Coding Effects for RSiena

Tom A.B. Snijders

University of Oxford
University of Groningen

October 2025

Section 18 in the RSiena manual treats the coding of new effects.

It is a cookbook-like treatment.

The following slides give a bit more insight into the internal structure.

Effects in RSiena – coding aspects

In the **data** directory of the source code

`https://github.com/snlab-nl/rsiena/blob/main/data`

there is the file **allEffects.csv**,
which in the RSiena package is available as an
internal data frame used to construct effect objects:
allEffects

In RSiena , you can request
dim(allEffects)
and view part of the variables in this data frame in the browser through
effectsDocumentation()

`xxxxxx, yyyyyy, zzzzzz` are names of variables to be filled in;
the ♯ is the internal effect parameter to be filled in.

effectGroups

The first column of **`allEffects`** is the **`effectGroup`**. By

`unique(allEffects$effectGroup)`

you can see that, currently, there are 59 effectGroups.

The **`effectGroup`** is based on combining the type of dependent variable `(oneMode - symmetric - bipartite - behavior - continuous)`
with the kind of explanatory variable (type of dependent variable & `actor covariate - dyadic covariate`)

This is done by **`getEffects`**, defined in the file

`https://github.com/snlab-nl/rsiena/blob/main/R/effects.r`

which calls internal function **`createEffects`** for each **`effectGroup`** separately, handling all variables in the **`Siena`** data set.

Creating new effects – R

The manual (Chapter 18) contains a tutorial about creating new effects.

If, for a new effect, you wish to create a new **`effectGroup`**,
you have to modify **`getEffects`** in `effects.r` accordingly,
and specify its rank order in `effectsDocumentation.r`.

But usually you can employ one of the existing effectGroups.
The RSiena manual (Section 18.2) contains a note about
how various effectGroups handle two-mode networks differently.

Coding effects in C++

The `EffectFactory`

`https://github.com/snlab-nl/rsiena/blob/main/src/model/effects/EffectFactory.cpp`

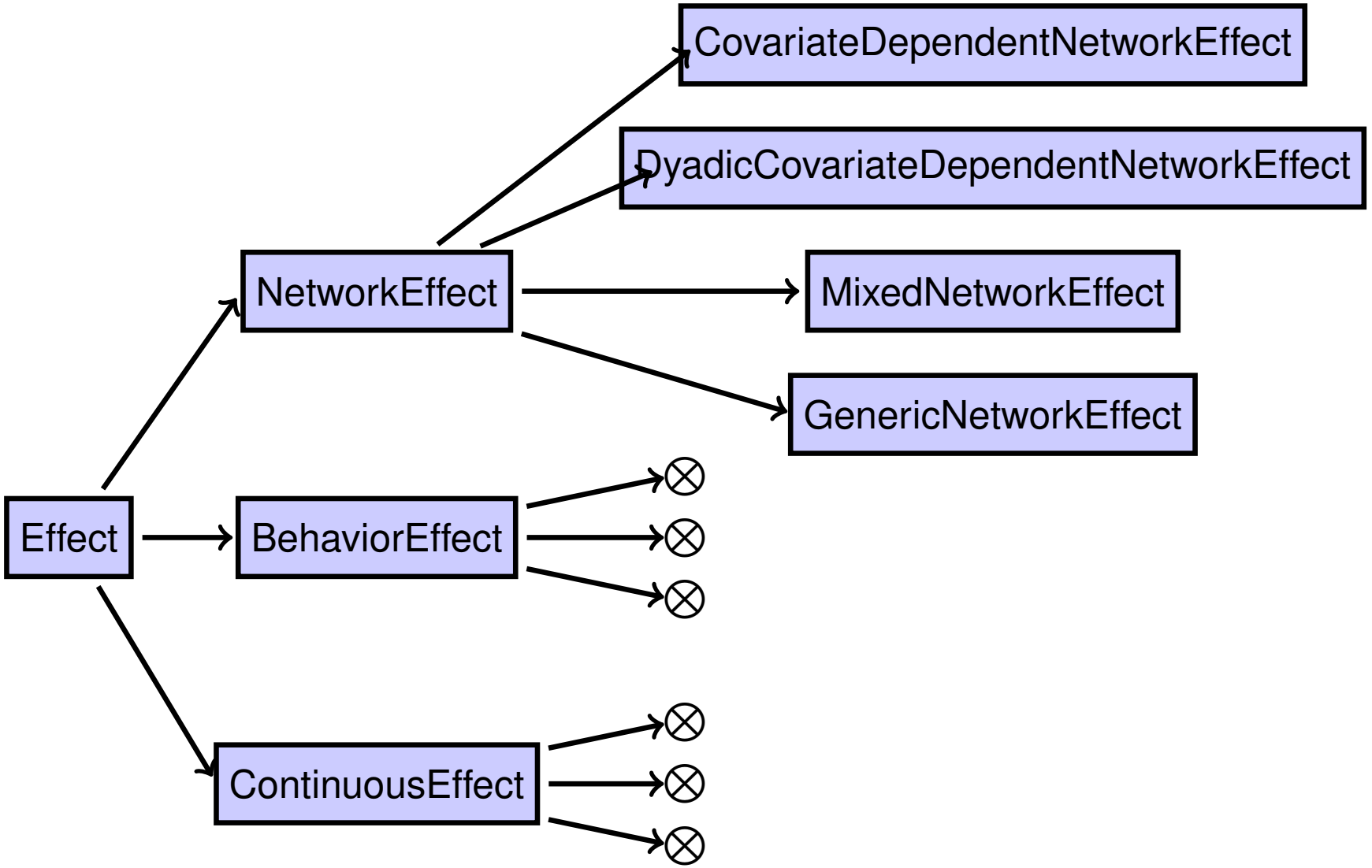
takes the included effects and constructs the computing machinery.

Note that for each effect, we need two things:

1. its contribution to the objective function (exception: `gmom`-type effects);
2. its estimation statistic for MoM (will not be used by ML).

The next page shows the structure of the main effect classes.

Effect classes in C++



NetworkEffect: change statistic

The NetworkEffect class has a given **`ego()`**.

The contribution to the objective function for network effects
is defined by function **`calculateContribution(alter)`**,
and calculations that are not specific to `alter`
are done in **`preprocessEgo(int ego)`**
(called elsewhere in C++, therefore `ego` is specified).
This can be of great help for efficient calculations.

`preprocessEgo` is a virtual function,
therefore you have to reckon with its definition in ancestor classes.

`calculateContribution(alter)` computes the *change statistic*.

NetworkEffect: estimation function

More details are given in **`Siena_algorithms.pdf`**.

`https://www.stats.ox.ac.uk/~snijders/siena/Siena_algorithms.pdf`

Here only the case for effects depending only on the network is treated.

The statistic used for estimation is

$$\sum_{m=2}^{M} s_k^X\left(x(t_m)\right) , \tag{1}$$

where s_k^X is the sum of the effect over all actors, defined by

$$s_k^X(x) = \sum_i s_{ik}^X(x) . \tag{2}$$

and s_{ik}^X is effect k for actor i.

The terms in (2) are the function **`egoStatistic()`**, and these are computed for some (not all) effects as

$$\text{egoStatistic}(i) = \sum_j x_{ij}\, \text{tieStatistic}(i,j)\ . \tag{3}$$

Effect-specific instances of **`egoStatistic()`** or **`tieStatistic()`** are defined in all functions defining specific effects.
It makes no sense to define **`egoStatistic()`** as well as **`tieStatistic()`** for any specific effect, because (3) is computed in **`NetworkEffect.cpp`**,
and will be valid unless **`egoStatistic()`** is defined for the specific effect (which then will avoid the use of **`tieStatistic()`**).

The endowment and creation effects will be computed by applying
`egoStatistic()`
to the network of lost or (respectively) newly created ties;
only if this is to be replaced by something else,
should **`endowmentStatistic()`** (or **`creationStatistic()`**)
be defined in the new effect.

Generic effects

The class **`GenericNetworkEffect`** may be used to specify an effect for a network variable X if its change statistic is given by

$$f_{ij}(x)$$

and the estimation statistic for the evaluation function by

$$\sum_{i,j} x_{ij} f_{ij}^0(x) \ ;$$

with the appropriate modifications for the endowment and creation functions. The functions $f_{ij}(x)$ and $f_{ij}^0(x)$ should be specified as instances of the **`AlterFunction`** class and passed as parameters when creating the effect.

The distinction between f_{ij} and f_{ij}^0 is made mainly to allow possibilities for taking missing covariate data into account.

Generic effects (2)

Composition of **`AlterFunction`**s is allowed, e.g., sum, product, square root, which opens the possibility of simple general definitions.

This can be studied by looking for `GenericNetworkEffect` in `https://github.com/snlab-nl/rsiena/blob/main/src/model/effects/EffectFactory.cpp`

Two examples

Example of a **`NetworkEffect`**:

`https://github.com/snlab-nl/rsiena/blob/main/src/model/effects/AverageDegreeEffect.cpp`

Example of a **`GenericNetworkEffect`**:

Search for `outAct_ego` in

`https://github.com/snlab-nl/rsiena/blob/main/src/model/effects/EffectFactory.cpp`

and then

`https://github.com/snlab-nl/rsiena/blob/main/src/model/effects/generic/EgoOutDegreeFunction.cpp`


The horserace of Siena,
in piazza was originated in medieval