SunFlow Tutorial Part I

Supply Network Modeling & Optimization

© 2020 AI-Technologies

SunFlow is a Python module for modelling and optimizing supply chain networks. Its name is derived from **Su**pply **n**etwork **Flow**.

This tutorial illustrates how you model and evaluate your supply chain networks with SunFlow. Step-by-step it tells you how to model simple and complex logistics networks including custom tariffs, normal and compound capacity constraints up to recipe based production and manufacturing networks covering substitutional products along the value chain.

- simple and complex logistics networks
- custom tariffs (coming up in the next release)
- normal and compound capacities
- networks with production/manufacturing incl. the usage of recipes
- substitutional products and alternatives

The source code of the examples described below are in the directory examples in the format of Anaconda Notebook files.

Chapter 1

Chapter 1: A Distribution Network

Building supply networks with SunFlow is quite easy. To see the steps look at the example below.

Let's assume we want to supply 20 pallets of wine (our product) from a european winery (our supplier) to Walmart (our customer) either through a warehouse in New York or New Orleans. Which option is the cheaper one? To answer this question we first build a model of the potential product flows and optimize it afterwards.

Create a new Net

A supply net is created by SupplyNet('name'). The given name identifies the net and becomes the name of its graphical chart file.

```
globals().clear(); import rwlib; from sunflow import SupplyNet

net = SupplyNet('Chapter 1 Distribution USA')
```

Build the Net

A supply net is a directed graph made by nodes and vertices. Once a new net is created, we start to build up its nodes step-by-step.

The first node is the initial source, the supplier.

```
supplier = net.source()
```

Now we add the two warehouses to the net. The link between the supplier's node and the warehouse nodes are made by the ditstibution() method's parameter which is set to 'supplier' which links the new node with their predecessor - here the supplier node.

```
new_york    = net.distribution(supplier)
new_orleans = net.distribution(supplier)
```

Finally add the customer node and connect it in the same way with its predecessors, the two warehouses. The customer's demand of 20 pallets is attached to this node by demand(20).

```
customer = net.delivery(new_york, new_orleans) .demand(20)
```

Compile the Net

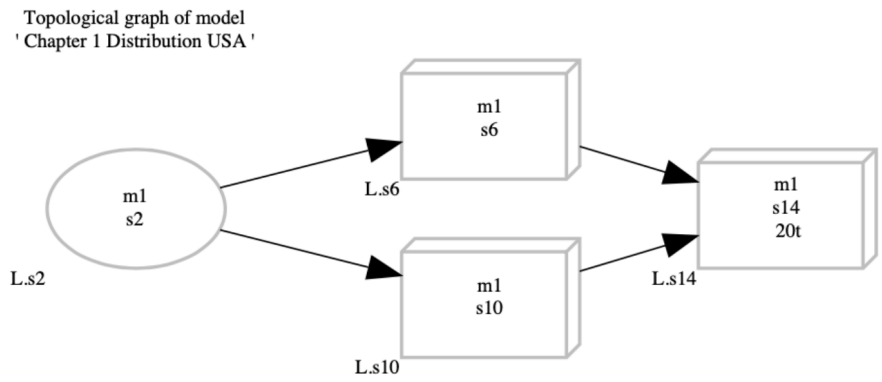
Compiling the net by the compile() method tells SunFlow to build and construct the internal model representation and prepare it for its further use.

```
net.compile()
```

Show the Net's Graph

Finally we let SunFlow create a graphical representation of the net by createGraph(). To display this graph use te method view().

```
net.showGraph().view()
```

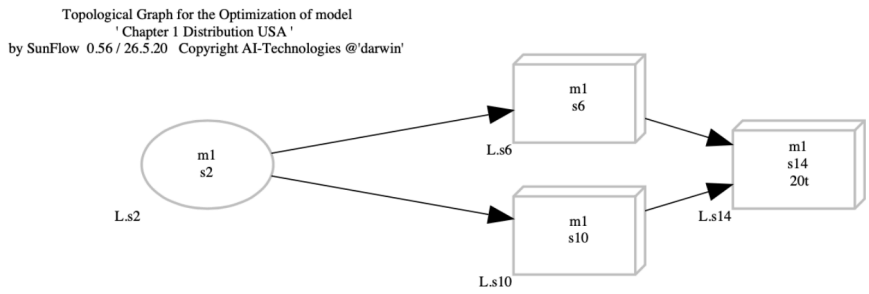


Chapter 2

Chapter 2: A Distribution Network

If we look at the network's graph from chapter 1, it doesn't look yet really good. Indeed we can see its structure, but the descriptions are not the way we want them.

So let's enhance it and add as well cost information.



Locations

The model contains a supplier, two warehouses and a customer. All of them are at physical locations whic we add now.

```
globals().clear(); import rwlib; from sunflow import SupplyNet,Logistics,Customer,Location,Freight,Producer

Europe      = Location('Europe')         # the Winery
New_York    = Location('New York')       # warehouse
New_Orleans = Location('New Orleans')    # warehouse
StLouis     = Location('StLouis')        # customer
```

Freight Costs

With these locations we can now add transportation costs to relevant relations. A relation is given by a tuple (from, to, freight rate). Freight rates are the transportation costs per unit. For instance, the tranportation cost of one container with 20 pallets is 2500€; therefore the freight rate is 2500/20 = 125€/pallet.

```
Freight(Europe,       New_York,      2500/20 )
Freight(Europe,       New_Orleans,   2900/20 )
Freight(New_York,     StLouis,       1200/20 )
Freight(New_Orleans,  StLouis,        820/20 )
print()
```

Producers, Customers, Logistics

Now we classify our actors into the groups Producers, Logistics and Customers and assign them a location enabling SunFlow to calculate the flow costs.

```
Winery          = Producer('Winery')          .at(Europe)
New_York_WH     = Logistics('New York WH')    .at(New_York)
New_Orleans_WH  = Logistics('New_Orleans WH') .at(New_Orleans)
Walmart         = Customer('Walmart')         .at(StLouis)
```

Create a new Net¶

```
net = SupplyNet('Chapter 2 Distribution USA')
```

Build the Net

As in chapter 1 we start with the first node - the supplier - which we name now as winery at location Winery. Normally there are 2 labels per node: the product name at this stage and the manufacturer, supplier, distribution hub or customer name. Since we dealing with the same product along the while change, we set the product name with .n(' ') to blank.

NOTE / HINT

It is helpful if you follow some naming policy. Here the node names start are always with lower letter, whereas the locations, customers, producers,... start always with capital letters. This allows to use the same names, but still referencong different objects.

```
winery = net.source().n(' ').by(Winery)
```

The same procedure with the logistics hubs.

```
new_york    = net.distribution(winery).n(' ')  .by(New_York_WH)
new_orleans = net.distribution(winery).n(' ')  .by(New_Orleans_WH)
```

And as well with the customer.

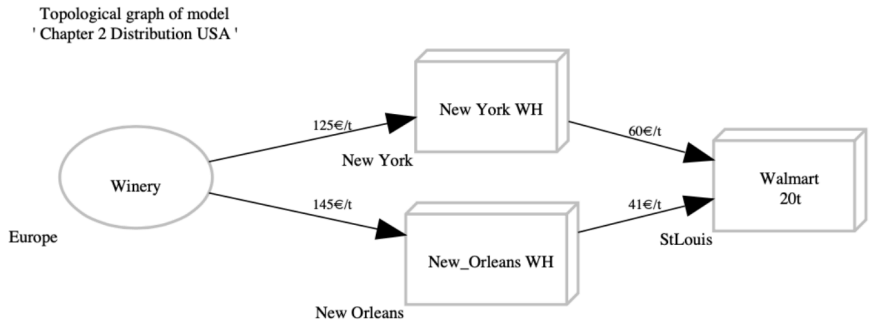
```
walmart = net.delivery(new_york, new_orleans).n(' ') .at(Walmart) .demand(20)
```

Compile and Graph the Net

Here is no change to chapter 1.

```
net.compile()
net.showGraph().view()
```

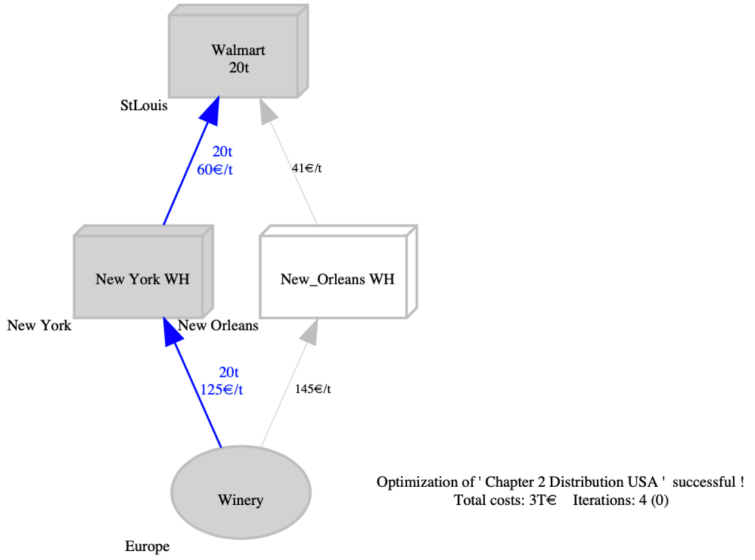
The net looks now much better. You see the labels and the freight rates attached to the relation arrows.



Execute the Net

To identify the cheapest path you got to excecute the net's flow, i.e. its optimizer will be excecuted and the result displayed.

```
net.execute().showGraph(orient='BT').view()
```



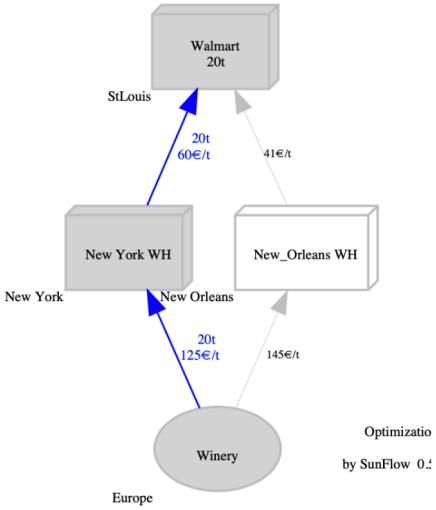
The blue lines indicate the cheapest path of the product flow and the quantities following the path ways.

Chapter 3

Chapter 3: A Distribution Network

Looking at the chart from chapter 2 we miss the handling costs at the warehouse locations which we will add now.

For this purpose we assume warehouse costs of 2€/pallet at New Orlenas and 4€/pallet at New York warehouse. The only change to chapter 2 are in sections "Conversion","Compile and Graph the Net" and "Excecute the Net" below. All other parts are identical to chapter 2.



Locations

The model contains a supplier, two warehouses and a customer. All of them are at physical locations whic we add now.

```
globals().clear(); import rwlib; from sunflow import SupplyNet,Logistics,Customer,Location,Freight,Producer

Europe      = Location('Europe')         # the Winery
New_York    = Location('New York')       # warehouse
New_Orleans = Location('New Orleans')    # warehouse
StLouis     = Location('StLouis')        # customer
```

Freight Costs

With these locations we can now add transportation costs to relevant relations. A relation is given by a tuple (from, to, freight rate). Freight rates are the transportation costs per unit. For instance, the tranportation cost of one container with 20 pallets is 2500€; therefore the freight rate is 2500/20 = 125€/pallet.

```
Freight(Europe,       New_York,      2500/20 )
Freight(Europe,       New_Orleans,   2900/20 )
Freight(New_York,     StLouis,       1200/20 )
Freight(New_Orleans,  StLouis,        820/20 )
print()
```

Producers, Customers, Logistics

Now we classify our actors into the groups Producers, Logistics and Customers and assign them a location enabling SunFlow to calculate the flow costs.

```
Winery          = Producer('Winery')          .at(Europe)
New_York_WH     = Logistics('New York WH')    .at(New_York)
New_Orleans_WH  = Logistics('New_Orleans WH') .at(New_Orleans)
Walmart         = Customer('Walmart')         .at(StLouis)
```

Create a new Net¶

```
net = SupplyNet('Chapter 3 Distribution USA')
```

Build the Net

As in chapter 1 we start with the first node - the supplier - which we name now as winery at location Winery. Normally there are 2 labels per node: the product name at this stage and the manufacturer, supplier, distribution hub or customer name. Since we dealing with the same product along the while change, we set the product name with .n(' ') to blank.

NOTE / HINT

It is helpful if you follow some naming policy. Here the node names start are always with lower letter, whereas the locations, customers, producers,... start always with capital letters. This allows to use the same names, but still referencong different objects.

```
winery = net.source().n(' ').by(Winery)
```

Conversion

Production, manufacturing or just handling products produce costs. These cost - as well costs per unit - are attached to those label with the varcosts() method as shown below.

```
new_york    = net.distribution(winery).n(' ')  .by(New_York_WH)     .varcost(4)
new_orleans = net.distribution(winery).n(' ')  .by(New_Orleans_WH)  .varcost(2)
```

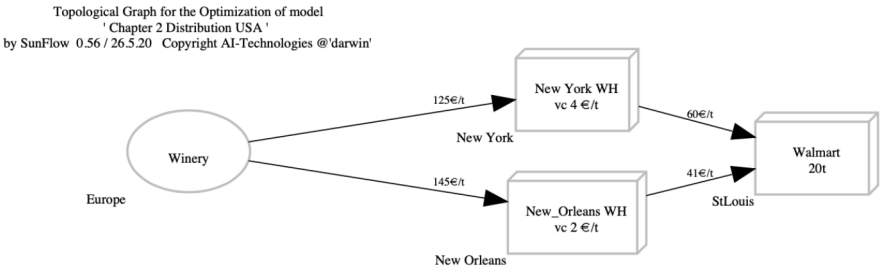
And as well with the customer.

```
walmart = net.delivery(new_york, new_orleans).n(' ') .at(Walmart) .demand(20)
```

Compile and Graph the Net

```
net.compile()
net.showGraph().view()
```

The net looks now much better. You see the labels and the freight rates attached to the relation arrows.

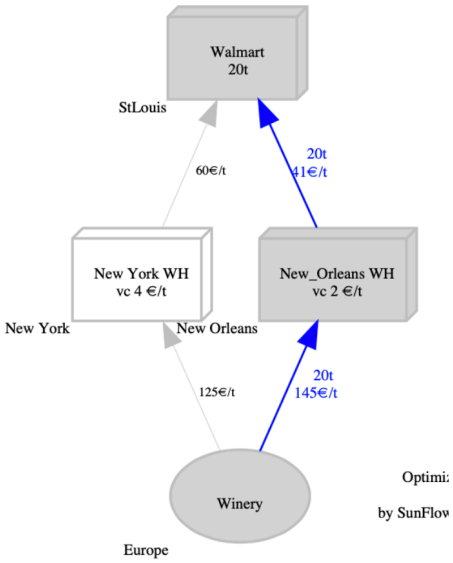


Execute the Net

To identify the cheapest path you got to excecute the net's flow, i.e. its optimizer will be excecuted and the result displayed.

If you compare the chart below with the one from chapter 2, you see that it changed. Instead of going through New York not the cheapest route is by New Orleans: the cheaper handling costs at New Orleans brought the change!

```
net.execute().showGraph(orient='BT').view()
```

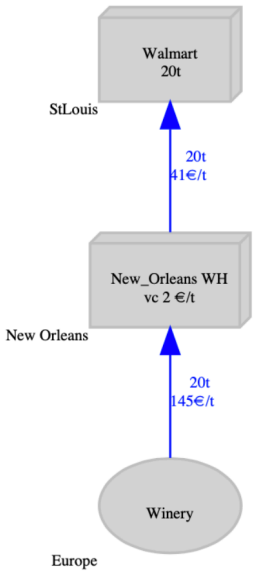


The blue lines indicate the cheapest path of the product flow and the quantities following the path ways.

The Optimal Flow

Finally you may just want to see the optimized network excluding all options not used, i.e. just show nodes receiving or sending blue arrows. This is done by setting the parameter flowOnly to True (default=False).

```
net.execute().showGraph(flowOnly=True, orient='BT').view()
```



Chapter 4

Chapter 4: A Distribution Network with Capacities

In the preceeding chapters we assumes an unlimited capacity which is not the case in reality. This chapter shows how to enhance your model with capacities. Here we assume that warehouse in New Orleans has a limit capacity of handling just 8 pallets, but one at NewYork 14 pallets. What will change in the model?

There are two ways of bringing capacities into a model: either by a defining a capacity by Capacity() or by assigning a capacity be capacity(). The first one (uppercase) is a creation of an object of class Capacity, where the second one (lowercase) is excecutung a method of class Net().

In our model we use both: the first one in section Capacities(), the second one section "Conversion Costs".

All other parts are identical to chapter 3 and the updated chart is shown in section "Excecute the Net" below.

Locations

```
globals().clear(); import rwlib;
from sunflow import SupplyNet,Logistics,Customer,Location,Freight,Producer, Capacity

Europe      = Location('Europe')         # the Winery
New_York    = Location('New York')       # warehouse
New_Orleans = Location('New Orleans')    # warehouse
StLouis     = Location('StLouis')        # customer
```

Freight Costs

```
Freight(Europe,       New_York,      2500/20 )
Freight(Europe,       New_Orleans,   2900/20 )
Freight(New_York,     StLouis,       1200/20 )
Freight(New_Orleans,  StLouis,        820/20 )
print()
```

Capacities

```
New_York_WH_capa = Capacity('New York', 14 )
```

Producers, Customers, Logistics

Capacity constraints can be applied at the time of a nodes declaration (here) or as an attachment to a net structure (see section Conversion Costs).

```
Winery          = Producer('Winery')          .at(Europe)
New_York_WH     = Logistics('New York WH')    .at(New_York)    .capacity(New_York_WH_capa)
New_Orleans_WH  = Logistics('New_Orleans WH') .at(New_Orleans)
Walmart         = Customer('Walmart')         .at(StLouis)
```

Create a new Net¶

```
net = SupplyNet('Chapter 2 Distribution USA').display(0)
```

Build the Net

```
winery = net.source().n(' ').by(Winery)
```

Conversion Costs

Here you find the second capacity constraint.

```
new_york    = net.distribution(winery).n(' ')  .by(New_York_WH)     .varcost(4)  #.capacity(New_York_WH_capa)
new_orleans = net.distribution(winery).n(' ')  .by(New_Orleans_WH)  .varcost(2)  .capacity(8)
```

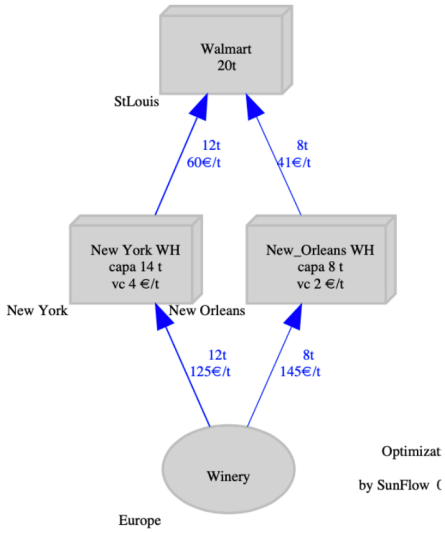
```
walmart = net.delivery(new_york, new_orleans).n(' ') .at(Walmart) .demand(20)
```

Compile and Graph the Net

```
net.compile()
```

Execute the Net

```
net.execute().showGraph(orient='BT').view()
```



Now, with the capacity contraints, only 8 pallets are flowing through New Orleans, the remaining are send through New York.

Chapter 5

Chapter 5: A Distribution Network with Compound Capacities

In the preceeding chapters we assumes an unlimited capacity which is not the case in reality. This chapter shows how to enhance your model with capacities. Here we assume that warehouse in New Orleans has a limit capacity of handling just 8 pallets, but one at NewYork 14 pallets. What will change in the model?

There are two ways of bringing capacities into a model: either by a defining a capacity by Capacity() or by assigning a capacity be capacity(). The first one (uppercase) is a creation of an object of class Capacity, where the second one (lowercase) is excecutung a method of class Net().

In our model we use both: the first one in section Capacities(), the second one section "Conversion Costs". All other parts are identical to chapter 3 and the updated chart is shown in section "Excecute the Net" below.

We assume two scenarios of which differ only by their compound capacities in section Capacities.

Locations

```
globals().clear(); import rwlib;
from sunflow import SupplyNet,Logistics,Customer,Location,Freight,Producer, Capacity

Europe      = Location('EUR')            # the Winery
New_York    = Location('New York')       # warehouse
New_Orleans = Location('New Orleans')    # warehouse
StLouis     = Location('StLouis')        # customer
Baltimore   = Location('Baltimore')      # !!!
```

Freight Costs

```
Freight(Europe,       New_York,      2500/20 )
Freight(Europe,       New_Orleans,   2900/20 )
Freight(New_York,     StLouis,       1200/20 )
Freight(New_Orleans,  StLouis,        820/20 )

Freight(Europe,       Baltimore,     2500/20 )  # !!!
Freight(Baltimore,    StLouis,       1500/20 )  # !!!

print()
```

Capacities

```
#NY_NOR_WH_capa = Capacity('NY + NOR', 50 )  # scenario 1
NY_NOR_WH_capa = Capacity('NY + NOR', 12 )  # scenario 2
```

Producers, Customers, Logistics

The warehouses at NewYork and NewOrleans got now a joint total capacity of 8 pallets and the warehouse at NewOrleans a single capacity of 5 pallets.

The conversion costs of Baltimore warehouse can be applied at declaration time (here) or later at building time (see NewYork and NewOrleans warehouses in section Conversions).

```
Winery          = Producer('Winery')          .at(Europe)
New_York_WH     = Logistics('New York WH')    .at(New_York)    .compoundCapacity(NY_NOR_WH_capa).capacity(30) # !!!
New_Orleans_WH  = Logistics('New_Orleans WH') .at(New_Orleans) .compoundCapacity(NY_NOR_WH_capa).capacity(9)  # !!!
Baltimore_WH    = Logistics('Baltimore WH')   .at(Baltimore)   .varcost(4)  # !!!
Walmart         = Customer('Walmart')         .at(StLouis)
```

Create a new Net¶

```
net = SupplyNet('Chapter 5 Distribution USA').display(0)
```

Build the Net

```
winery = net.source().n(' ').by(Winery)
```

Conversions

```
new_york    = net.distribution(winery).n(' ')  .by(New_York_WH)     .varcost(4)
new_orleans = net.distribution(winery).n(' ')  .by(New_Orleans_WH)  .varcost(2)
baltimore   = net.distribution(winery).n(' ')  .by(Baltimore_WH)    .varcost(4)
```

```
walmart = net.delivery(new_york, new_orleans, baltimore).n(' ') .at(Walmart) .demand(20)   # !!!
```

Compile and Graph the Net

```
net.compile()
```

Execute the Net

```
net.execute().showGraph(orient='BT').view()
```

Two Capacity Scenarios

We assumed the two scenarios:

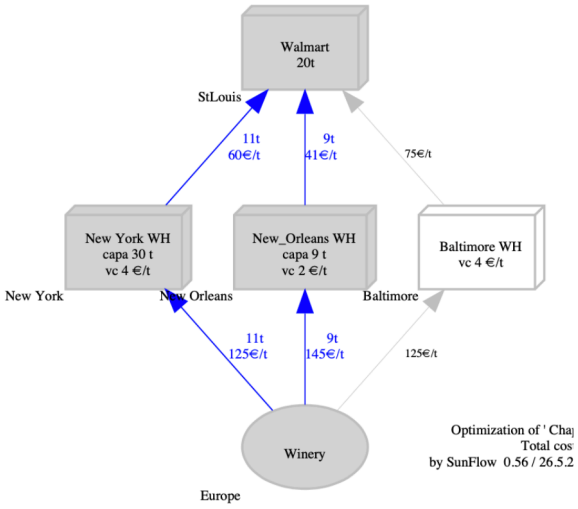
```
- scenario 1: capacity NewYork    = 30 pallets
              capacity NewOrleans = 9 pallets

- scenario 2: capacity NewYork    = 30 pallets
              capacity NewOrleans = 9 pallets
              compound capacity NewYork+NewOrleans = 12 pallets
```

In both scenarios we assume no capacity constraint from warehouse Baltimore. With the given freight and conversion costs the three path ways would cost:

```
Winery ==> Baltimore  ==> Walmart  = 125+4+75 = 205,-  ==> No.3
       ==> NewOrleans ==> Walmart  = 145+2+41 = 188,-  ==> No.1 cheapest
       ==> NewYork    ==> Wallmart = 125+4+60 = 189,-  ==> No.2
```

Scenario 1

For scenario 1 we would costwise expect that the maximum quantity would go through NewOrleans. Due to the capacity limitation of 9 pallets the remaining quantity would flow along the second best alternative NewYork with 11 pallets. This fits with the result below.

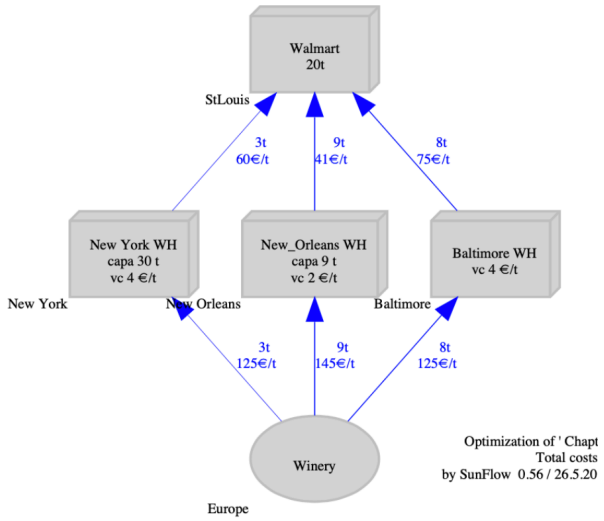


Scenario 2

The only difference between both scenarios is the introduction of a compound capacity which says that even if both or one of them has a higher single capacity the total capacity of both is limited to 12 pallets. Since the cost assumptions didn't change we would expect the following flows:

- cheapest path through NewOrleans: 9 pallets max. capacity
- runner-up is NewYork: NewYork got a single capacity of 30 but a joint one of 12 pallets of which 9 are already eaten-up from NewOrleans, the maximum flow is therefore limited to 3 pallets
- the remaining quantity of 8 pallets must flow through the most expensive hub Baltimore.

This is exacly what SunFlow figured out in the chart below.



Chapter 6

Watch the Algorithm

SunFlow's algorithm building blocks are

- a graph compiler,
- a graph visualizer using modules from graphviz and
- an optimizer using scipy, numpy, pandas and pprint modules.

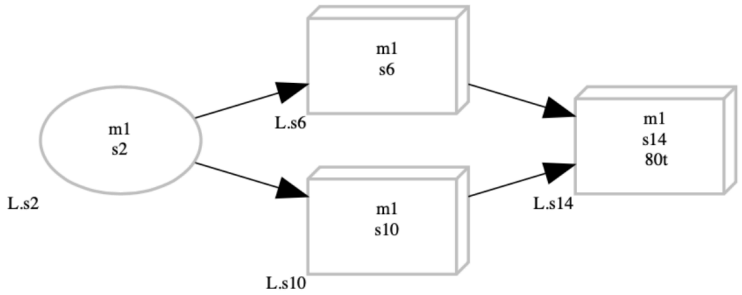
Backstage

To see the logic behind the algorithm you may use the display() method of class Net. By default its parameter is set to zero, but if you use instead a value of 1 or 2 you can get insights how the algorithm works.

```
globals().clear(); import rwlib;
from sunflow import SupplyNet


net = SupplyNet('Supply Net Backstage').display(1).title(False)

supplier    = net.source()
new_york    = net.distribution(supplier)
new_orleans = net.distribution(supplier)
customer    = net.delivery(new_york, new_orleans) .demand(80)
net.compile()
net.showGraph().view()
print()
```

```
List all Nodes
  endNodes      [s258]
  intermediates [s250, s254]
  terminals     [s246]
  allNodes      [s246, s250, s254, s258]

Flow equations of intermediate nodes
  m(s246,s250) - m(s250,s258)  = 0
  m(s246,s254) - m(s254,s258)  = 0

Flow equations of end nodes
  m(s250,s258) + m(s254,s258) = 80 : Demand(s258)

Capacities of intermediate nodes

Capacities of end nodes

Capacities of Terminals
```

This output isn't really neat. The names inside the boxes are SunFlow's internal node and vertex names which are not really readable.

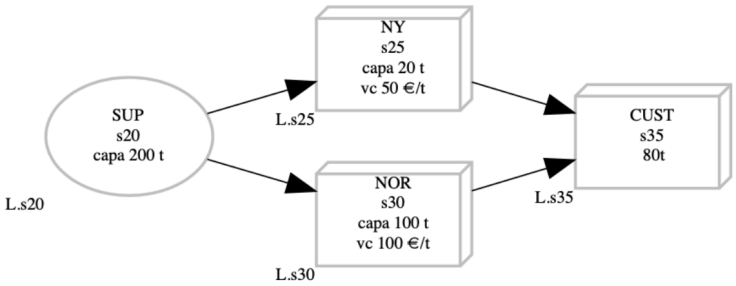


Compiler

To increase the readability of a graph you can assign each net node with the n() method an individual textual identification. Here we've added as well capacity constraints and cost figures - the service charges of the distribution sites - to see the compiler working. By the way, end nodes doesn't usually have capacities, but they could.

```
net = SupplyNet('Supply Net Compiler').display(1).title(False)

supplier    = net.source()                       .n('SUP')  .capacity(200)
new_york    = net.distribution(supplier)         .n('NY')   .capacity(20)  .varcost(50)
new_orleans = net.distribution(supplier)         .n('NOR')  .capacity(100) .varcost(100)
customer    = net.delivery(new_york, new_orleans).n('CUST') .demand(80)
net.compile()
net.showGraph().view()
```

```
List all Nodes
  endNodes      [CUST]
  intermediates [NY, NOR]
  terminals     [SUP]
  allNodes      [SUP, NY, NOR, CUST]

Flow equations of intermediate nodes
  m(SUP,NY) - m(NY,CUST)  = 0
  m(SUP,NOR) - m(NOR,CUST)  = 0

Flow equations of end nodes
  m(NY,CUST) + m(NOR,CUST) = 80 : Demand(CUST)

Capacities of intermediate nodes
  m(NY,CUST) <= 20 : capacity(NY)
  m(NOR,CUST) <= 100 : capacity(NOR)

Capacities of end nodes

Capacities of Terminals
  m(SUP,NY) +  m(SUP,NOR)   <= 200 : capacity(SUP)
```



The Math behind

The mathematical structure can be illustrated by using display(2) instead of display(1). With debug(1) in addition you will see as well SunFlow's variables and their connections between each other.

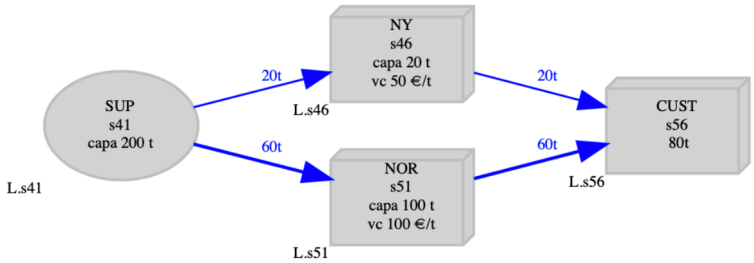
```
net = SupplyNet('Supply Net Math').display(2).title(False).debug(1)

supplier    = net.source()                      .n('SUP')  .capacity(200)
new_york    = net.distribution(supplier)        .n('NY')   .capacity(20)    .varcost(50)
new_orleans = net.distribution(supplier)        .n('NOR')  .capacity(100)   .varcost(100)
customer    = net.delivery(new_york, new_orleans).n('CUST') .demand(80)
net.compile()
```

```
List all Nodes
  endNodes      [CUST]
  intermediates [NY, NOR]
  terminals     [SUP]
  allNodes      [SUP, NY, NOR, CUST]

Flow equations of intermediate nodes
  m(SUP,NY) - m(NY,CUST)  = 0
  m(SUP,NOR) - m(NOR,CUST)  = 0

Flow equations of end nodes
  m(NY,CUST) + m(NOR,CUST) = 80 : Demand(CUST)

Capacities of intermediate nodes
  m(NY,CUST) <= 20 : capacity(NY)
  m(NOR,CUST) <= 100 : capacity(NOR)

Capacities of end nodes

Capacities of Terminals
  m(SUP,NY) +  m(SUP,NOR)   <= 200 : capacity(SUP)

NodeLinks
  variable [  0] = (s285,s290)  |  s285 => s290   L.s285 -> L.s290
  variable [  1] = (s285,s295)  |  s285 => s295   L.s285 -> L.s295
  variable [  2] = (s290,s300)  |  s290 => s300   L.s290 -> L.s300
  variable [  3] = (s295,s300)  |  s295 => s300   L.s295 -> L.s300

Optimization Parameters:
Flow Constraints =  3 equations with 4 variables
[[ 1.  0. -1.  0.]
 [ 0.  1.  0. -1.]
 [ 0.  0.  1.  1.]]
Flow Balance = [     0.0    0.0   80.0 ]

Upper limit constraints =  3 equations for 4 variables
[[0. 0. 1. 0.]
 [0. 0. 0. 1.]
 [1. 1. 0. 0.]]
Upper limits = [    20.0  100.0  200.0 ]

Total number of constraints = 6 for 4 variables ==> 24 cells to optimize

Total cost = [     0.0    0.0   50.0  100.0 ]
   freight = [     0.0    0.0    0.0    0.0 ]
   var     = [     0.0    0.0   50.0  100.0 ]
   fix     = [     0.0    0.0    0.0    0.0 ]
   capa    = [   200.0  200.0   20.0  100.0 ]
```

Optimizing

Once the net is compiled, you execute the optimizer by net.execute() which returns an instance of class Optimize which is used for graphics and provides the results of the optimization.

```
opt = net.execute()
opt.showGraph(flowOnly=False, orient='LR').view()
```



Results

There are three ways to get the result of an optimization .

As a Summary

You use the methods

- successful(): True if optimizer could find a solution
- iteration(): number of iterations required to find solution
- cost(): minimum network cost under the given constraints

```
print(f"Optimiziation was{'' if opt.successful() else ' not'} successful,")
print(f"  used {opt.iterations()} iterations")
print(f"  and minimum supply costs are{opt.cost():6.0f} currency units !")
```

```
Optimiziation was successful,
  used 4 iterations
  and minimum supply costs are  7000 currency units !
```

Details by Nodes & Links as a Pandas DataFrame

To get the details you use method frame() which delivers a pandas dataframe object on return. This object provides the information for each internal variable, i.e. by the number of links between the nodes of your model.

```
df = opt.frame()
print('Result of Optimization:\n'); print(df)
```

```
Result of Optimization:

     Decscription    m(0)    m(1)          m(2)          m(3)
0       Title from     SUP     SUP            NY           NOR
1               to      NY     NOR          CUST          CUST
2        Node from    s285    s285          s290          s295
3               to    s290    s295          s300          s300
4        Supplier    s285    s285          s290          s295
5   Location from  L.s285  L.s285        L.s290        L.s295
6               to  L.s290  L.s295        L.s300        L.s300
7        Activity  Buying  Buying  Distribution  Distribution
8        Material    m284    m284          m284          m284
9    Quantity [t]      20      60            20            60
10    Freight [€]       0       0             0             0
11          [€/t]       0       0             0             0
12    Varcost [€]       0       0          1000          6000
13          [€/t]       0       0            50           100
14    Fixcost [€]       0       0             0             0
15      Total [€]       0       0          1000          6000
16          [€/t]       0       0            50           100
17   Capacity [t]     200     200            20           100
18     Demand [t]      -1      -1            80            80
```

Details by Nodes & Links in Excel

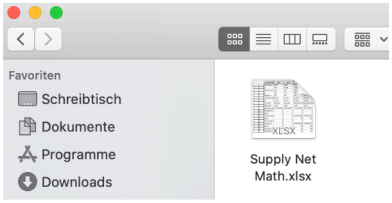
For your convenience the method save() stores the pandas dataframe inside the default data directory or at a place of your choice like

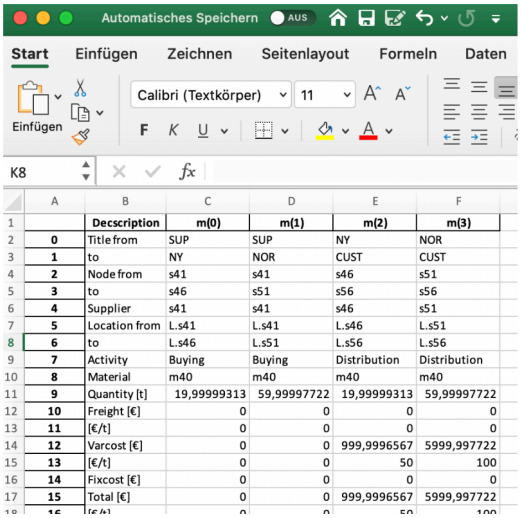
```
opt.save( filename='my optimization result', path='/users/xxx/data/')
```

Here we use the default setting, i.e. no parameters.

```
opt.save()
```

A file named "Supply Net Math.xlsx" - the name of the model - was created and shown below.





	A	B	C	D	E	F
1		Decscription	m(0)	m(1)	m(2)	m(3)
2	0	Title from	SUP	SUP	NY	NOR
3	1	to	NY	NOR	CUST	CUST
4	2	Node from	s41	s41	s46	s51
5	3	to	s46	s51	s56	s56
6	4	Supplier	s41	s41	s46	s51
7	5	Location from	L.s41	L.s41	L.s46	L.s51
8	6	to	L.s46	L.s51	L.s56	L.s56
9	7	Activity	Buying	Buying	Distribution	Distribution
10	8	Material	m40	m40	m40	m40
11	9	Quantity [t]	19,99999313	59,99997722	19,99999313	59,99997722
12	10	Freight [€]	0	0	0	0
13	11	[€/t]	0	0	0	0
14	12	Varcost [€]	0	0	999,9996567	5999,997722
15	13	[€/t]	0	0	50	100
16	14	Fixcost [€]	0	0	0	0
17	15	Total [€]	0	0	999,9996567	5999,997722

Chapter 7

Chapter 7: Loading Freight Rates

If your model contains lots of locations and has transfer betweeen these location, it could become time-consuming to create and maintain all data aligned with them. The following sections descibe the 'manual' (variant 1) and the 'semi-automatic' way (variant 2).

Variant 1: create and register Location and Freight Rates

In variant 1 (refering to the examples Ditribution Network) four locations and the freight between them.

```
globals().clear(); import aitlib; import importlib,sunflow; importlib.reload(sunflow)
from sunflow import SupplyNet,Logistics,Customer,Location,Freight,Producer,SunFlow
#
# init
#
sun = SunFlow()
#
# create locations
#
Europe      = Location('Europe')         # the Winery
New_York    = Location('New York')       # warehouse
New_Orleans = Location('New Orleans')    # warehouse
StLouis     = Location('StLouis')        # customer
#
# register freight rates
#
Freight(Europe,       New_York,      125 )
Freight(Europe,       New_Orleans,   145 )
Freight(New_York,     StLouis,        60 )
Freight(New_Orleans,  StLouis,        41 )
#
# display all creates entities
#
sun.show()
print()
```

```
SunFlow
Entities
  [  0] Location    Europe
  [  1] Location    New York
  [  2] Location    New Orleans
  [  3] Location    StLouis
  [  4] Freight     Europe => New York
  [  5] Freight     Europe => New Orleans
  [  6] Freight     New York => StLouis
  [  7] Freight     New Orleans => StLouis
```

The retrieved locations below can now used further in our model.

All participants like suppliers, warehouses/distribution hubs and customers are defined below.

Variant 2: create and register Location and Freight rates by loading from File

Variant 2 does a different approach. Instead of creating all registrations manual, it uses an Excel file with the locations and their transfer rates. For this example we use the file WineryFreights.xlsx which is shown below.

	A	B	C	D	E
1	from	Europe	NewYork	NewOrleans	StLouis
2	Europe		125	145	
3	NewYork				60
4	NewOrleans				41
5	StLouis				

The table above contains N+1 columns, where N is the number of locations, which has to be in the first line and the first column. And,

- it must (!!!) be symmetric, i.e. the locations in the first column must be the same as in the first line and
- the name of the first column (cell A1) have to be 'from'.

Cells containing a freight rate will be used to register its freight as you can see below. If no freight given or freight rate = 0, than SunFlow uses a freight=0 as rate.

```
globals().clear(); import aitlib; import importlib,sunflow; importlib.reload(sunflow)
from sunflow import SupplyNet,Logistics,Customer,Location,Freight,Producer, SunFlow
#
# init
#
sun = SunFlow()
#
# load freight from file WineryFreights.xlsx (see above)
#
sun.loadFreights('WineryFreights.xlsx')
#
# reference by loadFreights() created locations
#
Europe      = sun.location('Europe')         # the Winery
New_York    = sun.location('New York')       # warehouse
New_Orleans = sun.location('New Orleans')    # warehouse
StLouis     = sun.location('StLouis')        # customer
#
# display all creates entities
#
sun.show()
print()
```

```
SunFlow
Entities
  [  0] Location    Europe
  [  1] Location    NewYork
  [  2] Location    NewOrleans
  [  3] Location    StLouis
  [  4] Freight     Europe => NewYork
  [  5] Freight     Europe => NewOrleans
  [  6] Freight     NewYork => StLouis
  [  7] Freight     NewOrleans => StLouis
```

Instead of creating the locations by

```
Europe = Location( 'Europe' )
```

your reference the already created locations by

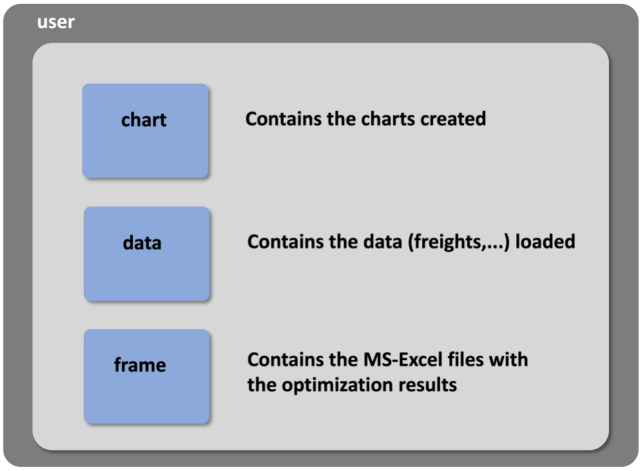
```
Europe = sun.location('Europe')
```

if you want to use them within your model.

Chapter 8

Setting the Directory Paths

By default SunFlow expects the following directories:



By default all paths point to the same relative directory path

```
'/users/USER_NAME/py/pydata/tempdata/sunflowtemp/'
```

These settings can be changed either fix, within the library file sunflow.py at the very beginning by setting the variables below to your respective path,

```
sunFlowGraphicsDir      = '/users/USER_NAME/py/pydata/tempdata/sunflowtemp/'    # chart directory
sunFlowDefaultSheetPath = '/users/USER_NAME/py/pydata/tempdata/sunflowtemp/'    # frame directory
sunFlowDefaultDataPath  = '/users/USER_NAME/py/pydata/datasets/'                # data directory
```

or dynamically within your program as show below.

Get a Path

The current definitions can be retrieved by the methods userDir(), chartDir(), frameDir() and dataDir():

- userDir(): cannot be changed and is read-only
- chartDir(): is the directory where the charts will be saved.
- frameDir(): is the directory where the excel file with the optimization result is saved.
- dataDir(): is the directory where the freight data (Excel file) are saved and loaded.

Keep in mind that these methods return the relative path to the user directory as you can see below.

```
globals().clear(); import aitlib; import importlib,sunflow; importlib.reload(sunflow)
from sunflow import SunFlow

sun = SunFlow()

print('Pre-defined default paths are:\n')
print('  userDir  = ',sun.userDir())
print('  chartDir = ',sun.chartDir())
print('  frameDir = ',sun.frameDir())
print('  dataDir  = ',sun.dataDir())
```

```
Pre-defined default paths are:

  userDir  =  /users/rainer/
  chartDir =  py/pydata/tempdata/sunflowtemp/
  frameDir =  py/pydata/tempdata/sunflowtemp/
  dataDir  =  py/pydata/datasets/
```

Building a Path

The directory paths above can be customized to your requirements by using the method buildPath(). In contrast to the methods chartDir(), frameDir() and dataDir() buildPath() creates an absolute path including the user directory shown in the example below.

If the parameter path is not given or equal to '' (default),then the current settings of the relative paths are used to create an absolute path.

```
fpath = sun.buildPath();                                    print('  buildPath(): default path      =',fpath)
fpath = sun.buildPath('peter.xlsx',   path='', kind='chart'); print('  buildPath(chart): file path    =',fpath)
fpath = sun.buildPath('mary.xlsx',    path='', kind='frame'); print('  buildPath(frame): file path    =',fpath)
fpath = sun.buildPath('freight.xlsx', path='', kind='data');  print('  buildPath(data): file path     =',fpath)
```

```
  buildPath(): default path      = /users/rainer/py/pydata/tempdata/sunflowtemp/
  buildPath(chart): file path    = /users/rainer/py/pydata/tempdata/sunflowtemp/peter.xlsx
  buildPath(frame): file path    = /users/rainer/py/pydata/tempdata/sunflowtemp/mary.xlsx
  buildPath(data): file path     = /users/rainer/py/pydata/datasets/freight.xlsx
```

If path is different from '' then it is added to the user path before the filename is appended.

```
fpath = sun.buildPath('freight.xlsx',                # file without path information
                      path='other_data/freights/',   # the path rel. to user dir
                      kind='data');
print('  buildPath(data): file path     =',fpath)
```

```
  buildPath(data): file path     = /users/rainer/other_data/freights/freight.xlsx
```

If path is left at '' (default), then the path is added after the current data directory path.

```
fpath = sun.buildPath('freight_data/freight.xlsx',   # file with path information
                      path='',                       # default or no path given
                      kind='data');
print('  buildPath(data): file path     =',fpath)
```

```
  buildPath(data): file path     = /users/rainer/py/pydata/datasets/freight_data/freight.xlsx
```

Set a Path

The directory paths are set by the same methods they are retrieved. In the example below they are set all to the directory 'py/pydata/tempdata/mytemp/'.

```
print('\n\nUser-defined default paths:\n')
sun.chartDir('py/pydata/tempdata/mytemp/')
sun.frameDir('py/pydata/tempdata/mytemp/')
sun.dataDir('py/pydata/tempdata/mytemp/')

print('  userDir  = ',sun.userDir())
print('  chartDir = ',sun.chartDir())
print('  frameDir = ',sun.frameDir())
print('  dataDir  = ',sun.dataDir())
```

```
User-defined default paths:

  userDir  =  /users/rainer/
  chartDir =  py/pydata/tempdata/mytemp/
  frameDir =  py/pydata/tempdata/mytemp/
  dataDir  =  py/pydata/tempdata/mytemp/
```

If now the same absolute paths build, you see that the directory names have changed and pointing to a subdirectory temp.

```
fpath = sun.buildPath();                                    print('  buildPath(): default path      =',fpath)
fpath = sun.buildPath('peter.xlsx',   path='', kind='chart'); print('  buildPath(chart): file path    =',fpath)
fpath = sun.buildPath('mary.xlsx',    path='', kind='frame'); print('  buildPath(frame): file path    =',fpath)
fpath = sun.buildPath('freight.xlsx', path='', kind='data');  print('  buildPath(data): file path     =',fpath)
```

```
  buildPath(): default path      = /users/rainer/py/pydata/tempdata/mytemp/
  buildPath(chart): file path    = /users/rainer/py/pydata/tempdata/mytemp/peter.xlsx
  buildPath(frame): file path    = /users/rainer/py/pydata/tempdata/mytemp/mary.xlsx
  buildPath(data): file path     = /users/rainer/py/pydata/tempdata/mytemp/freight.xlsx
```