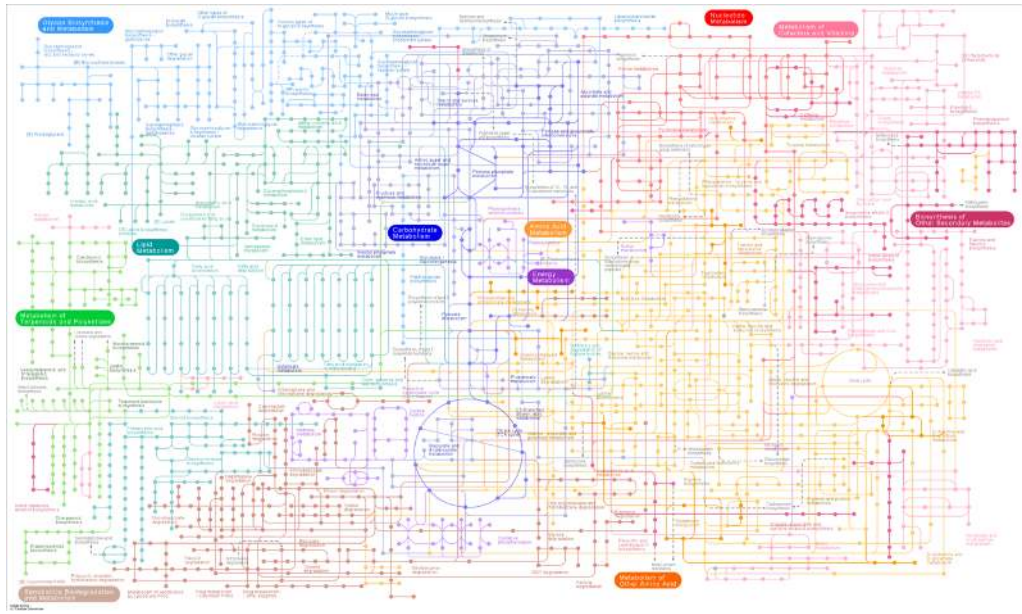
COMbinatoire et **B**io-**I**nformatique

Underlying tool : from genes to dynamical systems



Link between genes and functions

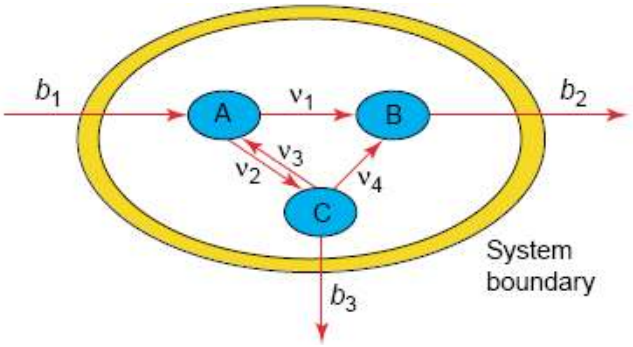
1 genome
⇛ 1 metabolic network
= bipartite directed graph



Large scale metabolic network

All expected metabolic capabilities of an organism

How to model fluxes ?



$$\frac{dA}{dt} = -v_1 - v_2 + v_3 + b_1$$

$$\frac{dB}{dt} = v_1 + v_4 - b_2$$

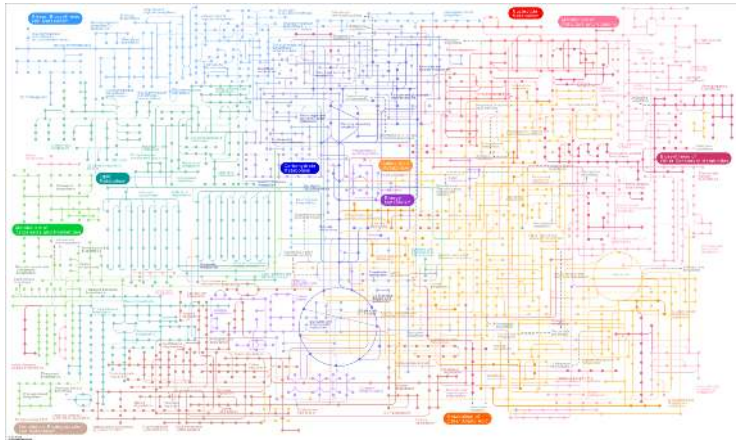
$$\frac{dC}{dt} = v_2 - v_3 - v_4 - b_3$$

$$\frac{dx}{dt} = S \cdot v(x)$$

$$v([substrat]) = Vm[Substrat] / (Km + [Substrat])$$

Back to high school chemistry

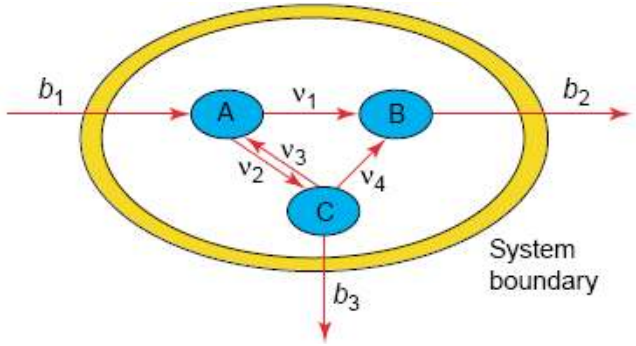
- Two parameters have to be estimated for each reaction



Intractable in practice !

- Overapproximation of the dynamics

Quasi-steady state hypothesis



$$\frac{dx}{dt} = S \cdot v(x) = 0 = S \cdot v$$

$$v([substrat]) = Vm[Substrat] / (Km + [Substrat])$$

= constant

Metabolic compounds do not accumulate

- Fluxes have constant values
- Fluxes are constrained by linear values
- The system optimises a global objective

r is *active* if

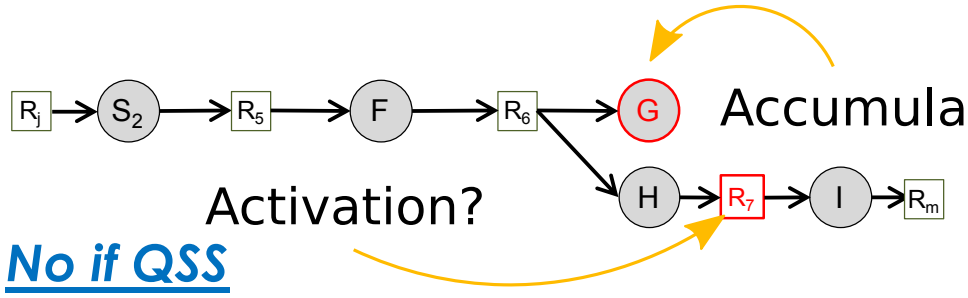
$v_r > 0$ and

$s.v = 0$ and

$lb < v < ub$

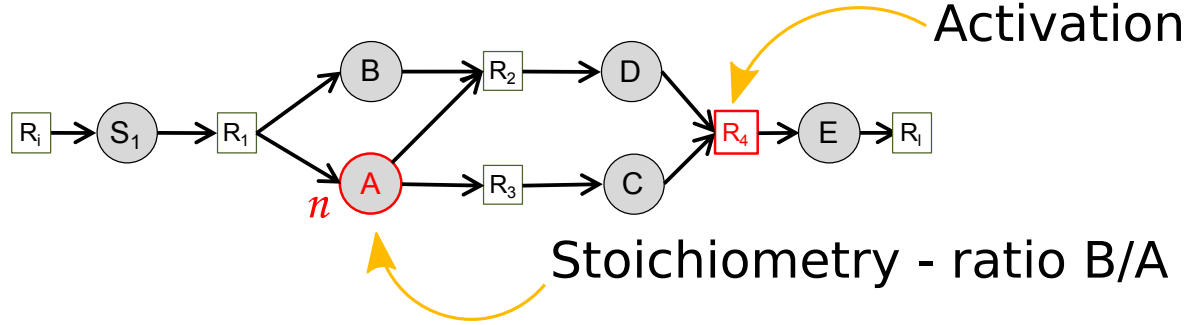
Replace kinetic constants by conservation laws and global optimisation hypotheses

Consequences of the approximation

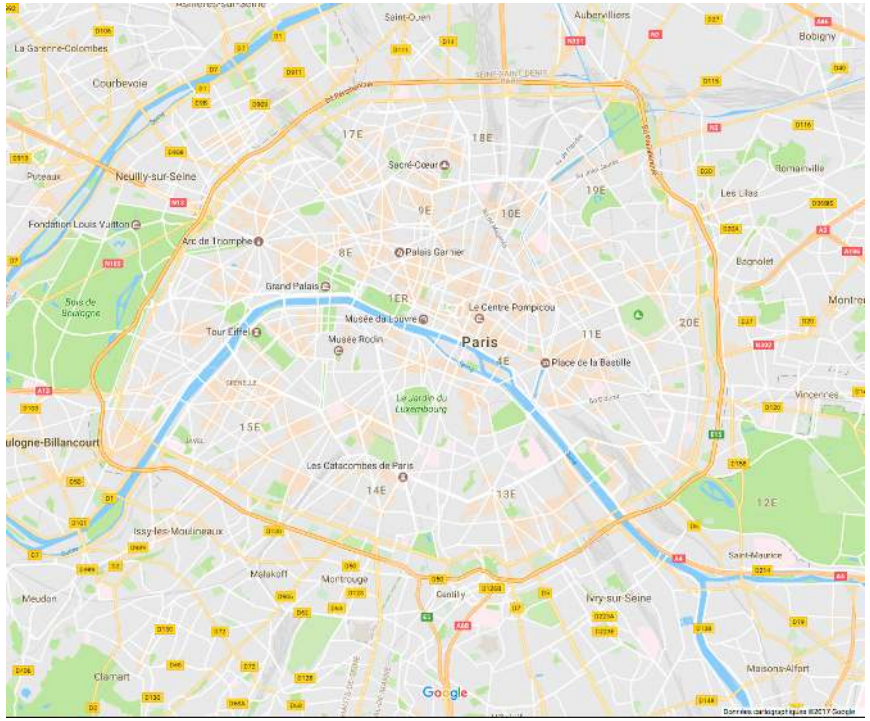


No if QSS

Yes if QSS and n=2



Modelling information



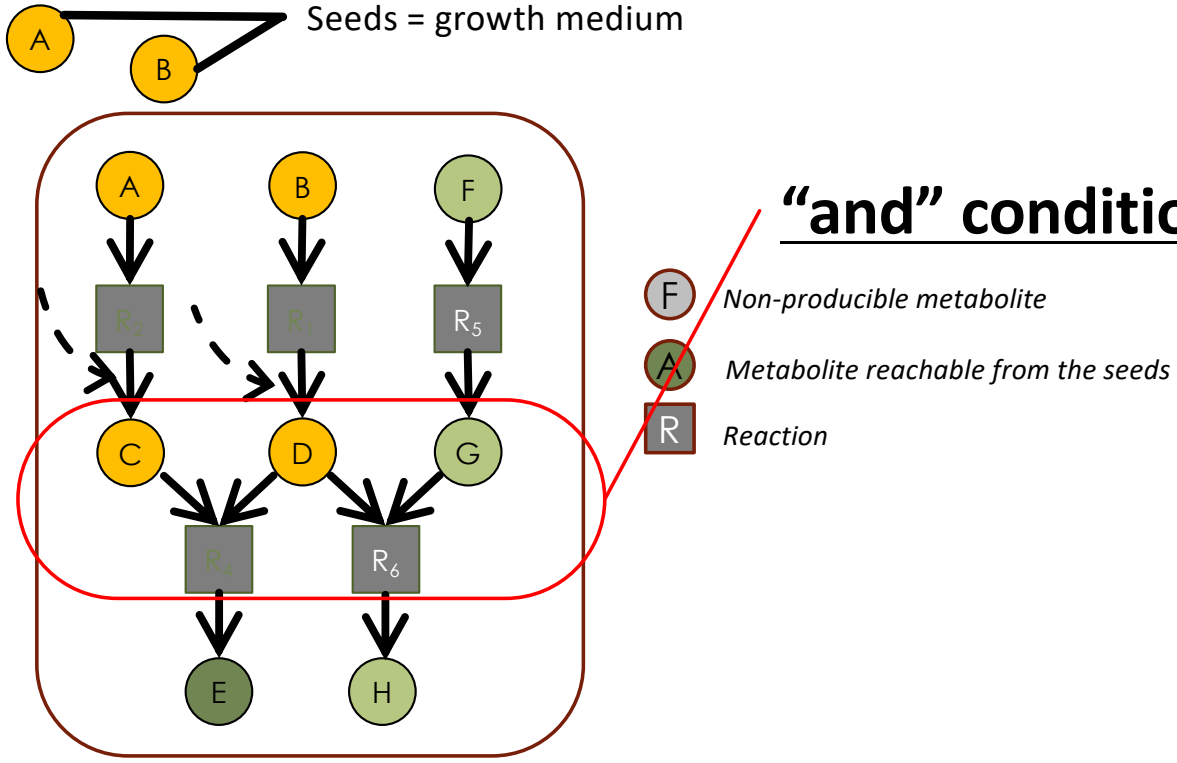
Model links between reactions:

fluxes… graphs… or intermediary formalism ?

Boolean approximation

Functionality: fixed-points of a Boolean network

→ recursive graph-based semantics

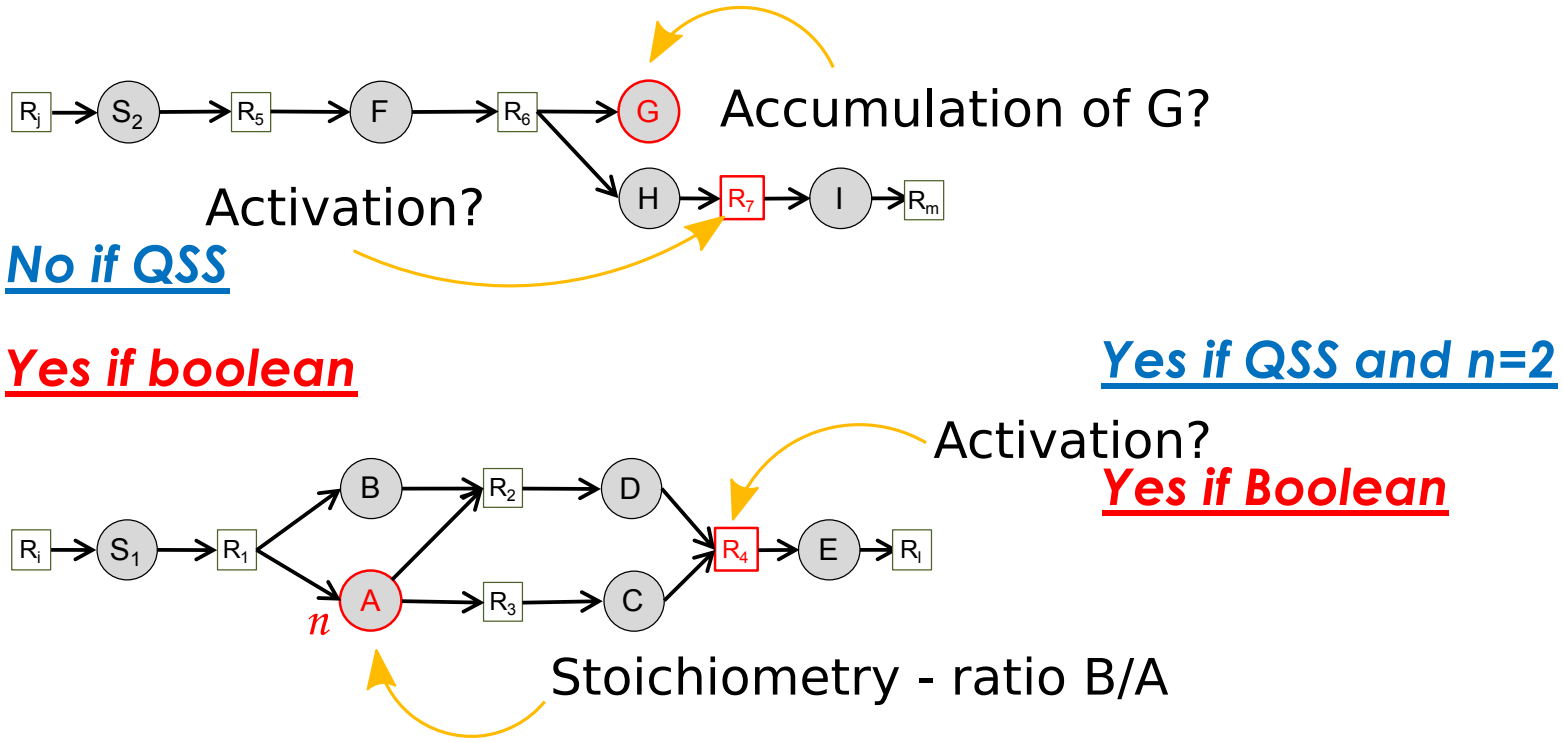


“and” condition checked recursively

```
scope(M):- seed(M).
scope(M):- product(M,R), reaction(R),  scope(M') : reactant(M',R).
```

Study paths in hypergraphs

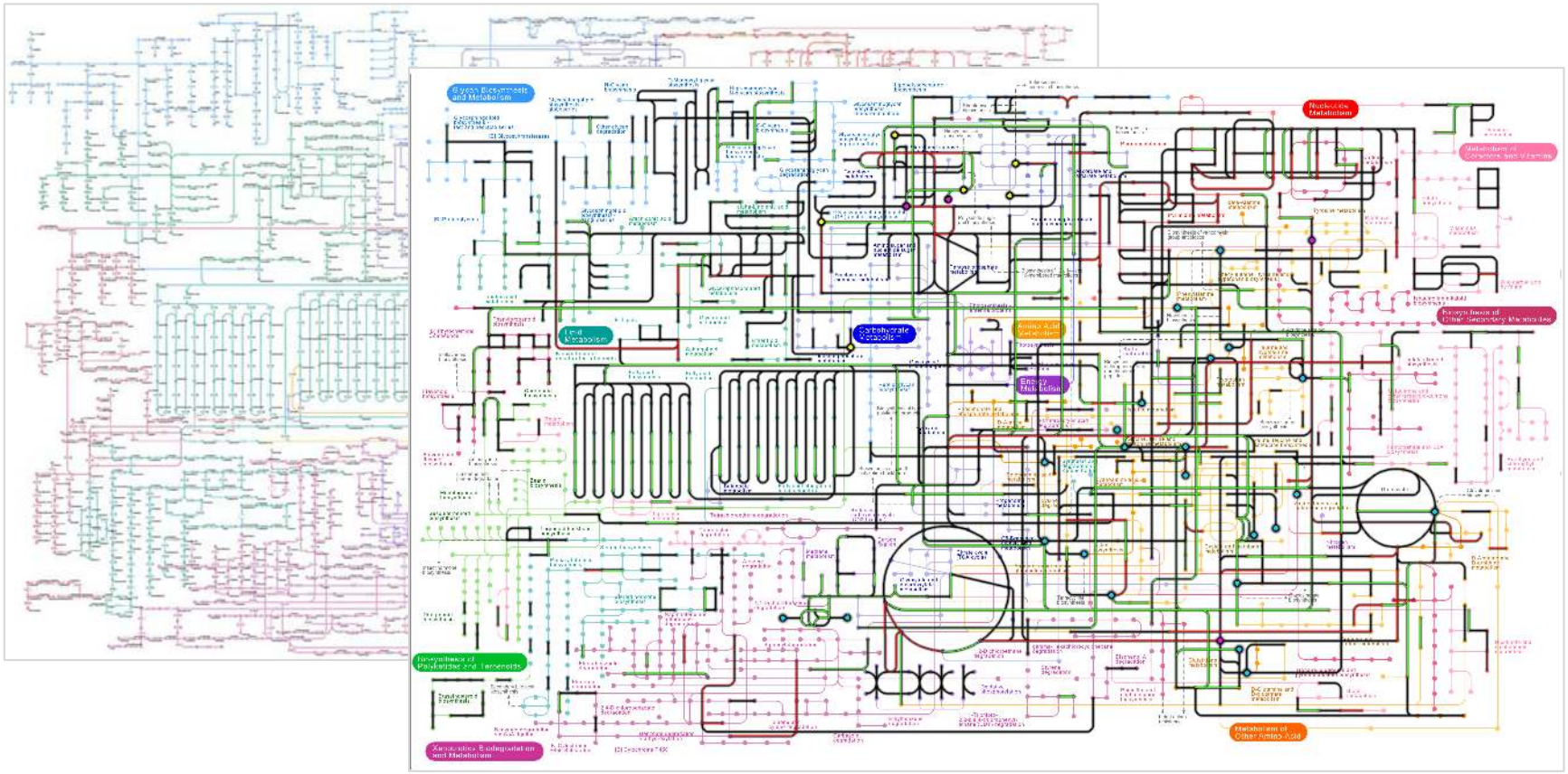
Everything is a matter of choices



The reaction status of the reactions is different according to the approximation

- No choice but dealing with such overapproximation !
- Use the flexibility of ASP language to handle these questions

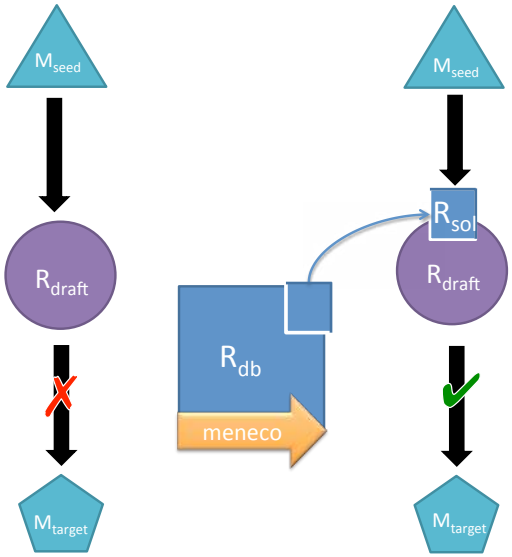
Data incompleteness



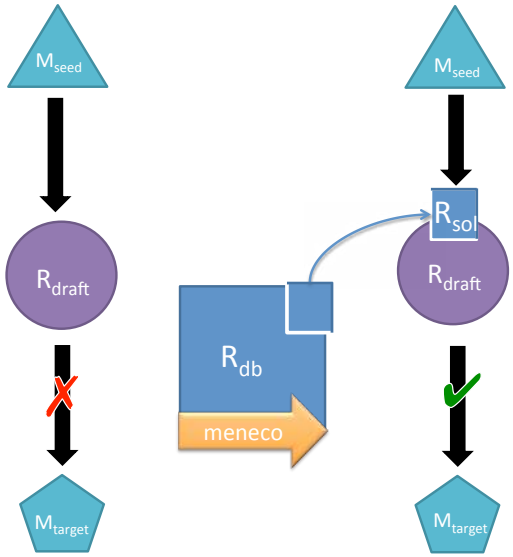
Metabolic networks built from NGS sequencing

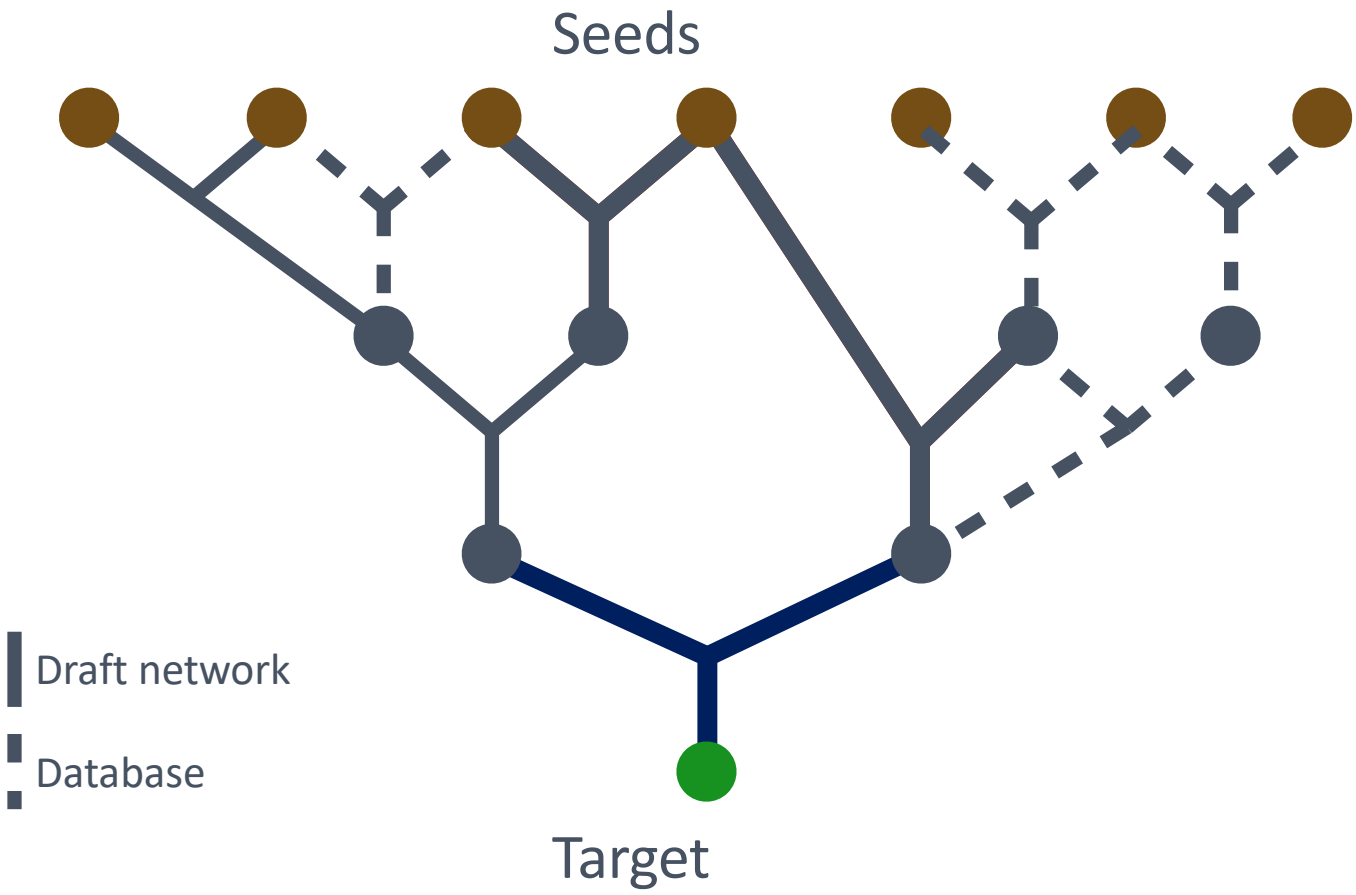
- no possible biomass production.

Gapfilling a metabolic network (nutshell)

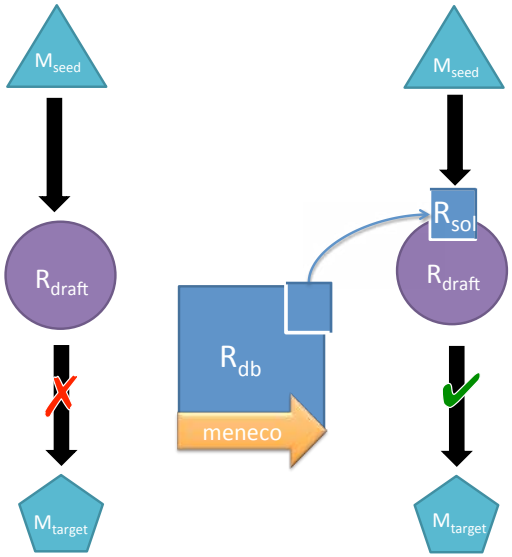


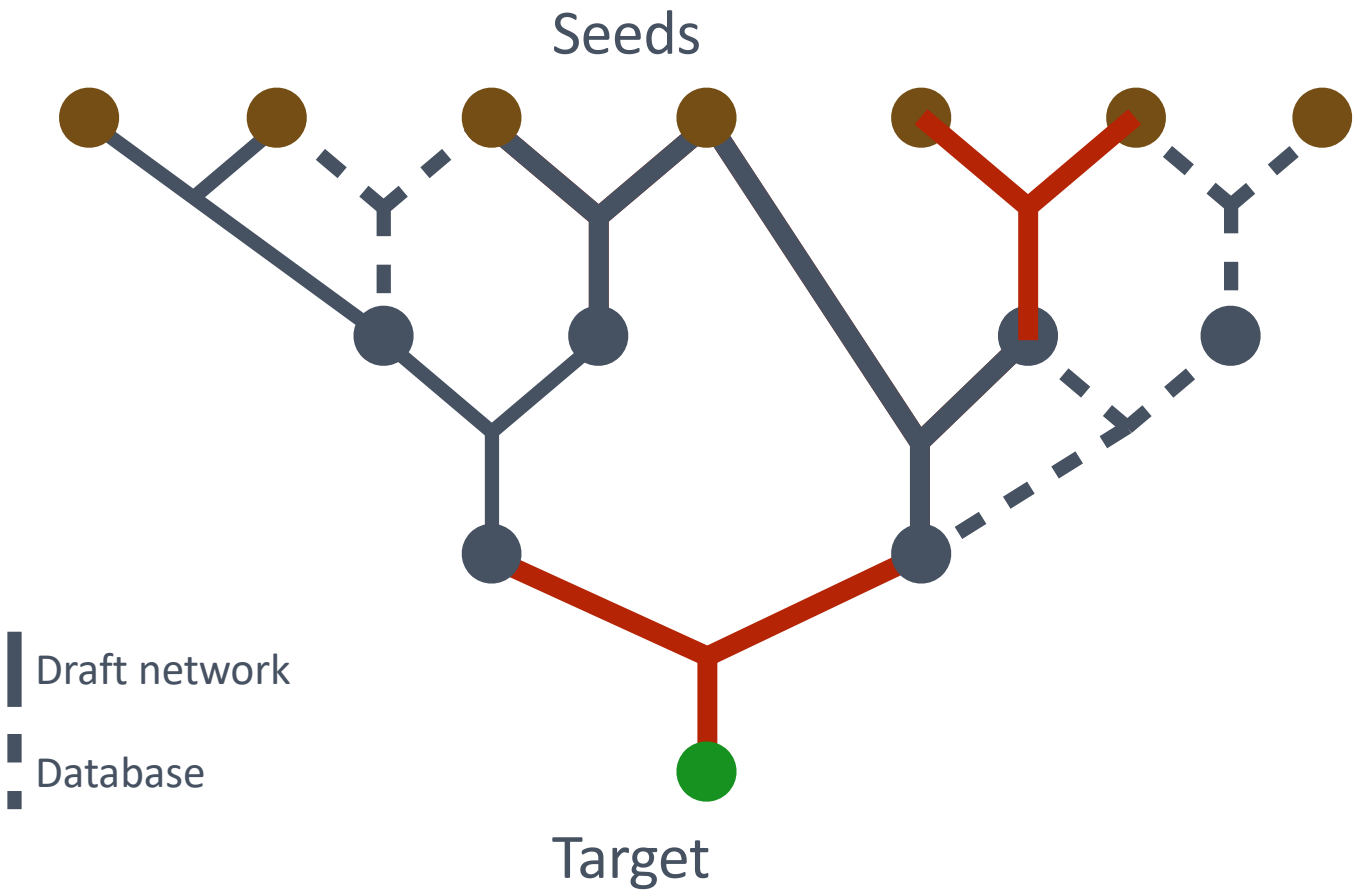
Gapfilling a metabolic network (nutshell)



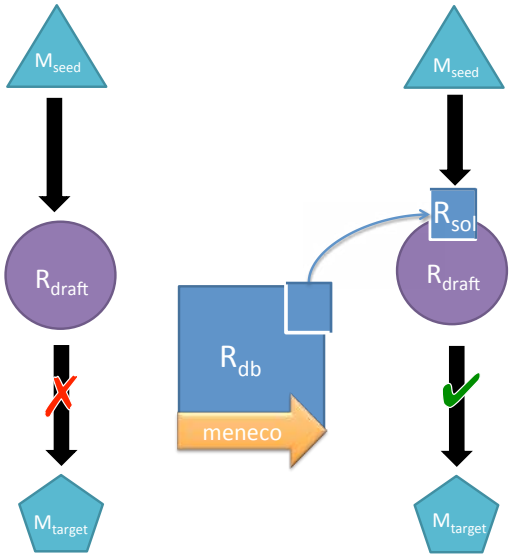


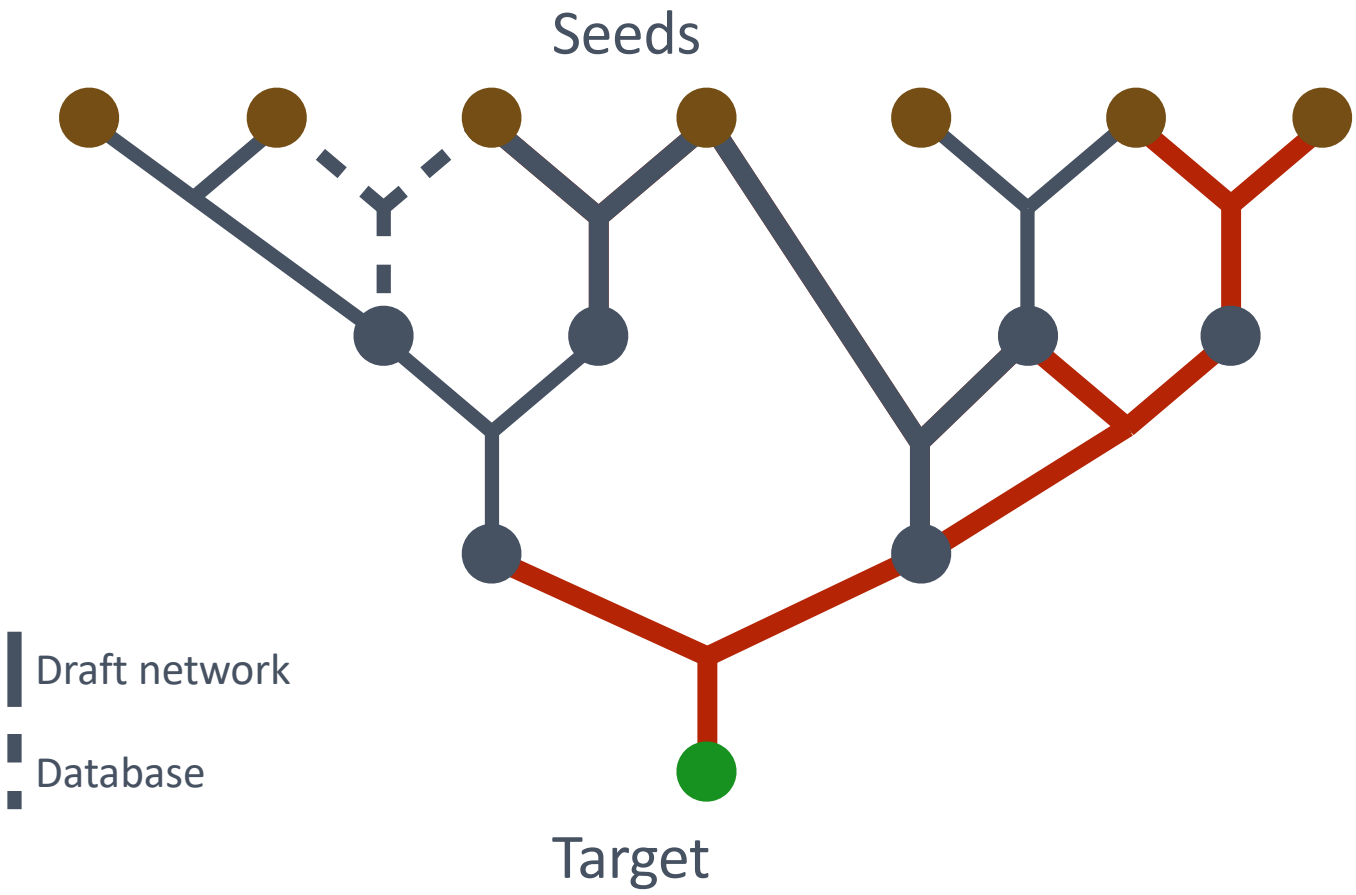
Gapfilling a metabolic network (nutshell)



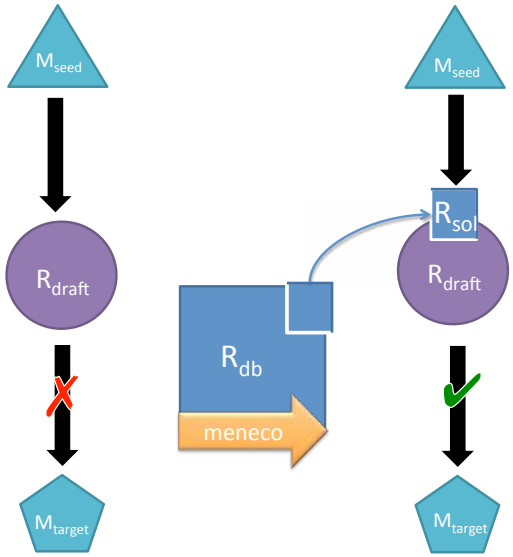


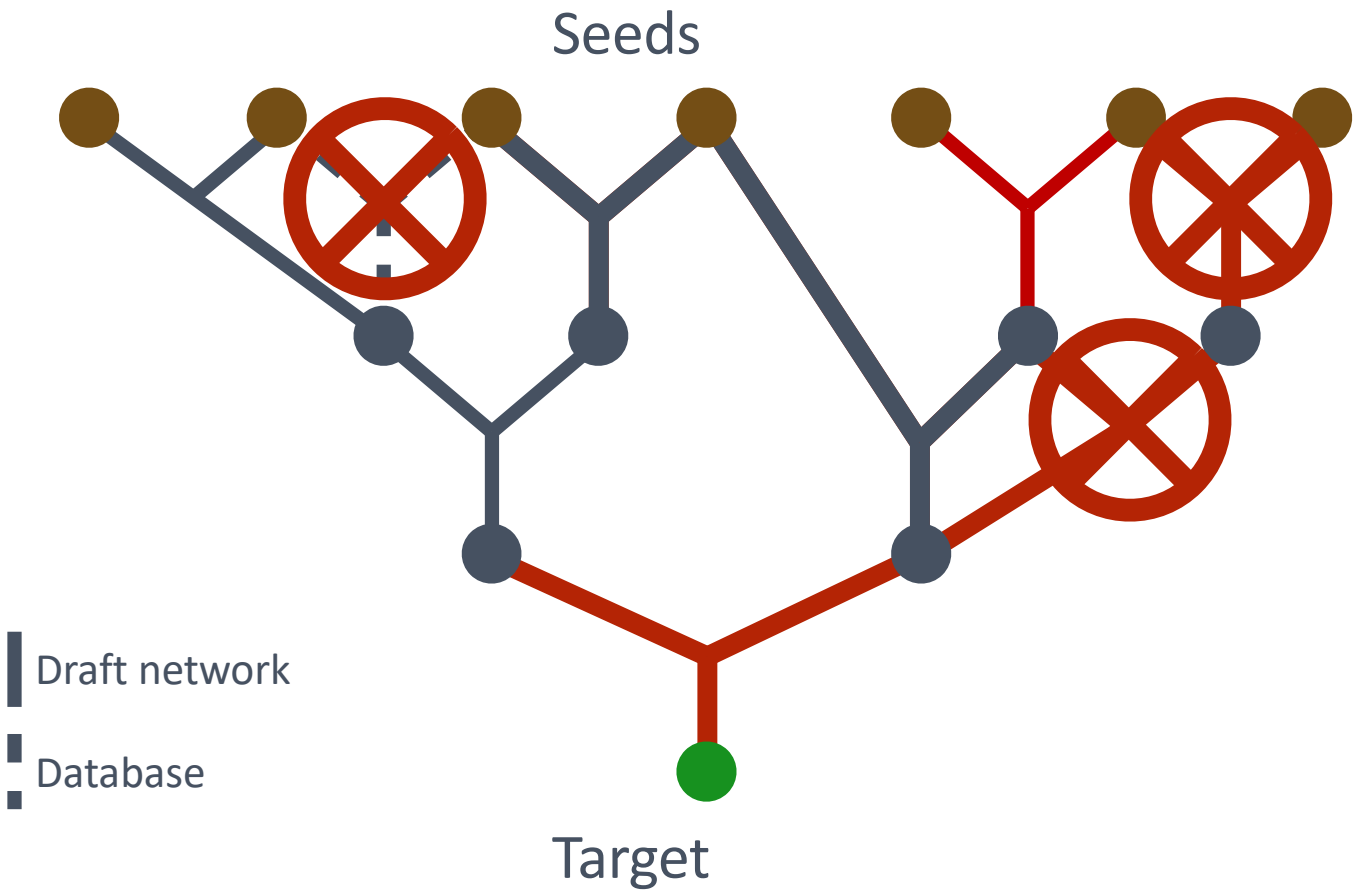
Gapfilling a metabolic network (nutshell)





Gapfilling a metabolic network (nutshell)

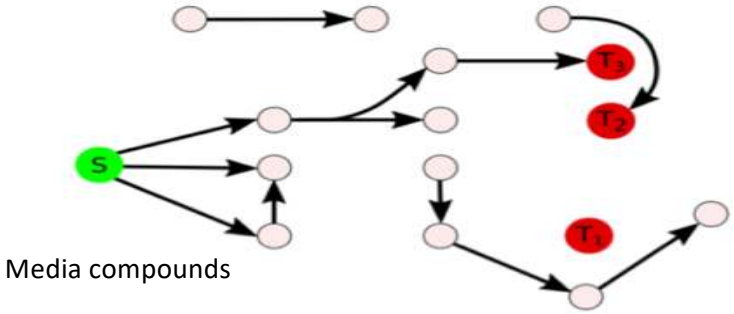




Gapfilling a metabolic network

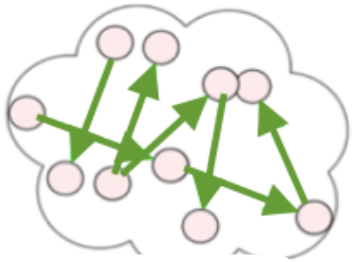
What we have

- Graph with **non-accessible target components**
- **Knowledge database** of possible edges



Media compounds

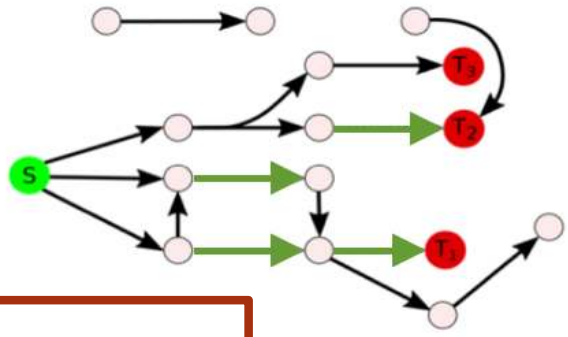
Experimentally observed compounds



Putative interactions from knowledge databases

Gap-filling problem:

- Restore target accessibility
- Minimal number of reactions



$$\text{gapfilling}(\mathrm{S}, \mathrm{R_T}, \mathrm{G_1}, \mathrm{G_{DB}}) = \underset{\{R_i..R_m\} \subset G_{DB}}{\arg\min} \left(\frac{size(reactants(R_T) \setminus scope(G_1 \cup \{R_i..R_m\})}{size\{R_i..R_m\}} \right)$$

Use our strategy…

Describe a system by a family of abstract models

Discrete dynamical systems

- Abstract flux-based non-monotonous criteria by a monotonous recursive topological criteria

(Logical) knowledge representation

- Implicit modeling of recursive propagation
- Integrity constraints to reduce the search space
  Use constraints propagators for fluxes

Solving optimisation problems

- Use relevant searching methods
- Enumerate all solutions !

[LPNMR'13, LPNMR'17]

Meneco: ASP-based gap-filling for non-model organisms

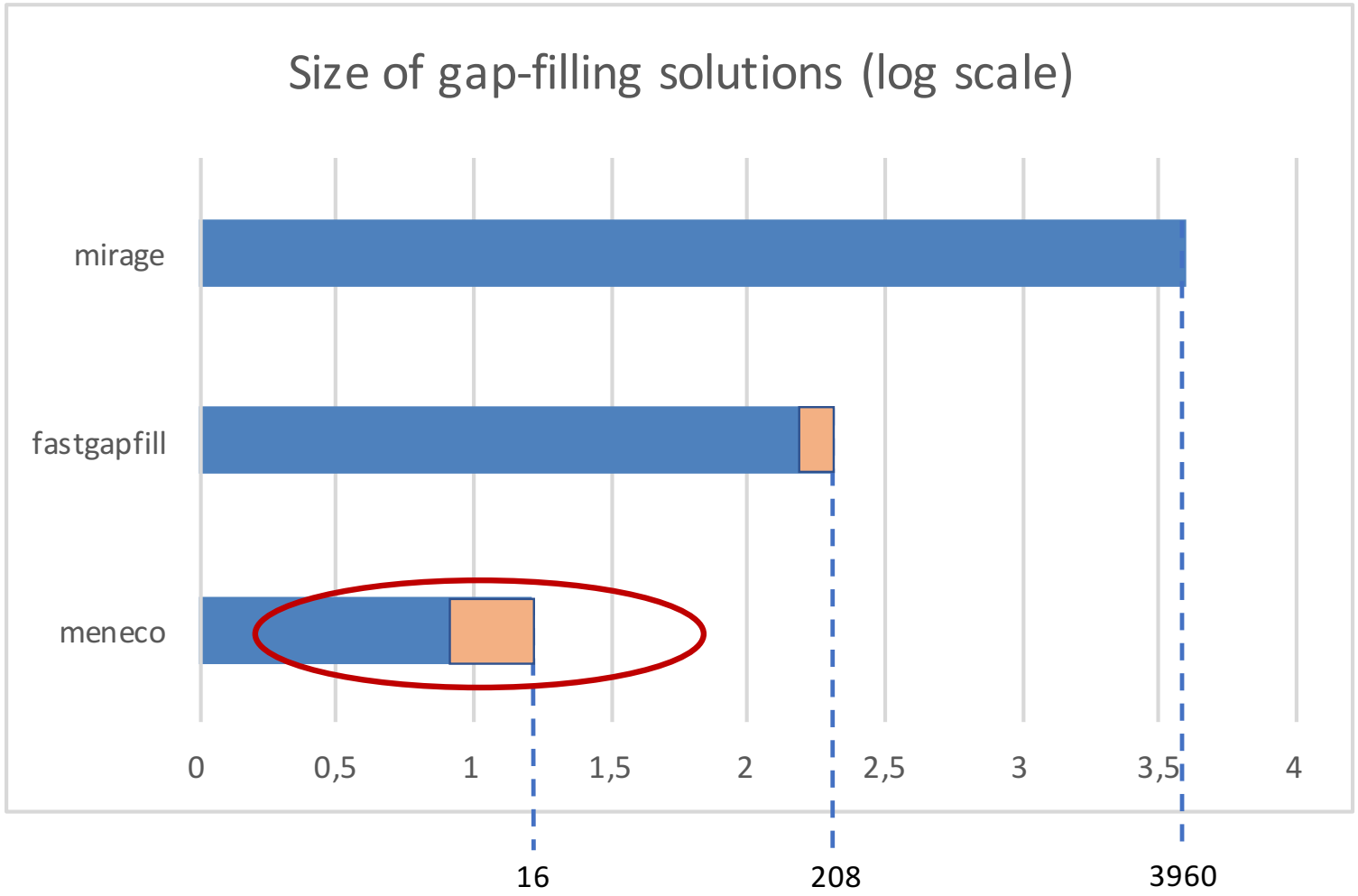
$$\text{gapfilling}(\mathrm{S}, \mathrm{R_T}, \mathrm{G_1}, \mathrm{G_{DB}}) = \underset{\{R_i..R_m\} \subset G_{DB}}{\arg\min} \begin{pmatrix} size(reactants(R_T) \setminus scope(G_1 \cup \{R_i..R_m\}) \\ size\{R_i..R_m\} \end{pmatrix}$$

```
{reaction(r)}.
scope(M):- seed(M).
scope(M):- product(M,R), reaction(R), scope(M') : reactant(M',R).
:- target(T), not scope(T).
#minimize{ reaction(r) }.
```

Meneco : ASP-based encoding

- Enumeration of all solutions.
- Very efficient.
- Requires cycles to be externally initiated

Benchmarking gap-filling strategies

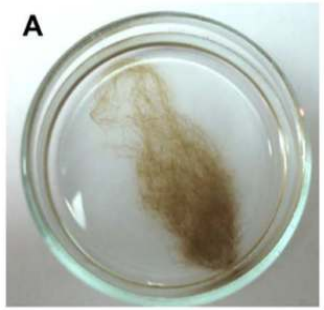


Benchmark of 10,800 bacterial networks

16 reactions in average are sufficient to restore degraded bacterial networks (PLOS CB 2017)

- MILP-based approaches required from 200 to 4000 reactions.

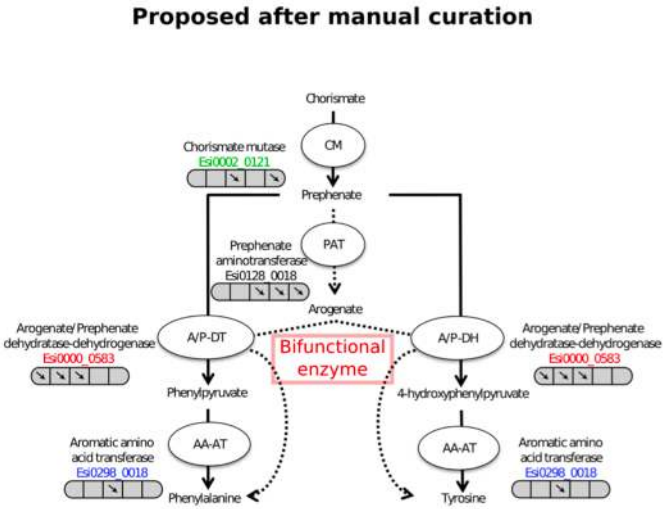
Example of application



Ectocarpus siliculosus

[Tapia2016]

- **Genome**: 1785 reactions, 1981 compounds
- **54 metabolites to produce:**
  - 25 are graph-based producible
  - None is FBA-based producible.
- **Gapfilling**
  - MILP : 500 reactions (untractable)
  - ASP: 50 reactions added to the network
    - Sufficient for fluxes
    - Manual curation



New bifunctional role of a specific enzym

(Plant Journal 2015)

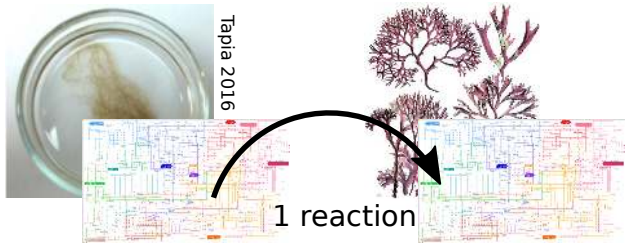
CNRS • SORBONNE UNIVERSITÉ
Station Biologique de Roscoff

Counter-example of application

Chondrus crispus

Network analysis (G. Markov, SBR)

- 1943 reactions
- 149 reactions added by ASP
- **No way to produce biomass**

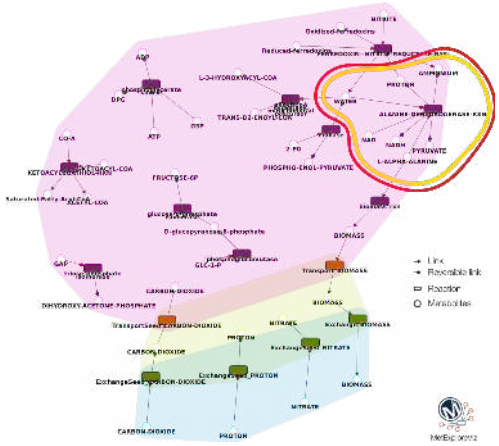


New problem to be solved

- **Hybrid problem** (TPLP 2018)
- Constraint propagator
- Reduce the database

$$\text{Hybgapfilling}(\mathrm{S}, \mathrm{R_T}, \mathrm{G_1}, \mathrm{G_{DB}}) = \underset{\{R_i..R_m\} \subset G_{DB}}{\arg\min} \begin{pmatrix} size(reactants(R_T) \setminus scope(G_1 \cup \{R_i..R_m\})) \\ size\{R_i..R_m\} \end{pmatrix}$$

$$\text{s.t} \quad s.v = 0, v_{R_T} > 0,\ lb < v < ub$$



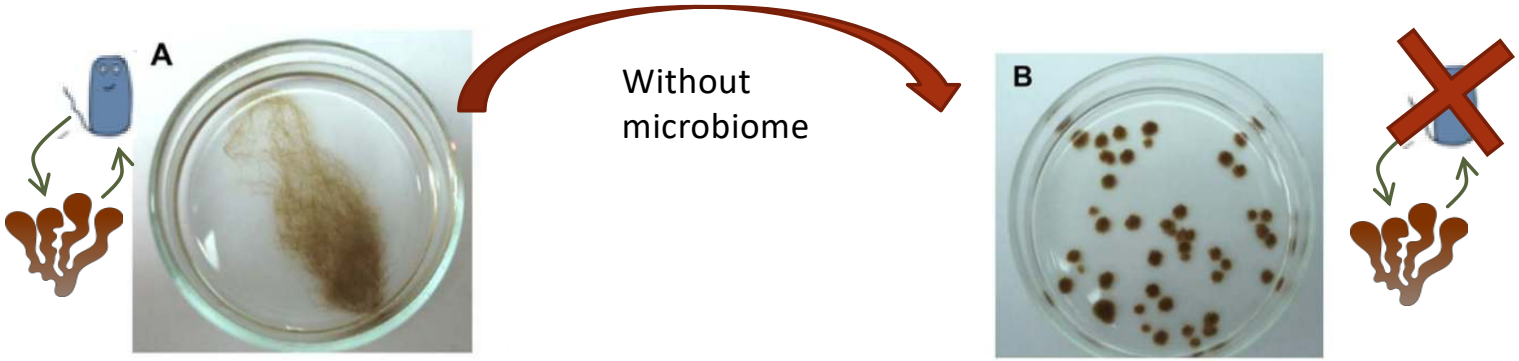
Essential reactions for alanine production in *CcrGem*

Still more complexity: microbial communities

- Prigent et al, PLOS computational Biology, 2017.
- Frioux et al, ECCB/bioinformatics

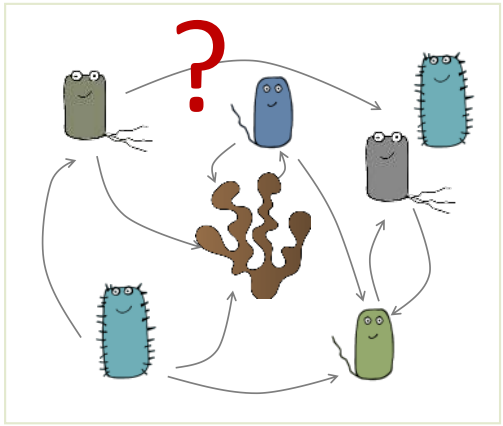
CNRS UPMC
Station Biologique
Roscoff

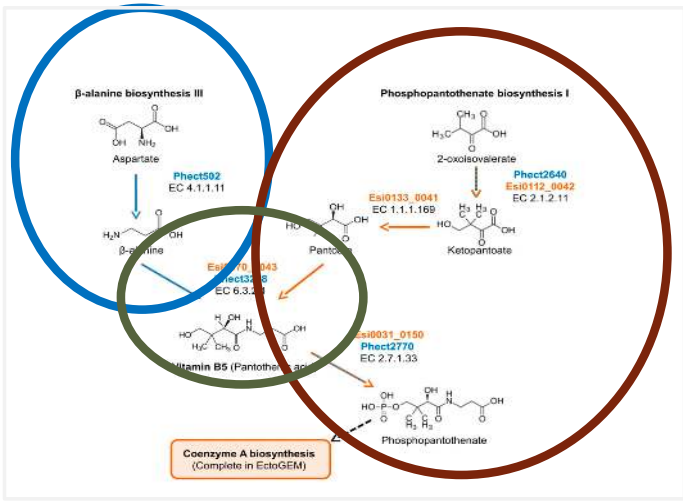
Role of environmental bacteria ?



Ectocarpus
[Dittami2014, Tapia2016, Prigent2015]

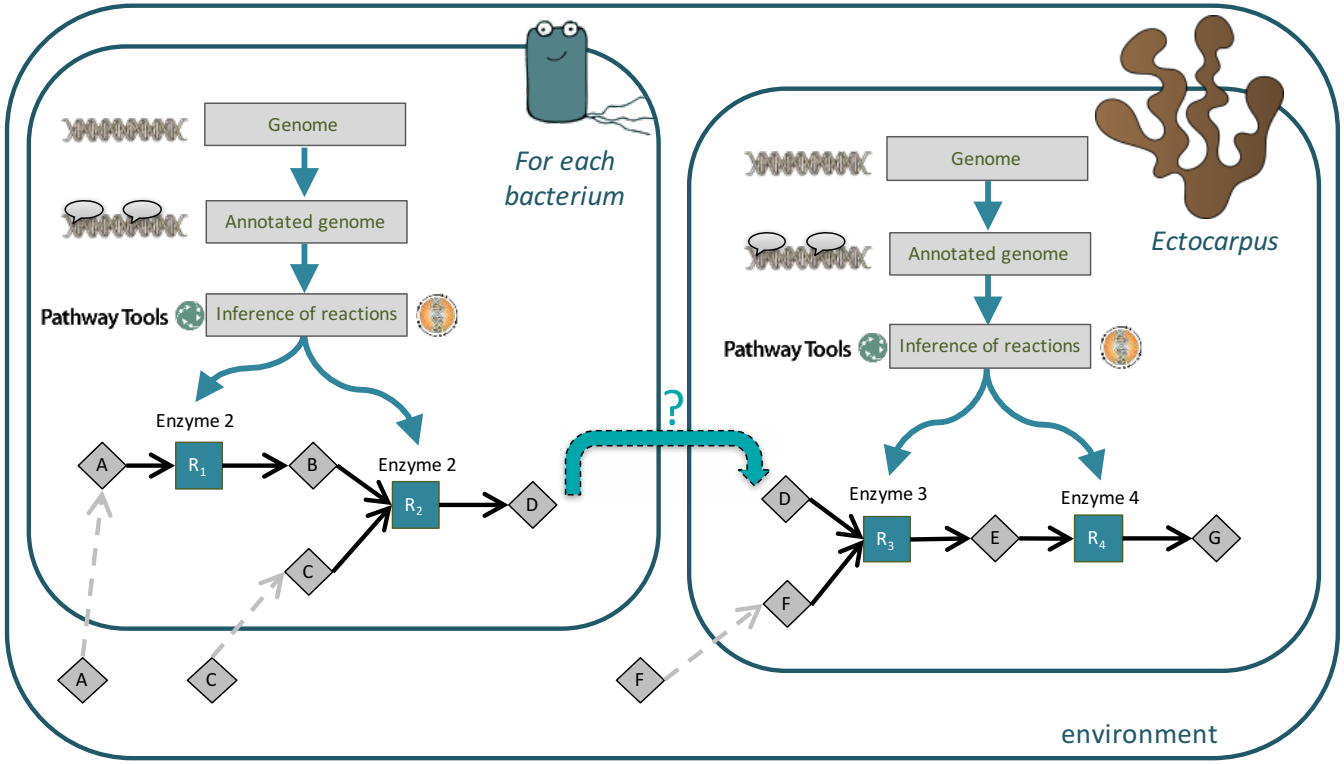
CNRS • SORBONNE UNIVERSITÉ
Station Biologique
de Roscoff





Metabolism may be an explanation
(PLOS CB 2017)

Systems ecology question

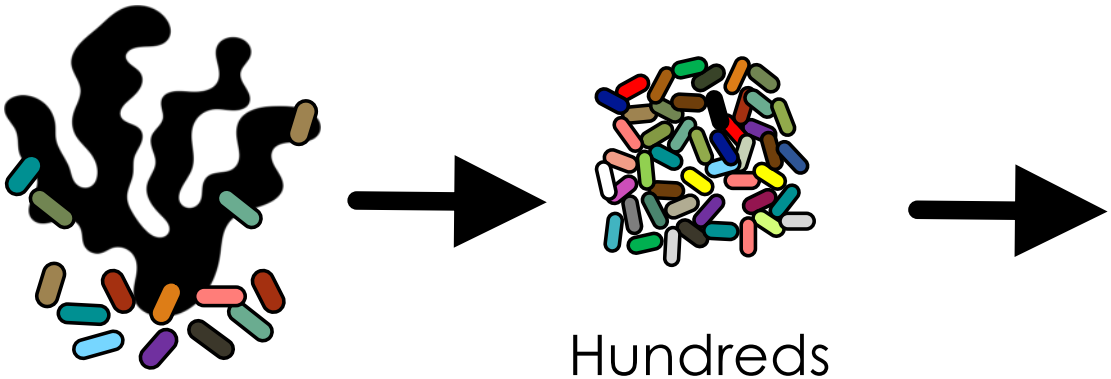


Can we suggest compound exchanges that could restore the production of targeted compounds ?

- New gap-filling problem !
- Steiner graph approach (Sagot team, 2017) or ASP implementation

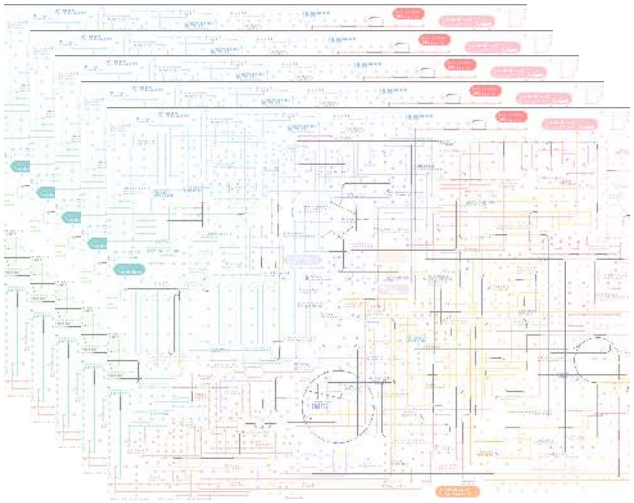
Scalability…

But… There are hundreds of bacteria in the environment



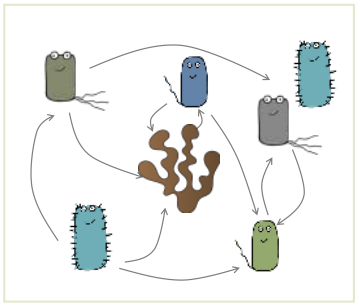
Marine biology

Hundreds
of bacteria



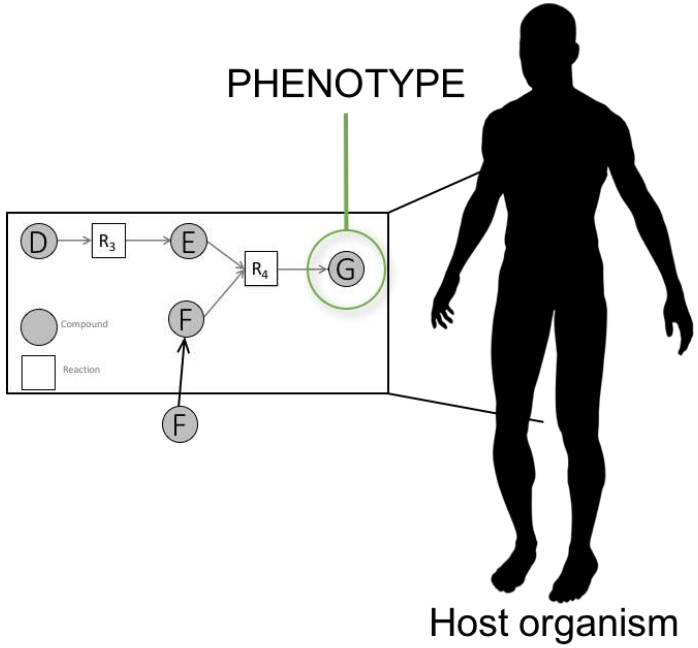
Hundreds of Genome-scale models (GSMs)

Happy few bacteria interact with the algae



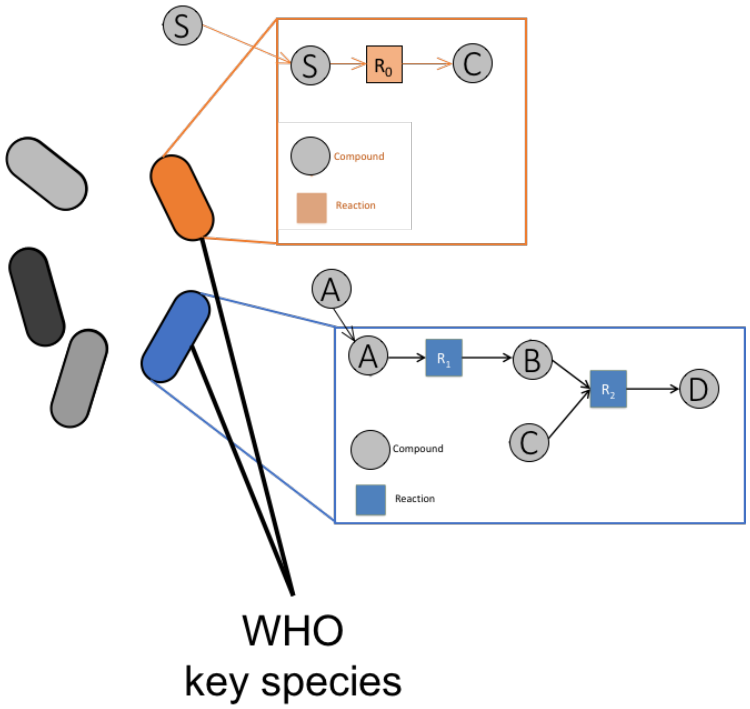
How to select communities within large microbiotas which explain the algal response to stress ?

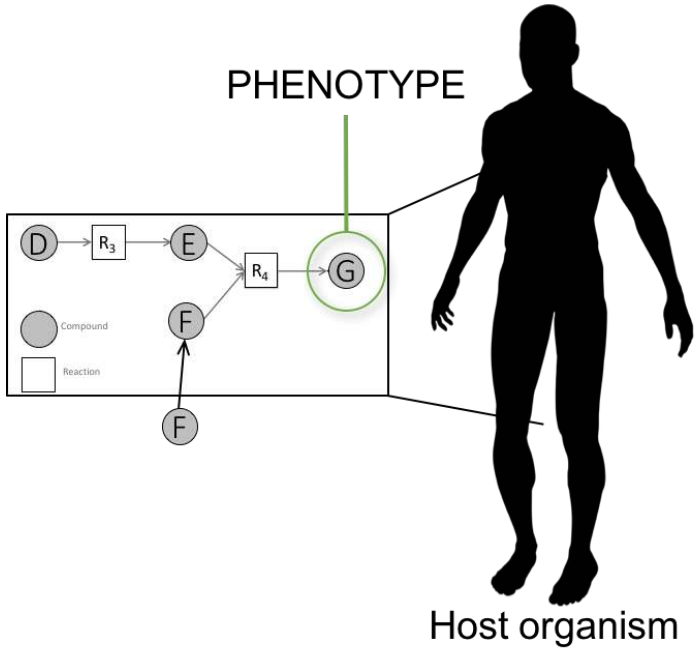
Selecting communities of interest within [large] microbiotas



The “who”, “how” challenges of community selection

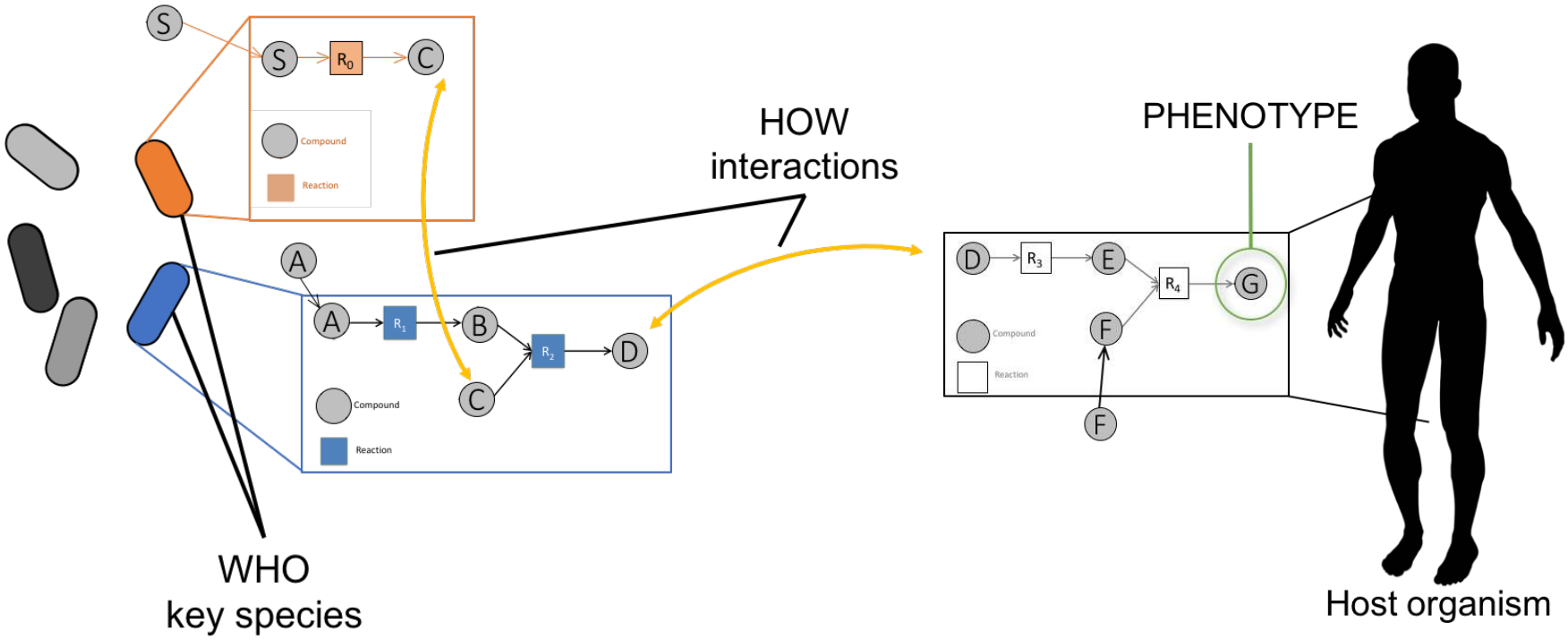
Selecting communities of interest within [large] microbiotas





The "who", "how" challenges of community selection

Selecting communities of interest within [large] microbiotas



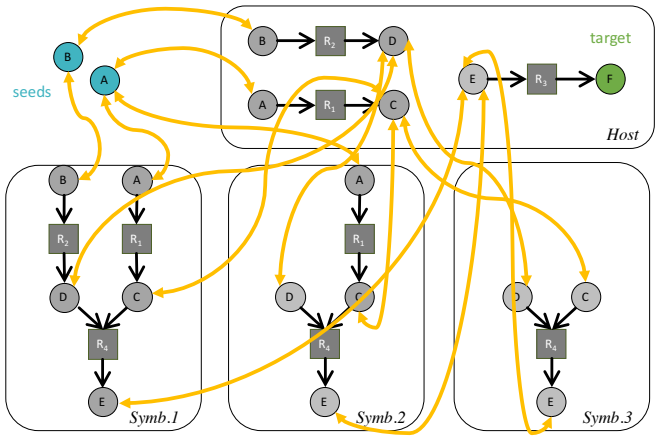
The "who", "how" challenges of community selection

Complexity

Community selection problem

- Switch from hundreds of symbiots to 3 or 4
- Pinpoint a few number of putative cross-feedings

$$\mathrm{Comsel}(S, T, G_1 .. G_n) = \underset{\{exchg(G_{i_1}..G_{i_L})\} \subset \{G_1..G_n\}}{\arg\min} \begin{pmatrix} size(T \setminus MBscope(G_{i_1}..G_{i_L}) \\ size\{\varepsilon \subset exchg(G_{i_1}..G_{i_L}) \mid \\ T \cap CPscope(G_{i_1}..G_{i_L}, \varepsilon, S) = \\ T \cap MBscope(G_{i_1}..G_{i_L}, S)\} \end{pmatrix}$$



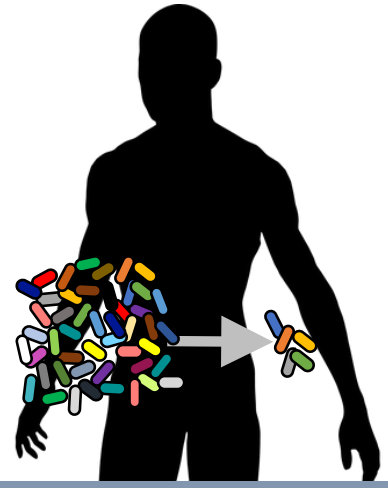
499,177 combinations of <6 exchanges

Gapfilling complexity

- depends on the number of hyperarcs

Size of the search space

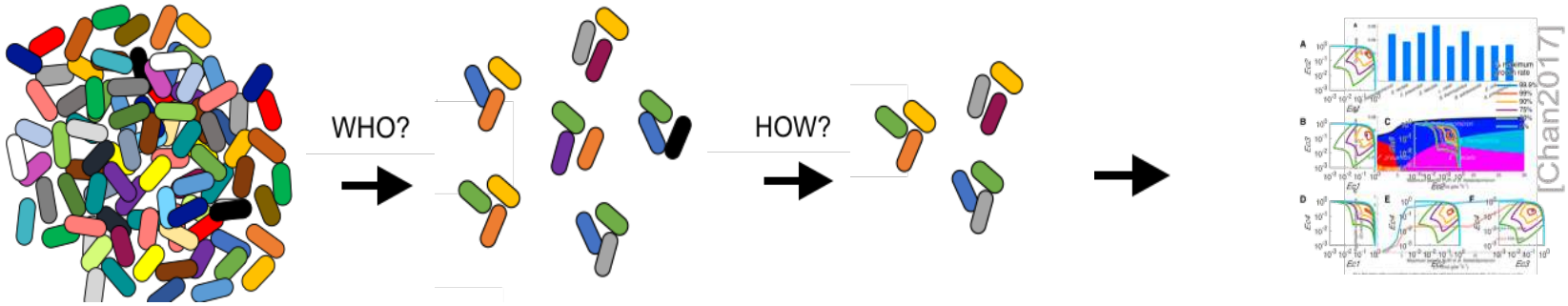
- depends on the number of symbionts



1.62.10^{81} combinations of <10 exchanges

Highly combinatorial problem

Two-step optimization procedure



Heuristics for the community selection problem

- **Who problem.**
  - Get rid of boundaries and select all minimal symbiot families

$$\text{mxdbagCnity}(S, T, G_1..G_N) = \underset{\{G_{i_1}..G_{i_L}\} \subset \{G_1..G_N\}}{\arg\min} \begin{pmatrix} \text{size}\left(T \backslash \text{mxdbagScope}(G_{i_1}..G_{i_L}, S)\right), \\ \text{size}\{G_{i_1}..G_{i_L}\}. \end{pmatrix}$$

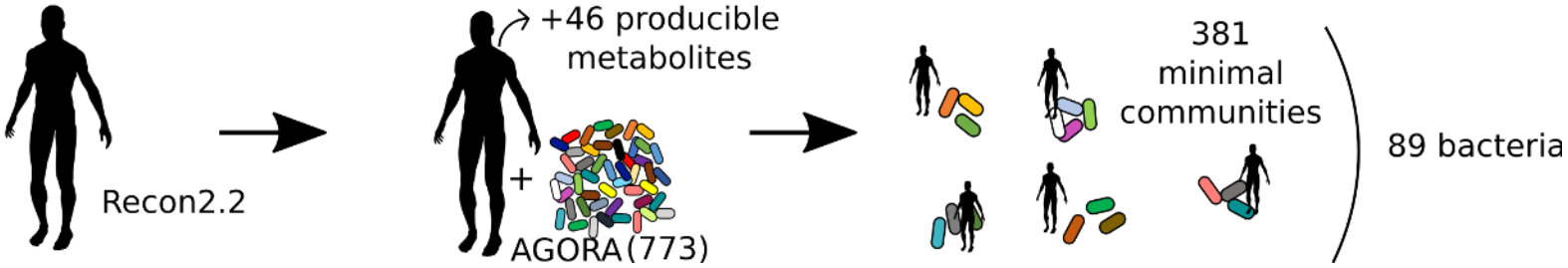
- **How problem.**
  - Sort the selected families according to the number of exchanges

$$\text{cptCnity}(S, T, G_1..G_N) = \underset{\substack{\{G_{i_1}..G_{i_L}\} \\ \subset \{G_1..G_N\}}}{\arg\min} \begin{pmatrix} \text{size}\left(T \backslash \text{mxdbagScope}(G_{i_1}..G_{i_L}, S)\right), \\ \text{size}\{G_{i_1}..G_{i_L}\}, \\ \text{size}\{\mathcal{E} \subset \text{exchg}(G_{i_1}..G_{i_L}) | \\ \quad T \cap \text{cptScope}(G_{i_1}..G_{i_L}, \mathcal{E}, S) \\ \quad = T \cap \text{mxdbagScope}(G_{i_1}..G_{i_L}, S)\}. \end{pmatrix}$$

- **Manual curation.**
  - Ask your favorite biologist to select the final one

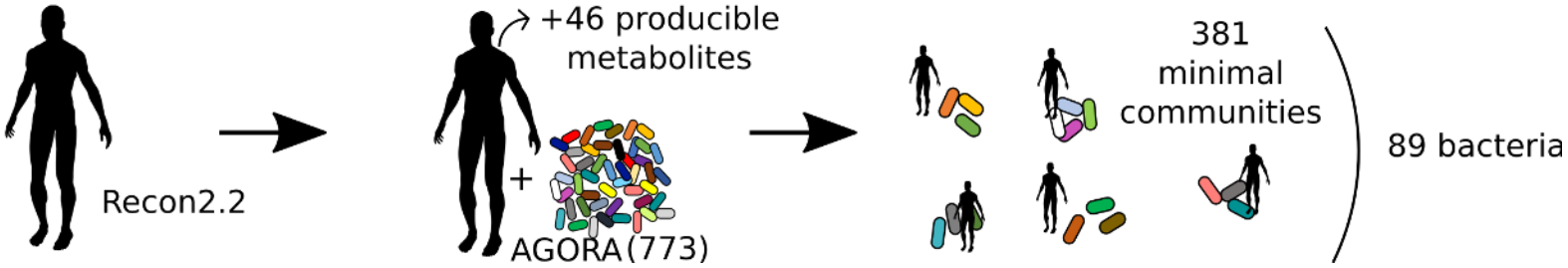
Validation/benchmarking on human microbiome project

Context of the study [Swainston et al., 2016] [Magnúsdóttir et al., 2016]

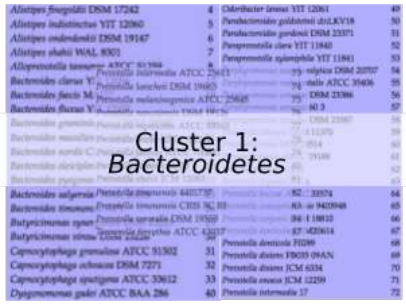


Validation/benchmarking on human microbiome project

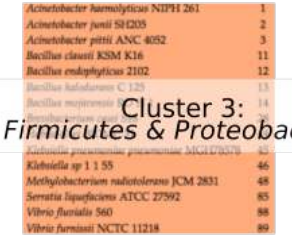
Context of the study [Swainston et al., 2016] [Magnúsdóttir et al., 2016]

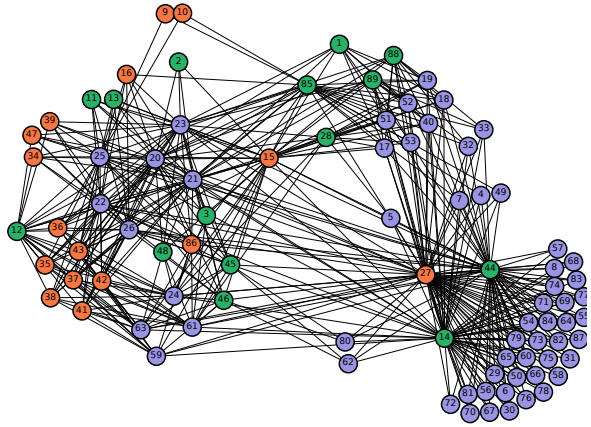


Clustering of bacteria


Cluster 1:
Bacteroidetes


Cluster 2:
Firmicutes & Proteobacteria


Cluster 3:
Firmicutes & Proteobacteria

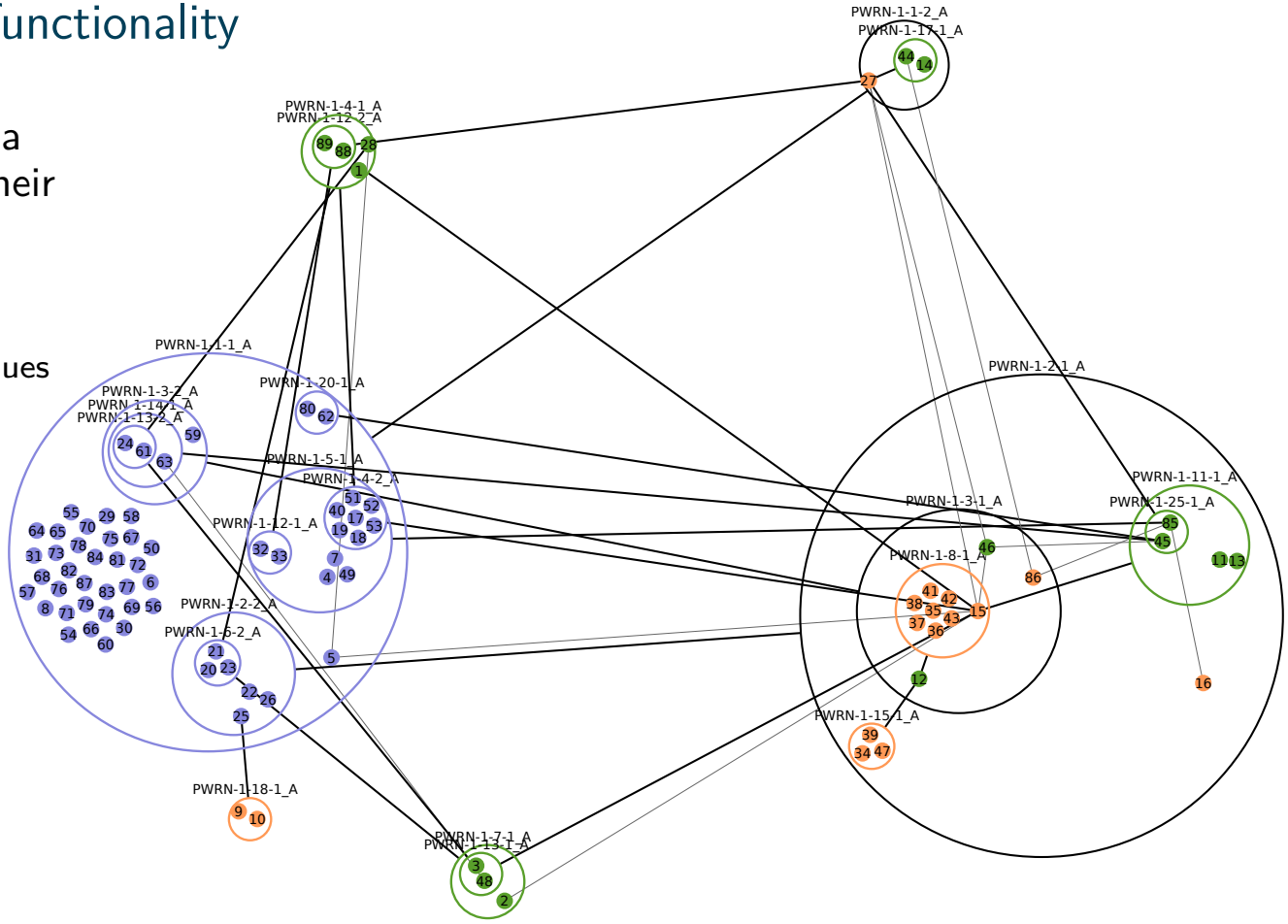


Each of the 381 communities is composed of
1 *Bacteroidetes* (/58) + 1 *Firmicute* or *Proteobacteria* (/15) + 1 *Firmicute* or *Proteobacteria* (/16)

Validation/benchmarking on human microbiome project

Association of bacteria & functionality

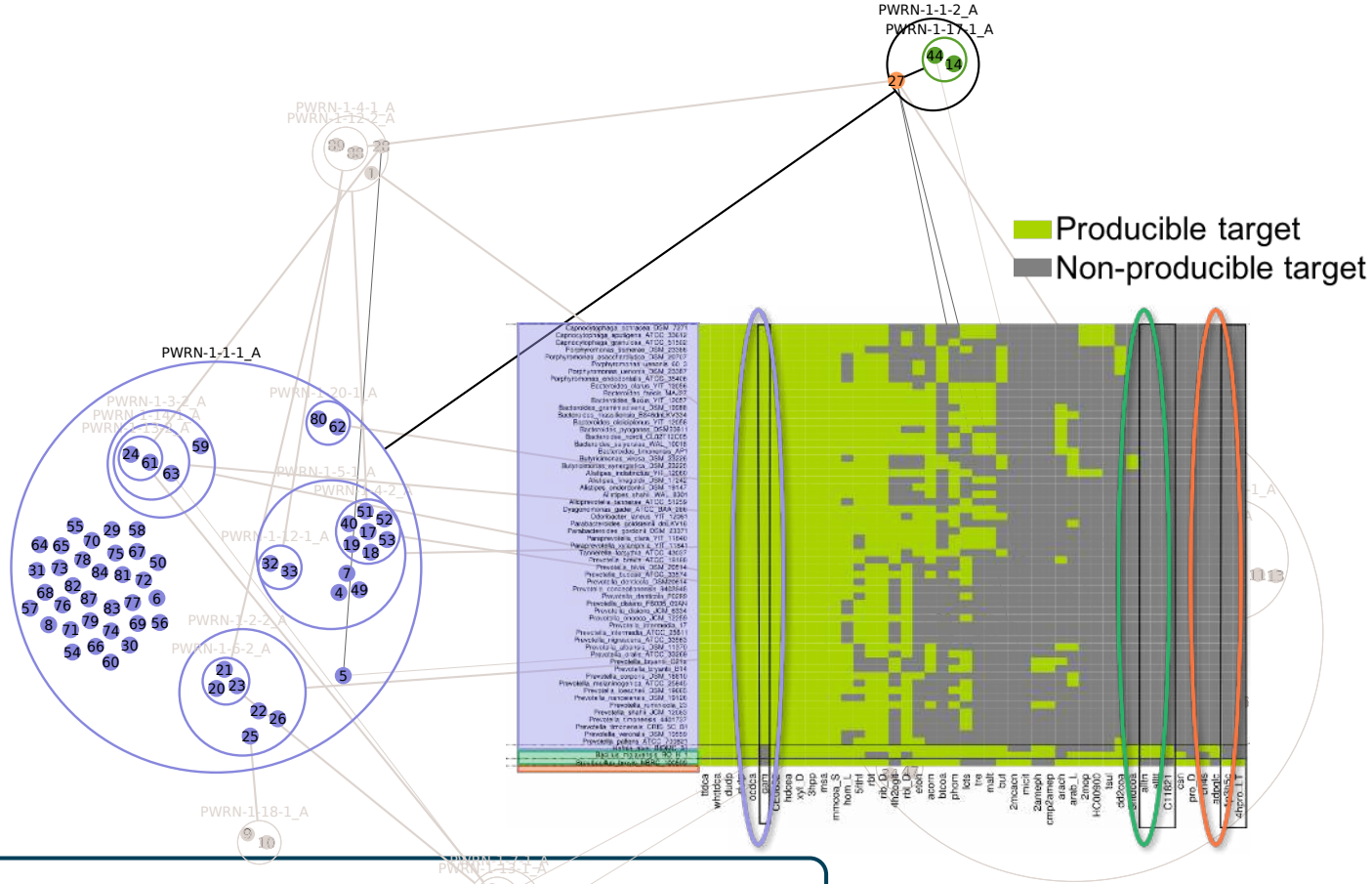
- Groups of equivalent bacteria in clusters with respect to their associations [Bourneuf et al., 2017]
  - **Powernodes:** groups of bacteria, parts of bicliques
  - **Poweredges:** connect bicliques



C. Frioux's Thesis. ECCB 2018

Validation/benchmarking on human microbiome project

- Producibility of individual targets explains the communities → screening



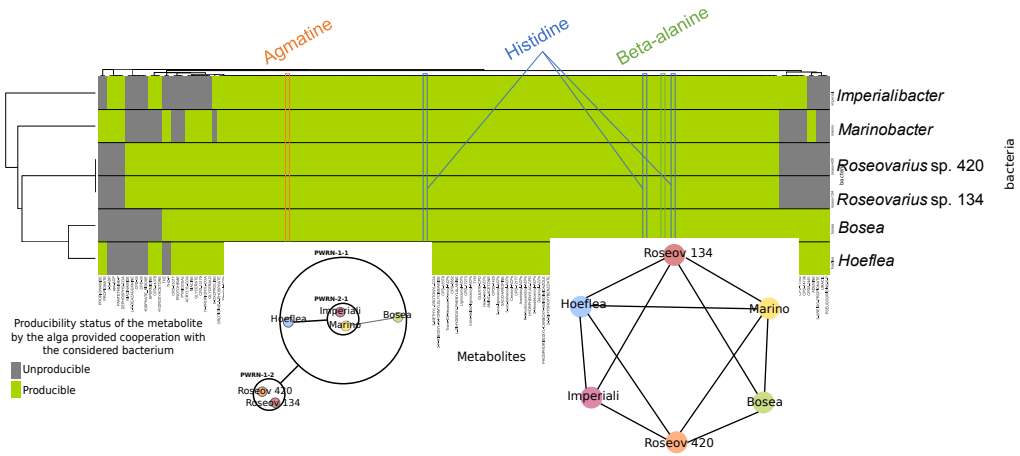
Community composition can be explained by the functional dependencies of the targets towards specific groups of bacteria

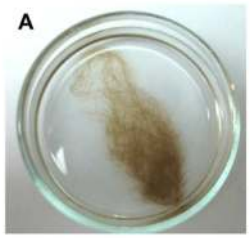
C. Frioux's Thesis. ECCB 2018

Validation on algae

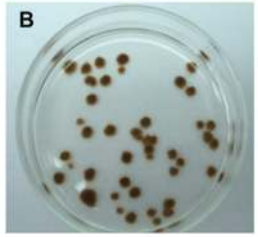
- *Ca.* P. ectocarpi not culturable
- 10 culturable bacteria → functional redundancy
- 6 equivalent communities of 3 bacteria

Joint work with Enora Fremy, Bertille Burgunter-Delamare & Simon Dittami

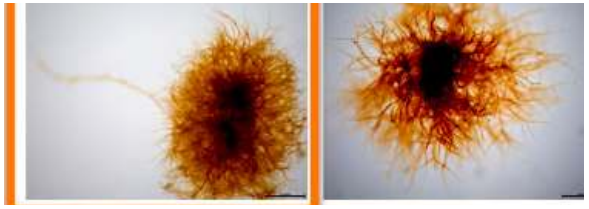




Without microbiome



+ 3 selected bacteria among 30 cultivable bacteria



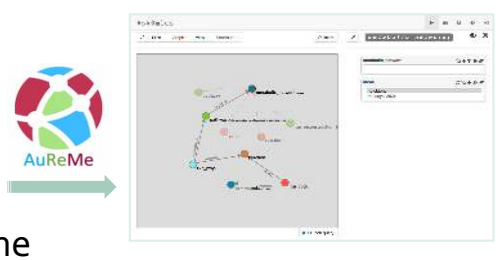
S. Dittami, Bertille Burgunter-Delamare

The algae grew again… But with strange behaviors

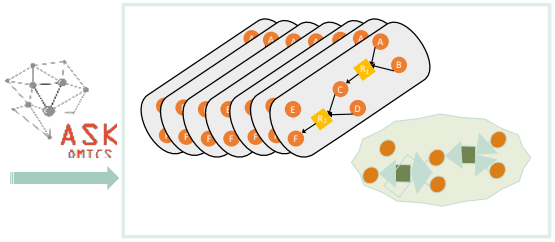
Prospective: semantic and combinatorial exploration of host-microbia consortia

- **Wiki** exploration of metabolic pathways metadata built from a-la-carte reconstruction pipeline (*AuReMe*)
- Construction and exploration of data links with **semantic queries** (*AskOmics*)
- **Functional screening of host-microbial consortia** (*MiSCoTo*)

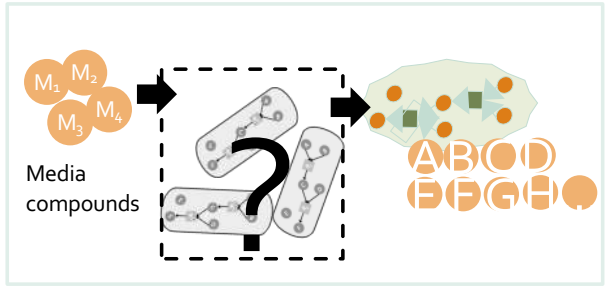
Host and microbiome genomic information



(1) Wiki GSMs **query**

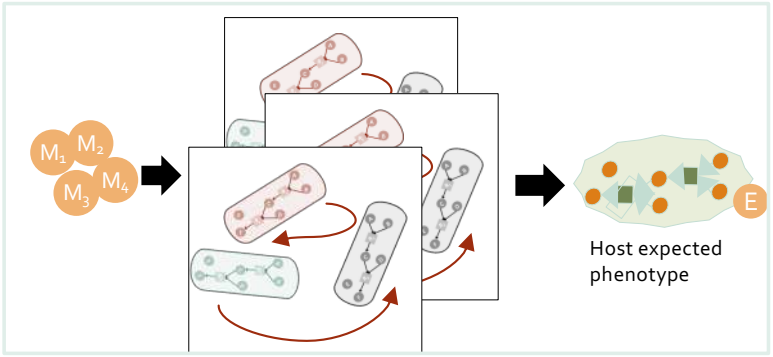


Bacteria and host metabolic models



(3) Compounds production screening
added-value of the consortium for the production of metabolic compounds

(4) Manual selection of communities
(functional explanation, co-culture, transport feasibility, ...)



(3) Microbial consortia prediction: families of bacteria facilitating an expected phenotype

Signaling networks ?

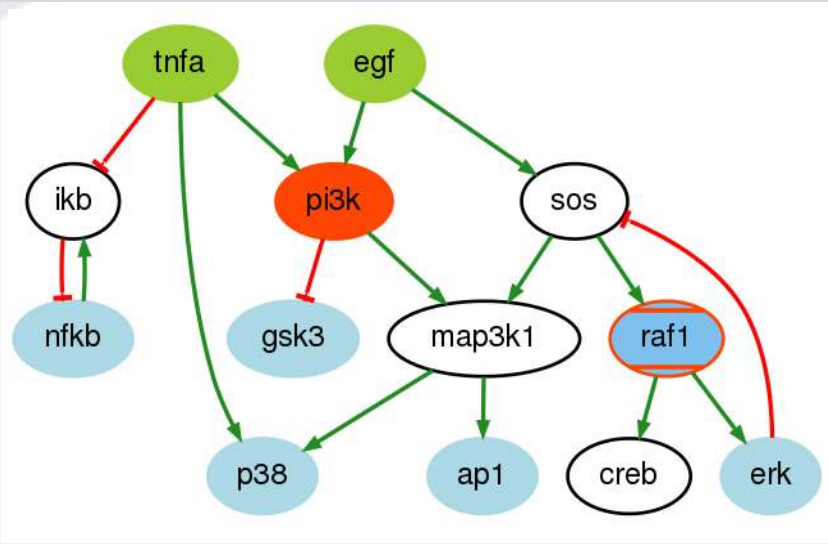
- Videla, Guziolowski, Eduati, Thiele, Gebser, Nicolas, Saez-Rodriguez, Schaub, Siegel, Journal of Theoretical Computer Science, 2015.
- Videla, Konokotina, Alexopoulos, Saez-Rodriguez, Schaub, Siegel, Guziolowski, *Frontiers in Bioengineering and Biotechnology,* 2015
- Videla, Schaub, Siegel, Chapter in *Logical Modeling of Biological Systems*, Wiley Online, 2015.
- Ostrowski, Paulevé, Schaub, Siegel, Guziolowski, CMSB 2015 & *Biosystems* 2016
- Videla et al, *Bioinformatics*, 2017

Equipe MeForBio

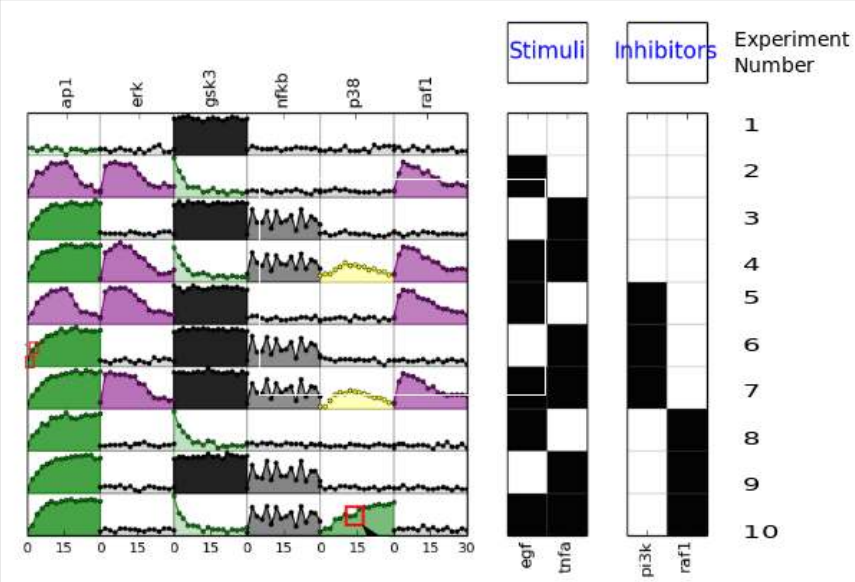
EMBL-EBI Hinxton

Universität Potsdam

The problem: describe combinatorial responses of a signaling network



Domain knowledge



Phosphoproteomic data

Goal. Explain individual data

- Propagation of regulations among a discrete system
- Common hidden mutation

Sample	mutation	RNA-seq
Patient	-	AP1 =0
Patient 1	SOS	AP1 ++
Patient 2	pi3K	P38 = 0

Genomics (inter-individual data)

Analogy…

Data

Inference

Several equivalent models

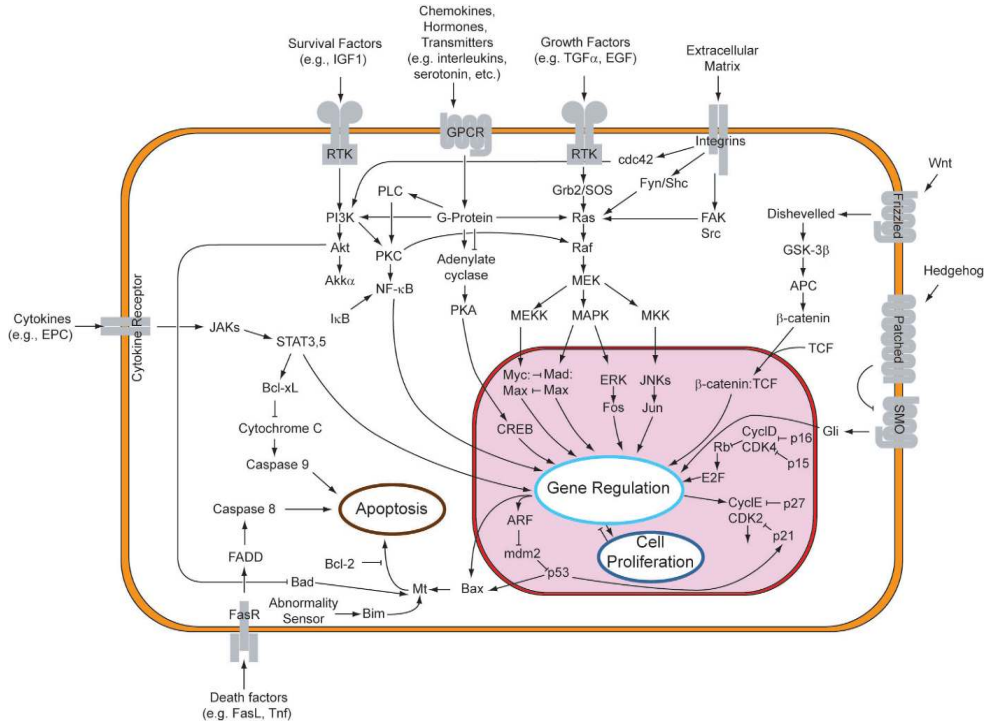
Experimental design

Speed limiting module

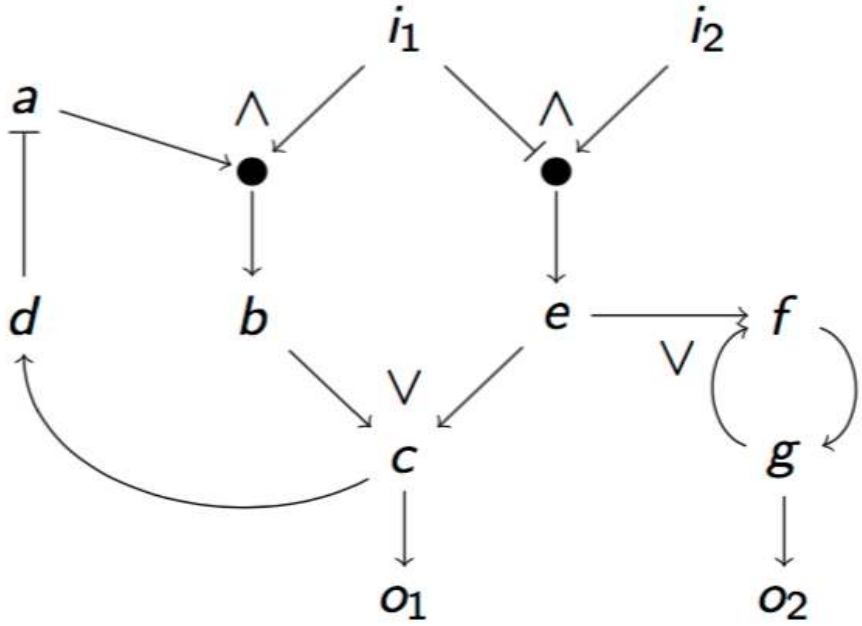
Controler identification

Reducing the number of models

Modeling hypothesis



Signaling network: on/off system



Models: synchronous boolean networks at quasi-steady state

$$\phi = \left\{ \begin{array}{llll} i_1 \mapsto \emptyset & a \mapsto \neg d & d \mapsto c & g \mapsto f \\ i_2 \mapsto \emptyset & b \mapsto a \wedge i_1 & e \mapsto \neg i_1 \wedge i_2 & o_1 \mapsto c \\ & c \mapsto b \vee e & f \mapsto e \vee g & o_2 \mapsto g \end{array} \right\}$$

Logical models

Reformulating the inference problem

Mathematical problem: Optimize $\Theta_{rss}((V,\phi)) + 0.1\Theta_{size}((V,\phi))$

1st objective ($\in \mathbb{R}$). difference between observations and predictions

$$\Theta_{rss}((V,\phi), \underbrace{(P_1,\ldots,P_n)}_{n \text{ experiments}}) = \sum_{i=1}^{n} \sum_{v \in dom(P_i)} \left(\underbrace{P_i(v)}_{\text{observations} \in [0,1]} - \underbrace{\pi_i(v)}_{\text{predictions} \in \{0,1\}} \right)^2$$

2nd objective ($\in \mathbb{N}$). model complexity

$$\Theta_{size}((V,\phi)) = \sum_{v \in dom(\phi)} |\phi(v)|$$

- Local minima
- No global enumeration
- No data noise

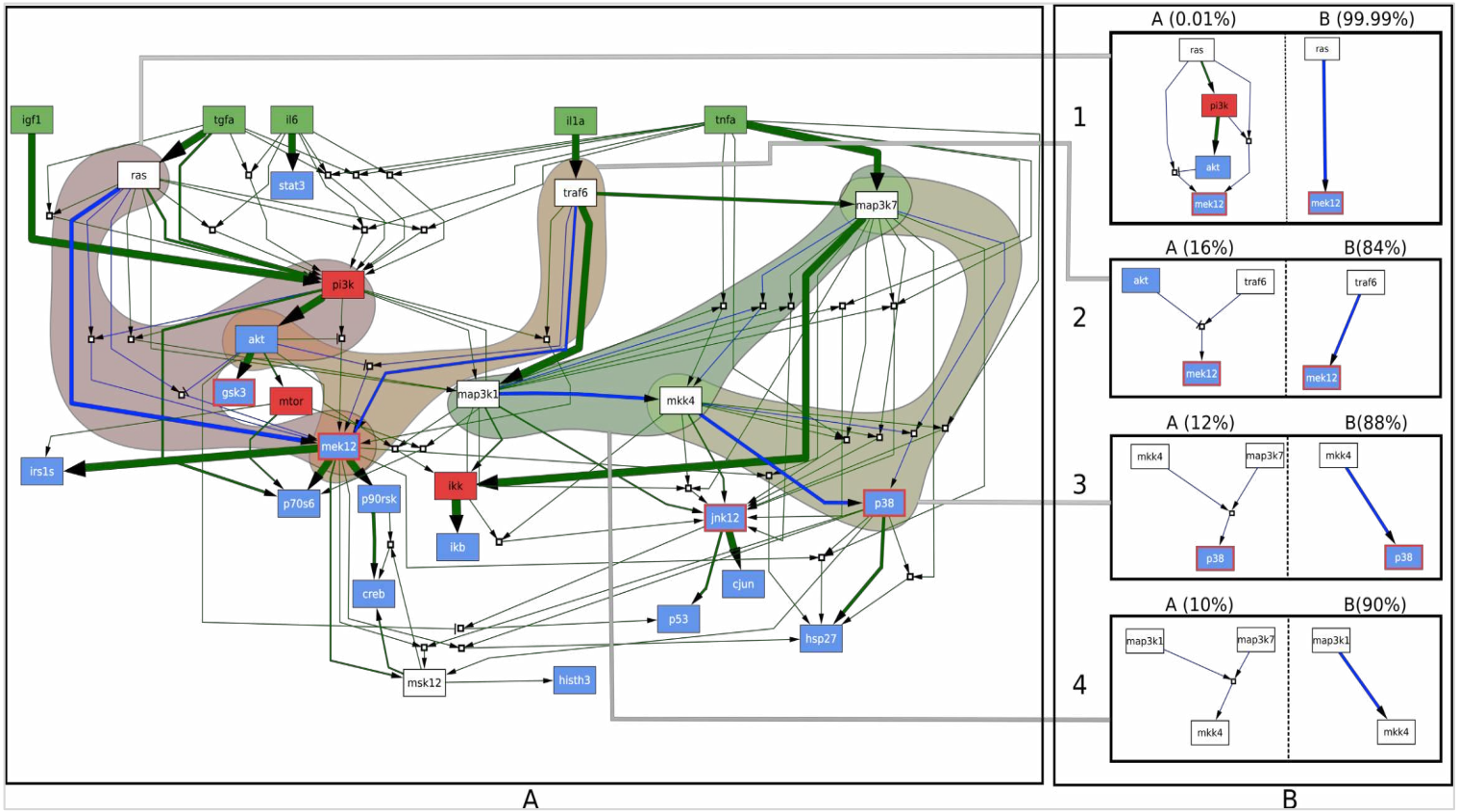
Impact of data discretization
on the solutions of the optimization problem

Computing all sub-optimal solutions
to a combinatorial **multi-optimization problem → ASP encoding**

$$\left\{ (V,\phi) \in \underbrace{\mathbb{M}_{(V,E,\sigma)}}_{\text{Structure}} \mid \underbrace{Score_{rss}((V,\phi), Exp\_Data)}_{\text{Fitting}} \leq \underbrace{(1 + 10\%) \min Score_{rss}}_{\text{Noise tolerance}} \text{ and } \underbrace{Score_{size}((V,\phi))}_{\text{Parsimony}} \leq \underbrace{\min Score_{rss} + 2}_{\text{Size tolerance}} \right\}$$

Videla, et al, Journal of Theoretical Computer Science, 2015

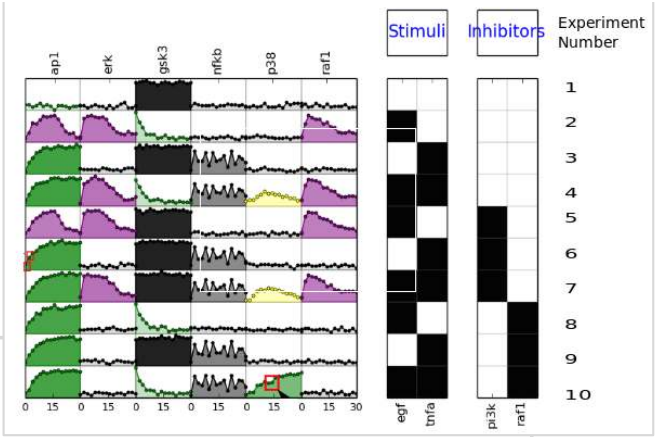
Network identification from two time points



Within the search space of 2^{87} models,
exactly 5306 boolean models explain equivalently well the 2009 Dream challenge data.

Videla, et al, Journal of Theoretical Computer Science, 2015
Guziolowski et al, *Bioinformatics*, 2013

Boolean network inference from time-series data



Main limitation:
ASP is not a model-checker: finite trajectories only.

➢ **How to take dynamical traces into account ?**

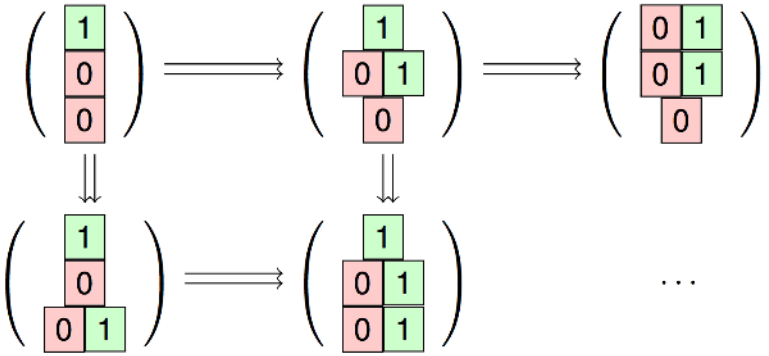
$$\begin{pmatrix} u_{1..i-1} \\ \boxed{a} \\ u_{i+1..n} \end{pmatrix} \rightrightarrows \begin{pmatrix} u_{1..i-1} \\ 0\ 1 \\ u_{i+1..n} \end{pmatrix} \quad \text{if } \exists x \in u : f_i(x) \neq a$$

Example

$f_1(x) = \neg x_2 \vee x_3$

$f_2(x) = x_1$

$f_3(x) = \neg x_2 \vee x_3$



➢ **Abstraction of logical states with meta-states**

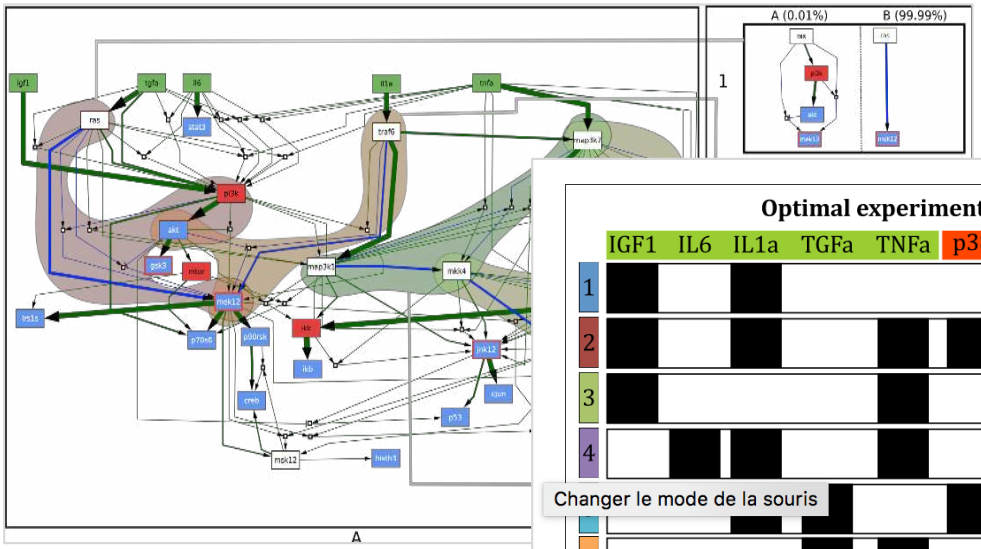
➢ A posteriori-filtering with model-checkers

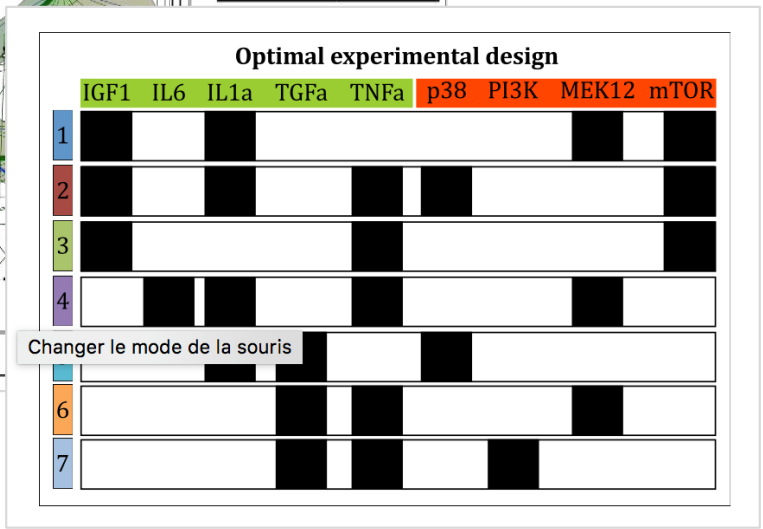
Paulevé et al, CMSB 2015 & *Biosystems* 2016

Discriminating models

Minimal number of perturbations needed to distinguish between a family of model.

- Lexicographic multi-objective optimization (incremental solving)
- Fits with phosphoproteomics arrays





$$(\forall \beta, \beta' \in \boldsymbol{B} :: (\exists p \in \boldsymbol{D} :: \beta(p) \neq \beta'(p))) .$$

Let us denote with $\mathcal{D}_{(k,s,i)}$ the set of all $\boldsymbol{D} \subseteq \boldsymbol{P}$ with $|\boldsymbol{D}| = k$

$$\Theta_{diff}(\boldsymbol{B}, \boldsymbol{D}) = \sum_{\beta,\beta' \in \boldsymbol{B}} \sum_{p \in \boldsymbol{D}} \mathcal{H}(\beta(p), \beta'(p)) \quad (2)$$

where $\mathcal{H}$ denotes the Hamming distance over Boolean vectors,

$$D^*_{(k,s,i)} = \arg\max_{\boldsymbol{D} \in \mathcal{D}_{(k,s,i)}} \Theta_{diff}(\boldsymbol{B}, \boldsymbol{D}) .$$

$$\forall \boldsymbol{D}^* \in \mathcal{D}^*_{(k,s,i)}, \quad \Theta_U(\boldsymbol{D}^*) = \sum_{p \in \boldsymbol{D}^*} \sum_{u_j \in U} p_j$$

$$\mathcal{D}_{opt} = \arg\min_{\boldsymbol{D}^* \in \mathcal{D}^*_{(k,s,i)}} (\Theta_{V_S}(\boldsymbol{D}^*), \Theta_{V_I}(\boldsymbol{D}^*))$$

7 experiments are needed to discriminate between the 5306 models

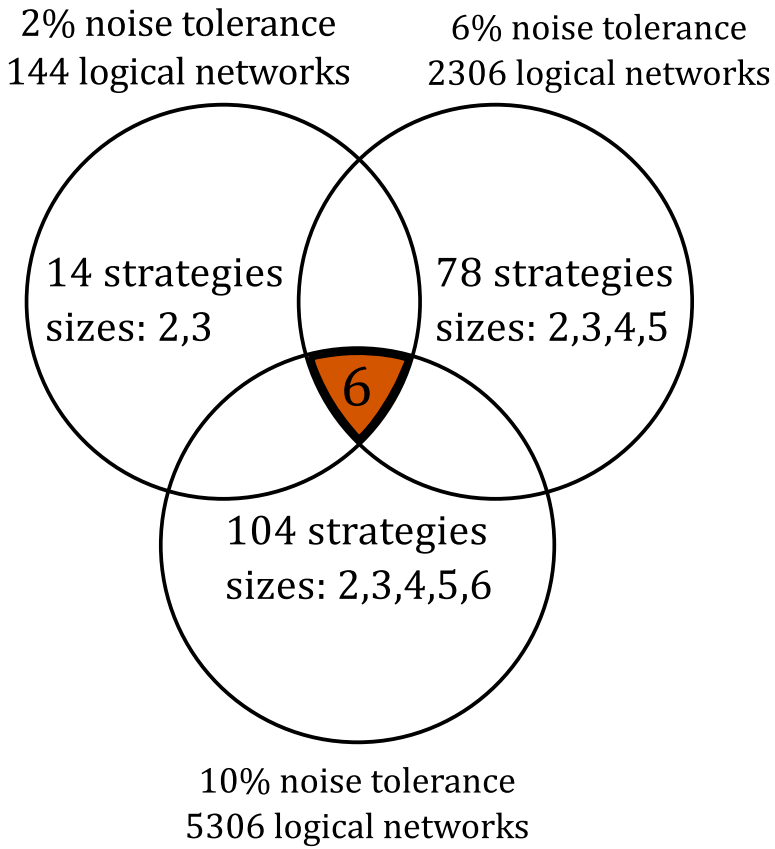
Videla, et al, *Frontiers in Bioengineering and Biotechnology*, 2015

Intervention sets of families of models

Sample	mutation	RNA-seq
Patient	-	AP1 =0
Patient 1	SOS	AP1 ++
Patient 2	pi3K	P38 = 0

Genomics

t	a	b	c	d	e	f
t_0	-1	*	1	*	*	*
t_1	-1	*	1	-1	1	*
t_2	-1	*	1	-1	1	-1



Efficient computation of **intervention sets shared by families of models**

- disjunctive solving heuristics
- encode the dynamics into a three-value logics

Videla et al, Chapter in *Logical Modeling of Biological Systems*, Wiley Online, 2015

Link with the general scheme

Describe a system by a family of abstract models

- Reason over a family of models instead of selecting a single one

(Logical) knowledge representation

- Model : CNF boolean formula
- Fixed-point : logical equation
- Trajectories : finite encodings

Solving combinatorial constraints

- Test new solving heuristics

Discrete dynamical systems: remove spurious information

- Noise: over-approximation issue
- Learning from two time-points: two-value logics
- Compute intervention sets: three-value logics
- Learning from kinetic data: static analysis- inspired abstraction.

Link with systems biology ?

Integrative and systems biology is a very relevant field to challenge ASP technologies

- Repair large-scale interaction graph with **branch and bound** solving heuristics (KR 2010)
- Scale metabolic network completion problem with **unsatisfiable core** solving strategy (LPNMR 2013)
- Design experiments with **incremental solving** (Frontiers 2015)
- Implement and benchmark **constrains propagators** (TPLP 2018)

Problem statement & modelling

Solving heuristics & problem reformulation

Linear constrains atoms

```
&sum{a1*x1;...;al*xl} <= k
```

Methodological conclusion

Constraints and abstractions to scale-up systems analysis

- Confront RNA-seq data with regulatory network models
- Fill metabolic models
- Infer, discriminate and control boolean networks with combinatorial optimization problems

What we plan to do next

- Scale-up to linked open data repositories
- **Compare networks or patient data.**
- Infer boolean networks from metabolic phenotypes
- Simulate multi-layer dynamical systems.

Pretty relevant for heterogeneous data on large-scale system…

- Phosphoproteomics data
- Microbiomes and genomes …

Philosophical conclusion: life science data integration ?

- **Life science data are multi-layer and heterogeneous**
  - Linked by underlying regulatory processes

- Systems biology ?
  - **study of complex systems which cannot be uniquely identified**

- **Handling complexity for**
  - Make (dynamical) hypotheses
  - Solve optimization problems instead of identify parameters
  - Win-win collaboration with your BFF ASP-tech developers

- **We will never replace biologists**

Molecular and cellular life science analysis is a user-assisted data science rather than a modeling system science