FlowCAr: Flow Network Construction and Assessment in R:

An R Package for Standardised Construction and Evaluation of Ecological Networks

Christopher Waspe, Ruchit Mahabir and Ursula Scharler

University of KwaZulu-Natal

Westville

Durban

Abstract

Network modelling requires a cogent, standardised methodology for the vital construction phase to facilitate appropriate interpretation of outcomes from the final analysis. Different approaches to network construction or single solutions have been commonplace in network modelling over the past decades and could be improved with multi-solution analysis of a single system to account for the inherent variability of input data. Linear inverse modelling (LIM) can be used to quantify complex flow networks by calculating network link values from under sampled data, typical for ecological networks. The resultant networks can be analysed using Ecological Network Analysis (ENA), a branch of network ecology used to holistically analyse the structure and dynamics of interactions in networks. The **FlowCAr** package provides a standardised methodology for the construction phase of network modelling, producing a range of mathematically and ecologically sound flow networks from given input data, solved using LIM. These networks can be visually assessed, to ensure the LIM solution space has been adequately sampled. **FlowCAr** restructures and packs the list of LIM solved networks, enabling ENA to be performed on the flow networks in R package enaR.

We provide a simple 4 node flow network to illustrate the use of the package.

If you use the **FlowCAr** package, please cite as: (Waspe, Mahabir, and Scharler 2018).

Keywords: Network construction, Linear inverse modelling, flow networks, food web, Ecological Network Analysis (ENA)

Table of Contents

Table of Figures

1.) Introduction

Quantification of flow networks such as food webs are often problematic due to data deficiency as direct flow measurement is particularly difficult, even in comparatively well sampled systems (Woodward *et al.*, 2005; Fath *et al.*, 2007; Soetaert and van Oevelen, 2009; van Oevelen *et al.*, 2010; Lau *et al.,* 2017). The difficulty can arise from the high cost and high labour required to measure all aspects of an ecological system. Most ecological data sets only consist of biomass estimates of aggregated functional groups and an occasional rate measurement (Woodward *et al.*, 2005; Soetaert and van Oevelen, 2009; Soetaert and van Oevelen, 2010). This *in situ* measured data is often referred to as hard data (Soetaert and van Oevelen, 2009; van Oevelen *et al.,* 2009). Due to food web complexity, system information based only on the limited field measurements is inadequate to derive a clear and accurate representation of the flows in these systems (Soetaert *et al.,* 2009; Soetaert and van Oevelen, 2009). Data from other, similar systems and literature is used to fill in some of these gaps. Such secondary data sources are referred to as soft data (Woodward *et al.*, 2005; Meersche *et al.*, 2009; Soetaert and van Oevelen, 2009; van Oevelen *et al.*, 2010; Soetaert and van Oevelen, 2010). Secondary data sources can be very helpful as it allows constraints to be applied to the flows. However, there may be consequences to using this non-system specific data. The manner in which system components react and interact may change between systems and over time e.g. there may be a species which occurs in two similar systems but may have different production rates due to an influence from a specific factor within the systems (Woodward *et al.*, 2005; Meersche *et al.*, 2009; Soetaert and van Oevelen, 2009; van Oevelen *et al.*, 2010; Soetaert and van Oevelen, 2010). Therefore, soft data are to be used sparingly and only where necessary. To overcome these data limitations, linear inverse modelling (LIM) was developed and has been expanded upon (Klepper and Van de Kamer, 1987; Vézina and Platt, 1988; Meersche *et al.*, 2009; Soetaert and van Oevelen, 2009; van Oevelen *et al.*, 2010; Soetaert and van Oevelen, 2010).

The LIM methodology allows for the quantification of interactions in a complex flow network, from incomplete and uncertain data (Klepper and Van de Kamer, 1987; Vézina and Platt, 1988; Soetaert and van Oevelen, 2009; van Oevelen *et al.*, 2010; Soetaert and van Oevelen, 2010). The flow network in a LIM is a linear function of its flows, which are estimated from empirical data. An infinite number of solutions exist to quantify flows consistent with the (sparse) input data when the system is under-sampled or underdetermined (Meersche *et al.*, 2009; Soetaert and van Oevelen, 2009; Soetaert and van Oevelen, 2010; van Oevelen *et al.*, 2010). Thus, LIM uses a set of equality and inequality equations from the input ecological data set (empirically measured and from similar systems and literature) and several constraints to set up a solution space in which an infinite number of flow solutions exist, within the parameters provided by this ecological data set. A ‘best’ or

parsimonious solution is selected from this infinite set using optimisation criteria (Vézina and Platt, 1988; Kones *et al.*, 2006). However, there is no theoretical or empirical evidence as to whether this criterion selects the 'best' solution (Vézina *et al.,* 2004; Kones *et al.,* 2006; Soetaert and van Oevelen, 2009; van Oevelen *et al.*, 2010). The selected solution results in some flows set to zero and others very close to their minimum or maximum values, which should be considered extreme values rather than likely ones (Vézina *et al.,* 2004; Kones *et al.,* 2006; Soetaert and van Oevelen, 2009; van Oevelen *et al.,* 2010). Therefore, a Likelihood Approach (LA-LIM) was suggested (van Oevelen *et al.*, 2010). The LA generates a marginal probability density function (mPDF) for each flow from which a distribution of flow values can be derived (van Oevelen *et al.*, 2010). The single 'best' solution is selected from the ensemble by taking the mean value for each flow (van Oevelen *et al.*, 2010). This method still results in just a single network out of an infinite number of possible solutions. Networks however, should rather represent the ubiquitous variability of the ecological input data, and therefore move away from single solutions to multiple network analysis to better represent real systems.

The LIM can be solved a selected number of times during a specified number of iterations (e.g. 10 000), providing a set of possible ecological networks which are within the specified parameters of the ecological input data (Soetaert and van Oevelen, 2009; Meersche *et al.*, 2009). Once these networks have been constructed using LIM many 'replicates' of networks can be analysed further. Each network represents one of the possible solved LIM networks.

Ecological Network Analysis (ENA) is an analytical tool used to understand various system components and attributes (Ulanowicz, 1986; Fath and Patten, 1999; Borrett and Lau, 2014). Due to the methods used to construct networks and the solution procedures for the unknowns, the outcome was only single networks such as the parsimonious or mean networks (Kones *et al.*, 2006; van Oevelen *et al.*, 2010). Thus, ENA was performed on single networks, representing only one state of the ecosystem (Christian *et al.,* 2005; Goerner *et al.,* 2009; Scharler, 2012; Borrett and Lau, 2014; Chrystal and Scharler, 2014; Ortega *et al.*, 2016). The analysis of the networks is greatly dependent on the quality of the information and parameters used to build the networks. However, methods to address the uncertainty in model parameters are underdeveloped (Ulanowicz, 2004; Dame and Christian, 2006; Fath *et al.,* 2007; Hines *et al.,* 2018).

Uncertainty analyses show how the variability which is inherent model parameters affects the outputs of these models (Crosetto and Tarantola, 2001; Hines *et al.,* 2018). An uncertainty methodology was proposed to alter flow values by a chosen percent on existing networks (Hines *et al.,* 2018). The flows may be individually altered (if uncertainty data is available) or all flows can be uniformly altered (Hines *et al.,* 2018). This tool allows researchers to assess the degree of reliability

in the network results depending on the amount of flow alteration (Hines *et al.,* 2018). As this technique is performed after a single network has been constructed, it does not allow for the representation of structural changes in the initial network model construction phase, even though the structure of models affects the model results (Allesina *et al.,* 2005; Dame and Christian, 2005; Patonai and Jordán, 2017; Hines *et al.,* 2018).

The single solution approach is a widely recognised issue in network ecology as it ignores the variability in flows and stocks values and the infinite number of solutions for an underdetermined flow network problem (Vézina *et al.,* 2004; Kones *et al.,* 2006; van Oevelen *et al.,* 2010). To provide standardisation in the vital construction phase of flow network modelling and move towards multiple network analysis we introduce the package **FlowCAr.**

The **FlowCAr** package provides the user with a novel approach to create a range of possible solved LIM food web networks and analyse this range using the ENA approach. This package also provides a link from the construction (and solving) of the networks using LIM (van Oevelen *et al.,* 2010), to the analysis using **enaR** (Borrett and Lau, 2014). With this package the user can ensure the generated networks have adequately sampled the LIM solution space and that these networks accurately represent the input ecological constraints. The **FlowCAr** package introduces several functions which evaluates its performance. Figure 1 indicates an example workflow of **FlowCAr**, from the input file, the functions used and the outputs of those functions, which are explained in Section 4 (Formatting of the Input File for the FlowCAr Package) of this document.

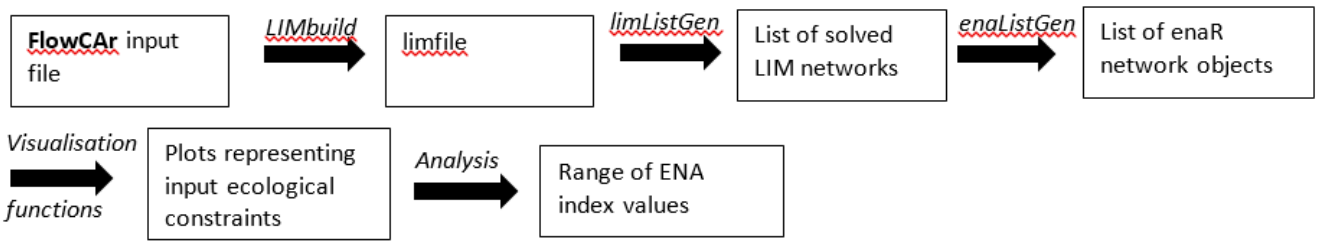

Figure 1: Simplified workflow of the FlowCAr package.

2.) Linear Inverse Modelling

Following is a brief description of the methods for Linear Inverse Modelling (van Oevelen *et al.*, 2009)**.**

Linear inverse problems can be expressed in matrix notation as:

$$\mathbf{A} \cdot \mathbf{x} \approx \mathbf{b} \qquad (1)$$

$$\mathbf{E} \cdot \mathbf{x} = \mathbf{f} \qquad (2)$$

$$\mathbf{G \cdot x \geq h} \qquad (3)$$

There are three sets of linear equations: equalities that must be met as closely as possible (1), equalities that must be met exactly (2) and inequalities (3).

Linear programming methods are used to solve for **x** in the above equations. In R, these techniques are available through the packages **LIM** (van Oevelen *et al.,* 2009) and **limSolve** (Soetaert *et al.,* 2009). These two packages are utilised by the package **FlowCAr**.

Depending on the set of equalities (2) and constraints (3), the system may either be underdetermined, even determined or overdetermined. Complex flow network problems are almost always underdetermined due to under sampling of the system. Solving the equalities and inequalities requires the following different mathematical approaches:

- If the model is even or over determined, there is only one solution that satisfies the equations exactly. This solution can be singled out by using the least squares method *lsei* from the package **limSolve**.
- If the model is underdetermined, such as with ecological flow network problems, there exists an infinite number of solutions. To solve these, there are several options (in the **LIM** package):
  - *ldei* – finds the "least distance" solution
  - *lsei* – minimises some other set of functions (equation 1) in a least squares sense
  - *linp* – finds the solution where one linear function (sum of the unknowns) is either minimised or maximised
  - *xranges* – finds the possible ranges (min, max) for each unknown
  - *xsample* – randomly samples the solution space using a Monte Carlo Markov Chain (MCMC). This method returns the marginal probability density function for each unknown (Van den Meersche, Soetaert and Van Oevelen, 2009)
- Our package (**FlowCAr**) uses the *Xsample* technique to create a list of possible solutions of each problem. The *Xsample* technique uses a sampling method based on the MCMC algorithm to retrieve an array of flow values from within the solution set, which is within the specified constraints of the ecological input data. Each set of flow values representing an individual network provide the flow matrices for the LIM networks, which can then be converted to **enaR** network objects for analyses.

3.) Flow networks

Flow networks are represented as a set of nodes (compartments), which are connected by flows (arrows). The flows have direction:

$$A \rightarrow B$$

Indicates a flow from *A* to *B*, while:

$$A \rightleftarrows B$$

Indicates two flows in both directions.

There can only be one flow (with direction) connecting two nodes directly. Solving the LIM problem consists of finding values for all specified flows.

Flow networks can describe a wide range of systems and interactions including food webs, biogeochemical cycles and systems containing mostly non-living components such as urban metabolism networks (Niquil *et al.,* 1999; Borrett *et al.*, 2016; Zhang *et al.,* 2016). A food web is a representation of the energy flow between biotic and abiotic components of an ecosystem and any energy exchange across the system boundary (Niquil *et al.,* 1999; Fath *et al.*, 2007; van Oevelen *et al.,* 2010). Organisms both consume and are consumed by other organisms. Part of consumption is used for biomass production and reproduction and part is expelled as faeces or respired. Autotrophic organisms produce biomass from light energy and nutrients, whilst non-living matter (detritus) may be consumed by animals and bacteria and be added to from dead organisms and faeces (Fath *et al.*, 2007; van Oevelen *et al.,* 2010).

LIM can be used to solve the missing or unknown information in flow networks (Soetaert and van Oevelen, 2009; van Oevelen *et al.,* 2010). The unknown information of the system is often the weights of the ecological network flows, connecting the biotic and abiotic components, and the external flows crossing the system boundary such as inputs, exports and respirations (van Oevelen *et al.,* 2010). LIM solves for the unknowns which are themselves linear combinations of other unknowns. This can be explained as such: the flow from a living node to a detrital node (faecal production and death) is specified as a fraction of the amount of food ingested. Ingested food (consumption) is the sum of all flows to the consumer. The consumption, defaecation and respiration flows are therefore not independent of one another (Klepper and Van de Kamer, 1987; Vézina and Platt, 1988; Soetaert and van Oevelen, 2010; van Oevelen *et al.*, 2010). For instance, organisms cannot produce more faeces than the amount of food they consume. Organisms also respire and require energy for growth, the food that is assimilated (i.e. not defecated) is used to

produce new biomass (Klepper and Van de Kamer, 1987; Vézina and Platt, 1988; Soetaert and van Oevelen, 2010; van Oevelen *et al.*, 2010).

In some cases, direct flow measurements exist and are used to set up equality equations. Other empirically measured system attributes such as node production and respiration rates from literature or similar systems are used as parameters to set up inequality equations. Even if these parts of the system are well sampled, the empirically measured data still represents ecological variability. These parameters will therefore be used to constrain the flows within a range, accounting for the inherent variability within a system (Klepper and Van de Kamer, 1987; Vézina and Platt, 1988; Soetaert and van Oevelen, 2010; van Oevelen *et al.*, 2010).

A simple ecological network (Figure 2) is introduced here to explain the functioning of the **FlowCAr** package. The ecological network has four nodes: detritus, a plant (autotroph) and two animals (heterotrophs): an invertebrate which feeds on detritus and the plant, and a vertebrate which feeds on the invertebrate. The ecological network is assumed to be mass balanced and in a steady state i.e. masses (flow weights and node masses) remain unchanged in time. There are ten flows (Table 1) which connect the food web components with each other and across the system boundary.

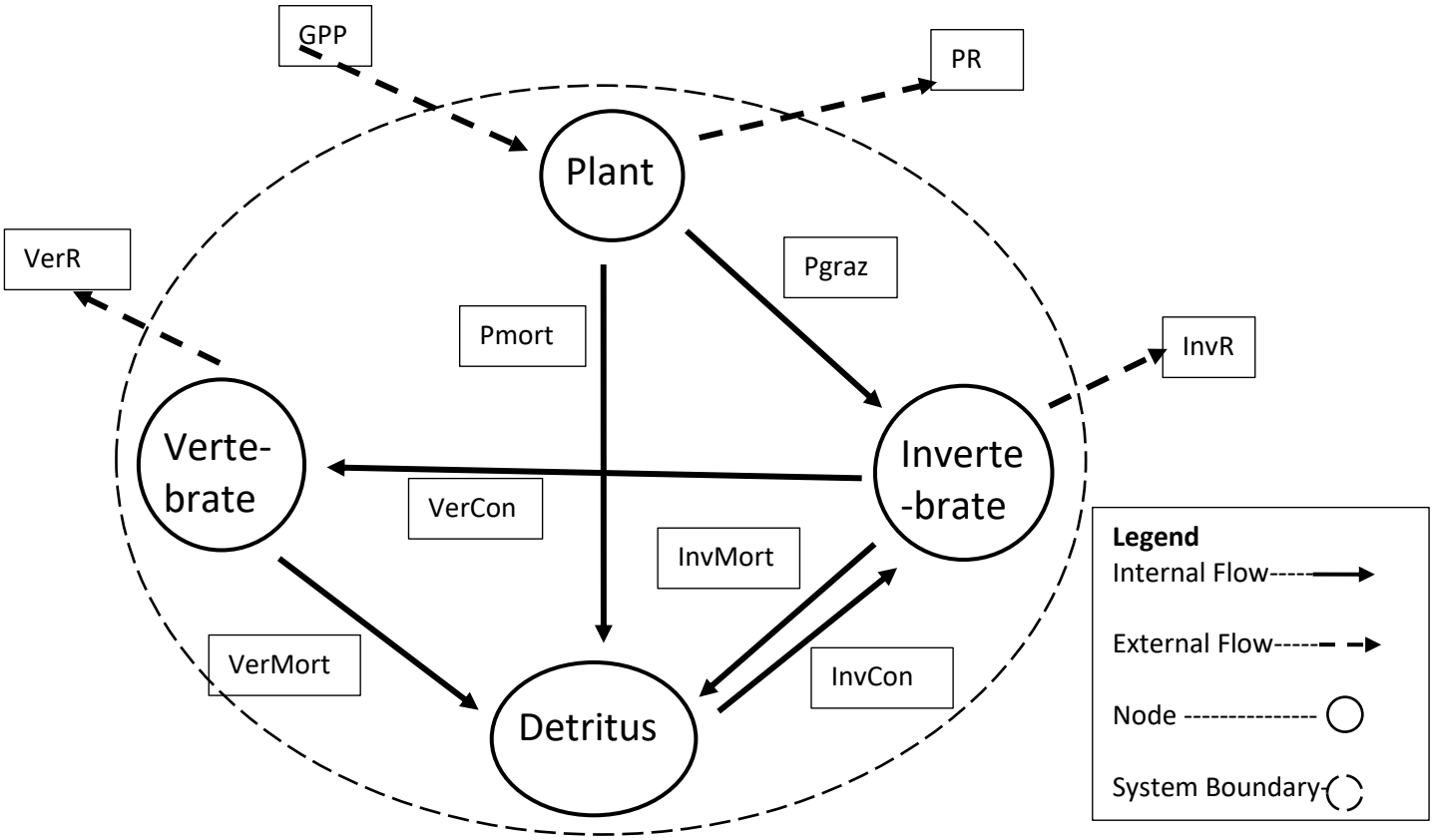

Figure 2: Schematic of a hypothetical four node food web. Flow name abbreviations are explained in Table 1.

Table 1: Flow names and abbreviations for the 4-node network.

Flow Name	**Abbrevaition**
Gross Primary Production	GPP
Grazing on Plant by Invertebrate	Pgraz
Plant Respiration	PR
Plant Mortality	Pmort
Feeding on detritus (Invertebrate Consumption)	InvCon
Invertebrate Respiration	InvR
Invertebrate Mortality	InvMort
Vertebrate Consumption	VerCon
Vertebrate Respiration	VerR
Vertebrate Mortality	VerMort

The mass balance equations for the internal flows between the four components, with the rate of change = 0, is given by:

$$\frac{dPLANT}{dt} = 0 = Gross\ primary\ production - grazing\ on\ plant - plant\ respiration - plant\ mortality$$

$$\frac{dINVERTEBRATE}{dt} = 0 = grazing\ on\ plant + feeding\ on\ detritus - invertebrate\ respiration - invertebrate\ mortality$$

$$\frac{dVERTEBRATE}{dt} = 0 = vertebrate\ consumption - vertebrate\ respiration - vertebrate\ mortality$$

$$\frac{dDETRITUS}{dt} = 0 = plant\ mortality + invertebrate\ mortality + vertebrate\ mortality - feeding\ on\ detritus$$

Mass balances for each node can be expressed as:

0 = 1·*GPP* - 1·*Pgraz* - 1·*Pmort* - 1·*PR* + 0·*InvCon* + 0·*InvR* + 0·*InvMort* + 0·*VerCon* + 0·*VerR* + 0·*VerMort*

0 = 0·*GPP* + 1·*Pgraz* + 0·*Pmort* + 0·*PR* + 1·*InvCon* - 1·*InvR* - 1·*InvMort* - 1·*VerCon* + 0·*VerR* + 0·*VerMort*

0 = 0·*GPP* + 0·*Pgraz* + 0·*Pmort* + 0·*PR* + 0·*InvCon* + 0·*InvR* + 0·*InvMort* + 1·*VerCon* - 1·*VerR* - 1·*VerMort*

0 = 0·*GPP* + 0·*Pgraz* + 1·*Pmort* + 0·*PR* - 1·*InvCon* + 0·*InvR* + 1·*InvMort* + 0·*VerCon* + 0·*VerR* + 1·*VerMort*

The zero on the left-hand side of the equation relates to a sum of products, where each product is composed of the flows and a coefficient. The coefficient indicates if and by how much these flows contribute to the rate of change.

Certain flows within a system may be directly measured. Assume that the respiration for the vertebrate (VerR) has been directly measured. We can add an extra equation (equality):

$$VerR = 0.75$$

For just the internal flows we obtain the following matrix:

$$\begin{bmatrix} 1 & -1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & -1 & -1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & -1 \\ 0 & 0 & 1 & 0 & -1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} GPP \\ Pgraz \\ Pmort \\ PR \\ InvCon \\ InvR \\ InvMort \\ VerCon \\ VerR \\ VerMort \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ -0.75 \end{bmatrix}$$

We can now illustrate the full system in matrix notation with internal flows and flows between the system and the external environment (boundary flows). Flows across the system boundary are inputs and exports:

$$\begin{bmatrix} 1 & -1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & -1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & -1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & -1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} GPP \\ Pgraz \\ Pmort \\ PR \\ InvCon \\ InvR \\ InvMort \\ VerCon \\ VerR \\ VerMort \\ ppIn \\ detIn \\ ppEx \\ VerEx \\ detEx \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ -0.75 \end{bmatrix}$$

The parameters which are used to set up the inequality equations are as follows:

0.6*Plant biomass < Plant Net Primary Production < 1.1*Plant biomass

0.4*Plant Net Primary Production < Plant Respiration < 0.7*Plant Net Primary Production

0.004*Invertebrate biomass < Invertebrate Production < 0.06*Invertebrate biomass

0.0003* Vertebrate biomass < Vertebrate Production < 0.005*Vertebrate biomass

0.4< (Invertebrate Consumption – Invertebrate Mortality)/Invertebrate Consumption<0.75

1000 < Input to plant < 1600 (Input across the system boundary to the plant may vary between 1000 and 1600 units)

500 < Input to detritus < 1500

1000< Export from plant < 1600

0.21< Export from vertebrate < 0.36

200< Export from detritus < 400

Once the inequality equations are calculated the following matrix can be produced incorporating the internal and boundary flows, equalities (empirically measured flows) and inequalities (constraints on flows):

$$\begin{bmatrix}
-1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 4 & 0 & 0 & 4 & -5 & -4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 0 & 0 & -1 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & -5 & -4 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -1.5 & 2.5 & 1.5 & 0 & 0 & 0 & 0 & 0 \\
0 & 0.6 & 0 & 0 & 0.6 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -0.25 & 0 & 0 & -0.25 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix} \cdot \begin{bmatrix} GPP \\ Pgraz \\ Pmort \\ PR \\ InvCon \\ InvR \\ InvMort \\ VerCon \\ VerR \\ VerMort \\ ppIn \\ detIn \\ ppEx \\ VerEx \\ detEx \end{bmatrix} = \begin{bmatrix} -880 \\ 480 \\ 0 \\ 0 \\ -120 \\ 8 \\ -2.50 \\ 0.15 \\ 0 \\ 0 \\ 0 \\ 0 \\ -1600 \\ 1000 \\ -1500 \\ 500 \\ -1600 \\ 1000 \\ -0.36 \\ 0.21 \\ -400 \\ 200 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

This model has 5 equations and 22 inequalities: there are 10 unknown flows. This problem is solved with **LIM** (using *Xsample*) which is incorporated in the **FlowCAr** package and produces the ensemble of flow values.

4.) Formatting of the Input File for the FlowCAr Package

The required input file for this package is a ".R" text file (referred to as the **FlowCAr** input file from now). The structure of the **FlowCAr** input file follows a similar structure to that of the **LIM** package (van Oevelen *et al.,* 2009) input file, with some modifications.

All variables of the model must be defined, e.g. invertebrate consumption (InvCon). 'Invertebrate consumption (InvCon)' is the sum of the two feeding flows (Pgraz + InvCon). The system parameters are used to set up the inequalities which constrain the variables. Structuring the **FlowCAr** input file in this manner assists with adding or removing elements. **limSolve** generates the mass balances for each node, based on the flows that were defined.

The basic structure of the declaration file must obey the following guidelines:

- All guidelines outlined in "5. Structure of the LIM input file" in **LIM** package (Soetaert and van Oevelen, 2010). Provided in the appendix of this document.
- In the '###Component' section, all non-living (dead) component names (e.g. detritus) are required to have 'NLNode' in the component name i.e. detritus would be detNLNode or detritusNLNode. It is not case sensitive as the **FlowCAr** '*LIMbuild*' function converts all components to uppercase.
- In the '###Externals' section, the respiration element (e.g. CO_2) is required to feature FIRST.
- In the '###Externals' section, all Inputs (e.g. ppInput) require 'Input' at the end of the name, with the 'I' in UPPERCASE, i.e. plant Input would be plantInput. This is case sensitive.
- In the '###Externals' section, all Exports (e.g. ppExport) require 'Export' at the end of the name, with the 'E' in UPPERCASE, i.e. plant export would be plantExport. This is case sensitive.

The input file for the 4-node ecological network example is documented below. The input file includes the names of the components and their steady state biomasses (###Component section) and a section stating the external components (###Externals). The information in the Variable,

Flows, Parameters, Equalities and Inequalities sections are typically obtained from empirically measured data from the system and data from other systems and literature.

```
###Example four node network named “4node.R”

###Component
!Steady state biomasses: mgC/m2
pp = 800 !Plant
inv = 2000 !Invertebrate
ver = 500 !Vertebrate
detNLNode = 10000 !Detritus
###End component

###Externals
CO2
ppInput
ppExport
verExport
detNLNodeInput
detNLNodeExport
###End Externals

###Variables
!Plant
ppGPP = GPP
ppR = PR
ppNPP = ppGPP - ppR
!Organisms
invC = Pgraz + InvCon
invR = InvR
invP = invC - invR - InvMort
verC = VerCon
verR = VerR
verP = verC - verR - VerMort
```

```
###End Variables

###Flows
GPP : CO2 -> pp !Plant energy gain
Pgraz : pp -> inv !Grazing on plant
Pmort : pp -> detNLNode !Plant mortality
PR : pp -> CO2 !Plant respiration
InvCon : detNLNode -> inv !Feeding on detritus
InvR : inv -> CO2 !Invertebrate respiration
InvMort : inv -> detNLNode !Invertebrate mortality
VerCon : inv -> ver !Feeding on invertebrate
VerR : ver -> CO2 !Vertebrate respiration
VerMort : ver -> detNLNode !Vertebrate mortality
ppIn : ppInput -> pp !Plant Input
detIn : detNLNodeInput -> detNLNode !Detritus Input
ppEx : pp -> ppExport !Plant export
VerEx : ver -> verExport !Vertebrate export
detEx : detNLNode -> detNLNodeExport !Detritus export
###End Flows

###Parameters
ppNPPratioBmin = 0.6
ppNPPratioBmax = 1.1
ppRratioNPPmin = 0.4
ppRratioNPPmax = 0.7
invPratioBmin = 0.004
invPratioBmax = 0.06
invRratioPmin = 1
invRratioPmax = 4
verPratioBmin = 0.0003
verPratioBmax = 0.005
###End parameters

###Equalities
```

VerR = 0.75

###End Equalities

###Inequalities
ppNPP = pp * [ppNPPratioBmin, ppNPPratioBmax]
ppR = ppNPP * [ppRratioNPPmin, ppRratioNPPmax]
invP = inv * [invPratioBmin, invPratioBmax]
invR = invP * [invRratioPmin, invRratioPmax]
verP = ver * [verPratioBmin, verPratioBmax]
invC - InvMort > 0.4*invC
invC - InvMort < 0.75*invC
ppIn < 1600
ppIn > 1000
detIn < 1500
detIn > 500
ppEx < 1600
ppEx > 1000
verEx < 0.36
verEx > 0.21
detEx < 400
detEx > 200
###End Inequalities

It is important to note the use of '###SectionName…###End SectionName' to declare items, and that the sections '###Stocks and '###Externals' define the names. A mass balance equation is only generated for stocks of internal nodes but not for the external nodes (e.g. CO_2). A name is declared as "name: ", and exclamation mark (!) demarcates the start of a comment, which is ignored when read.

Although this can be a lengthy process, the problem formulation is elegant, flexible, robust and can easily be adjusted at a later point.

5.) Using the FlowCAr functions: creating a list of LIM solved networks, restructuring into enaR network objects and validating and visualising these networks:

The **FlowCAr** input file is used to create the matrices and vectors that constitute the LIM problem. This is solved using *Xsample* to create a list of multiple possible LIM networks within the constraints of the ecological input data. Thereafter, this LIM list of solved possible networks is converted to a list of **enaR** network objects, ready to be used for network analysis in the R package **enaR** (Borrett and Lau, 2014). There are several network validations and visualisation tools available. Functions of the **FlowCAr** package are summarised in Table 2 and Figure 3.

Table 2: **FlowCAr** *functions.*

Procedure	**FlowCAr Functions**
1) Network Construction 1.1) Solving 1.2) Optional Preparations	1.1.1) LIMbuild
	1.1.2) limListGen
	1.2.1) flowCheck
	1.2.2) flowLimit
2) Restructuring and Packing	2.1) internalFlowGen
	2.2) inputVector
	2.3) respVector
	2.4) exportVector
	2.5) ouputVector
	2.6) biomassVector
	2.7) livingVector
	2.8) PackNet
3) Main Function (Do-all Function, 1) +2))	3) enaListGen
4) Network Validation and Visualisation	4.1) plotNodeFlows
	4.2) plotRange
	4.3) plotFlowRange

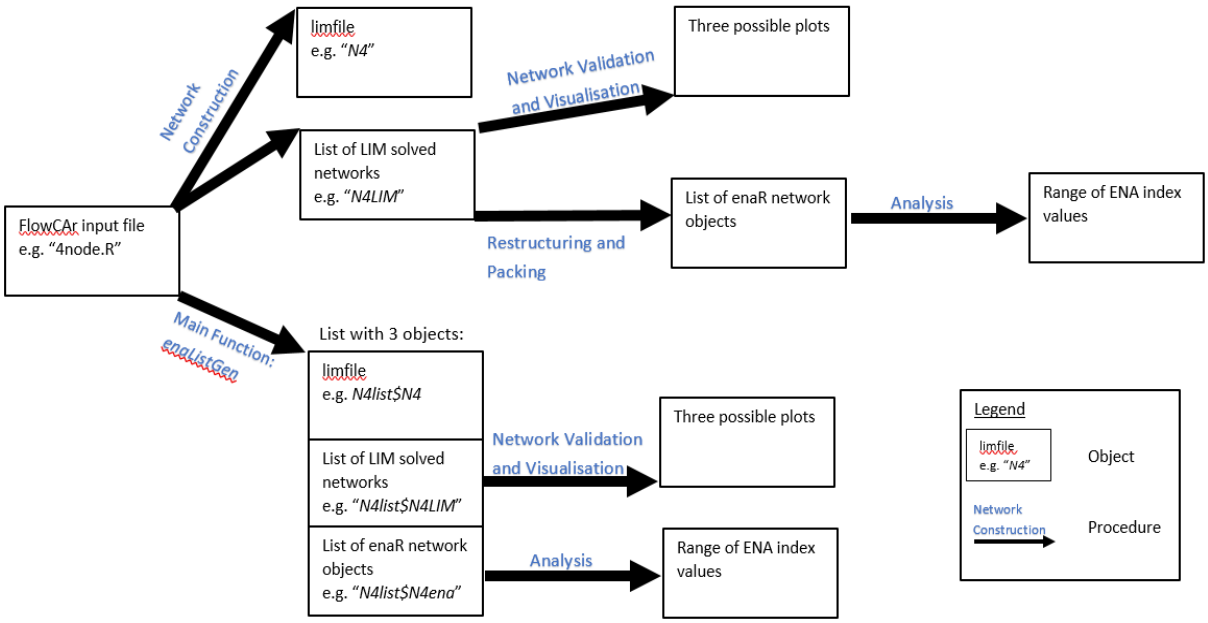

Figure 3: Operational procedure of the FlowCAr package.

Individual functions are presented for full understanding of the procedure, however the main function, **enaListGen** provides a one function shortcut. The four-node network input file ("4node.R") given as an example earlier in the document is used to illustrate the use of all **FlowCAr** functions.

5.1.) Network Construction

Once the **FlowCAr** input file ("4node.R") has been created, it needs to be reformatted into a LIM readable file (limfile) and the unknown flows are solved to generate the ensemble of networks.

5.1.1) Solving

5.1.1.1) LIMbuild

The *LIMbuild()* function outputs the limfile and a text-output which re-iterates the network components. The limfile is an object which takes the input file and rewrites it as a function of the unknowns, creating the matrices and vectors of the problem (Soetaert *et al.,* 2009; van Oevelen *et al.,* 2010).

Running the ***LIMbuild()*** function:

N4 <- LIMbuild(Rfile = "4node.R")

This function outputs the result:

LIM files require to be 'packed' to be used for analysis with enaR.
To properly complete the packing process, the FlowCAr file structure must be correct.
Please check the following information is correct:

Network nodes: PP INVERTEBRATE VERTEBRATE DETNLNODE

Externals: CO2 ppInput ppExport detNLNodeInput vertebrateExport detNLNodeExport

Input nodes: ppInput detNLNodeInput

Export nodes: ppExport vertebrateExport detNLNodeExport

Respiration Element: CO2

Non-Living nodes: DETNLNODE

If the above details do not represent your model correctly, please adjust your FlowCAr input file and re-run this function

The object created is named *N4.* Various aspects of this object can be viewed, such as the network nodes and externals using the codes 'N4$Components' and 'N4$Externals'. The network nodes and externals are displayed as laid out in the **FlowCAr** input file.

Once the information has been verified, multiple possible LIM networks are generated from this output, *N4* using the **limListGen** function.

5.1.1.2) limListGen

The **limListGen** function creates a list of solved possible networks from the limfile which is the output of the *LIMbuild()* function. The **limListGen** function uses the *Xsample()* technique from the **LIM** package (van Oevelen *et al.*, 2010). The *Xsample()* technique solves the solution space a chosen number of times (iterations) using the Monte Carlo Markov Chain (MCMC) algorithm. The distance the algorithm must move between solutions is the jump size. The number of iterations and jump size can be defined within the **limListGen** function or, if unspecified, is default (3000 iterations and NULL jump size). If the jump size is defaulted, *Xsample()* attempts to calculate the ideal size. This requires more processing power and time.

The output of **limListGen** function is a list of two objects. The first object [[1]]AllNetworks is in the form of a matrix. Each row in the matrix represents the value of that flow for each possible network solved using the *Xsample()* technique. The second object [[2]]FlowMatrices is a list of flow matrices which the **limListGen** function has restructured from the first object, created from applying the function '*Flowmatrix()*' (**LIM** package) onto the results of *Xsample()*. Each flow matrix represents a possible network.

Running the **limListGen()** function:

N4LIM <- limListGen(limfile = N4, …)

Produces the output:

Generating Networks
==========
Networks created

The variable "N4LIM" is in the form of a list with two objects.
List object 1 can be accessed through: N4LIM$AllNetworks.
List object 2 can be accessed through: N4LIM$FlowMatrices.

5.1.2) Optional Preparations:

These include checks on whether the generated matrices conform to the ecological input data specified in the **FlowCAr** input file. The user may also want to further restrict the range of certain flows. This can be done with two functions.

5.1.2.1) flowCheck

The first optional check is the *flowCheck()* function. This function cross-checks the flow matrices generated in the *limListGen()* function against the original flow matrix generated by the *LIMbuild()* function run on the **FlowCAr** input file. The MCMC algorithm can result in flow matrices having flows with a value of zero when the desire by the user is to have a flow greater than zero. If there is a cell value larger than 0 in the original flow matrix (i.e. a flow was originally specified to exist in the **FlowCAr** input file), there should be a cell value larger than 0 in the generated list of flow matrices. In the same way, if there is a 0 (i.e. no flow was originally specified) in the original flow matrix, there should be a 0 in the generated list of flow matrices.

Running the **flowCheck()** function:

N4LIMc <- flowCheck(limfile = N4, limList = N4LIM)

Produces a list variable *N4LIMc* which contains 2 objects.

The first object is a list of flow matrices which will replace the first list of flow matrices (generated by the *limListGen()* function).

Object 1 can be accessed though: N4LIMc$FlowMatrices

The second object contains the removed flow matrices because they did not abide by the mentioned rules. This is saved if the user requires the flow matrices later.

Object 2 can be accessed through: N4LIMc$DiscardedList

5.1.2.2) flowLimit

The second optional check is the *flowLimit*() function. In this function, restrictions can be applied to a specific flow. The arguments of the function are:

flowLimit(limfile, limList, flow, MinVal, MaxVal)

A restriction will be applied to the original matrix of flow values, outputting a new matrix which adheres to the restriction. This is then reformatted into a list of flow matrices. The function outputs a list containing two objects in the same format as the output from the *limListGen* function. The first object [[1]] is the restricted matrix of flow values, the second object [[2]] is the restricted list of flow

matrices. This function is useful as it provides the user with the ability to change flows after solving. These changes, and the effects they have on other flows can be visualised (with the **plotRange()** function) within this package.

Additionally, both minVal and maxVal need not be specified, as only providing one will act as a value boundary. By specifying the minimum value (minVal) only, all flows above that are saved and conversely by specifying the maximum value (maxVal) only, all flows below that (and above zero) are saved. If the minVal and maxVal are specified, the flows between that range are saved.

The function **flowLimit()** can be run in 3 possible ways:

NodesMin <- flowLimit(limfile = N4, limList = N4LIM, flow = "invertebrate->detNLNode",minVal = val1)

NodesMin is a list containing the two restricted objects where the values for the flow from *PP* to *INVERTEBRATE* fall above val1.

Object 1 can be accessed though: *NodesMin$AllNetworks*

Object 2 can be accessed though: *NodesMin$FlowMatrices*

NodesMax<- flowLimit(limfile = N4, limList = N4LIM, flow = "invertebrate->detNLNode", maxVal = val2)

NodesMax is a list containing the two restricted objects where the values for the flow from *PP* to *INVERTEBRATE* fall below val2.

Object 1 can be accessed though: *NodesMax$AllNetworks*

Object 2 can be accessed though: *NodesMax$FlowMatrices*

NodesRange<- flowLimit(limfile = N4,limList = N4LIM, flow = "invertebrate->detNLNode",minVal = val1, maxVal = val2)

NodesRange is a list containing the two restricted objects where the values for the flow from *PP* to *INVERTEBRATE* fall between val1 and val2.

Object 1 can be accessed though: *NodesRange$AllNetworks*

Object 2 can be accessed though: *NodesRange$FlowMatrices*

5.2) Packing LIM networks into enaR network objects

To analyse generated networks existing ENA algorithms (e.g. Ulanowicz and Kay, 1991; Allesina and Bondavalli, 2004; Christensen and Walters, 2004; Latham, 2006; Fath and Borrett; 2006, Kazanci, 2007; Kones *et al.,* 2007) have been recoded in R, in the package **enaR** (Borrett and Lau, 2014). **enaR**

requires the network objects to be in a specific format (Butts, 2008). Once multiple flow networks have been generated (10 000 in our example), the **FlowCAr** package provides the required functions to restructure or 'pack' these flow networks into **enaR** network objects so they can be read into **enaR** for further analysis.

To create **enaR** network objects, certain requirements need to be met.

The enaR package stores the model data in the network class defined in the network package (Butts, 2008). In this software, a complete ecosystem network model description includes:

- F is the n × n internal flow matrix, oriented row-to-column
- z a vector of inputs
- r a vector of respirations
- e a vector of exports
- y a vector of outputs, which are respirations plus exports
- X a vector of biomass or storage values
- Living = logical vector indicating if the node is living (TRUE) or non-living (FALSE)

The following section outlines the step by step functions required to reformat and pack the LIM network list into an **enaR** network object list, by creating the objects F, z, e, r, y, X and Living using the **FlowCAr** package (Table 2):

F - The function *internalFlowGen***()** pulls the flow matrices from N4LIM and removes the externals (which are defined in the **FlowCAr** input file), thus producing a list of flow matrices consisting of only the internal flows (F).

InternalFlowMatrices <- internalFlowGen(limfile = N4, limList = N4LIM)

z - The function *inputVector***()** generates vectors by processing the rows of "input" nodes in the flow matrices. The length of the vector is the number of internal nodes in the system, in this instance it will be of length 4. For each node, the total input (all flows originating from outside the system, crossing the boundary) is summed up and a single value is stored as a vector. This is repeated for each matrix in the *N4LIM* list and a list of vectors is created.

inputVectorList <- inputVector (limfile = N4, limList = N4LIM)

r - The function *respVector***()** creates a list of vectors for the respiration flows. Each vector will be the length of the number of internal nodes and the list will be the length of the *N4LIM* list. This vector receives its values from the respiration node column.

respVectorList <- respVector(limfile = N4, limList = N4LIM)

e - The function *exportVector***()** works in the same way as the inputVector() function, however, this function processes the columns of "export" nodes in the flow matrices.

exportVectorList <- exportVector(limfile = N4, limList = N4LIM)

y - The function *outputVector***()** simply uses vector addition for the respective export vector and the respiration vector in the vector lists and produces a list of vectors.

outputVecList <- outputVector(limfile = N4, expVec = exportVectorList, rspVec = respVectorList)

x - The function *biomassVector***()** creates a single vector of values representing the biomass of the nodes. These values are pulled from the "STOCKS" section in the **FlowCAr** file ("4node.R").

biomassVecList <-biomassVector(limfile = N4)

Living - The function *livingVector***()** generates a single logical (TRUE/FALSE) vector based on the naming structure of the **FlowCAr** file. Node names with "NLnode" (not case sensitive) are viewed as non-living nodes and are thus "FALSE" values in the vector. All living nodes have a "TRUE" value.

livingVecList <- livingVector(limfile = N4)

This concludes the preparation for "packing" the **FlowCAr** file into a network object that is usable in **enaR**. Hereafter, the packing can commence.

*PackNet***()** is the function that combines all lists and vectors created for each object and "packs" the **enaR** network object list, utilising the "*pack*" function from the **enaR** package.

PackedNetworkObjectList <- PackNet(InternalFlowMatrices, inputVectorList, respVectorList, exportVectorList, outputVecList, biomassVecList, livingVecList)

5.3) Main Function

The main function **enaListGen** provides the user with a 'shortcut' function. The **enaListGen** function produces a list of three objects. The first object is the [[1]] limfile created using the *LIMbuild* function. The second object is the [[2]] list of solved networks using LIM (*limList*) created using the *limListGen* function. The third object is the [[3]] list of packed **enaR** network objects. This function is versatile. The user can run through all the steps of the preparation phase and then pack into the list of **enaR** network objects, or the **enaListGen** function can be run from the initial **FlowCAr** input file. The latter will run through the whole preparation phase, creating the *limfile,* generating the list of LIM networks (*limList*) and then packing this list into a list of **enaR** network objects (*enaList*). *The limName, limListName* and *enaListName* arguments are used to assign names to the objects outputted by the *enaListGen* function. This function saves the *limfile* and *limList* so the user may run the visualisation functions. The following table briefly explains all the **enaListGen** arguments.

Table 3: **enaListGen** *arguments*

Argument	**Purpose**
Rfile	This is the user generated **FlowCAr** input file
limfile	The created limfile (*LIMbuild*)
limList	The list of solved LIM networks (*limListGen*)
limName	If no *limfile* exists, the name of the *limfile* for the function to output
limListName	If no *limList* exists, the name of the *limList* for the function to output
enaListName	The name of the enaR object list for the function to output
storeAll	Boolean: indicates whether to store each enaR component as a separate list
flowCheck	Boolean: Indicates whether to run flowCheck function

…	Parameters available using the *xsample* function, e.g. number of iterations and jump size

The **enaListGen** function is run is follows:

```
N4list <- enaListGen("4node.R", limName = "N4",
                limListName = "N4LIM", enaListName = "N4ena", storeAll = FALSE,
                flowCheck = FALSE, iter = 10000, jmp = NULL)
```

This will produce the following output:

LIM files are required to be 'packed' for analysis with enaR.

To properly complete the packing process, the FlowCAr file structure must be correct.

The following details indicate vital information.

Network nodes: PP INVERTEBRATE VERTEBRATE DETNLNODE

Externals: CO2 ppInput ppExport vertebrateExport detNLNodeInput detNLNodeExport

Input nodes: ppInput detNLNodeInput

Export nodes: ppExport vertebrateExport detNLNodeExport

Respiration Element: CO2

Non-Living nodes: DETNLNODE

If the above details do not represent your model correctly, please adjust your FlowCAr input file and re-run this function.

Generating Networks

==========

Networks created

[1] "Internal Flows list created"

[1] "Input vector list created"

[1] "Export vector list created"

[1] "Respiration vector list created"

[1] "Output vector list created"

[1] "Living vector list created"

==========

[1] "Network objects packed"

The first section of the result checks the structure of the **FlowCAr** input file. This is vital as **FlowCAr** requires the previously specified rules to accurately create the **enaR** network object. The next section of the result indicates that the list of solved possible networks has been created. The final section of the result indicates the various components of the **enaR** network objects and the list of **enaR** network objects have been created.

The list created by the *enaListGen* function containing the three objects is named *N4List*.

The first object [[1]] can be accessed through: *N4list$N4*

This is the *limfile*

The second object [[2]] can be accessed through: *N4list$N4LIM*

This is the list of LIM networks

The third object can be accessed through: *N4list$N4ena*

This is the list of enaR network objects

If the preparation phase had already been followed, the **enaListGen** function can still be run to pack the generated list of LIM networks into a list of **enaR** network objects. The arguments *limfile* and *limList* are used to specify the already created *limfile* and list of LIM networks respectively. The function will output the list of three objects, the *limfile, limList* and *enaList*. It is only necessary to name the *enaList* (*enaListName*) as the function will extract the names of the *limfile* and *limList* objects. The **enaListGen** function, after the preparation phase is run as follows:

```
N4list <- enaListGen(limfile = N4, limList = N4LIM, enaListName = "N4ena")
```

The objects can be accessed as before.

5.4) Network Validation and Visualisation

There is no standardisation in selecting the required number of iterations and jump size in solving flow networks through LIM (Klepper and Van de Kamer, 1987; Vézina and Platt, 1988; Meersche *et al.*, 2009; Soetaert and van Oevelen, 2010; van Oevelen *et al.*, 2010). There has also not been a means to illustrate whether the solution space has been adequately sampled by the LIM algorithms. Due to the inherent variability and complexity in ecological systems it is not possible to standardise the required number of iterations and jump size. Less variable and complex systems may require smaller number of iterations and jump sizes, while more variable and complex systems may require a larger number of iterations and jump sizes to adequately sample the solution space (Klepper and Van de Kamer, 1987; Vézina and Platt, 1988; Meersche *et al.*, 2009; Soetaert and van Oevelen, 2010; van Oevelen *et al.*, 2010). However, depending on the quality of the data used to set up the initial **FlowCAr** input file the required number of iterations and jump size may vary even more. This package provides functions which allow the user to visually assess whether the solution space has been adequately sampled by the chosen number of iterations and jumps. These plotting functions are run on the list of LIM networks created previously. If this process is being performed during the preparation phase the name of the *limfile* will be the name assigned when the function *LIMbuild* was used (e.g. N4) and the name of the *limList* will be the name assigned when the function *limListGen* was used (e.g. N4LIM). If the visualisation process is performed after the main function *enaListGen* was run, the *limfile* will be the name of the three-object list followed by the dollar sign ($) followed by the name of the *limfile* (e.g. N4List$N4) and the *limList* will be the name of the three-object list followed by the dollar sign ($) followed by the name of the *limList* (e.g. N4List$N4LIM).

5.4.1) plotRange

The **plotRange** function allows the user to generate a plot illustrating the possible range of each flow, each iteration (a point-symbol for the value of that flow in each of the possible LIM networks) calculated using the **limListGen** function, the parsimonious solution (Kones *et al.*, 2006) for each flow and the mean of the ensemble of flow values (van Oevelen *et al.,* 2010) calculated using the **limListGen** function. This visualisation shows how well the solution space has been sampled and if the parsimonious solution is near the extremes of the range. Having the entire set of possible networks plotted indicates the difference between single solution networks chosen using ‘best’ solutions and considering the whole ensemble of possible ecological networks created from a range of input data.

If the **flowLimit** function has been applied, the resulting restrictions can be plotted. This visualisation indicates how much the flow has been limited and how much the applied restrictions influence other flows (Figure 5). In figure 4, the lines represent the flow range, the black point gives

the parsimonious solution, the green point indicates the mean of the iterations and each red cross represents an iteration (Figure 4).

plotRange(limfile = N4, limList = N4LIM, legend = TRUE)

Or

plotRange(limfile = N4List$N4, limList = N4List$N4LIM, legend = TRUE)

This will produce the following plot:

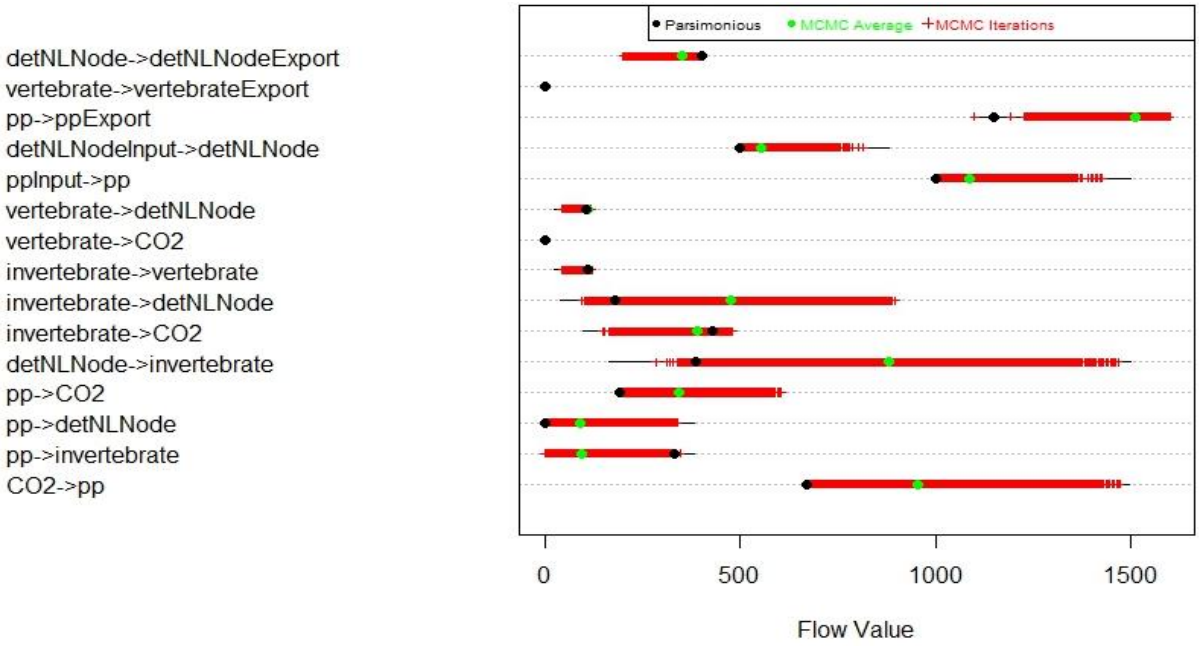

Figure 4: Output from the plotRange *function.*

The parsimonious solution can be seen to lie to the extremes of the flow range (Figure 4) (Kones *et al.*, 2006). The mean solution (van Oevelen *et al.*, 2010) of the ensemble of networks (10 000 networks in this example), in many cases does not adequately represent the flow as it also falls to the extremes and can occur close to the parsimonious solution (Figure 4). Figure 4 illustrates the difference between considering all possible solutions and attempting to select a 'best' solution.

If the **flowLimit** function (as done in section 5.1.2.2) *flowLimit*) has been used to apply restrictions the *plotRange* function is run as follows:

plotRange(limfile = N4, limList = N4LIM, limListLimit = NodeRange, legend = TRUE)

Or

plotRange(limfile = N4List$N4, limList = N4List$N4LIM, limListLimit = NodesRange, legend = TRUE)

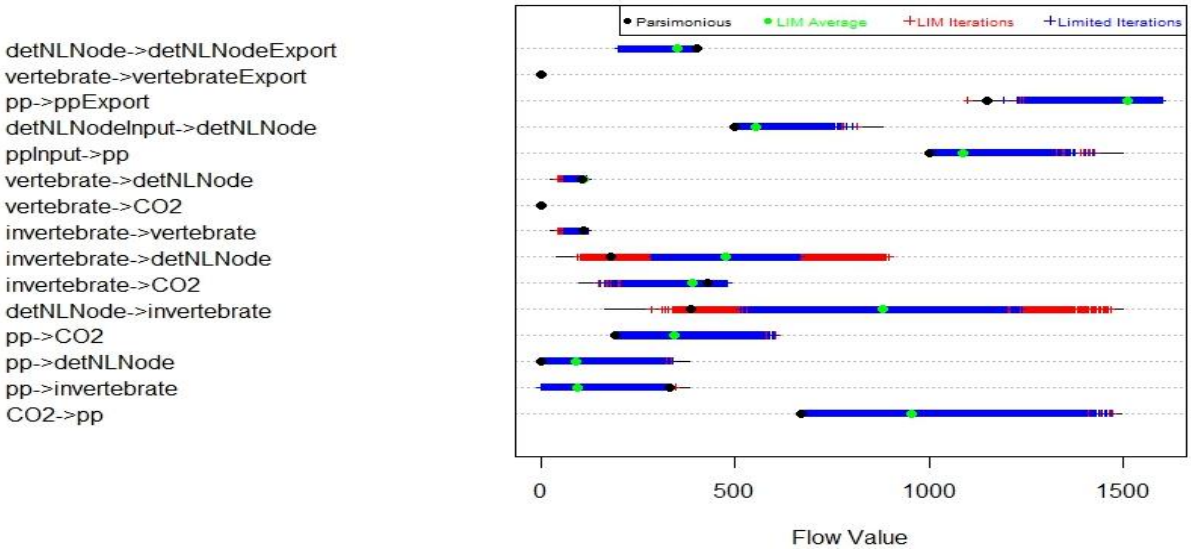

Figure 5: plotRange *output with* flowLimit *restrictions. The blue crosses represent the restricted ranges.*

The colour and style of the points can be changed using the arguments *avcol, allcol, limitcol, avpch, allpch* and *limitpch*. The legend argument (defaulted to TRUE) indicates whether to include the legend or not.

Further arguments are defaulted to the *Plotranges* function from **LIM** package e.g. main title and x axis label.

5.4.2) plotNodeFlows

The **plotNodeFlows()** function allows the user to generate a plot illustrating the frequency of values for each flow over all generated networks. This function only illustrates all flows leaving a defined node rather than all nodes, as one plot cannot accommodate all flows. The **plotNodeFlows** function produces a density plot, histogram and boxplot showing the range of values for each flow leaving the specified node (Figure 6). Note the need to have the whole node name in uppercase as the *LIMbuild* function creates it in this manner and the need for “Quotation marks”.

plotNodeFlows(limfile = N4, limList = N4LIM, flow= “INVERTEBRATE”)

Or

plotNodeFlows(limfile = N4List$N4, limList = N4List$N4LIM, flow= "INVERTEBRATE")

This will produce the following plots:

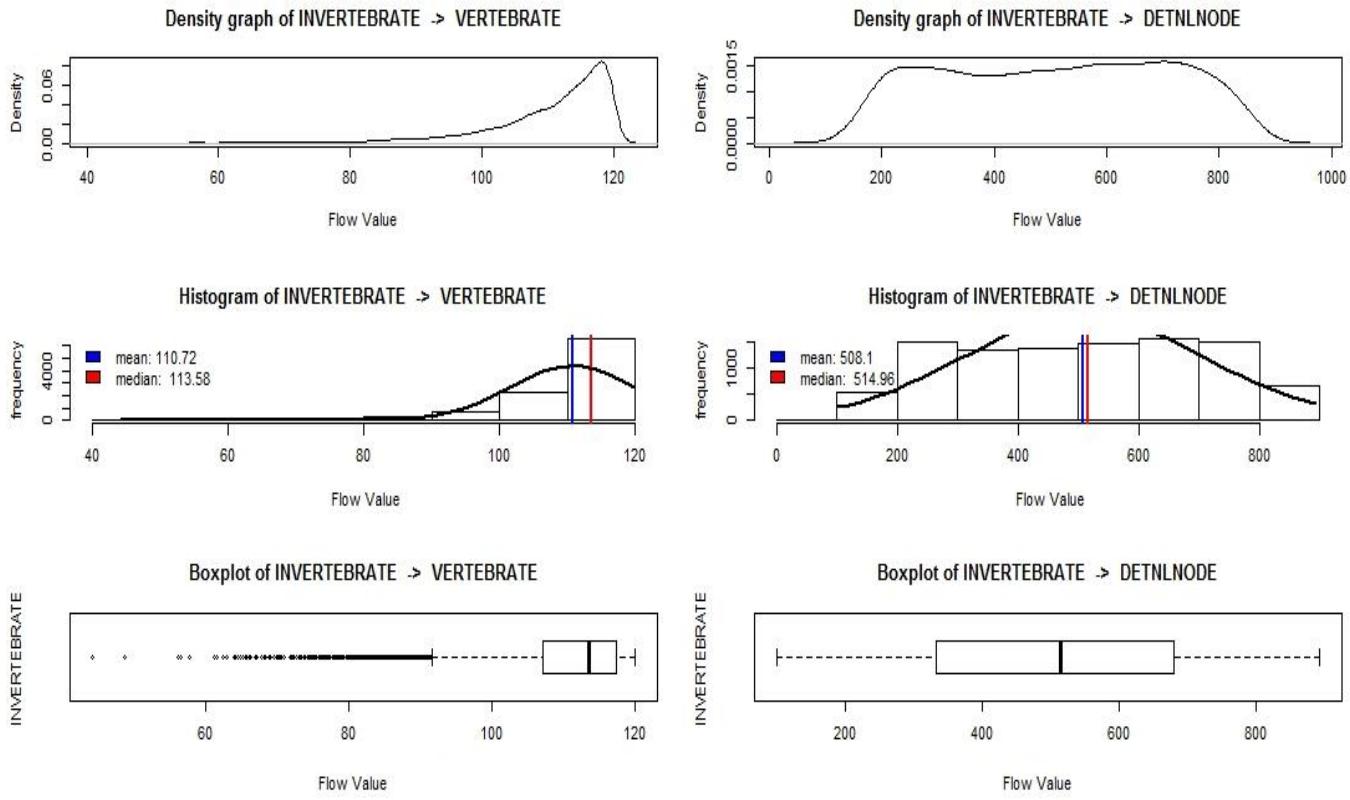

Figure 6: Output from the plotNodeFlows *function showing the range of flow values leaving the invertebrate node. Depicted through a density plot, histogram and boxplot.*

5.4.3) plotFlowRange

The **plotFlowRange** function allows the user to generate a plot which illustrates the frequency an individual flow has been sampled. The plot will illustrate the MCMC sampling of the flow, for example the InvMort (invertebrate mortality, invertebrate->detritus) flow. The flow name must be in "Quotation marks". If the flows were not named in the **FlowCAr** input file, the flow name can be replaced with the flow itself, "*invertebrate->detNLNode*".

plotFlowRange(limList = N4LIM, flow = "invertebrate->detNLNode")

Or

plotFlowRange(limList = N4List$N4LIM, flow = "invertebrate->detNLNode")

This will produce the following plot:

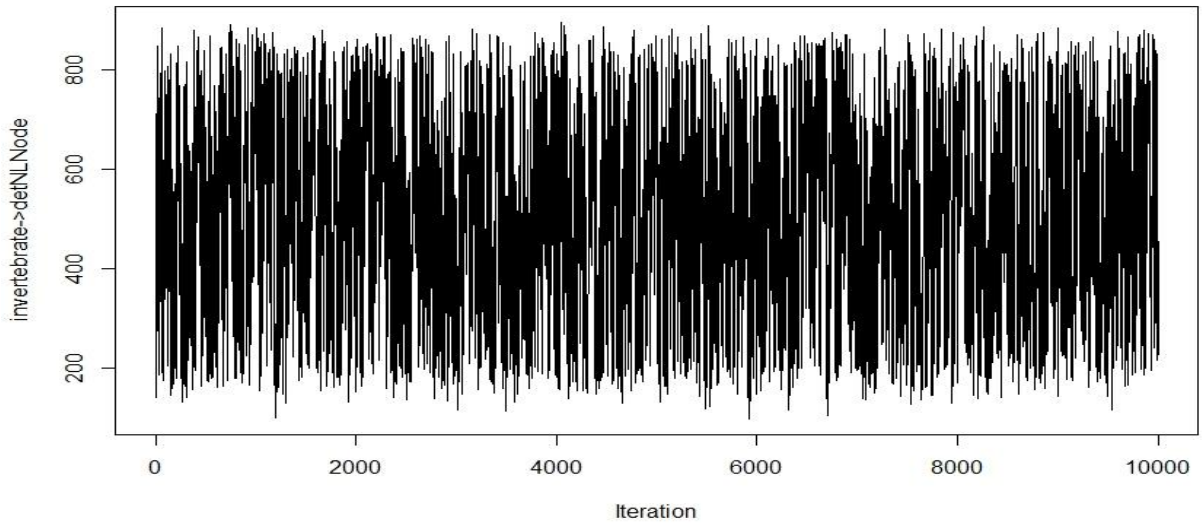

Figure 7: Output from the plotFlowRange *function, illustrating the invertebrate to detritus flow.*

The flow from "*invertebrate->detNLNode*" has been well sampled with a normal distribution of flow values (Figure 7).

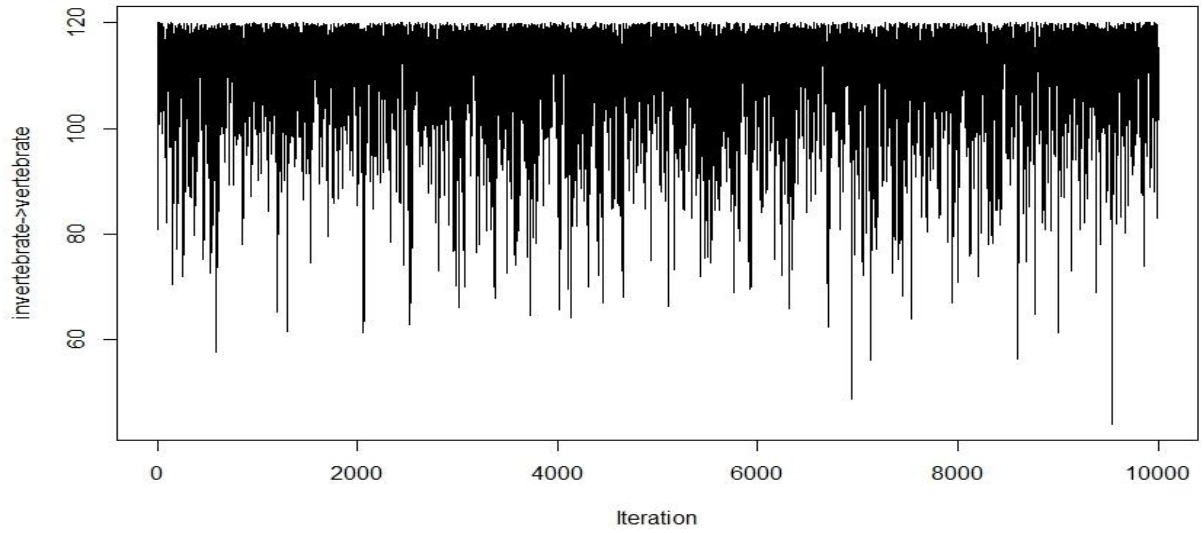

Figure 8: Output from the plotFlowRange *function, illustrating the invertebrate to vertebrate flow.*

The flow from "*invertebrate->vertebrate*" has been well sampled but is skewed towards the higher values (Figure 8). This plot shows the importance of having a high number of iterations to ensure the values which are sampled proportionally less (in the *invertebrate->vertebrate* flow, the lower values) still get represented in the ensemble.

6.) Producing Indices from packed enaRList using enaR package:

Traditionally, Ecological Network Analysis (ENA) has been used to examine the structure, function and dynamics of single networks. One of the goals of the **FlowCAr** package is to facilitate the analysis of multiple networks representing the ranges of input data. Once the list of **enaR** network objects has been packed they can be analysed using **enaR**. This will produce a range of values for a single ENA metric.

Refer to **enaR** vignette (Borrett and Lau, 2014; Lau *et al.,* 2017) for a detailed description of available functions in that package. Here we present a few examples of how to use **enaR** functions to analyse the multiple networks generated using the **FlowCAr** package.

To run analysis on a list in R, the *lapply* function can be used.

For example, a basic flow analysis (Ulanowicz, 1986, Fath and Patten, 1999) can be run on the enaRList:

Flow4 <- lapply(N4List$ena4, enaFlow)

Incorporated in the Flow Analysis are various ENA indices, such as Total System Throughput (TSTp) (Ulanowicz, 1986). The TSTp gives an indication of the amount of energy moving through the system (Ulanowicz, 1986). This can be calculated from the list of **enaR** network objects as follows:

N4TSTp <- lapply(N4List$ena4, function(x) enaFlow(x)$ns[3])

N4TSTp is in list format. To run basic R statistical functions, the object needs to be 'unlisted'.

N4TSTp <- unlist(N4TSTp)

TSTp is the third network statistic of the flow analysis function.

Following on from the example, 10 000 networks were generated, packed into 10 000 **enaR** network objects and then analysed, producing 10 000 unique values for TSTp. If only the parsimonious (first) network had been packed and analysed, the TSTp value would be 5296.386 (*N4TSTp[1]*) or if just the mean network had been packed and analysed, the TSTp value would be 6928.636 (*mean(N4TSTp)*). However, the range of values of TSTp is 5296.386 to 8282.308 (*range(N4TSTp*)). The TSTp values can be assessed in a plot:

A density plot:

Density plots of a single index:

```
dr4 <- density(N4TSTp)
plot(dr4,type ="l",lwd = 5,
   xlab="Total System Throughput (TST)", cex.axis = 1.5,
   ylab="Density",cex.lab = 1.5,main="")
```

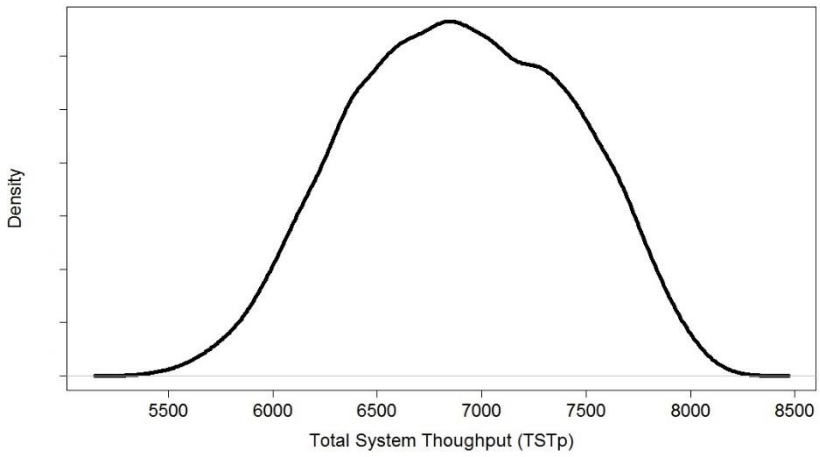

Figure 9: Density plot of the range of values for Total System Throughput (TSTp).

This plot (Figure 9) shows the range of values of the TSTp ENA index. The range of TSTp is normally distributed. This plot indicates the importance of presenting this range and not just the mean or median as there is high variability in the amount of energy passing through (TSTp) the various generated networks. While the distribution of this ENA metric (TSTp) is normally distributed, this is not always the case (Figure 11). The distribution of values may be skewed or bimodal (Figure 11).

A very useful function from **enaR** is *get.ns*. This function extracts all the network statistics (ns) for each network. The *do.call* function is used to reshape the network statistics into a single data frame:

```
N4NS <- lapply(N4List$ena4, get.ns)
N4NSstats <- do.call(rbind, N4NS)
```

From here any of the various ENA index networks statistics can easily be extracted, such as the Finn Cycling Index (FCI) (Finn, 1980). The FCI gives an indication of the amount of energy recycled within the system (Finn, 1980):

range(N4NSstats$FCI)

[1] 0.0704 0.3640

Traditionally, where indices only have a single value their relationship to other indices calculated for the same network could not be investigated. However, now that we have a range of potential values, the relationship between two indices from the same network can be assessed:

From this data frame we can create an interesting plot for further analysis:

Relational plot of two indices:

```
plot(N4NSstats$FCI, N4NSstats$ID.F,pch=20,col="blue",cex=2,
    ylab="Indirect-to-Direct Flow Ratio (I/D, Realised)",
    xlab="Finn Cycling Index (FCI)",cex.axis=1.5,cex.lab=1.5)
```

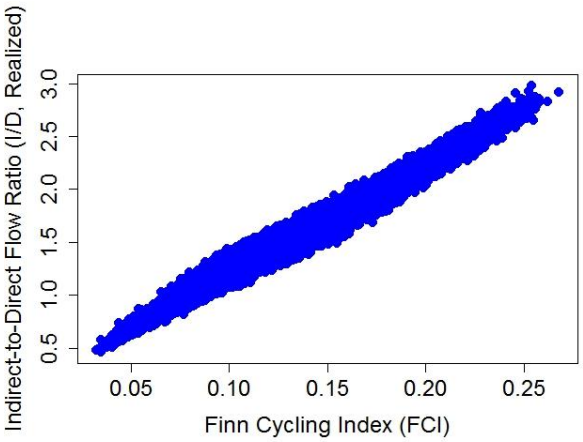

Figure 10: Finn Cycling Index (FCI) vs Indirect to Direct Flow Ratio (I/D).

The index depicted on the x-axis is the Finn Cycling Index (FCI). The I/D ratio indicates the amount of direct to indirect flows (Borrett *et al.*, 2006). The relationship of these two indices produced from the ensemble of generated networks from the one system can be assessed. It is interesting to notice that as the cycling within the system increases so does the I/D ratio (Figure 10). This pattern would normally be unidentifiable if only one value for each index was available.

Having a range of index values allows the a more accurate comparison of index values between networks of the same system. The following plot shows the relative ascendency (A/DC (Ulanowicz, 1986)) for an estuarine system over four different seasons. The A/DC ratio gives an indication of the ability of the system to recover after a perturbation (Ulanowicz, 1986):

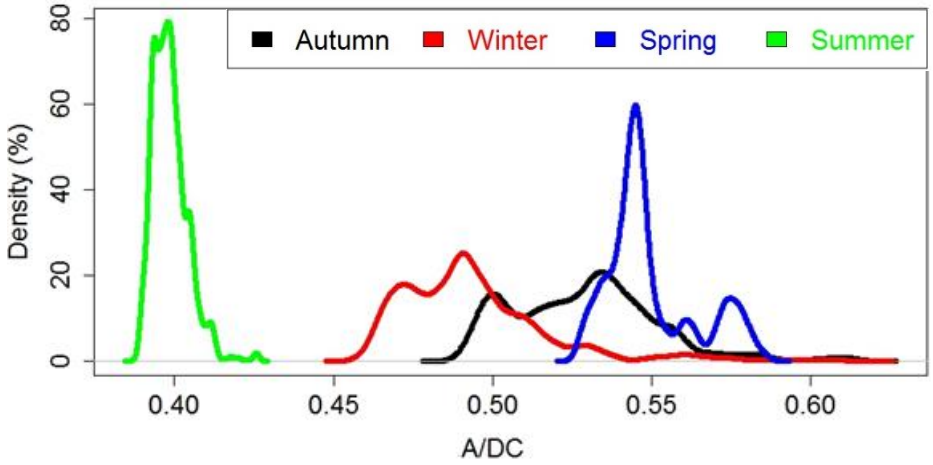

Figure 11: A/DC ratio over four different seasons.

There is an overlap in the A/DC ratio values for three of the seasons (Figure 11). If only one value was available for each seasonal network, such as the parsimonious solution, the overlap of index values would not be evident (Figure 11). Therefore, when comparing the networks, having a range of index values and plotting this range for comparison is necessary, to indicate similarities between networks and highlighting their differences.

There is a large selection of network statistics in the **enaR** package for which this kind of analysis can be done. A description of all **enaR** functions are available in the **enaR** vignette (Borrett and Lau, 2014).

7.) Conclusions

Network analysis requires standardisation from the vital construction phase through to the final analysis. The methodology needs to apply rigorous mathematical solutions to solve the unknown flows while incorporating the inherent ecological variability in the model (Klepper and Van de Kamer, 1987; Vézina and Platt, 1988; Meersche *et al.*, 2009; van Oevelen *et al.*, 2010; Soetaert and van Oevelen, 2010). van Oevelen *et al.,* (2010) identified the power of using Linear Inverse Modelling (LIM) (Woodward *et al.,* 2005) to solve the unknowns in a network. However, the initial parsimonious solution (Kones *et al.*, 2006) and the suggested mean solution (van Oevelen et al., 2010) only provide a single solution out of a possible infinite number of solutions. This single

solution approach that has been used in network analysis has traditionally only focused on the analysis of single networks, producing only a single ENA metric for that network. Hines *et al.,* (2018) identified the need to measure the affect uncertainty has on the analysis of networks through altering flow values by a certain percent and identifying the effect on ENA indices. It is however, vital to incorporate this variability in the initial construction phase of network modelling.

The **FlowCAr** package provides a standardised, robust method to construct networks, solving the unknows using LIM and producing a multitude of possible ecological networks. The accuracy of sampling of the solution space can be visualised, along with the variability from the previously used parsimonious and mean flow solutions. These networks are restructured and packed, ready to be analysed in an ENA format.

The **FlowCAr** package (i) provides accessibility to the code and (ii) easy to use functions and visualisation. Borrett and Lau (2014) outlined the need for a link from LIM solved networks to be used in **enaR.** The uncertainty function (Hines *et al.,* 2018) provides a link from an **enaR** network object to LIM and then back to **enaR.** The great advantage of the **FlowCAr** package is that the networks are ecologically plausible, representing the range of input values and once the networks are solved they can be validated. After this process a link to the **enaR** package is provided. Collaboration between **enaR** and **FlowCAr** is in place to assist using **FlowCAr** for construction and validation of networks and **enaR** for analysis. There is still potential for development of the **FlowCAr** package through the addition of validation functions, improving the construction and assessment phases in network modelling. This package requires knowledge of the R software and the initial set up of the input file may be tedious.

In conclusion, the **FlowCAr** package provides the tools to move from a single ENA metric per ecosystem to a range of values. **FlowCAr** helps to ensure the solutions to the under determined problem of complex ecosystems is ecologically sound. ENA is a rapidly growing field (Proulx *et al.*, 2005; Borrett *et al.*, 2014) with great potential and through the **FlowCAr** package, we hope to assist in this growth.

8.) References

Allesina, S., Bondavalli, C., & Scharler, U. M. (2005). The consequences of the aggregation of detritus pools in ecological networks. *Ecological Modelling*, *189*(1–2), 221–232.

Borrett, S. R., Carter, M., & Hines, D. E. (2016). Six general ecosystem properties are more intense in biogeochemical cycling networks than food webs. *Journal of Complex Networks*, *4*(4). http://doi.org/10.1093/comnet/cnw001

Borrett, S. R., Moody, J., & Edelmann, A. (2014). *The rise of Network Ecology: Maps of the topic diversity and scientific collaboration*. *Ecological Modelling* (Vol. 293). http://doi.org/10.1016/j.ecolmodel.2014.02.019

Christensen, V., & Walters, C. J. (2004). Ecopath with Ecosim: Methods, capabilities and limitations. *Ecological Modelling*. http://doi.org/10.1016/j.ecolmodel.2003.09.003

Christian, R. R., Baird, D., Luczkovich, J., Johnson, J. C., Scharler, U. M., & Ulanowicz, R. E. (2007). Role of network analysis in comparative ecosystem ecology of estuaries. In *Aquatic Food Webs: An ecosystem approach*. http://doi.org/10.1093/acprof:oso/9780198564836.003.0004

Chrystal, R. A., & Scharler, U. M. (2014). Network analysis indices reflect extreme hydrodynamic conditions in a shallow estuarine lake (Lake St Lucia), South Africa. *Ecological Indicators*, *38*, 130–140. http://doi.org/10.1016/j.ecolind.2013.10.025

Crosetto, M., & Tarantola, S. (2001). Uncertainty and sensitivity analysis: tools for GIS-based model implementation. *Int. J. Geographical Information Science*, *15*(5), 415–437. http://doi.org/10.1080/13658810110053125

Dame, J. K., & Christian, R. R. (2006). Uncertainty and the use of network analysis for ecosystem-based fishery management. *Fisheries*, *31*(7), 331–341.

Ecosystems, B. D., & Guide, P. (2007). Food Web Interactions in Benthic Deep-Sea Ecosystems.

Oceanography, *22*(1), 128–143. http://doi.org/10.5670/oceanog.2009.13

Fath, B. D., & Patten, B. C. (1999). Review of the foundations of network environ analysis. *Ecosystems*. http://doi.org/10.1007/s100219900067

Fath, B. D., Scharler, U. M., Ulanowicz, R. E., & Hannon, B. (2007). Ecological network analysis: network construction. *Ecological Modelling*, *208*(1), 49–55. http://doi.org/10.1016/j.ecolmodel.2007.04.029

Goerner, S. J., Lietaer, B., & Ulanowicz, R. E. (2009). Quantifying economic sustainability: Implications for free-enterprise theory, policy and practice. *Ecological Economics*, *69*(1), 76–81. http://doi.org/10.1016/j.ecolecon.2009.07.018

Hines, D. E., Ray, S., & Borrett, S. R. (2018). Uncertainty analyses for Ecological Network Analysis enable stronger inferences. *Environmental Modelling and Software*, *101*, 117–127. http://doi.org/10.1016/j.envsoft.2017.12.011

Klepper, O., & Van De Kamer, J. P. G. (1987). The use of mass balances to test and improve the estimates of carbon fluxes in an ecosystem. *Mathematical Biosciences*, *85*(1), 37–49. http://doi.org/10.1016/0025-5564(87)90098-8

Kones, J. K., Soetaert, K., van Oevelen, D., Owino, J. O., & Mavuti, K. (2006). Gaining insight into food webs reconstructed by the inverse method. *Journal of Marine Systems*, *60*(1–2), 153–166. http://doi.org/10.1016/j.jmarsys.2005.12.002

Lau, M. K., Borrett, S. R., Baiser, B., Gotelli, N. J., & Ellison, A. M. (2017). Ecological network metrics: Opportunities for synthesis. *Ecosphere*, *8*(8). http://doi.org/10.1002/ecs2.1900

Meersche, K. Van Den, Soetaert, K., & van Oevelen, D. (2009). xsample (): An R function for sampling linear inverse problems. *Journal of Statistical …*, *30*(April), 1–15. http://doi.org/10.1002/wics.10

Niquil, N., Arias-González, J. E., Delesalle, B., & Ulanowicz, R. E. (1999). Characterization of the planktonic food web of Takapoto Atoll lagoon, using network analysis. *Oecologia*, *118*(2), 232–241.

Ortega-Cisneros, K., Sharler, U.M., Whitfield, A. K. (2016). Carbon and nitrogen system dynamics in three small South African estuaries, with particular emphasis on the influence of seasons, river flow and mouth state. *Marine Ecology Progress Series*.

Patonai, K., & Jordán, F. (2017). Aggregation of incomplete food web data may help to suggest sampling strategies. *Ecological Modelling*, *352*, 77–89.

Proulx, S. R., Promislow, D. E. L., & Phillips, P. C. (2005). Network thinking in ecology and evolution. *Trends in Ecology and Evolution*. http://doi.org/10.1016/j.tree.2005.04.004

Scharler, U. M. (2012). Ecosystem development during open and closed phases of temporarily open/closed estuaries on the subtropical east coast of South Africa. *Estuarine, Coastal and Shelf Science*, *108*, 119–131. http://doi.org/10.1016/j.ecss.2011.08.003

Soetaert, K., & Oevelen, D. Van. (2009). Modeling food web interactions in benthic deep-sea ecosystems: a practical guide. *Oceanography*, *22*(1), 128–143. http://doi.org/10.5670/oceanog.2009.13

Soetaert, K., & Van Oevelen, D. (2010). Package LIM, implementing linear inverse models in R, 37.

Ulanowicz, R. E. (1986). Growth and development: Ecosystem phenomenology. *New York*, 203.

Ulanowicz, R. E. (2004). Quantitative methods for ecological network analysis. *Computational Biology and Chemistry*, *28*(5–6), 321–339. http://doi.org/10.1016/j.compbiolchem.2004.09.001

van Oevelen, D., van den Meersche, K., Meysman, F. J. R., Soetaert, K., Middelburg, J. J., & Vézina, A. F. (2010). Quantifying food web flows using linear inverse models. *Ecosystems*, *13*(1), 32–45. http://doi.org/10.1007/s10021-009-9297-6

Vézina, A. F., & Platt, T. (1988). Food Web Dynamics in the Ocean. I. Best-estimates of flow networks using inverse methods. *Marine Ecology Progress Series*, *42*, 269–287. http://doi.org/10.3354/meps042269

Vézina, A. F., Berreville, F., & Loza, S. (2004). Inverse reconstructions of ecosystem flows in investigating regime shifts: Impact of the choice of objective function. *Progress in Oceanography*, *60*(2–4), 321–341. http://doi.org/10.1016/j.pocean.2004.02.012

Woodward, G., Speirs, D. C., & Hildrew, A. G. (2005). Quantification and Resolution of a Complex, Size-Structured Food Web. *Advances in Ecological Research*, *36*, 85–135. http://doi.org/10.1016/S0065-2504(05)36002-8

Zhang, Y. (2013). Urban metabolism: A review of research methodologies. *Environmental Pollution*, *178*, 463–473.

Zhang, Y., Lu, H., Fath, B. D., Zheng, H., Sun, X., & Li, Y. (2016). A Network Flow Analysis of the Nitrogen Metabolism in Beijing, China. *Environmental Science & Technology*, *50*(16), 8558–8567.

9.) Appendix

From: Soetaert, K., & Van Oevelen, D. (2010). Package LIM, implementing linear inverse models in R, 37. With a few edits.

The structure of the input file must obey the following rules:

• Declarations are case-INsensitive: flows, Flows, FLOWS are all the same.

• The declaration file is divided into several sections, each contained between '### section name' and '### END section name'. Only the text embraced by "###" and "### END" couples is considered by the parser. The number of #s does not matter. Only the first four characters of the section names are considered, e.g. to designate the parameter section,

we can write ## PARAM or ##PARAMETERS. The declaration sections allowed are summarised in the LIM vignette (Soetaert and van Oevelen, 2010).

• Text in between the declaration sections is ignored (and can be used to write comments).

In the foodweb example for instance, all text positioned in front of "### EXTERNAL" will

be ignored.

• An input file can contain declarations for externals, components, flows, parameters, variables

and defines the additional equalities (i.e. not the mass balances) and inequalities,

costs, and profits (see below).

• Any line that starts with "!" or any blank line is ignored. The exclamation mark can also

be used to discard part of an input line (i.e. everything past "!" is ignored).

• Simple calculations are allowed, i.e. addition and multiplication. The use of brackets for

a calculation is not allowed.

• Continuation of a line is allowed via the use of the "&" sign, at the end of the line.

• Flows can also be given a name using ("name:"). Although this is not mandatory, it may

make the equations more readable.

• Equalities and inequalities can also be given a name. This is only used for output.

A number of shorthand notations are available:

• If the LIM is a flow network, then *FLOWfrom(x)* is shorthand for the sum of all flows

directed out of component x, while *FLOWto(x)* is shorthand for all flows directed into

component x. In the foodweb model example, we wrote:

```
###VARIABLES
invC = Pgraz + InvCon
###END VARIABLES
```

This could have been written as:

```
###VARIABLES
invC = Flowto(inv)
###END VARIABLES
```

• In the inequality section, using [] assigns in one statement lower and upper bounds. In the foodweb example for instance:

```
ppNPP = pp * [ppNPPratioBmin, ppNPPratioBmax]
```