Table of Contents

In [2]:

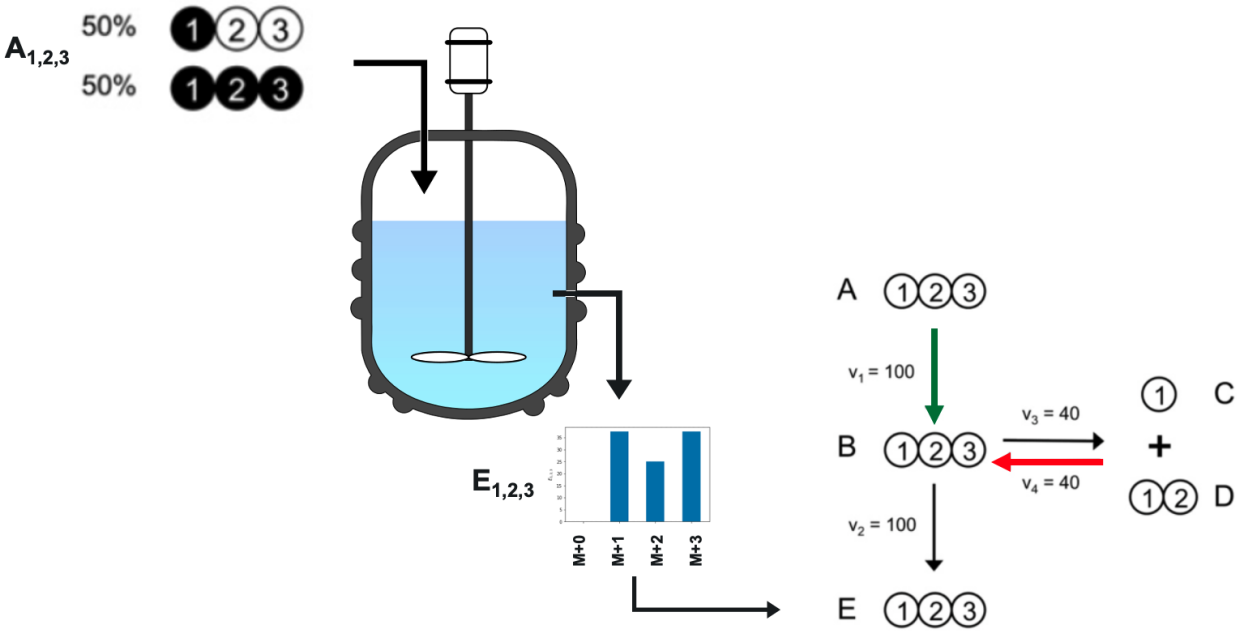
```
%load_ext nb_black


from IPython.display import display, Image, Markdown, Latex, HTML
import pandas as pd
import numpy as np
from cobra import Metabolite, Reaction, Model
from cobra.util import create_stoichiometric_matrix
import cobra
cobra_config = cobra.Configuration()
cobra_config.solver='glpk_exact'
import sys, re,os
#sys.path.append(os.path.join(os.environ['HOME'],'Projects/src'))
from pyefm import calculate_elementary_modes, calculate_minimum_cut_se
from d3flux import flux_map
from IPython.display import display, Image, SVG, Markdown,Latex, HTML
from cobra.flux_analysis.geometric import geometric_fba
from pyemu import *

```

```
The nb_black extension is already loaded. To reload it, use:
  %reload_ext nb_black
```

Goals for Fluxomics Tutorial

In a ^{13}C fluxomics experiment, an organism is fed a substrate with a mixture of isotopic labels where the location of each labeled atom is known. To fully specify the **isotopic labeling state** of the mixture, both the percentage of each component of the mixture and which atoms are labeled in each component must be described.

For today's fluxomics tutorial, we will assume all samples are taken when the organism is in both **metabolic steady state**, where the concentration of each metabolite does not change, and **isotopic stationary state**, where the isotopic labeling state of each metabolite does not change.



If these assumptions are met, mass spectrometry still cannot determine the isotopic labeling state of each metabolite it measures. The best it can do is measure the **Mass Isotopomer Distribution (MID)**, which represents a substrate with a mixture of labels where the location of each labeled atom is not known. For a molecule with n atoms, $M_0, M_1, \ldots, M_n$ specifies the percentage of atoms that are unlabeled, 1-labeled, up to fully labeled.

Metabolic flux analysis (MFA) takes as input:

1. the isotopic labeling state of the substrate,
2. the MID of a set of measured metabolites,
3. a metabolic model of the organism,
4. knowledge of atom transitions
5. and a known uptake rate

and uses this information to calculate the material and energy flows through the organism.

The goal of today's tutorial is to understand how this is done. We will start with a simple model, introduce some new concepts that build upon what we learned this morning about flux balance analysis and elementary modes, and apply these concepts to understand:

1. How measurements of stationary-state MIDs can be used to analytically solve for steady-state fluxes.
2. Why the choice of tracers is critical for identifying flux distributions
3. and how to distinguish good tracers from bad ones.

While analytic solutions are useful for gaining intuition, they do not scale to larger problems. We will thus demonstrate how metabobolic flux analysis actually works with an extended model.

1. Guess a plausible flux distribution
2. Solve for the MIDs that this flux distribution entails.
3. Compare the measured MIDs with the calculated MIDs
4. If they are not close enough, return the flux distribution.
5. Otherwise, repeat step 1.

2 Simple metabolic model

The network in Fig. 1 is described below

In this system, metabolite A is the substrate, metabolite E is the product, and the intermediate metabolites B, C and D are assumed to be at metabolic and isotopic steady state. This network has two free fluxes, namely the substrate consumption rate v_1 and the flux of the reversible cleavage/condensation reaction v_4. We would like to determine flux v_4 based solely on the measurement of the Mass Isotope Distribution (MID) of E and a known uptake rate of A. Three key questions that we would like to answer are:

1. Which labeling pattern(s) of A can be used to resolve flux v_4?
2. Are there certain tracers that will never allow flux v_4 to be determined?
3. Is there a rational criterion that can be used a priori to identify good tracers from poor tracers?

2.1 Simple Metabolic Reaction Network

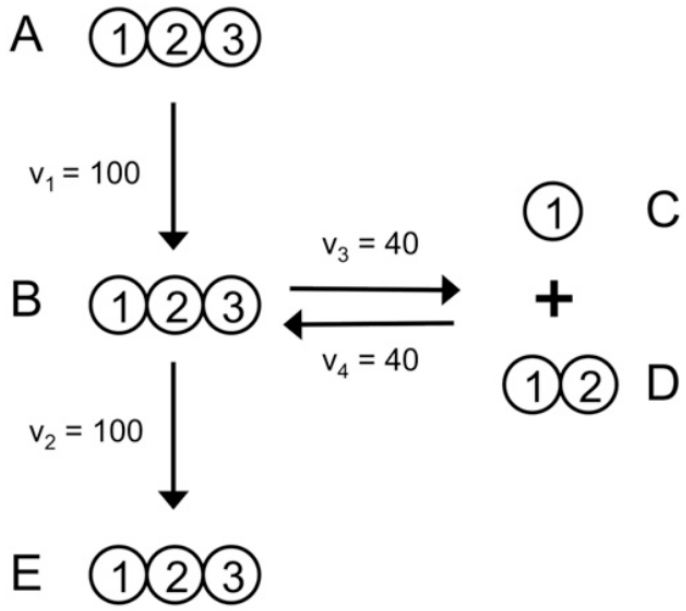


Fig. 1. Simple metabolic network model used to demonstrate the need for a more rational approach to tracer design. The assumed fluxes have arbitrary units.

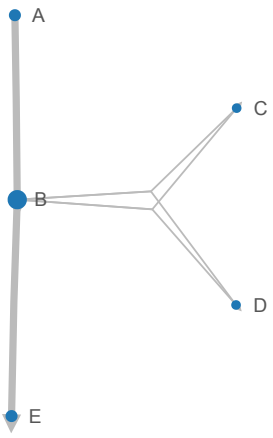
Table 1
Stoichiometry and atom transitions for the reactions in the example network model.

Reaction number	Reaction stoichiometry	Atom transitions
1	A → B	abc → abc
2	B → E	abc → abc
3 and 4	B ↔ C + D	abc ↔ c + ab

In [11]:

```
from IPython.display import Image, display, Markdown

simple_metabolic_network = Model("figure1_simple_metabolic_network")
v1, v2, v3, v4 = 100, 100, 40, 40
A = Metabolite("A")
B = Metabolite("B")
C = Metabolite("C")
D = Metabolite("D")
E = Metabolite("E")
R1 = Reaction("R1")
R2 = Reaction("R2")
R3 = Reaction("R3")
R4 = Reaction("R4")
B1 = Reaction("B1")
B2 = Reaction("B2")
simple_metabolic_network.add_metabolites([A, B, C, D, E])
simple_metabolic_network.add_reactions([R1, R2, R3, R4, B1, B2])
simple_metabolic_network.reactions.R1.build_reaction_from_string("A --
simple_metabolic_network.reactions.R2.build_reaction_from_string("B --
simple_metabolic_network.reactions.R3.build_reaction_from_string("B --
simple_metabolic_network.reactions.R4.build_reaction_from_string("C +
simple_metabolic_network.reactions.B1.build_reaction_from_string("-->
simple_metabolic_network.reactions.B2.build_reaction_from_string("E --

simple_metabolic_network.reactions.R1.notes["CARBON TRANSITIONS"] = "A
simple_metabolic_network.reactions.R2.notes["CARBON TRANSITIONS"] = "B
simple_metabolic_network.reactions.R3.notes[
    "CARBON TRANSITIONS"
] = "B -- C + D\tabc : c + ab"
simple_metabolic_network.reactions.R4.notes[
    "CARBON TRANSITIONS"
] = "C + D -- B\tc + ab : abc"

S = get_stoichiometric_matrix(simple_metabolic_network)
rxns, mets = S.columns, S.index
display(S.astype(int))
display(
    flux_map(
        cobra.io.load_json_model("figure1_simple_metabolic_network.jso
        # display_name_format=lambda x: str(x.id),
        flux_dict=dict(R1=v1, R2=v2, R3=v3, R4=v4, B1=100, B2=100),
        default_flux_width=1,
    )
)
```

	R1	R2	R3	R4	B1	B2
A	-1	0	0	0	1	0
B	1	-1	-1	1	0	0
C	0	0	1	-1	0	0
D	0	0	1	-1	0	0
E	0	1	0	0	0	-1



Save JSON | Show Reaction Nodes | Download SVG

simple network with fluxes (simple-network-with-fluxes.png)

2.1.1 Elementary modes of Simple Metabolic Reaction Network

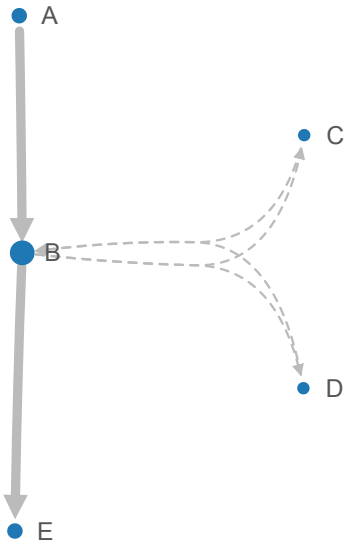
In [76]:
```
elmo = calculate_elementary_modes(simple_metabolic_network, verbose=Fa
display(elmo)
```

```
/Users/zuck016/.pyenv/versions/anaconda3-2020.11/lib/python3.8/subproces
s.py:849: RuntimeWarning: line buffering (buffering=1) isn't supported in
binary mode, the default buffer size will be used
```

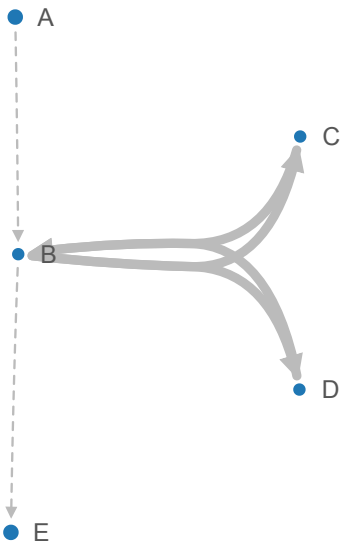
	R1	R2	R3	R4	B1	B2
EM1	1.0	1.0	0.0	0.0	1.0	1.0
EM2	0.0	0.0	1.0	1.0	0.0	0.0

In [77]:

```
display(
    flux_map(
        cobra.io.load_json_model("figure1_simple_metabolic_network.jsc
        display_name_format=lambda x: str(x.id),
        flux_dict={
            rxn.id: elmo.loc["EM1", rxn.id]
            for rxn in simple_metabolic_network.reactions
        },
        figsize=(800, 600),
    )
)
display(
    flux_map(
        cobra.io.load_json_model("figure1_simple_metabolic_network.jsc
        display_name_format=lambda x: str(x.id),
        flux_dict={
            rxn.id: elmo.loc["EM2", rxn.id]
            for rxn in simple_metabolic_network.reactions
        },
        figsize=(800, 600),
    )
)
```



Save JSON | Show Reaction Nodes | Download SVG

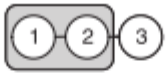
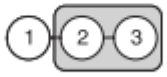
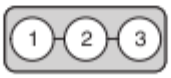


Save JSON | Show Reaction Nodes | Download SVG

simple network elmo 1 (simple-network-elmo-1.png) simple network elmo 2 (simple-network-elmo-2.png)

2.2 Definitions

- **Elementary Metabolite Units (EMU)** are distinct subsets of the compound's atoms. There are 7 EMU's for a 3-atom metabolite A shown in the figure from Antoniewicz 2007 (http://linkinghub.elsevier.com/retrieve/pii/S109671760600084X) below. The subscript in the first column and the shaded areas in the second column denote the atoms that are included in the EMU. The EMU size is the number of atoms included in the EMU.

Elementary metabolite unit	Atoms included in the EMU	EMU size
A_1		1
A_2		1
A_3		1
A_{12}		2
A_{13}		2
A_{23}		2
A_{123}		3

2.3 EMU Networks of the Simple metabolic model

- An **EMU network** is a set of EMU's and the transitions between them. The network can be restricted to EMUs of a particular size (Figure 3A), or it can be combined into a larger network composed of EMU's of all sizes (Figure 3B). Note that not all possible EMUs participate in a transition. This insight allows us to significantly reduce the size of the problem.

2.3.1 Figure 3

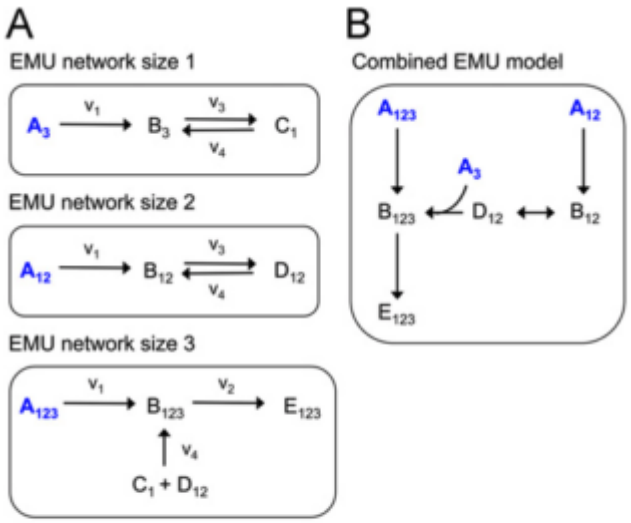
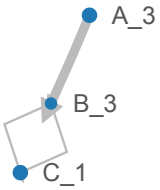


Fig. 3. (A) Decoupled EMU reaction networks for the example network model. (B) Combined EMU reaction network showing the flow of isotopic labeling from the substrate *A* to the network product E. The two EMU basis vectors in the model were A_{123} and $A_{12} \times A_3$.

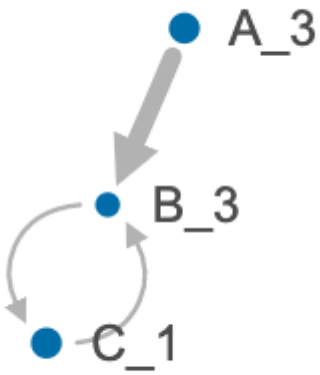
2.3.1.1 EMU network size 1

In [12]:

```
emu_network_size_1 = Model("figure3a_EMU_network_size_1")

A_3 = Metabolite("A_3")
B_3 = Metabolite("B_3")
C_1 = Metabolite("C_1")

R1 = Reaction("R1")
R3 = Reaction("R3")
R4 = Reaction("R4")
B1 = Reaction("B1")
B2 = Reaction("B2")

emu_network_size_1.add_metabolites([A_3, B_3, C_1])
emu_network_size_1.add_reactions([R1, R3, R4, B1, B2])

emu_network_size_1.reactions.R1.build_reaction_from_string("A_3 --> B_
emu_network_size_1.reactions.R3.build_reaction_from_string("B_3 --> C_
emu_network_size_1.reactions.R4.build_reaction_from_string("C_1 <--> E
emu_network_size_1.reactions.B1.build_reaction_from_string("--> A_3")
emu_network_size_1.reactions.B2.build_reaction_from_string("C_1 -->")

# cobra.io.saRe_json_model(emu_network, "figure3_combined_emu_network.
EMU1 = get_stoichiometric_matrix(emu_network_size_1).astype(int)
display(
    flux_map(
        # cobra.io.load_json_model("figure1_simple_metabolic_network.j
        emu_network_size_1,
        display_name_format=lambda x: str(x.id),
        figsize=(800, 600),
        flux_dict=dict(R1=v1, R3=v3, R4=v4),
    )
)
```



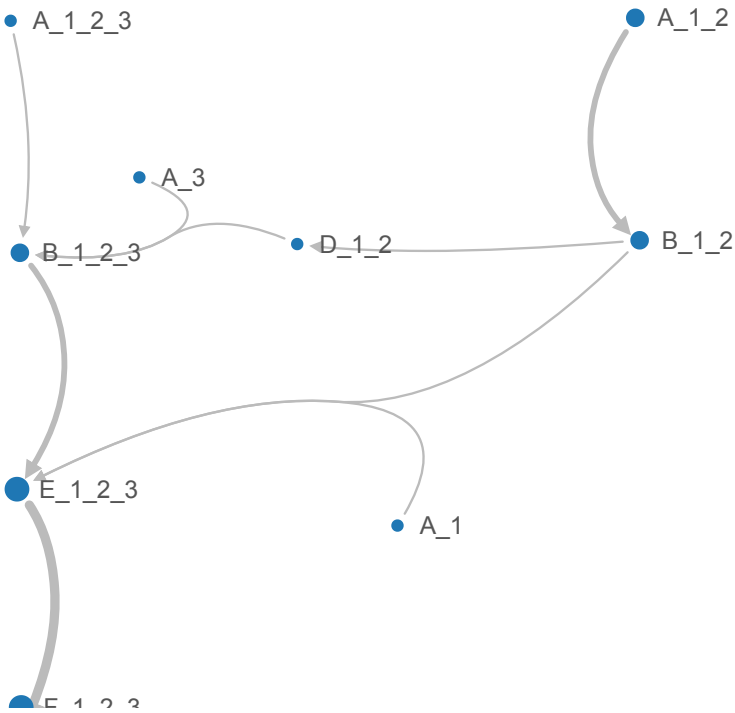
Save JSON | Show Reaction Nodes | Download SVG

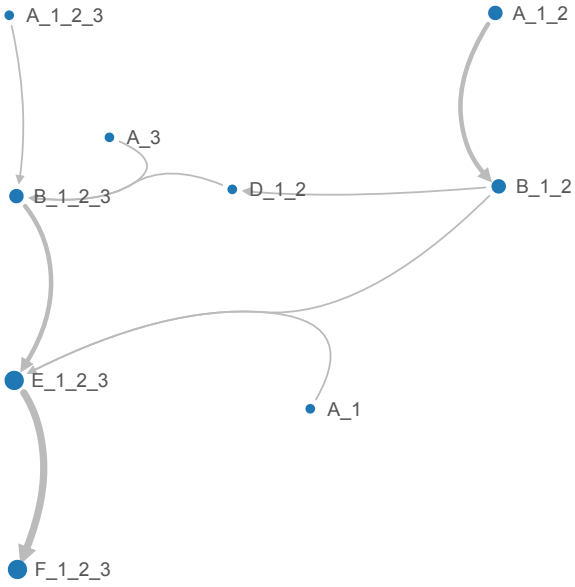


2.3.1.2 EMU Network size 2

In [13]:

```
emu_network_size_2 = Model("figure3a_EMU_network_size_2")

A_1_2 = Metabolite("A_1_2")
B_1_2 = Metabolite("B_1_2")
D_1_2 = Metabolite("D_1_2")

R1 = Reaction("R1")
R3 = Reaction("R3")
R4 = Reaction("R4")
B1 = Reaction("B1")
B2 = Reaction("B2")

emu_network_size_2.add_metabolites([A_1_2, B_1_2, D_1_2])
emu_network_size_2.add_reactions([R1, R3, R4, B1, B2])

emu_network_size_2.reactions.R1.build_reaction_from_string("A_1_2 -->
emu_network_size_2.reactions.R3.build_reaction_from_string("B_1_2 -->
emu_network_size_2.reactions.R4.build_reaction_from_string("D_1_2 <-->
emu_network_size_2.reactions.B1.build_reaction_from_string("--> A_1_2'
emu_network_size_2.reactions.B2.build_reaction_from_string("D_1_2 -->'

# cobra.io.save_json_model(emu_network, "figure3_combined_emu_network.
EMU2 = get_stoichiometric_matrix(emu_network_size_2).astype(int)
display(
    flux_map(
        # cobra.io.load_json_model("figure1_simple_metabolic_network.j
        emu_network_size_2,
        display_name_format=lambda x: str(x.id),
        figsize=(800, 600),
        flux_dict=dict(R1=v1, R3=v3, R4=v4),
    )
)
```





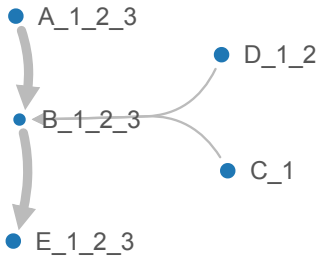
Save JSON | Show Reaction Nodes | Download SVG

EMU size 2 (emu-size-2.png)

2.3.1.3 EMU Network size 3

In [78]:

```
emu_network_size_3 = Model("figure3a_EMU_network_size_3")

A_1_2_3 = Metabolite("A_1_2_3")
B_1_2_3 = Metabolite("B_1_2_3")
C_1 = Metabolite("C_1")
D_1_2 = Metabolite("D_1_2")
E_1_2_3 = Metabolite("E_1_2_3")

R1 = Reaction("R1")
R2 = Reaction("R2")
R4 = Reaction("R4")
B1 = Reaction("B1")
B2 = Reaction("B2")
B3 = Reaction("B3")
B4 = Reaction("B4")

emu_network_size_3.add_metabolites([A_1_2_3, B_1_2_3, D_1_2, C_1, E_1_
emu_network_size_3.add_reactions([R1, R2, R4, B1, B2, B3, B4])

emu_network_size_3.reactions.R1.build_reaction_from_string("A_1_2_3 --
emu_network_size_3.reactions.R2.build_reaction_from_string("B_1_2_3 --
emu_network_size_3.reactions.R4.build_reaction_from_string("D_1_2 + C_
emu_network_size_3.reactions.B1.build_reaction_from_string("--> A_1_2_
emu_network_size_3.reactions.B2.build_reaction_from_string("E_1_2_3 --
emu_network_size_3.reactions.B3.build_reaction_from_string("--> D_1_2'
emu_network_size_3.reactions.B4.build_reaction_from_string("--> C_1")


# cobra.io.save_json_model(emu_network, "figure3_combined_emu_network.
EMU3 = get_stoichiometric_matrix(emu_network_size_3).astype(int)
display(
    flux_map(
        cobra.io.load_json_model(f"{emu_network_size_3}.json"),
        display_name_format=lambda x: str(x.id),
        figsize=(800, 600),
        flux_dict=dict(R1=v1, R2=v2, R4=v4),
    )
)
```



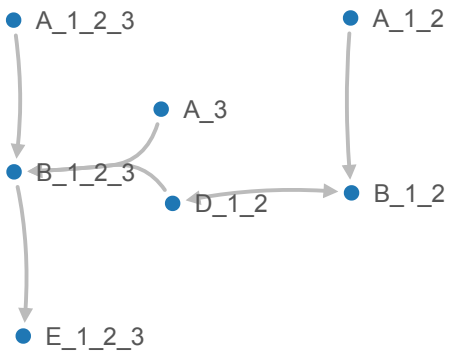
Save JSON | Show Reaction Nodes | Download SVG

emu-size-3 (emu-size-3.png)

2.3.1.4 Combined EMU Network

In [79]:

```
emu_network = Model("figure3_combined_EMU_network")
A_1_2_3 = Metabolite("A_1_2_3")
A_1_2 = Metabolite("A_1_2")
A_3 = Metabolite("A_3")
B_1_2_3 = Metabolite("B_1_2_3")
D_1_2 = Metabolite("D_1_2")
B_1_2 = Metabolite("B_1_2")
E_1_2_3 = Metabolite("E_1_2_3")
R1 = Reaction("R1")
R2 = Reaction("R2")
R3 = Reaction("R3")
R4 = Reaction("R4")
R5 = Reaction("R5")
B1 = Reaction("B1")
B2 = Reaction("B2")
B3 = Reaction("B3")
B4 = Reaction("B4")
emu_network.add_metabolites([A_1_2_3, A_1_2, A_3, B_1_2_3, B_1_2, D_1_
emu_network.add_reactions([R1, R2, R3, R4, R5, B1, B2, B3, B4])

emu_network.reactions.R1.build_reaction_from_string("A_1_2_3 --> B_1_2
emu_network.reactions.R2.build_reaction_from_string("B_1_2_3 --> E_1_2
emu_network.reactions.R3.build_reaction_from_string("A_3 + D_1_2 --> E
emu_network.reactions.R4.build_reaction_from_string("B_1_2 <--> D_1_2"
emu_network.reactions.R5.build_reaction_from_string("A_1_2 --> B_1_2")
emu_network.reactions.B1.build_reaction_from_string("--> A_1_2_3")
emu_network.reactions.B2.build_reaction_from_string("--> A_1_2")
emu_network.reactions.B3.build_reaction_from_string("--> A_3")
emu_network.reactions.B4.build_reaction_from_string("E_1_2_3 -->")

# cobra.io.save_json_model(emu_network, "figure3_combined_emu_network.
EMU_Combined = get_stoichiometric_matrix(emu_network).astype(int)

display(
    flux_map(
        cobra.io.load_json_model("figure3_combined_emu_network.json"),
        display_name_format=lambda x: str(x.id),
        figsize=(800, 600),
    )
)
```



Save JSON | Show Reaction Nodes | Download SVG

Combined emu (emu-combined.png)

2.4 EMU basis vectors

Each **EMU basis vector** can be viewed as a unique way for assembling substrate EMUs to form the product. By definition, EMU basis vectors have the same EMU size as the product. Importantly, given a network and a labeled substrate, one can associate a Mass Isotopic Distribution (MID) with each EMU basis vector. Ultimately, one can interpret the MID of the product as a linear combination of EMU basis vector MIDs, where the coefficients are solely a function of free fluxes. The coefficients quantify the fractional contribution of each EMU basis vector to the product. Thus, by definition, the sum of the coefficients must equal one.

For the simple model, an analytic solution exists that can be written in terms of the EMU basis vectors and the free fluxes:

$$E_{1,2,3} = \frac{v_1}{v_1 + v_4} A_{1,2,3} + \frac{v_4}{v_1 + v_4} A_{1,2} \times A_3 \quad (1)$$

In [80]:
```
display(Markdown("### EMU basis vectors for $E_{1,2,3}$\n"))
emu_transitions, emus = EMU_Combined.columns, EMU_Combined.index
external_emus = ["A_1_2_3", "A_1_2", "A_3", "E_1_2_3"]
external_emu_transitions = ["B1", "B2", "B3", "B4"]
output_emu = "E_1_2_3"
output_emu_transition = "B4"
ebv, emu_elmo = get_EMU_basis_vectors_and_pathways(
    emu_network,
    external_emu_transitions,
    external_emus,
    output_emu,
    output_emu_transition,
)
ebv
```

EMU basis vectors for $E_{1,2,3}$

```
/Users/zuck016/.pyenv/versions/anaconda3-2020.11/lib/python3.8/subproces
s.py:849: RuntimeWarning: line buffering (buffering=1) isn't supported in
binary mode, the default buffer size will be used
```

Out[80]:

	A_1_2_3	A_1_2	A_3	E_1_2_3	EMU Basis Vector
EM1	-1.0	-0.0	-0.0	1.0	$A_{1,2,3}$
EM2	-0.0	-1.0	-1.0	1.0	$A_{1,2} \times A_3$

2.5 EMU pathways

In addition to the EMU basis vector, Crown et al also define an **EMU pathway**. Each EMU basis vector specifies a unique EMU pathway from substrate(s) to product.

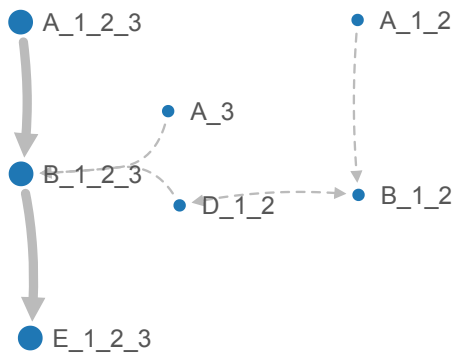
- An **Elementary mode (ELMO)** of a network is a minimal cardinality set of reactions that form a steady-state. Minimal cardinality means that no steady-state flux distribution can be a subset of an elementary mode. The set of elementary modes forms a basis for steady-state flux space.
- Assertion: The elementary modes of the EMU network are the EMU pathways. Evidence for this assertion is provided below.

In [82]:
```
display(Markdown("### EMU Pathways for $E_{1,2,3}$\n"))
emu_elmo["EMU Basis Vector"] = ebv["EMU Basis Vector"]

display(emu_elmo)

for em in emu_elmo.index:
    display(
        Markdown(
            "#### EMU Pathway {} (".format(em) + ebv.loc[em, "EMU Basi
        )
    )
    display(
        flux_map(
            cobra.io.load_json_model("figure3_combined_emu_network.jsc
            display_name_format=lambda x: str(x.id),
            flux_dict={
                emu_transition.id: emu_elmo.loc[em, emu_transition.id]
                for emu_transition in emu_network.reactions
            },
            figsize=(800, 600),
        )
    )
```

EMU Pathways for $E_{1,2,3}$

	R1	R2	R3	R4	R5	B1	B2	B3	B4	EMU Basis Vector
EM1	1.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	$A_{1,2,3}$
EM2	0.0	1.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	$A_{1,2} \times A_3$

EMU Pathway EM1 ($A_{1,2,3}$)



Save JSON | Show Reaction Nodes | Download SVG

EMU Pathway EM2 ($A_{1,2} \times A_3$)

EMU pathway 1 (emu-pathway-1.png)

EMU pathway 2 (emu-pathway-2.png)

2.6 Tracer Compositions for Simple EMU network

Given 4 different tracer compositions of:

- $100\%\ A_{100}$,
- $100\%\ A_{111}$,
- $50\%\ A_{101}$ and $50\%\ A_{111}$,
- $50\%\ A_{100}$ and $50\%\ A_{111}$,

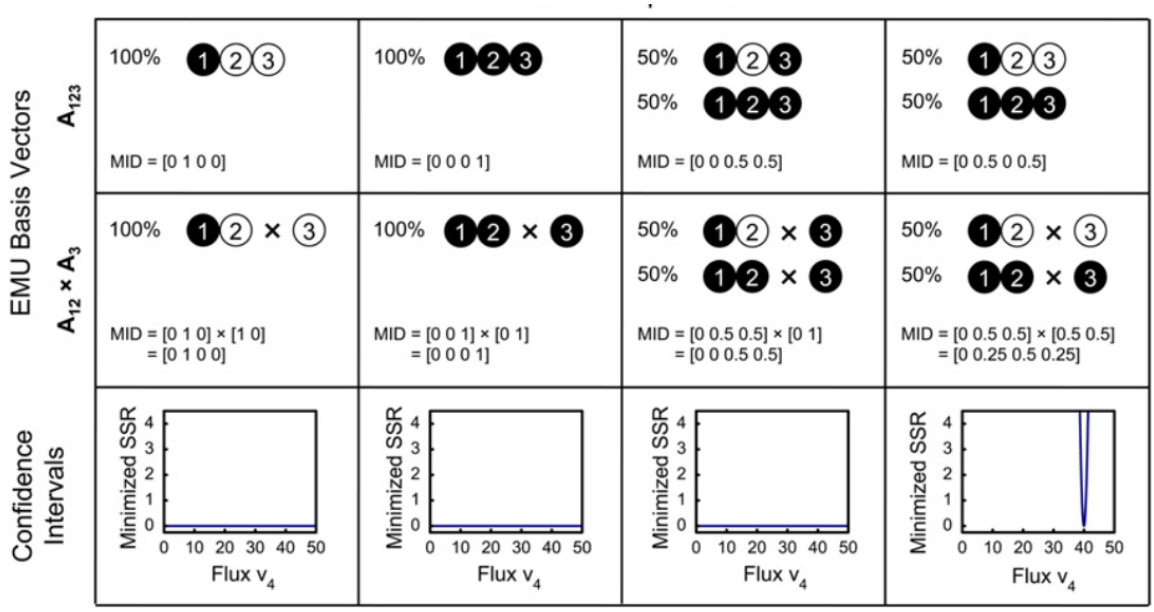
what are the isotopic labeling states and mass isotopomer distributions for each EMU Basis Vector?

Compare the results with Figure 4.

In [20]:

```
display(Image("TracerCompositions.png"))
isotopic_labeling_states = [
    pd.DataFrame([{"%": 100, 1: 1, 2: 0, 3: 0}], index=["$100\%\ A_{10
    pd.DataFrame([{"%": 100, 1: 1, 2: 1, 3: 1}], index=["$100\%\ A_{11
    pd.DataFrame(
        [{"%": 50, 1: 1, 2: 0, 3: 1}, {"%": 50, 1: 1, 2: 1, 3: 1}],
        index=["$50\%\ A_{101}$", "$50\%\ A_{111}$"],
    ),
    pd.DataFrame(
        [{"%": 50, 1: 1, 2: 0, 3: 0}, {"%": 50, 1: 1, 2: 1, 3: 1}],
        index=["$50\%\ A_{100}$", "$50\%\ A_{111}$"],
    ),
]
tracer_mixtures = [dict(A=Tracer("A", 3, ils)) for ils in isotopic_lab
emu_basis_vectors = [
    EMU_Basis_Vector(ebv.iloc[0, :-1]),
    EMU_Basis_Vector(ebv.iloc[1, :-1]),
]
mid_comparison = {}
isl_comparison = {}
for tracers in tracer_mixtures:
    for t in tracers:
        for emu_basis_vector in emu_basis_vectors:
            for emu in emu_basis_vector.get_input_emus():
                emu_isl = emu.get_isotopic_labeling_state(tracers)
                emu_isl.index = pd.MultiIndex.from_tuples(
                    [(tracers[t].get_name(), a) for a in emu_isl.index
                )
                for atom in emu_isl.columns:
                    key = (emu_basis_vector.get_name(), emu.get_name(
                    if key in isl_comparison:
                        isl_comparison[key] = isl_comparison[key].appe
                    else:
                        isl_comparison[key] = emu_isl[atom]
            mid_comparison[
                tracers[t].get_name(), emu_basis_vector.get_name()
            ] = emu_basis_vector.get_mid(tracers)

display(Markdown("### Isotopomer labeling states"))
isl_df = pd.DataFrame(isl_comparison)
isl_df.columns = isl_df.columns.set_names(["EMU Basis Vectors", "EMUs'
isl_df.index = isl_df.index.set_names(["Tracer Compositions", "Tracers
isl_df.style.set_table_styles([dict(selector="th", props=[("text-align
isl_df.style.set_properties(**{"text-align": "left"})
display(isl_df.T)
display(Markdown("### Mass Isotopomer Distribution"))

mid = pd.DataFrame(mid_comparison).T
mid.index.set_names(["Tracer compositions", "Tracers"], inplace=True)
display(mid)
```

Tracer Composition



Isotopomer labeling states

		Tracer Compositions	100% A_{100}	100% A_{111}	50% A_{101} and 50% A_{111}		50% A_{100} and 50% A_{111}	
		Tracers	100% A_{100}	100% A_{111}	50% A_{101}	50% A_{111}	50% A_{100}	50% A_{111}
EMU Basis Vectors	**EMUs**	**Atoms**						
$A_{1,2,3}$	$A_{1,2,3}$	**1**	1	1	1	1	1	1
		2	0	1	0	1	0	1
		3	0	1	1	1	0	1
$A_{1,2} \times A_3$	$A_{1,2}$	**1**	1	1	1	1	1	1
		2	0	1	0	1	0	1
	A_3	**3**	0	1	1	1	0	1

Mass Isotopomer Distribution

			M+0	M+1	M+2	M+3
Tracer compositions		**Tracers**				
100% A_{100}		$A_{1,2,3}$	0.0	100.0	0.0	0.0
	$A_{1,2} \times A_3$		0.0	100.0	0.0	0.0
100% A_{111}		$A_{1,2,3}$	0.0	0.0	0.0	100.0

Tracer compositions		Tracers	M+0	M+1	M+2	M+3
	$A_{1,2} \times A_3$		0.0	0.0	0.0	100.0
50% A_{101} and 50% A_{111}		$A_{1,2,3}$	0.0	0.0	50.0	50.0
	$A_{1,2} \times A_3$		0.0	0.0	50.0	50.0
50% A_{100} and 50% A_{111}		$A_{1,2,3}$	0.0	50.0	0.0	50.0
	$A_{1,2} \times A_3$		0.0	25.0	50.0	25.0

In [21]:
```
observed_mid = mid.groupby(level="Tracer compositions").sum() / 2
observed_mid
```

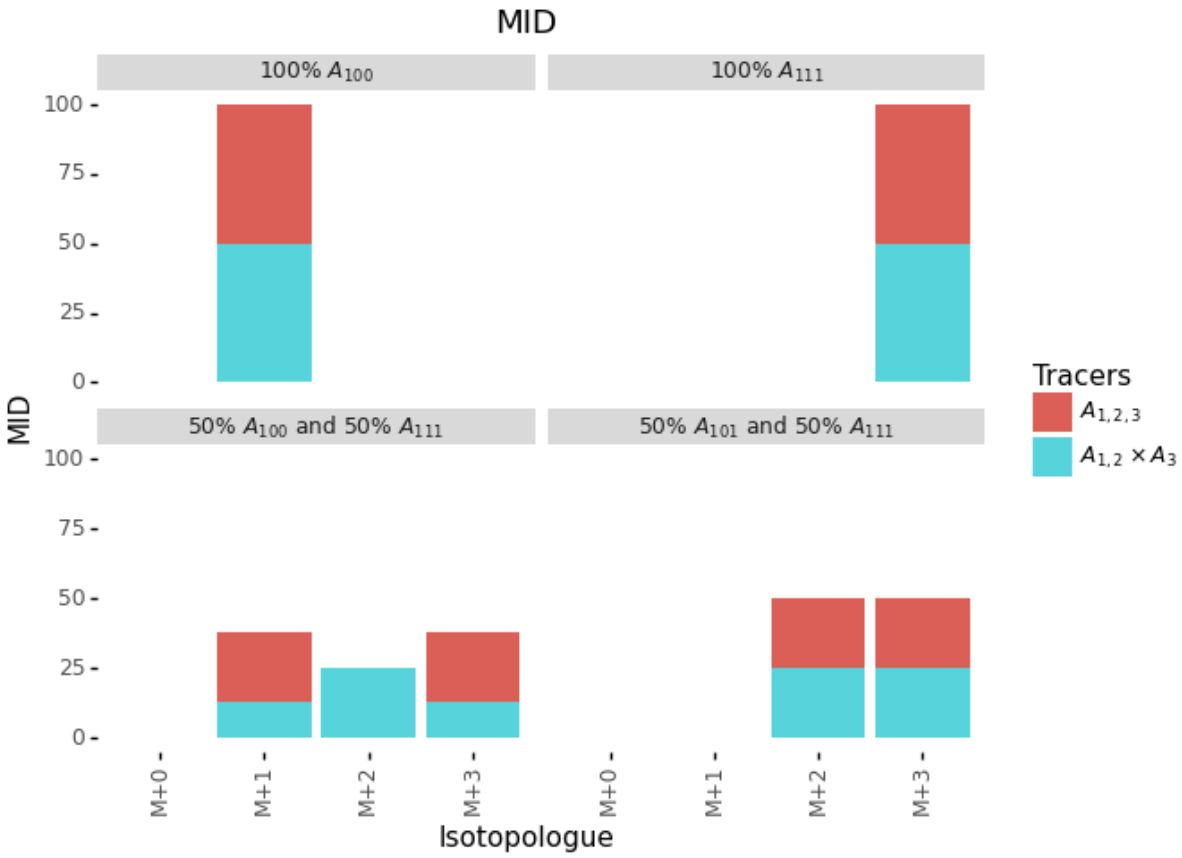
Out[21]:

Tracer compositions	M+0	M+1	M+2	M+3
100% A_{100}	0.0	100.0	0.0	0.0
100% A_{111}	0.0	0.0	0.0	100.0
50% A_{100} and 50% A_{111}	0.0	37.5	25.0	37.5
50% A_{101} and 50% A_{111}	0.0	0.0	50.0	50.0

In [70]:
```
from plotnine import (
    position_dodge,
    ggplot,
    geom_bar,
    aes,
    ggtitle,
    theme,
    element_rect,
    element_text,
    geom_col,
    facet_wrap,
    save_as_pdf_pages,
)


```

In [73]:

```
mid.columns.name = "Isotopologue"
mid_gg = mid.stack().reset_index().rename(columns={0: "MID"})
mid_gg.Tracers = mid_gg.Tracers.str.replace(r"\\text{A}", "A")
mid_gg.MID = mid_gg.MID / 2
mid_gg
plot = (
    ggplot(
        mid_gg,
        aes(x="Isotopologue", y="MID", fill="Tracers"),
    )
    + ggtitle("MID")
    + theme(
        panel_background=element_rect(fill="white"),
        axis_text_x=element_text(angle=90),
    )
    + geom_col()
    + facet_wrap("Tracer compositions")
)
display(plot)
```



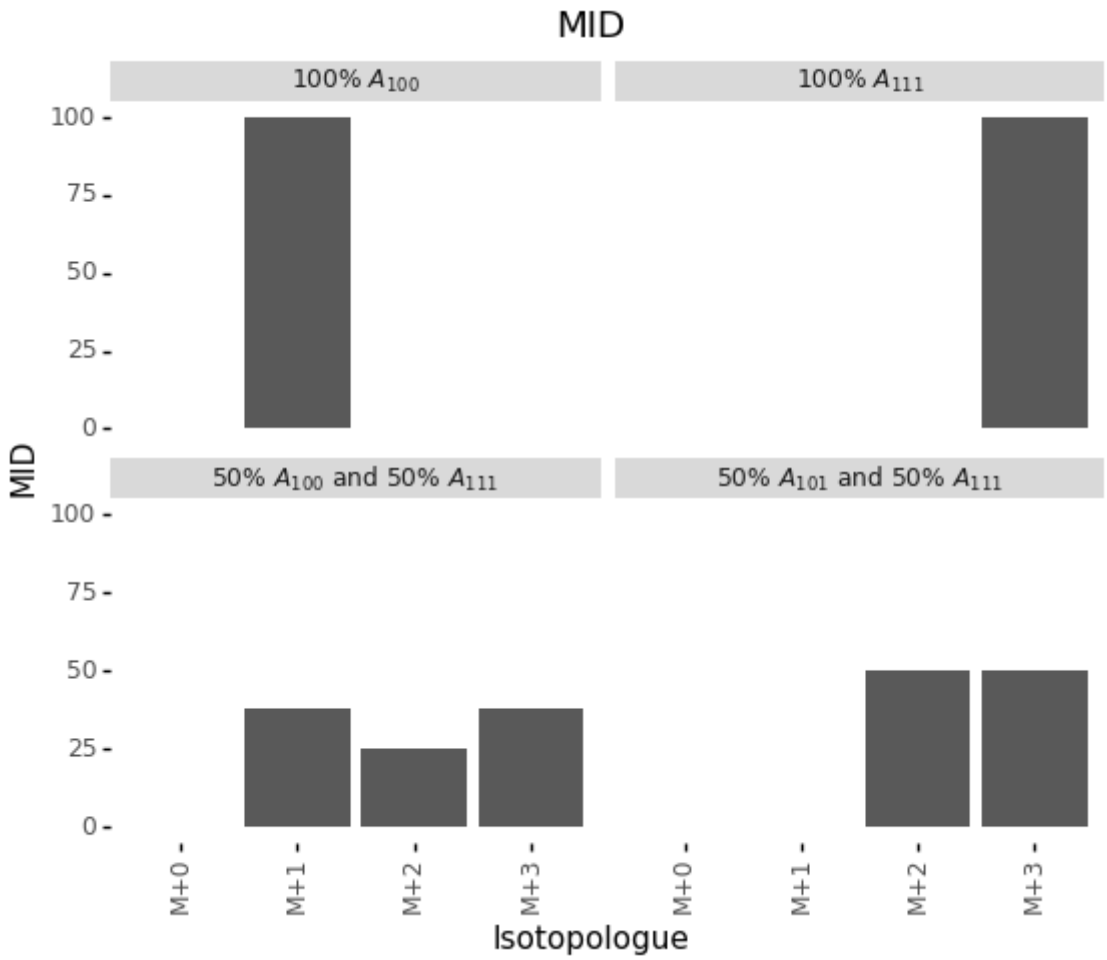

```
<ggplot: (8791977696886)>
```

In [54]:
```
observed_mid.columns.name = "Isotopologue"
observed_mid_gg = observed_mid.stack().reset_index().rename(columns={(
observed_mid_gg
```

Out[54]:

	Tracer compositions	Isotopologue	MID
0	100% A_{100}	M+0	0.0
1	100% A_{100}	M+1	100.0
2	100% A_{100}	M+2	0.0
3	100% A_{100}	M+3	0.0
4	100% A_{111}	M+0	0.0
5	100% A_{111}	M+1	0.0
6	100% A_{111}	M+2	0.0
7	100% A_{111}	M+3	100.0
8	50% A_{100} and 50% A_{111}	M+0	0.0
9	50% A_{100} and 50% A_{111}	M+1	37.5
10	50% A_{100} and 50% A_{111}	M+2	25.0
11	50% A_{100} and 50% A_{111}	M+3	37.5
12	50% A_{101} and 50% A_{111}	M+0	0.0
13	50% A_{101} and 50% A_{111}	M+1	0.0
14	50% A_{101} and 50% A_{111}	M+2	50.0
15	50% A_{101} and 50% A_{111}	M+3	50.0

In [56]:
```
plot = (
    ggplot(
        observed_mid_gg,
        aes(x="Isotopologue", y="MID"),
    )
    + ggtitle("MID")
    + theme(
        panel_background=element_rect(fill="white"),
        axis_text_x=element_text(angle=90),
    )
    + geom_col()
    + facet_wrap("Tracer compositions")
)
display(plot)
```



```
<ggplot: (8791942342891)>
```

3 Extended metabolic model

The Simple metabolic model is extended with three new reactions, and the atom transitions are shown below

In [17]:
```
display(Markdown("## Extended Metabolic Reaction Network"))
display(Image("Extended_EMU_network.png"))
display(Image("Table2.png"))
```

Extended Metabolic Reaction Network

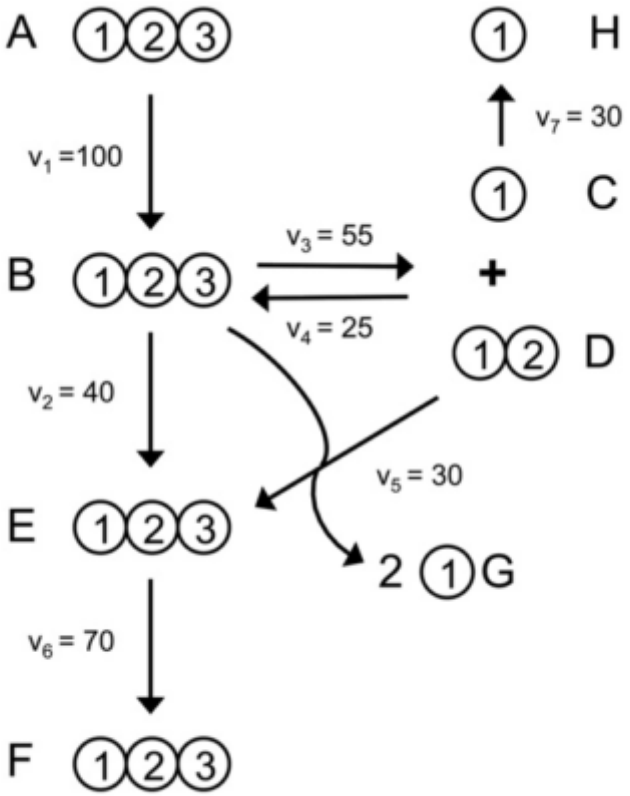


Fig. 5. Extended metabolic network model. The assumed fluxes have arbitrary units.

Table 2
Stoichiometry and atom transitions for the reactions in the extended network model.

Reaction number	Reaction stoichiometry	Atom transitions
1	A → B	abc → abc
2	B → E	abc → abc
3 and 4	B ↔ C + D	abc ↔ c + ab
5	B + D → E + 2G	abc + de → abd + c + e
6	E → F	abc → abc
7	C → H	a → a

In [86]:

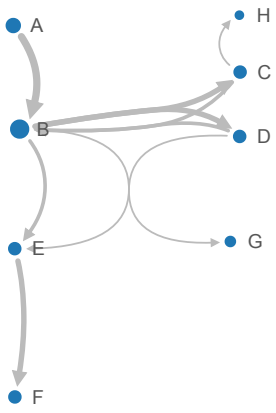
```
metabolic_network = Model("extended_metabolic_network")
A = Metabolite("A")
B = Metabolite("B")
C = Metabolite("C")
D = Metabolite("D")
E = Metabolite("E")
F = Metabolite("F")
G = Metabolite("G")
H = Metabolite("H")
v1 = Reaction("v1")
v2 = Reaction("v2")
v3 = Reaction("v3")
v4 = Reaction("v4")
v5 = Reaction("v5")
v6 = Reaction("v6")
v7 = Reaction("v7")
b1 = Reaction("b1")
b2 = Reaction("b2")
b3 = Reaction("b3")
b4 = Reaction("b4")
metabolic_network.add_metabolites([A, B, C, D, E, F, G, H])
metabolic_network.add_reactions([v1, v2, v3, v4, v5, v6, v7, b1, b2, b
metabolic_network.reactions.v1.build_reaction_from_string("A --> B")
metabolic_network.reactions.v2.build_reaction_from_string("B --> E")
metabolic_network.reactions.v3.build_reaction_from_string("B --> C + D
metabolic_network.reactions.v4.build_reaction_from_string("C + D --> B
metabolic_network.reactions.v5.build_reaction_from_string("B + D --> B
metabolic_network.reactions.v6.build_reaction_from_string("E --> F")
metabolic_network.reactions.v7.build_reaction_from_string("C --> H")
metabolic_network.reactions.b1.build_reaction_from_string("--> A")
metabolic_network.reactions.b2.build_reaction_from_string("F -->")
metabolic_network.reactions.b3.build_reaction_from_string("G -->")
metabolic_network.reactions.b4.build_reaction_from_string("H -->")

metabolic_network.reactions.v1.notes["CARBON TRANSITIONS"] = "A -- B\t
metabolic_network.reactions.v2.notes["CARBON TRANSITIONS"] = "B -- E\t
metabolic_network.reactions.v3.notes["CARBON TRANSITIONS"] = "B -- C +
metabolic_network.reactions.v4.notes["CARBON TRANSITIONS"] = "C + D --
metabolic_network.reactions.v5.notes[
    "CARBON TRANSITIONS"
] = "B + D -- E + 2 G\tabc + de : abd + c + e"
metabolic_network.reactions.v6.notes["CARBON TRANSITIONS"] = "E -- F\t
metabolic_network.reactions.v7.notes["CARBON TRANSITIONS"] = "C -- H\t

S = get_stoichiometric_matrix(metabolic_network)
rxns, mets = S.columns, S.index


# display(Markdown('### Nullspace of Extended Metabolic Reaction Netwo
# display(get_nullspace( S ).T)
display(Markdown("### Elementary modes of Extended Metabolic Reaction
elmo = calculate_elementary_modes(metabolic_network, verbose=False)
display(elmo)
display(
    flux_map(
        cobra.io.load_json_model("extended_metabolic_network.json"),
```

```
        display_name_format=lambda x: str(x.id),
        flux_dict={
            rxn.id: elmo.sum(axis=0)[rxn.id] for rxn in metabolic_netw
        },
    )
)
```

Elementary modes of Extended Metabolic Reaction Network

```
/Users/zuck016/.pyenv/versions/anaconda3-2020.11/lib/python3.8/subproces
s.py:849: RuntimeWarning: line buffering (buffering=1) isn't supported in
binary mode, the default buffer size will be used
```

	v1	v2	v3	v4	v5	v6	v7	b1	b2	b3	b4
EM1	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
EM2	1.0	1.0	0.0	0.0	0.0	1.0	0.0	1.0	1.0	0.0	0.0
EM3	1.0	0.0	0.5	0.0	0.5	0.5	0.5	1.0	0.5	1.0	0.5

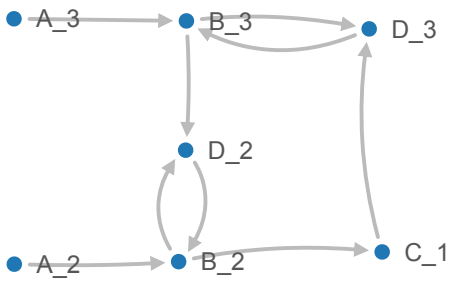


Save JSON | Show Reaction Nodes | Download SVG

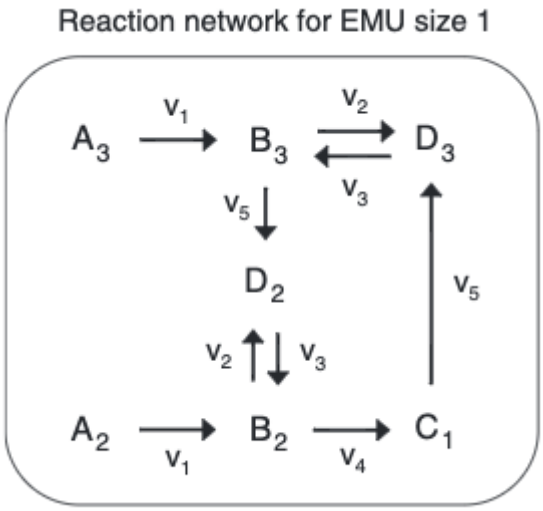
3.1 Reaction network for EMU size 1

In [92]:

```
extended_emu_network_size_1 = Model("extended_emu_network_size_1")

A_2 = Metabolite("A_2")
A_3 = Metabolite("A_3")
B_2 = Metabolite("B_2")
B_3 = Metabolite("B_3")
C_1 = Metabolite("C_1")
D_2 = Metabolite("D_2")
D_3 = Metabolite("D_3")


rxns = [Reaction(f"R{i}") for i in range(1, 10)]


extended_emu_network_size_1.add_metabolites([A_2, A_3, B_2, B_3, C_1,
extended_emu_network_size_1.add_reactions(rxns)

extended_emu_network_size_1.reactions.R1.build_reaction_from_string("A
extended_emu_network_size_1.reactions.R2.build_reaction_from_string("B
extended_emu_network_size_1.reactions.R3.build_reaction_from_string("D
extended_emu_network_size_1.reactions.R4.build_reaction_from_string("B
extended_emu_network_size_1.reactions.R5.build_reaction_from_string("B
extended_emu_network_size_1.reactions.R6.build_reaction_from_string("D
extended_emu_network_size_1.reactions.R7.build_reaction_from_string("B
extended_emu_network_size_1.reactions.R8.build_reaction_from_string("A
extended_emu_network_size_1.reactions.R9.build_reaction_from_string("C


# cobra.io.saRe_json_model(emu_network, "figure3_combined_emu_network.
EMU1 = get_stoichiometric_matrix(extended_emu_network_size_1).astype(i
display(
    flux_map(
        cobra.io.load_json_model("extended_emu_network_size_1.json"),
        # extended_emu_network_size_1,
        display_name_format=lambda x: str(x.id),
        figsize=(800, 600),
    )
)
```



Save JSON | Show Reaction Nodes | Download SVG



3.1.1 EMU size 1 equation

$$\begin{bmatrix} -v_4 & v_4 & 0 & 0 & 0 \\ 0 & -v_1 - v_3 & v_3 & 0 & 0 \\ 0 & v_2 & -v_2 - v_5 & v_5 & 0 \\ 0 & 0 & 0 & -v_1 - v_3 & v_3 \\ v_5 & 0 & 0 & v_2 & -v_2 - v_5 \end{bmatrix} \begin{bmatrix} C_1 \\ B_2 \\ D_2 \\ B_3 \\ D_3 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ -v_1 & 0 \\ 0 & 0 \\ 0 & -v_1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} A_2 \\ A_3 \end{bmatrix}, \quad (2a)$$

3.1.2 EMU size 1 solution

Solution of EMU balances for reaction network of EMU size 1:

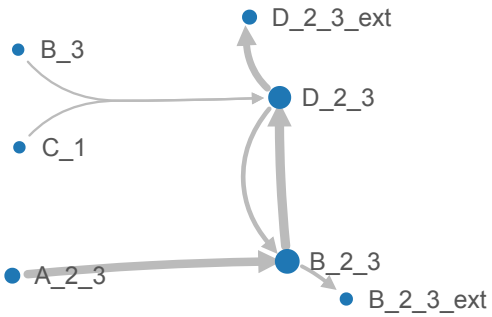
$$\begin{bmatrix} C_1 \\ B_2 \\ D_2 \\ B_3 \\ D_3 \end{bmatrix} = \begin{bmatrix} -20 & 20 & 0 & 0 & 0 \\ 0 & -150 & 50 & 0 & 0 \\ 0 & 110 & -130 & 20 & 0 \\ 0 & 0 & 0 & -150 & 50 \\ 20 & 0 & 0 & 110 & -130 \end{bmatrix}^{-1} \begin{bmatrix} 0 & 0 \\ -100 & 0 \\ 0 & 0 \\ 0 & -100 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 0.0667 & 0.9333 \\ 0.0667 & 0.9333 \\ 0.2000 & 0.8000 \\ 0.9333 & 0.0667 \\ 0.8000 & 0.2000 \end{bmatrix}.$$

Reaction network for EMU size 2

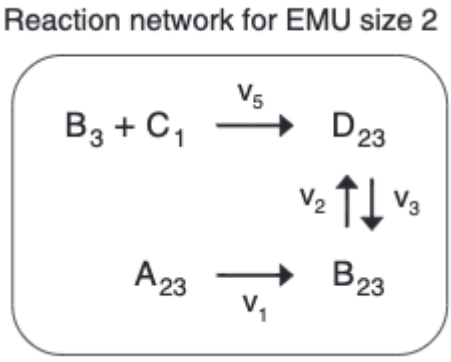
In [9]:

```
xextended_emu_network_size_2 = Model("figure4_extended_emu_network_siz
v1, v2, v3, v5 = 100, 110, 50, 20
A_2_3 = Metabolite("A_2_3")
B_2_3 = Metabolite("B_2_3")
B_3 = Metabolite("B_3")
C_1 = Metabolite("C_1")
D_2_3 = Metabolite("D_2_3")


rxns = [Reaction(f"R{i+1}") for i in range(6)]


extended_emu_network_size_2.add_metabolites([A_2_3, B_2_3, B_3, C_1, I
extended_emu_network_size_2.add_reactions(rxns)

extended_emu_network_size_2.reactions.R1.build_reaction_from_string(
    "B_3 + C_1 --> D_2_3"
)
extended_emu_network_size_2.reactions.R2.build_reaction_from_string("I
extended_emu_network_size_2.reactions.R3.build_reaction_from_string("I
extended_emu_network_size_2.reactions.R4.build_reaction_from_string("A
extended_emu_network_size_2.reactions.R5.build_reaction_from_string(
    "D_2_3 --> D_2_3_ext"
)
extended_emu_network_size_2.reactions.R6.build_reaction_from_string(
    "B_2_3 --> B_2_3_ext"
)


# cobra.io.saRe_json_model(emu_network, "figure3_combined_emu_network.
EMU2 = get_stoichiometric_matrix(extended_emu_network_size_2).astype(i
display(
    flux_map(
        # cobra.io.load_json_model("figure1_simple_metabolic_network.j
        extended_emu_network_size_2,
        display_name_format=lambda x: str(x.id),
        figsize=(800, 600),
        flux_dict=dict(R1=v5, R2=v3, R3=v2, R4=v1, R5=v5 + v2 - v3, R6
    )
)
```

```
unknown metabolite 'D_2_3_ext' created
unknown metabolite 'B_2_3_ext' created
```



Save JSON | Show Reaction Nodes | Download SVG



EMU size 2 equation

$$\begin{bmatrix} -v_5 - v_2 & v_2 \\ v_3 & -v_1 - v_3 \end{bmatrix} \begin{bmatrix} D_{23} \\ B_{23} \end{bmatrix} = \begin{bmatrix} -v_5 & 0 \\ 0 & -v_1 \end{bmatrix} \begin{bmatrix} B_3 \times C_1 \\ A_{23} \end{bmatrix}, \tag{2b}$$

Expansion of $D_{2,3}$ and $B_{2,3}$

$$\begin{bmatrix} D_{23} \\ B_{23} \end{bmatrix} = \begin{bmatrix} D_{23,M+0} & D_{23,M+1} & D_{23,M+2} \\ B_{23,M+0} & B_{23,M+1} & B_{23,M+2} \end{bmatrix}, \tag{3}$$

Expansion of $B_3 \times C_1$ and $A_{2,3}$

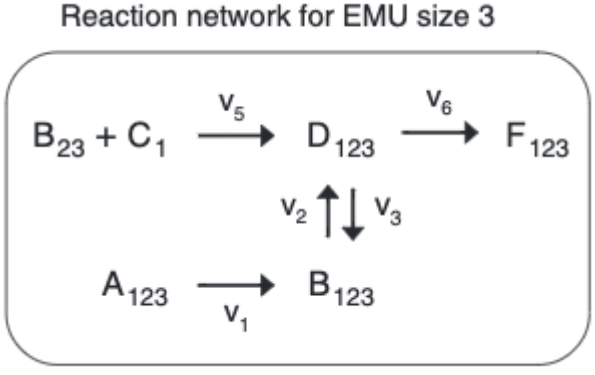
$$\begin{bmatrix} B_3 \times C_1 \\ A_{23} \end{bmatrix} = \begin{bmatrix} B_{3,M+0}C_{1,M+0} & (B_{3,M+0}C_{1,M+1} + B_{3,M+1}C_{1,M+0}) & B_{3,M+1}C_{1,M+1} \\ A_{23,M+0} & A_{23,M+1} & A_{23,M+2} \end{bmatrix}. \tag{4}$$

EMU size 2 solution

Solution of EMU balances for reaction network of EMU size 2:

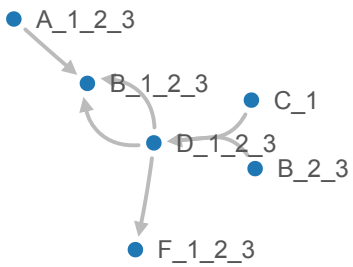
$$\begin{bmatrix} D_{23} \\ B_{23} \end{bmatrix} = \begin{bmatrix} -130 & 110 \\ 50 & -150 \end{bmatrix}^{-1} \begin{bmatrix} -20 & 0 \\ 0 & -100 \end{bmatrix} \begin{bmatrix} 0.0622 & 0.8756 & 0.0622 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0.0133 & 0.9733 & 0.0133 \\ 0.0044 & 0.9911 & 0.0044 \end{bmatrix}.$$

3.3 Reaction network for EMU size 3



In [93]:

```
extended_emu_network_size_3 = Model("extended_emu_network_size_3")

A_1_2_3 = Metabolite("A_1_2_3")
B_2_3 = Metabolite("B_2_3")
B_1_2_3 = Metabolite("B_1_2_3")
C_1 = Metabolite("C_1")
D_1_2_3 = Metabolite("D_1_2_3")
F_1_2_3 = Metabolite("F_1_2_3")


rxns = [Reaction(f"R{i+1}") for i in range(5)]


extended_emu_network_size_3.add_metabolites(
    [A_1_2_3, B_2_3, B_1_2_3, C_1, D_1_2_3, F_1_2_3]
)
extended_emu_network_size_3.add_reactions(rxns)

extended_emu_network_size_3.reactions.R1.build_reaction_from_string(
    "B_2_3 + C_1 --> D_1_2_3"
)
extended_emu_network_size_3.reactions.R2.build_reaction_from_string(
    "D_1_2_3 --> B_1_2_3"
)
extended_emu_network_size_3.reactions.R3.build_reaction_from_string(
    "B_1_2_3 --> D_1_2_3"
)
extended_emu_network_size_3.reactions.R3.build_reaction_from_string(
    "D_1_2_3 --> B_1_2_3"
)
extended_emu_network_size_3.reactions.R4.build_reaction_from_string(
    "A_1_2_3 --> B_1_2_3"
)
extended_emu_network_size_3.reactions.R5.build_reaction_from_string(
    "D_1_2_3 --> F_1_2_3"
)

# cobra.io.saRe_json_model(emu_network, "figure3_combined_emu_network.
EMU3 = get_stoichiometric_matrix(extended_emu_network_size_3).astype(i
display(
    flux_map(
        # cobra.io.load_json_model("figure1_simple_metabolic_network.j
        extended_emu_network_size_3,
        display_name_format=lambda x: str(x.id),
        figsize=(800, 600),
    )
)
```



Save JSON | Show Reaction Nodes | Download SVG

3.3.1 EMU size 3 equation

$$\begin{bmatrix} -v_6 & v_6 & 0 \\ 0 & -v_5 - v_2 & v_2 \\ 0 & v_3 & -v_1 - v_3 \end{bmatrix} \begin{bmatrix} F_{123} \\ D_{123} \\ B_{123} \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ -v_5 & 0 \\ 0 & -v_1 \end{bmatrix} \begin{bmatrix} B_{23} \times C_1 \\ A_{123} \end{bmatrix}. \tag{2c}$$

3.3.2 EMU size 3 solution

Solution of EMU balances for reaction network of EMU size 3

$$\begin{bmatrix} F_{123} \\ D_{123} \\ B_{123} \end{bmatrix} = \begin{bmatrix} -80 & 80 & 0 \\ 0 & -130 & 110 \\ 0 & 50 & -150 \end{bmatrix}^{-1} \begin{bmatrix} 0 & 0 \\ -20 & 0 \\ 0 & -100 \end{bmatrix} \begin{bmatrix} 0.0003 & 0.0702 & 0.9253 & 0.0041 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0.0001 & 0.8008 & 0.1983 & 0.0009 \\ 0.0001 & 0.8008 & 0.1983 & 0.0009 \\ 0.0000 & 0.9336 & 0.0661 & 0.0003 \end{bmatrix}.$$

3.4 Combined EMU transition network of extended

metabolic reaction network

In [60]:
```
1 display(Markdown("## Figure 7\n"))
2 display(Image("Figure7.png"))
```

Figure 7

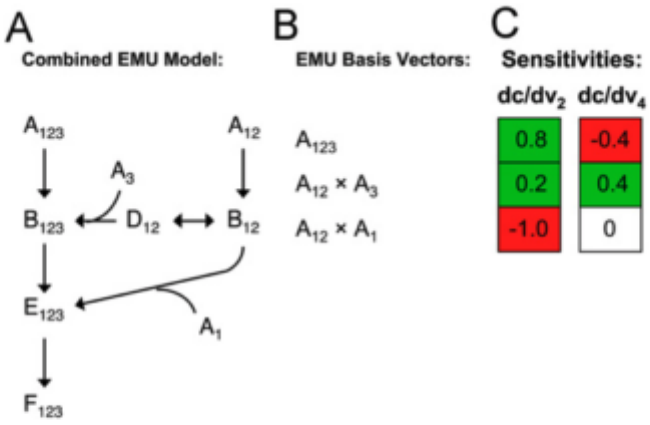


Fig. 7. (A) Combined EMU reaction network for the extended model, showing the flow of isotopic labeling from substrate A to product E. (B) EMU basis vectors for the extended network model. (C) Coefficient sensitivities ($\times 10^2$) with respect to free fluxes, v_2 and v_4. If A was chosen such that $A_{123}=A_{12} \times A_3$, functionality with respect to v_4 was lost.

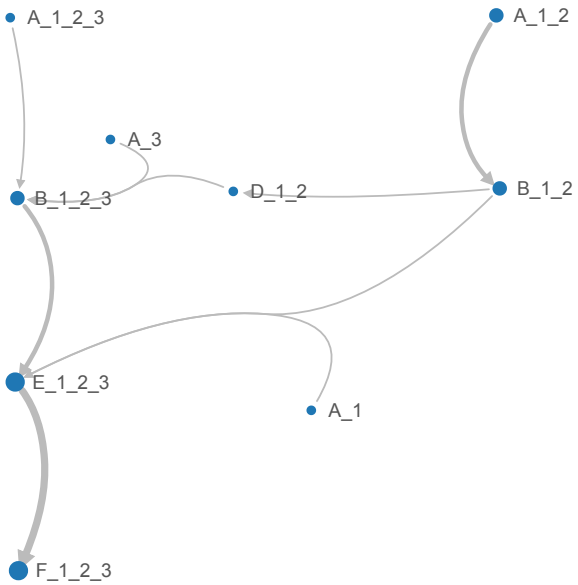
In [94]:

```
emu_network = Model('figure7_combined_EMU_network')
A_1_2_3 = Metabolite('A_1_2_3')
A_1_2 = Metabolite('A_1_2')
A_3 = Metabolite('A_3')
A_1 = Metabolite('A_1')
B_1_2_3 = Metabolite('B_1_2_3')
D_1_2 = Metabolite('D_1_2')
B_1_2 = Metabolite('B_1_2')
E_1_2_3 = Metabolite('E_1_2_3')
F_1_2_3 = Metabolite('F_1_2_3')

R1 = Reaction('R1')
R2 = Reaction('R2')
R3 = Reaction('R3')
R4 = Reaction('R4')
R5 = Reaction('R5')
R6 = Reaction('R6')
R7 = Reaction('R7')
R8 = Reaction('R8')
R9 = Reaction('R9')
R10 = Reaction('R10')
R11 = Reaction('R11')
R12 = Reaction('R12')

emu_network.add_metabolites([A_1, A_1_2_3, A_1_2, A_3, B_1_2_3, B_1_2,
emu_network.add_reactions([R1, R2, R3, R4, R5, R6, R7,R8, R9, R10, R1

emu_network.reactions.R1.build_reaction_from_string('A_1_2_3 --> B_1_2
emu_network.reactions.R2.build_reaction_from_string('B_1_2_3 --> E_1_2
emu_network.reactions.R3.build_reaction_from_string('A_3 + D_1_2 --> E
emu_network.reactions.R4.build_reaction_from_string('B_1_2 <--> D_1_2'
emu_network.reactions.R5.build_reaction_from_string('A_1_2 --> B_1_2')
emu_network.reactions.R6.build_reaction_from_string('--> A_1_2_3')
emu_network.reactions.R7.build_reaction_from_string('--> A_1_2')
emu_network.reactions.R8.build_reaction_from_string('--> A_3')
emu_network.reactions.R9.build_reaction_from_string('E_1_2_3 --> F_1_2
emu_network.reactions.R10.build_reaction_from_string('B_1_2 + A_1 -->
emu_network.reactions.R11.build_reaction_from_string('F_1_2_3 -->')
emu_network.reactions.R12.build_reaction_from_string('--> A_1')

S = get_stoichiometric_matrix( emu_network )
rxns, mets = S.columns, S.index
external_mets = [ 'A_1','A_3', 'A_1_2', 'A_1_2_3', 'F_1_2_3']
external_rxns = ['R6', 'R7', 'R8', 'R11', 'R12']
output_met = 'F_1_2_3'
output_rxn = 'R11'
ebv, emu_elmo = get_EMU_basis_vectors_and_pathways( emu_network, exter

#cobra.io.save_json_model(emu_network, "figure7_combined_emu_network.j

display(Markdown('## Combined EMU network\n'))
display(flux_map(cobra.io.load_json_model("figure7_combined_EMU_networ
         flux_dict={rxn.id: emu_elmo.sum(axis=0)[rxn.id] for rxn in em
display(Markdown('## EMU basis vectors for $F_{123}$ \n'))

```

```
display(ebv)


display(Markdown('## EMU Pathways \n'))
emu_elmo['EMU Basis Vector']  = ebv['EMU Basis Vector']

display(emu_elmo)

for em in emu_elmo.index:
    with emu_network:
        emu_network.remove_reactions([r for r in emu_elmo.columns if e
        display(Markdown('### EMU Pathway {} ('.format(em) + ebv.loc[e
        display(flux_map(emu_network,
                         display_name_format=lambda x: str(x.id),
                         flux_dict={rxn.id: emu_elmo.loc[em,rxn.id]
                                    for rxn in emu_network.reactions}))

```

```
/Users/zuck016/.pyenv/versions/anaconda3-2020.11/lib/python3.8/subproces
s.py:849: RuntimeWarning: line buffering (buffering=1) isn't supported in
binary mode, the default buffer size will be used
```

Combined EMU network



Save JSON | Show Reaction Nodes | Download SVG

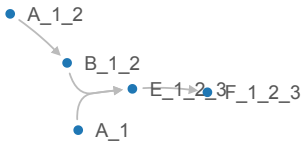
EMU basis vectors for F_{123}

	A_1	A_3	A_1_2	A_1_2_3	F_1_2_3	EMU Basis Vector
EM1	-1.0	-0.0	-1.0	-0.0	1.0	$A_1 \times A_{1,2}$
EM2	-0.0	-1.0	-1.0	-0.0	1.0	$A_3 \times A_{1,2}$
EM3	-0.0	-0.0	-0.0	-1.0	1.0	$A_{1,2,3}$

EMU Pathways

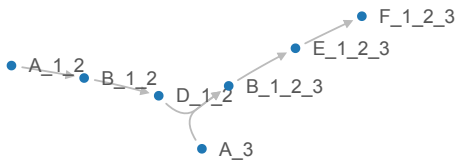
	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	EMU Basis Vector
EM1	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0	1.0	1.0	1.0	$A_1 \times A_{1,2}$
EM2	0.0	1.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	0.0	1.0	0.0	$A_3 \times A_{1,2}$
EM3	1.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	1.0	0.0	$A_{1,2,3}$

EMU Pathway EM1 ($A_1 \times A_{1,2}$)



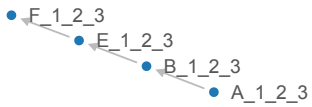
Save JSON | Show Reaction Nodes | Download SVG

EMU Pathway EM2 ($A_3 \times A_{1,2}$)



Save JSON | Show Reaction Nodes | Download SVG

EMU Pathway EM3 ($A_{1,2,3}$)



3.5 Tracer Compositions for Extended EMU network

Given all possible 2 tracer compositions of 3 atoms

.	A_{000}	A_{100}	A_{010}	A_{001}	A_{110}	A_{101}	A_{011}	A_{111}
A_{000}								
A_{100}								
A_{010}								
A_{001}								
A_{110}								
A_{101}								
A_{011}								
A_{111}								

what are the isotopic labeling states and mass isotopomer distributions for each EMU Basis Vector?

Compare the results with Figure 6.

In [21]:

```
display(Image("Figure6.png"))
num_atoms = 3
isotopic_labeling_states = []
for i in range(2 ** num_atoms):
    for j in range(i + 1):
        if i == j:
            isotopic_labeling_states.append(
                pd.DataFrame(
                    [{"%": 100, 1: (i) & 1, 2: (i >> 1) & 1, 3: (i >>
                    index=[
                        r"$100\%"
                        + r"\ A_{%d%d%d}$" % ((i) & 1, (i >> 1) & 1, (
                    ],
                )
            )
        else:
            isotopic_labeling_states.append(
                pd.DataFrame(
                    [
                        {"%": 50, 1: (i) & 1, 2: (i >> 1) & 1, 3: (i >
                        {"%": 50, 1: (j) & 1, 2: (j >> 1) & 1, 3: (j >
                    ],
                    index=[
                        r"$50\%"
                        + r"\ A_{%d%d%d}$" % ((i) & 1, (i >> 1) & 1, (
                        r"$50\%"
                        + r"\ A_{%d%d%d}$" % ((j) & 1, (j >> 1) & 1, (
                    ],
                )
            )

tracer_mixtures = [
    dict(A=Tracer("A", num_atoms, ils)) for ils in isotopic_labeling_s
]
emu_basis_vectors = [EMU_Basis_Vector(ebv.iloc[e, :-1]) for e in range
mid_comparison = {}
isl_comparison = {}
for tracers in tracer_mixtures:
    for t in tracers:
        for emu_basis_vector in emu_basis_vectors:
            for emu in emu_basis_vector.get_input_emus():
                emu_isl = emu.get_isotopic_labeling_state(tracers)
                emu_isl.index = pd.MultiIndex.from_tuples(
                    [(tracers[t].get_name(), a) for a in emu_isl.index
                )
                for atom in emu_isl.columns:
                    key = (emu_basis_vector.get_name(), emu.get_name(]
                    if key in isl_comparison:
                        isl_comparison[key] = isl_comparison[key].appe
                    else:
                        isl_comparison[key] = emu_isl[atom]
            mid_comparison[
                tracers[t].get_name(), emu_basis_vector.get_name()
            ] = emu_basis_vector.get_mid(tracers)

display(Markdown("### Isotopomer labeling states"))
```

```
isl_df = pd.DataFrame(isl_comparison)
isl_df.columns = isl_df.columns.set_names(["EMU Basis Vectors", "EMUs'
isl_df.index = isl_df.index.set_names(["Tracer Compositions", "Tracers
display(HTML(isl_df.to_html()))
display(Markdown("### Mass Isotopomer Distribution"))

display(HTML(pd.DataFrame(mid_comparison).T.to_html()))
```

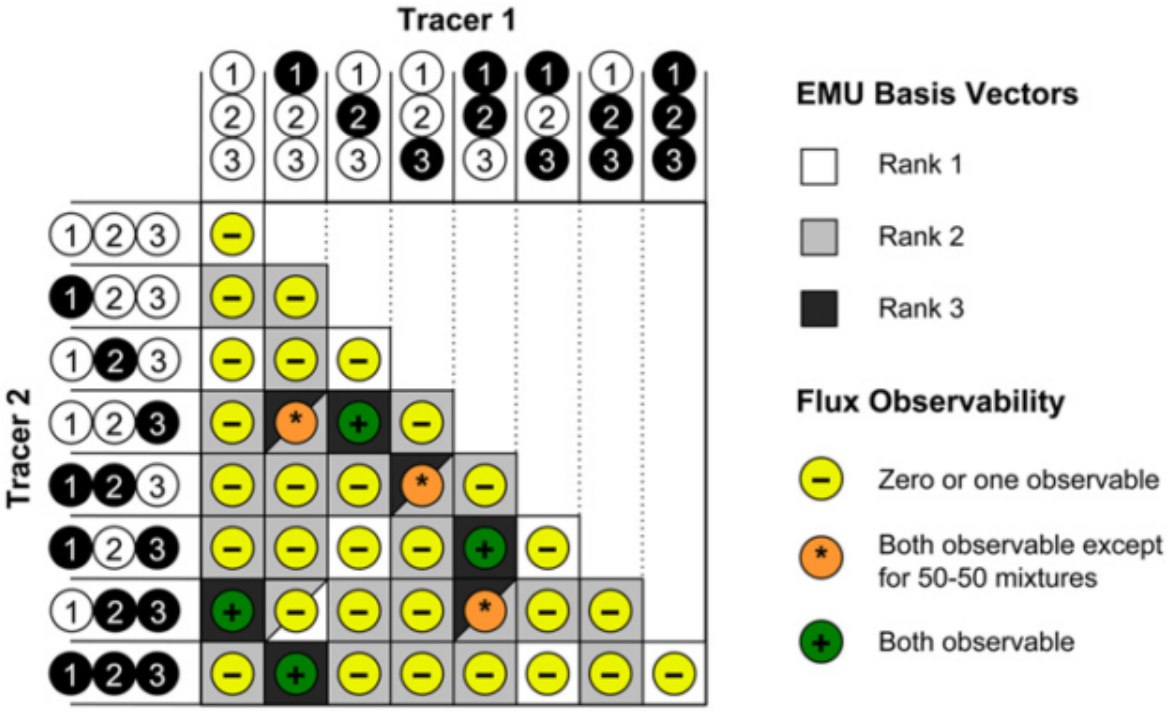


Fig. 6. Analysis of flux observability and EMU basis vector rank in the extended network model. Pure tracers are shown on the diagonal and binary mixtures of tracers are shown on the off-diagonal elements. The square colors denote the EMU basis vector rank (i.e. number of independent EMU basis vectors) and the circles correspond to flux observability. Observability of the free fluxes correlated with the independence of EMU basis vectors. Squares with upper and lower colored triangles denote cases where the EMU basis vector rank was different for 50/50 mixtures of tracers, i.e. compared to 25/75 and 75/25 mixtures.

Isotopomer labeling states

	EMU Basis Vectors	$A_{1,2} \times A_1$			$A_{1,2} \times A_3$			$A_{1,2,3}$		
	EMUs	$A_{1,2}$		A_1	$A_{1,2}$		A_3	$A_{1,2,3}$		
	Atoms	**1**	**2**	**1**	**1**	**2**	**3**	**1**	**2**	**3**
Tracer Compositions	**Tracers**									
100% A_{000}	100% A_{000}	0	0	0	0	0	0	0	0	0
50% A_{100}	50% A_{100}	1	0	1	1	0	0	1	0	0
and 50% A_{000}	50% A_{000}	0	0	0	0	0	0	0	0	0
100% A_{100}	100% A_{100}	1	0	1	1	0	0	1	0	0
50% A_{010}	50% A_{010}	0	1	0	0	1	0	0	1	0
and 50% A_{000}	50% A_{000}	0	0	0	0	0	0	0	0	0
50% A_{010}	50% A_{010}	0	1	0	0	1	0	0	1	0
and 50% A_{100}	50% A_{100}	1	0	1	1	0	0	1	0	0
100% A_{010}	100% A_{010}	0	1	0	0	1	0	0	1	0
50% A_{110}	50% A_{110}	1	1	1	1	1	0	1	1	0
and 50% A_{000}	50% A_{000}	0	0	0	0	0	0	0	0	0
50% A_{110}	50% A_{110}	1	1	1	1	1	0	1	1	0
and 50% A_{100}	50% A_{100}	1	0	1	1	0	0	1	0	0
50% A_{110}	50% A_{110}	1	1	1	1	1	0	1	1	0
and 50% A_{010}	50% A_{010}	0	1	0	0	1	0	0	1	0
100% A_{110}	100% A_{110}	1	1	1	1	1	0	1	1	0
50% A_{001}	50% A_{001}	0	0	0	0	0	0	1	0	0
and 50% A_{000}	50% A_{000}	0	0	0	0	0	0	0	0	0
50% A_{001}	50% A_{001}	0	0	0	0	0	0	1	0	0
and 50% A_{100}	50% A_{100}	1	0	1	1	0	0	1	0	0
50% A_{001}	50% A_{001}	0	0	0	0	0	0	1	0	0
and 50% A_{010}	50% A_{010}	0	1	0	0	1	0	0	1	0
50% A_{001}	50% A_{001}	0	0	0	0	0	0	1	0	0
and 50% A_{110}	50% A_{110}	1	1	1	1	1	0	1	1	0
100% A_{001}	100% A_{001}	0	0	0	0	0	0	1	0	0
50% A_{101}	50% A_{101}	1	0	1	1	0	0	1	1	0
and 50% A_{000}	50% A_{000}	0	0	0	0	0	0	0	0	0

	EMU Basis Vectors	$A_{1,2} \times A_1$			$A_{1,2} \times A_3$			$A_{1,2,3}$		
	EMUs	$A_{1,2}$		A_1	$A_{1,2}$		A_3	$A_{1,2,3}$		
	Atoms	**1**	**2**	**1**	**1**	**2**	**3**	**1**	**2**	**3**
Tracer Compositions	**Tracers**									
50% A_{101} and 50% A_{100}	50% A_{101}	1	0	1	1	0		1	1	0 1
	50% A_{100}	1	0	1	1	0		0	1	0 0
50% A_{101} and 50% A_{010}	50% A_{101}	1	0	1	1	0		1	1	0 1
	50% A_{010}	0	1	0	0	1		0	0	1 0
50% A_{101} and 50% A_{110}	50% A_{101}	1	0	1	1	0		1	1	0 1
	50% A_{110}	1	1	1	1	1		0	1	1 0
50% A_{101} and 50% A_{001}	50% A_{101}	1	0	1	1	0		1	1	0 1
	50% A_{001}	0	0	0	0	0		0	0	0 0
100% A_{101}	100% A_{101}	1	0	1	1	0		1	1	0 1
50% A_{011} and 50% A_{000}	50% A_{011}	0	1	0	0	1		1	0	1 1
	50% A_{000}	0	0	0	0	0		0	0	0 0
50% A_{011} and 50% A_{100}	50% A_{011}	0	1	0	0	1		1	0	1 1
	50% A_{100}	1	0	1	1	0		0	1	0 0
50% A_{011} and 50% A_{010}	50% A_{011}	0	1	0	0	1		1	0	1 1
	50% A_{010}	0	1	0	0	1		0	0	1 0
50% A_{011} and 50% A_{110}	50% A_{011}	0	1	0	0	1		1	0	1 1
	50% A_{110}	1	1	1	1	1		0	1	1 0
50% A_{011} and 50% A_{001}	50% A_{011}	0	1	0	0	1		1	0	1 1
	50% A_{001}	0	0	0	0	0		0	0	0 0
50% A_{011} and 50% A_{101}	50% A_{011}	0	1	0	0	1		1	0	1 1
	50% A_{101}	1	0	1	1	0		0	1	0 0
100% A_{011}	100% A_{011}	0	1	0	0	1		1	0	1 1
50% A_{111} and 50% A_{000}	50% A_{111}	1	1	1	1	1		1	1	1 1
	50% A_{000}	0	0	0	0	0		0	0	0 0
50% A_{111} and 50% A_{100}	50% A_{111}	1	1	1	1	1		1	1	1 1
	50% A_{100}	1	0	1	1	0		0	1	0 0
50% A_{111}	50% A_{111}	1	1	1	1	1		1	1	1 1

	EMU Basis Vectors	$A_{1,2} \times A_1$			$A_{1,2} \times A_3$			$A_{1,2,3}$			
	EMUs	$A_{1,2}$		A_1	$A_{1,2}$		A_3	$A_{1,2,3}$			
	Atoms	1	2	1	1	2	3	1	2	3	
Tracer Compositions	Tracers										
$50\%\ A_{010}$ and $50\%\ A_{000}$	$50\%\ A_{010}$	0	1	0	0	1		0	0	1	0
$50\%\ A_{111}$ and $50\%\ A_{110}$	$50\%\ A_{111}$	1	1	1	1	1		1	1	1	1
	$50\%\ A_{110}$	1	1	1	1	1		0	1	1	0
$50\%\ A_{111}$ and $50\%\ A_{001}$	$50\%\ A_{111}$	1	1	1	1	1		1	1	1	1
	$50\%\ A_{001}$	0	0	0	0	0		0	0	0	0
$50\%\ A_{111}$ and $50\%\ A_{101}$	$50\%\ A_{111}$	1	1	1	1	1		1	1	1	1
	$50\%\ A_{101}$	1	0	1	1	0		0	1	0	0
$50\%\ A_{111}$ and $50\%\ A_{011}$	$50\%\ A_{111}$	1	1	1	1	1		1	1	1	1
	$50\%\ A_{011}$	0	1	0	0	1		0	0	1	0
$100\%\ A_{111}$	$100\%\ A_{111}$	1	1	1	1	1		1	1	1	1

Mass Isotopomer Distribution

			M+0	M+1	M+2	M+3
$100\%\ A_{000}$		$A_{1,2} \times A_1$	100.0	0.0	0.0	0.0
		$A_{1,2} \times A_3$	100.0	0.0	0.0	0.0
		$A_{1,2,3}$	100.0	0.0	0.0	0.0
$50\%\ A_{100}$ and $50\%\ A_{000}$		$A_{1,2} \times A_1$	25.0	50.0	25.0	0.0
		$A_{1,2} \times A_3$	50.0	50.0	0.0	0.0
		$A_{1,2,3}$	50.0	50.0	0.0	0.0
$100\%\ A_{100}$		$A_{1,2} \times A_1$	0.0	0.0	100.0	0.0
		$A_{1,2} \times A_3$	0.0	100.0	0.0	0.0
		$A_{1,2,3}$	0.0	100.0	0.0	0.0
$50\%\ A_{010}$ and $50\%\ A_{000}$		$A_{1,2} \times A_1$	50.0	50.0	0.0	0.0
		$A_{1,2} \times A_3$	50.0	50.0	0.0	0.0
		$A_{1,2,3}$	50.0	50.0	0.0	0.0
$50\%\ A_{010}$ and $50\%\ A_{100}$		$A_{1,2} \times A_1$	0.0	50.0	50.0	0.0
		$A_{1,2} \times A_3$	0.0	100.0	0.0	0.0

			M+0	M+1	M+2	M+3
		$A_{1,2,3}$	0.0	100.0	0.0	0.0
100% A_{010}		$A_{1,2} \times A_1$	0.0	100.0	0.0	0.0
		$A_{1,2} \times A_3$	0.0	100.0	0.0	0.0
		$A_{1,2,3}$	0.0	100.0	0.0	0.0
50% A_{110}		$A_{1,2} \times A_1$	25.0	25.0	25.0	25.0
	and					
50% A_{000}		$A_{1,2} \times A_3$	50.0	0.0	50.0	0.0
		$A_{1,2,3}$	50.0	0.0	50.0	0.0
50% A_{110}		$A_{1,2} \times A_1$	0.0	0.0	50.0	50.0
	and					
50% A_{100}		$A_{1,2} \times A_3$	0.0	50.0	50.0	0.0
		$A_{1,2,3}$	0.0	50.0	50.0	0.0
50% A_{110}		$A_{1,2} \times A_1$	0.0	25.0	50.0	25.0
	and					
50% A_{010}		$A_{1,2} \times A_3$	0.0	50.0	50.0	0.0
		$A_{1,2,3}$	0.0	50.0	50.0	0.0
100% A_{110}		$A_{1,2} \times A_1$	0.0	0.0	0.0	100.0
		$A_{1,2} \times A_3$	0.0	0.0	100.0	0.0
		$A_{1,2,3}$	0.0	0.0	100.0	0.0
50% A_{001}		$A_{1,2} \times A_1$	100.0	0.0	0.0	0.0
	and					
50% A_{000}		$A_{1,2} \times A_3$	50.0	50.0	0.0	0.0
		$A_{1,2,3}$	50.0	50.0	0.0	0.0
50% A_{001}		$A_{1,2} \times A_1$	25.0	50.0	25.0	0.0
	and					
50% A_{100}		$A_{1,2} \times A_3$	25.0	50.0	25.0	0.0
		$A_{1,2,3}$	0.0	100.0	0.0	0.0
50% A_{001}		$A_{1,2} \times A_1$	50.0	50.0	0.0	0.0
	and					
50% A_{010}		$A_{1,2} \times A_3$	25.0	50.0	25.0	0.0
		$A_{1,2,3}$	0.0	100.0	0.0	0.0
50% A_{001}		$A_{1,2} \times A_1$	25.0	25.0	25.0	25.0
	and					
50% A_{110}		$A_{1,2} \times A_3$	25.0	25.0	25.0	25.0
		$A_{1,2,3}$	0.0	50.0	50.0	0.0
100% A_{001}		$A_{1,2} \times A_1$	100.0	0.0	0.0	0.0
		$A_{1,2} \times A_3$	0.0	100.0	0.0	0.0
		$A_{1,2,3}$	0.0	100.0	0.0	0.0
50% A_{101}		$A_{1,2} \times A_1$	25.0	50.0	25.0	0.0
	and					
50% A_{000}						

			M+0	M+1	M+2	M+3
		$A_{1,2} \times A_3$	25.0	50.0	25.0	0.0
		$A_{1,2,3}$	50.0	0.0	50.0	0.0
50% A_{101}	**and**	$A_{1,2} \times A_1$	0.0	0.0	100.0	0.0
50% A_{100}		$A_{1,2} \times A_3$	0.0	50.0	50.0	0.0
		$A_{1,2,3}$	0.0	50.0	50.0	0.0
50% A_{101}	**and**	$A_{1,2} \times A_1$	0.0	50.0	50.0	0.0
50% A_{010}		$A_{1,2} \times A_3$	0.0	50.0	50.0	0.0
		$A_{1,2,3}$	0.0	50.0	50.0	0.0
50% A_{101}	**and**	$A_{1,2} \times A_1$	0.0	0.0	50.0	50.0
50% A_{110}		$A_{1,2} \times A_3$	0.0	25.0	50.0	25.0
		$A_{1,2,3}$	0.0	0.0	100.0	0.0
50% A_{101}	**and**	$A_{1,2} \times A_1$	25.0	50.0	25.0	0.0
50% A_{001}		$A_{1,2} \times A_3$	25.0	50.0	25.0	0.0
		$A_{1,2,3}$	50.0	0.0	50.0	0.0
100% A_{101}		$A_{1,2} \times A_1$	0.0	0.0	100.0	0.0
		$A_{1,2} \times A_3$	0.0	0.0	100.0	0.0
		$A_{1,2,3}$	0.0	0.0	100.0	0.0
50% A_{011}	**and**	$A_{1,2} \times A_1$	50.0	50.0	0.0	0.0
50% A_{000}		$A_{1,2} \times A_3$	25.0	50.0	25.0	0.0
		$A_{1,2,3}$	50.0	0.0	50.0	0.0
50% A_{011}	**and**	$A_{1,2} \times A_1$	0.0	50.0	50.0	0.0
50% A_{100}		$A_{1,2} \times A_3$	0.0	50.0	50.0	0.0
		$A_{1,2,3}$	0.0	50.0	50.0	0.0
50% A_{011}	**and**	$A_{1,2} \times A_1$	0.0	100.0	0.0	0.0
50% A_{010}		$A_{1,2} \times A_3$	0.0	50.0	50.0	0.0
		$A_{1,2,3}$	0.0	50.0	50.0	0.0
50% A_{011}	**and**	$A_{1,2} \times A_1$	0.0	25.0	50.0	25.0
50% A_{110}		$A_{1,2} \times A_3$	0.0	25.0	50.0	25.0
		$A_{1,2,3}$	0.0	0.0	100.0	0.0
50% A_{011}	**and**	$A_{1,2} \times A_1$	50.0	50.0	0.0	0.0
50% A_{001}		$A_{1,2} \times A_3$	25.0	50.0	25.0	0.0
		$A_{1,2,3}$	50.0	0.0	50.0	0.0

			M+0	M+1	M+2	M+3
50% A_{011} and 50% A_{101}		$A_{1,2} \times A_1$	0.0	50.0	50.0	0.0
		$A_{1,2} \times A_3$	0.0	50.0	50.0	0.0
		$A_{1,2,3}$	0.0	50.0	50.0	0.0
100% A_{011}		$A_{1,2} \times A_1$	0.0	100.0	0.0	0.0
		$A_{1,2} \times A_3$	0.0	0.0	100.0	0.0
		$A_{1,2,3}$	0.0	0.0	100.0	0.0
50% A_{111} and 50% A_{000}		$A_{1,2} \times A_1$	25.0	25.0	25.0	25.0
		$A_{1,2} \times A_3$	25.0	25.0	25.0	25.0
		$A_{1,2,3}$	50.0	0.0	0.0	50.0
50% A_{111} and 50% A_{100}		$A_{1,2} \times A_1$	0.0	0.0	50.0	50.0
		$A_{1,2} \times A_3$	0.0	25.0	50.0	25.0
		$A_{1,2,3}$	0.0	50.0	0.0	50.0
50% A_{111} and 50% A_{010}		$A_{1,2} \times A_1$	0.0	25.0	50.0	25.0
		$A_{1,2} \times A_3$	0.0	25.0	50.0	25.0
		$A_{1,2,3}$	0.0	50.0	0.0	50.0
50% A_{111} and 50% A_{110}		$A_{1,2} \times A_1$	0.0	0.0	0.0	100.0
		$A_{1,2} \times A_3$	0.0	0.0	50.0	50.0
		$A_{1,2,3}$	0.0	0.0	50.0	50.0
50% A_{111} and 50% A_{001}		$A_{1,2} \times A_1$	25.0	25.0	25.0	25.0
		$A_{1,2} \times A_3$	25.0	25.0	25.0	25.0
		$A_{1,2,3}$	50.0	0.0	0.0	50.0
50% A_{111} and 50% A_{101}		$A_{1,2} \times A_1$	0.0	0.0	50.0	50.0
		$A_{1,2} \times A_3$	0.0	25.0	50.0	25.0
		$A_{1,2,3}$	0.0	50.0	0.0	50.0
50% A_{111} and 50% A_{011}		$A_{1,2} \times A_1$	0.0	25.0	50.0	25.0
		$A_{1,2} \times A_3$	0.0	25.0	50.0	25.0
		$A_{1,2,3}$	0.0	50.0	0.0	50.0
100% A_{111}		$A_{1,2} \times A_1$	0.0	0.0	0.0	100.0
		$A_{1,2} \times A_3$	0.0	0.0	0.0	100.0
		$A_{1,2,3}$	0.0	0.0	0.0	100.0

4 TCA Cycle metabolic model

In [22]:
```
display(Image('TCA.png'))

```

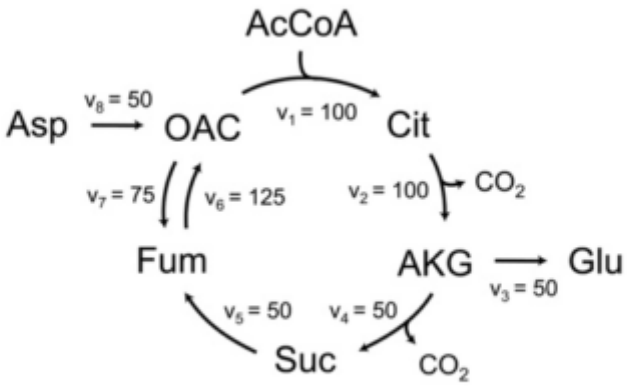


Fig. 10. Simplified model of the TCA cycle. The assumed fluxes have arbitrary units.

Table 3
Stoichiometry and atom transitions for the reactions in the TCA cycle model.

Reaction number	Reaction stoichiometry	Carbon atom transitions
1	OAC + AcCoA → Cit	abcd + ef → dcbfea
2	Cit → AKG + CO_2	abcdef → abcde + f
3	AKG → Glu	abcde → abcde
4	AKG → Suc + CO_2	abcde → ½ bcde + ½ edcb + a
5	Suc → Fum	½ abcd + ½ dcba → ½ abcd + ½ dcba
6	Fum → OAC	½ abcd + ½ dcba → abcd
7	OAC → Fum	abcd → ½ abcd + ½ dcba
8	Asp → OAC	abcd → abcd

4.1 TCA Cycle metabolic reaction network

In [23]:

```
tca_metabolic_network = Model('TCA_Metabolic_Network')
Asp = Metabolite('Asp')
OAC = Metabolite('OAC')
AcCoA = Metabolite('AcCoA')
Cit = Metabolite('Cit')
CO2 = Metabolite('CO2')
AKG = Metabolite('AKG')
Glu = Metabolite('Glu')
Suc = Metabolite('Suc')
Fum = Metabolite('Fum')

v1 = Reaction('v1')
v2 = Reaction('v2')
v3 = Reaction('v3')
v4 = Reaction('v4')
v5 = Reaction('v5')
v6 = Reaction('v6')
v7 = Reaction('v7')
v8 = Reaction('v8')
b1 = Reaction('b1')
b2 = Reaction('b2')
b3 = Reaction('b3')
tca_metabolic_network.add_metabolites([Asp,OAC,AcCoA,Cit,CO2,AKG,Glu,S
tca_metabolic_network.add_reactions([v1,v2,v3,v4,v5,v6,v7,v8,b1,b2,b3,

tca_metabolic_network.reactions.v1.build_reaction_from_string('OAC + A
tca_metabolic_network.reactions.v2.build_reaction_from_string('Cit -->
tca_metabolic_network.reactions.v3.build_reaction_from_string('AKG -->
tca_metabolic_network.reactions.v4.build_reaction_from_string('AKG -->
tca_metabolic_network.reactions.v5.build_reaction_from_string('Suc -->
tca_metabolic_network.reactions.v6.build_reaction_from_string('Fum -->
tca_metabolic_network.reactions.v7.build_reaction_from_string('OAC -->
tca_metabolic_network.reactions.v8.build_reaction_from_string('Asp -->

tca_metabolic_network.reactions.b1.build_reaction_from_string('--> Asp
tca_metabolic_network.reactions.b2.build_reaction_from_string('--> AcC
tca_metabolic_network.reactions.b3.build_reaction_from_string('Glu -->
tca_metabolic_network.reactions.b4.build_reaction_from_string('CO2 -->

tca_metabolic_network.reactions.v1.notes['CARBON TRANSITIONS'] = 'OAC
tca_metabolic_network.reactions.v2.notes['CARBON TRANSITIONS'] = 'CIT
tca_metabolic_network.reactions.v3.notes['CARBON TRANSITIONS'] = 'AKG
tca_metabolic_network.reactions.v4.notes['CARBON TRANSITIONS'] = 'AKG
tca_metabolic_network.reactions.v5.notes['CARBON TRANSITIONS'] = 'Suc
tca_metabolic_network.reactions.v6.notes['CARBON TRANSITIONS'] = 'Fum
tca_metabolic_network.reactions.v7.notes['CARBON TRANSITIONS'] = 'OAC
tca_metabolic_network.reactions.v8.notes['CARBON TRANSITIONS'] = 'Asp

S = get_stoichiometric_matrix( tca_metabolic_network )
rxns, mets = S.columns, S.index


#display(Markdown('### Nullspace of Extended Metabolic Reaction Netwo
#display(get_nullspace( S ).T)
display(Markdown('### Elementary modes of Extended Metabolic Reaction
```

```
elmo = calculate_elementary_modes( tca_metabolic_network,verbose=False
display(elmo)
display(flux_map(tca_metabolic_network, display_name_format=lambda x:
         flux_dict={rxn.id: elmo.sum(axis=0)[rxn.id] for rxn in tca_me

for em in elmo.index:
    with tca_metabolic_network:
        tca_metabolic_network.remove_reactions([r for r in elmo.columr
        display(Markdown('### Elementary mode {} '.format(em)))
        display(flux_map(tca_metabolic_network,
                         display_name_format=lambda x: str(x.id),
                         flux_dict={rxn.id: elmo.loc[em,rxn.id]
                                for rxn in tca_metabolic_network.reactions

```

Elementary modes of Extended Metabolic Reaction Network

```
/Users/zuck016/.pyenv/versions/anaconda3-2020.11/lib/python3.8/subproces
s.py:849: RuntimeWarning: line buffering (buffering=1) isn't supported in
binary mode, the default buffer size will be used
  self.stderr = io.open(errread, 'rb', bufsize)
```

	v1	v2	v3	v4	v5	v6	v7	v8	b1	b2	b3	b4
EM1	1.0	1.0	1.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0
EM2	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0
EM3	0.5	0.5	0.0	0.5	0.5	0.5	0.0	0.0	0.0	0.5	0.0	1.0

Save JSON | Show Reaction Nodes | Download SVG

Elementary mode EM1

Save JSON | Show Reaction Nodes | Download SVG

Elementary mode EM2

Save JSON | Show Reaction Nodes | Download SVG

Elementary mode EM3

4.2 Combined EMU network for TCA Cycle

In [24]:
```
1
2 display(Markdown('### Figure 12\n'))
3 display(Image('Figure12.png'))
4
```

Figure 12

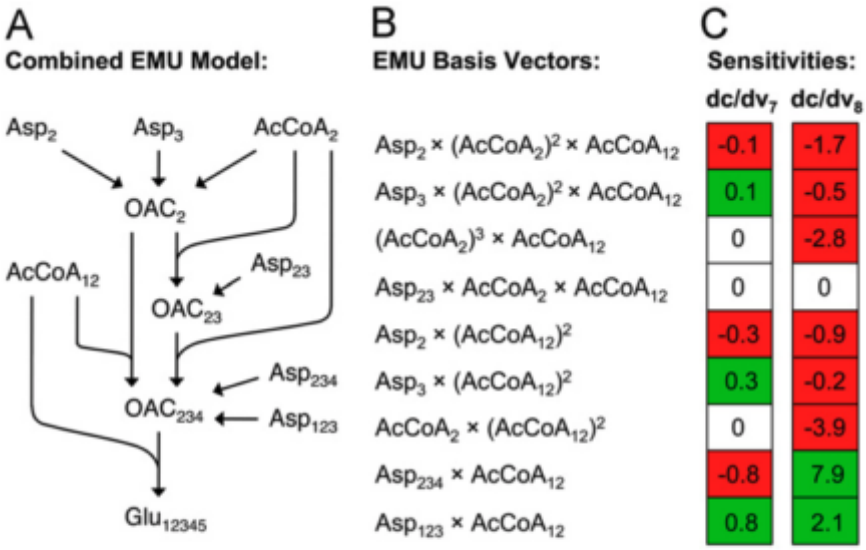


Fig. 12. (A) Combined EMU reaction network for the TCA cycle showing the flow of isotopic labeling from aspartate and acetyl-coenzyme A to glutamate. (B) Nine EMU basis vectors corresponding to the production of glutamate in the TCA cycle. (C) Sensitivity of contribution coefficients ($\times 10^3$) with respect to the free fluxes v_7 and v_8. In the case of dc/dv_7, there were three equal but opposite paired sensitivities. As a result, if an aspartate tracer was chosen such that $Asp_2 = Asp_3$ and $Asp_{123} = Asp_{234}$, each pair of sensitivities canceled, and functionality with respect to v_7 was lost.

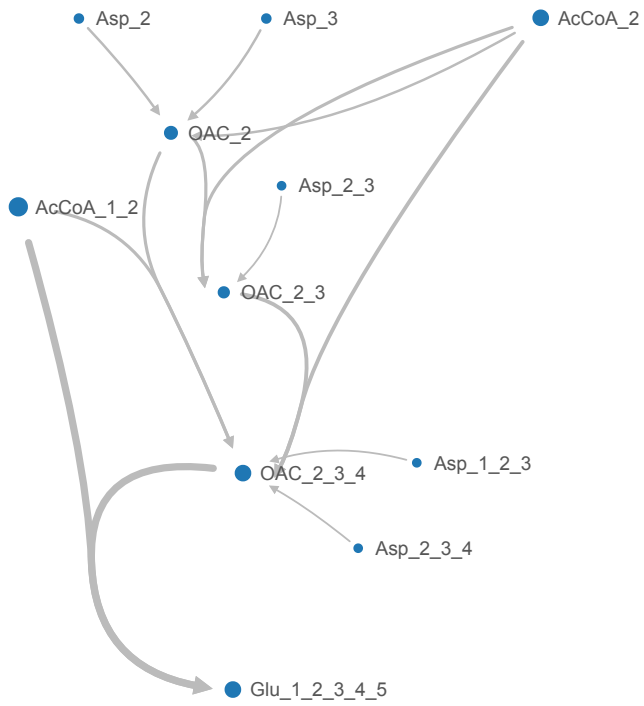
In [25]:

```
emu_network = Model('figure12_combined_EMU_network')
Asp_2 = Metabolite('Asp_2')
Asp_3 = Metabolite('Asp_3')
Asp_2_3 = Metabolite('Asp_2_3')
Asp_2_3_4 = Metabolite('Asp_2_3_4')
Asp_1_2_3 = Metabolite('Asp_1_2_3')
AcCoA_2 = Metabolite('AcCoA_2')
AcCoA_1_2 = Metabolite('AcCoA_1_2')
OAC_2 = Metabolite('OAC_2')
OAC_2_3_4 = Metabolite('OAC_2_3_4')
OAC_2_3 = Metabolite('OAC_2_3')
Glu_1_2_3_4_5 = Metabolite('Glu_1_2_3_4_5')

R1 = Reaction('R1')
R2 = Reaction('R2')
R3 = Reaction('R3')
R4 = Reaction('R4')
R5 = Reaction('R5')
R6 = Reaction('R6')
R7 = Reaction('R7')
R8 = Reaction('R8')
R9 = Reaction('R9')
R10 = Reaction('R10')
R11 = Reaction('R11')
R12 = Reaction('R12')
R13 = Reaction('R13')
R14 = Reaction('R14')
R15 = Reaction('R15')
R16 = Reaction('R16')
R17 = Reaction('R17')
R18 = Reaction('R18')

emu_network.add_metabolites(
    [Asp_2, Asp_3, Asp_2_3, Asp_2_3_4, Asp_1_2_3,
     AcCoA_2,AcCoA_1_2,
     OAC_2, OAC_2_3, OAC_2_3_4,
     Glu_1_2_3_4_5])

emu_network.add_reactions([R1, R2, R3, R4, R5, R6, R7,R8, R9, R10, R1

emu_network.reactions.R1.build_reaction_from_string('--> Asp_2')
emu_network.reactions.R2.build_reaction_from_string('--> Asp_3')
emu_network.reactions.R3.build_reaction_from_string('--> Asp_2_3')
emu_network.reactions.R4.build_reaction_from_string('--> Asp_2_3_4')
emu_network.reactions.R5.build_reaction_from_string('--> Asp_1_2_3')
emu_network.reactions.R6.build_reaction_from_string('--> AcCoA_2')
emu_network.reactions.R7.build_reaction_from_string('--> AcCoA_1_2')
emu_network.reactions.R8.build_reaction_from_string('Asp_2 --> OAC_2'
emu_network.reactions.R9.build_reaction_from_string('Asp_3 --> OAC_2'
emu_network.reactions.R10.build_reaction_from_string('AcCoA_2 --> OAC
emu_network.reactions.R11.build_reaction_from_string('AcCoA_2 + OAC_2
emu_network.reactions.R12.build_reaction_from_string('AcCoA_2 + OAC_2
emu_network.reactions.R13.build_reaction_from_string('Asp_2_3 --> OAC
emu_network.reactions.R14.build_reaction_from_string('Asp_2_3_4 --> C
```

```
emu_network.reactions.R15.build_reaction_from_string('Asp_1_2_3 --> C
emu_network.reactions.R16.build_reaction_from_string('OAC_2 + AcCoA_1
emu_network.reactions.R17.build_reaction_from_string('OAC_2_3_4 + AcC
emu_network.reactions.R18.build_reaction_from_string('Glu_1_2_3_4_5 -

S = get_stoichiometric_matrix( emu_network )
rxns, mets = S.columns, S.index
external_mets = [ 'Asp_2','Asp_3', 'AcCoA_2', 'AcCoA_1_2', 'Asp_2_3',
external_rxns = ['R1', 'R2', 'R3', 'R4', 'R5', 'R6','R7', 'R18']
output_met = 'Glu_1_2_3_4_5'
output_rxn = 'R18'

bv_order = ['$Asp_{2}\\times \\left(AcCoA_{2}\\right)^{2}\\times AcCo
            '$Asp_{3}\\times \\left(AcCoA_{2}\\right)^{2}\\times AcCo
            '$\\left(AcCoA_{2}\\right)^{3}\\times AcCoA_{1,2}$',
            '$AcCoA_{2}\\times AcCoA_{1,2}\\times Asp_{2,3}$',
            '$Asp_{2}\\times \\left(AcCoA_{1,2}\\right)^{2}$',
            '$Asp_{3}\\times \\left(AcCoA_{1,2}\\right)^{2}$',
            '$AcCoA_{2}\\times \\left(AcCoA_{1,2}\\right)^{2}$',
            '$AcCoA_{1,2}\\times Asp_{2,3,4}$',
            '$AcCoA_{1,2}\\times Asp_{1,2,3}$']

ebv, emu_elmo = get_EMU_basis_vectors_and_pathways( emu_network, exte
#cobra.io.save_json_model(emu_network, "figure12_combined_emu_network

display(Markdown('### D3Flux Combined EMU transition network\n'))
display(flux_map(cobra.io.load_json_model("figure12_combined_EMU_netw
         flux_dict={rxn.id: emu_elmo.sum(axis=0)[rxn.id] for rxn in e

display(Markdown('### EMU basis vectors \n'))
display(HTML(ebv.reset_index().set_index('EMU Basis Vector').loc[bv_o

#display(ebv['EMU Basis Vector'].values)


display(Markdown('### EMU Pathways \n'))
emu_elmo['EMU Basis Vector']  = ebv['EMU Basis Vector']
display(HTML(emu_elmo.reset_index().set_index('EMU Basis Vector').loc

for em in emu_elmo.index:
    with emu_network:
        emu_network.remove_reactions([r for r in emu_elmo.columns if
        display(Markdown('#### EMU Pathway {} ('.format(em) + ebv.loc
        display(flux_map(emu_network,
                         display_name_format=lambda x: str(x.id),
                         flux_dict={rxn.id: emu_elmo.loc[em,rxn.id]
                                    for rxn in emu_network.reactions}))
```

```
/Users/zuck016/.pyenv/versions/anaconda3-2020.11/lib/python3.8/subproces
s.py:849: RuntimeWarning: line buffering (buffering=1) isn't supported in
binary mode, the default buffer size will be used
  self.stderr = io.open(errread, 'rb', bufsize)
```

D3Flux Combined EMU transition network



Save JSON | Show Reaction Nodes | Download SVG

EMU basis vectors

	index	Asp_2	Asp_3	AcCoA_2	AcCoA_1_2	Asp_2_3	Asp_1_2_3	Asp_2_3_4	Glu_1_2_
EMU Basis Vector									
$Asp_2 \times (AcCoA_2)^2 \times AcCoA_{1,2}$	EM5	-1.0	-0.0	-2.0	-1.0	-0.0	-0.0	-0.0	
$Asp_3 \times (AcCoA_2)^2 \times AcCoA_{1,2}$	EM6	-0.0	-1.0	-2.0	-1.0	-0.0	-0.0	-0.0	

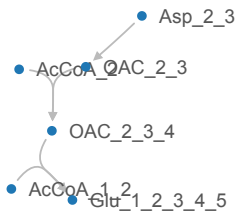
EMU Basis Vector	index	Asp_2	Asp_3	AcCoA_2	AcCoA_1_2	Asp_2_3	Asp_1_2_3	Asp_2_3_4	Glu_1_2_
$(AcCoA_2)^3 \times AcCoA_{1,2}$	EM9	-0.0	-0.0	-3.0	-1.0	-0.0	-0.0	-0.0	
$AcCoA_2 \times AcCoA_{1,2} \times Asp_{2,3}$	EM1	-0.0	-0.0	-1.0	-1.0	-1.0	-0.0	-0.0	
$Asp_2 \times \left(AcCoA_{1,2}\right)^2$	EM7	-1.0	-0.0	-0.0	-2.0	-0.0	-0.0	-0.0	
$Asp_3 \times \left(AcCoA_{1,2}\right)^2$	EM4	-0.0	-1.0	-0.0	-2.0	-0.0	-0.0	-0.0	
$AcCoA_2 \times \left(AcCoA_{1,2}\right)^2$	EM8	-0.0	-0.0	-1.0	-2.0	-0.0	-0.0	-0.0	
$AcCoA_{1,2} \times Asp_{2,3,4}$	EM2	-0.0	-0.0	-0.0	-1.0	-0.0	-0.0	-1.0	
$AcCoA_{1,2} \times Asp_{1,2,3}$	EM3	-0.0	-0.0	-0.0	-1.0	-0.0	-1.0	-0.0	

EMU Pathways

EMU Basis Vector	index	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15	R16	I
$Asp_2 \times (AcCoA_2)^2 \times AcCoA_{1,2}$	EM5	1.0	0.0	0.0	0.0	0.0	2.0	1.0	1.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	
$Asp_3 \times (AcCoA_2)^2 \times AcCoA_{1,2}$	EM6	0.0	1.0	0.0	0.0	0.0	2.0	1.0	0.0	1.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	
$(AcCoA_2)^3 \times AcCoA_{1,2}$	EM9	0.0	0.0	0.0	0.0	0.0	3.0	1.0	0.0	0.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	
$AcCoA_2 \times AcCoA_{1,2} \times Asp_{2,3}$	EM1	0.0	0.0	1.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	
$Asp_2 \times \left(AcCoA_{1,2}\right)^2$	EM7	1.0	0.0	0.0	0.0	0.0	0.0	2.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	
$Asp_3 \times \left(AcCoA_{1,2}\right)^2$	EM4	0.0	1.0	0.0	0.0	0.0	0.0	2.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	

EMU Basis Vector	index	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15	R16
$AcCoA_2 \times \left(AcCoA_{1,2}\right)^2$	EM8	0.0	0.0	0.0	0.0	0.0	1.0	2.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0
$AcCoA_{1,2} \times Asp_{2,3,4}$	EM2	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
$AcCoA_{1,2} \times Asp_{1,2,3}$	EM3	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0

EMU Pathway EM1 ($AcCoA_2 \times AcCoA_{1,2} \times Asp_{2,3}$)

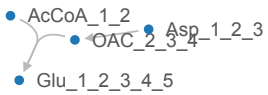


Save JSON | Show Reaction Nodes | Download SVG

EMU Pathway EM2 ($AcCoA_{1,2} \times Asp_{2,3,4}$)

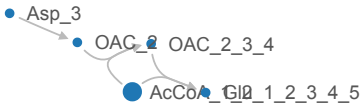
Save JSON | Show Reaction Nodes | Download SVG

EMU Pathway EM3 ($AcCoA_{1,2} \times Asp_{1,2,3}$)



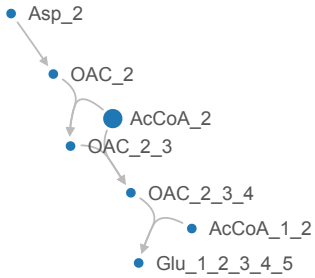
Save JSON | Show Reaction Nodes | Download SVG

EMU Pathway EM4 ($Asp_3 \times \left(AcCoA_{1,2}\right)^2$)



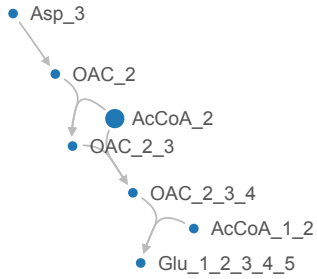
Save JSON | Show Reaction Nodes | Download SVG

EMU Pathway EM5 ($Asp_2 \times (AcCoA_2)^2 \times AcCoA_{1,2}$)

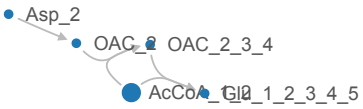


Save JSON | Show Reaction Nodes | Download SVG

EMU Pathway EM6 ($Asp_3 \times (AcCoA_2)^2 \times AcCoA_{1,2}$)

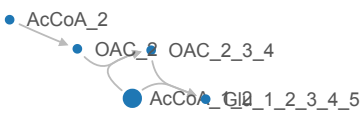


EMU Pathway EM7 ($Asp_2 \times \left(AcCoA_{1,2}\right)^2$)



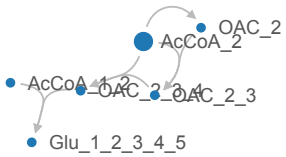
Save JSON | Show Reaction Nodes | Download SVG

EMU Pathway EM8 ($AcCoA_2 \times \left(AcCoA_{1,2}\right)^2$)



Save JSON | Show Reaction Nodes | Download SVG

EMU Pathway EM9 ($(AcCoA_2)^3 \times AcCoA_{1,2}$)



Save JSON | Show Reaction Nodes | Download SVG

4.3 Tracer Compositions for TCA EMU Network

In [26]:

```
import itertools

display(Image("Figure11.png"))
Asp_1_2_3_4 = EMU("Asp_1_2_3_4")
AcCoA_1_2 = EMU("AcCoA_1_2")

Asp_isotopic_labeling_states = []
AcCoA_isotopic_labeling_states = []
for i in range(2 ** Asp_1_2_3_4.get_num_atoms()):
    Asp_isotopic_labeling_states.append(
        pd.DataFrame(
            [{"%": 100, 1: (i) & 1, 2: (i >> 1) & 1, 3: (i >> 2) & 1,
            index=[
                r"$100\%"
                + r"\ \text{Asp}_{%d%d%d%d}$"
                % ((i) & 1, (i >> 1) & 1, (i >> 2) & 1, (i >> 3) & 1)
            ],
        )
    )
for j in range(2 ** AcCoA_1_2.get_num_atoms()):
    AcCoA_isotopic_labeling_states.append(
        pd.DataFrame(
            [{"%": 100, 1: (j) & 1, 2: (j >> 1) & 1}],
            index=[r"$100\%" + r"\ \text{AcCoA}_{%d%d}$" % ((j) & 1, (
        )
    )

tracer_mixtures = [
    dict(
        Asp=Tracer("Asp", Asp_1_2_3_4.get_num_atoms(), Asp_ils),
        AcCoA=Tracer("AcCoA", AcCoA_1_2.get_num_atoms(), AcCoA_ils),
    )
    for Asp_ils, AcCoA_ils in itertools.product(
        Asp_isotopic_labeling_states, AcCoA_isotopic_labeling_states
    )
]
emu_basis_vectors = [EMU_Basis_Vector(ebv.iloc[e, :-1]) for e in range
mid_comparison = {}
isl_comparison = {}
for tracers in tracer_mixtures:
    for emu_basis_vector in emu_basis_vectors:
        for emu in emu_basis_vector.get_input_emus():
            emu_isl = emu.get_isotopic_labeling_state(tracers)
            emu_isl.index = pd.MultiIndex.from_tuples(
                [
                    (" and ".join([tracers[t].get_name() for t in sort
                    for a in emu_isl.index
                ]
            )
            for atom in emu_isl.columns:
                key = (emu_basis_vector.get_name(), emu.get_name(latex
                if key in isl_comparison:
                    isl_comparison[key] = isl_comparison[key].append(e
                else:
                    isl_comparison[key] = emu_isl[atom]
        mid_comparison[
```

```
            " and ".join([tracers[t].get_name() for t in sorted(tracer
            emu_basis_vector.get_name(),
        ] = emu_basis_vector.get_mid(tracers)

display(Markdown("### Isotopomer labeling states"))
isl_df = pd.DataFrame(isl_comparison)
isl_df.columns = isl_df.columns.set_names(["EMU Basis Vectors", "EMUs'
isl_df.index = isl_df.index.set_names(["Tracer Compositions", "Tracers
# display(HTML(isl_df.to_html()))
display(Markdown("### Mass Isotopomer Distribution"))
mid_df = pd.DataFrame(mid_comparison)
mid_df.columns = mid_df.columns.set_names(["Tracer Compositions", "EMU


# display(HTML(mid_df.T.to_html()))
rank4 = mid_df.xs(
    r"$100\%" + r"\ \text{AcCoA}_{00}$ and $100\%" + r"\ \text{Asp}_{1
    axis=1,
    level="Tracer Compositions",
)
display(rank4)
np.linalg.matrix_rank(rank4)
table_template = "|{}|\n" * 6

accoa_isl = [accoa.index[0] for accoa in AcCoA_isotopic_labeling_state
asp_isl = [asp.index[0] for asp in Asp_isotopic_labeling_states]

rank_df = pd.DataFrame(0, index=accoa_isl, columns=asp_isl)
for accoa in accoa_isl:
    for asp in asp_isl:
        rank_df.loc[accoa, asp] = np.linalg.matrix_rank(
            mid_df.xs(accoa + " and " + asp, axis=1, level="Tracer Cor
        )
display(rank_df)
# table = table_template.format( ' | '.join([' | '] + asp_isl),
#              '|'.join(['-----'] + ['-----' for _ in asp_isl]),
#              ' | '.join([accoa_isl[0]] + [str(np.linalg.matrix_rank(
#              ' | '.join([accoa_isl[1]] + [str(np.linalg.matrix_rank(
#              ' | '.join([accoa_isl[2]] + [str(np.linalg.matrix_rank(
#              ' | '.join([accoa_isl[3]] + [str(np.linalg.matrix_rank(
# display(Markdown(table))
```

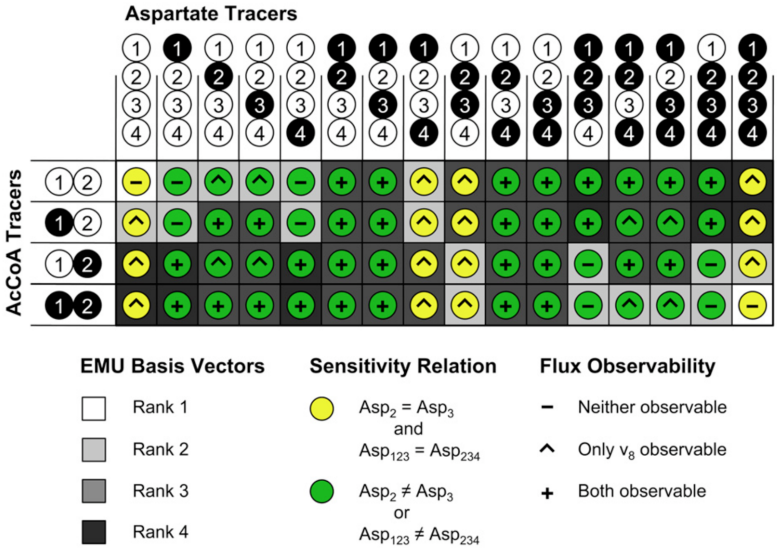


Fig. 11. Analysis of flux observability and EMU basis vector classification for the TCA cycle. Flux observability was evaluated for all combinations of aspartate tracers and acetyl-CoA tracers. The square colors denote the EMU basis vector rank, the circle colors correspond to whether v_7 functionality was lost due to tracer selection, and the symbol within the circle denotes the flux observability. Observability of both free fluxes in the model required an EMU basis vector rank of 3 or 4, and required that $Asp_2 \neq Asp_3$ and/or $Asp_{123} \neq Asp_{234}$.

Isotopomer labeling states

Mass Isotopomer Distribution

EMU Basis Vectors	$Asp_{2,3} \times AcCoA_{1,2} \times AcCoA_2$	$Asp_{2,3,4} \times AcCoA_{1,2}$	$Asp_{1,2,3} \times AcCoA_{1,2}$	$Asp_3 \times (AcCoA_{1,2})^2$	$Asp_2 \times (AcCoA_2)^2 \times AcCoA_{1,2}$	$Asp_3 \times (AcCoA_2)^2 \times$
M+0	0.0	0.0	0.0	0.0	0.0	
M+1	0.0	0.0	0.0	100.0	100.0	
M+2	100.0	100.0	0.0	0.0	0.0	
M+3	0.0	0.0	100.0	0.0	0.0	
M+4	0.0	0.0	0.0	0.0	0.0	
M+5	0.0	0.0	0.0	0.0	0.0	

	$100\%\ Asp_{0000}$	$100\%\ Asp_{1000}$	$100\%\ Asp_{0100}$	$100\%\ Asp_{1100}$	$100\%\ Asp_{0010}$	$100\%\ Asp$
$100\%\ AcCoA_{00}$	1	2	2	3	2	
$100\%\ AcCoA_{10}$	2	2	3	3	3	
$100\%\ AcCoA_{01}$	4	4	3	3	3	
$100\%\ AcCoA_{11}$	4	4	3	3	3	