Estimating Temporal Exponential Random Graph Models by Bootstrapped Pseudolikelihood

`R` package vignette for **xergm** 1.1

Philip Leifeld*, Skyler J. Cranmer†, and Bruce A. Desmarais‡

*University of Konstanz
†University of North Carolina, Chapel Hill
‡University of Massachusetts, Amherst

August 7, 2014

This package vignette is designed as a hands-on tutorial for estimating temporal exponential random graph models (TERGMs) (Desmarais and Cranmer 2010, 2012b; Hanneke et˜al. 2010) and assessing goodness of fit and predictive performance (Cranmer and Desmarais 2011; Leifeld and Cranmer 2014) using the **xergm** package (Leifeld et˜al. 2014) for the statistical computing environment **R** (**R** Core Team 2014).

The **xergm** package is compatible with the syntax of **statnet** (Handcock et˜al. 2008; Goodreau et˜al. 2008; Morris et˜al. 2008) and uses some of its functions, particularly from the **ergm** package (Hunter et˜al. 2008) and the **network** package (Butts 2008).

A basic familiarity with ERGMs and their estimation, as in **ergm**, and network data management, as in **statnet**, is assumed and thus not treated extensively in this tutorial.

Throughout the examples provided below, a dataset collected by Andrea Knecht on the dynamics of adolescent friendship networks in a Dutch school class is used as an illustration (Knecht 2006, 2008; Knecht et˜al. 2010; Steglich and Knecht 2009). This dataset is delivered with the **xergm** package and is the classic textbook example for estimating stochastic actor-oriented models (SAOM) using **SIENA** and **RSiena** (Ripley et˜al. 2011; Snijders et˜al. 2010).

1 TERGMs without cross-temporal dependencies

1.1 Preparatory steps

Several **R** packages should be loaded for running the examples presented below: the **texreg** package (Leifeld 2013) will be used for displaying estimation results; the **statnet**

suite of network analysis packages provides basic functions for handling network data (Handcock et˜al. 2008), and the **xergm** package (Leifeld et˜al. 2014).

```
R> require("statnet")
R> require("texreg")
R> require("xergm")
```

After loading the packages, we attach the Knecht dataset to the workspace. The dataset contains four waves of a friendship network (stored as matrices in a `list` object called `friendship`) and several nodal and dyadic covariates (see `help(knecht)` for details). Of particular interest are the sex of the pupils (stored in a data frame called `demographics`) and a network called `primary`, which contains dyadic information on which pupils co-attended the same primary school.

```
R> data("knecht")
```

1.2 Exploring the dataset

To get a first impression of the networks, we plot them using methods from the **statnet** suite of packages. The resulting networks are shown in Figure˜1.

```
R> par(mfrow = c(2, 2), mar = c(0, 0, 1, 0))
R> for (i in 1:length(friendship)) {
+    plot(network(friendship[[i]]), main = paste("t =", i),
+         usearrows = FALSE, edge.col = "grey50")
+  }
```

We would like to explain edge formation at these four time steps, first by assuming independence between the time steps (this section) and in subsequent examples by modeling network evolution as a process with cross-temporal dependencies (sections˜2 and˜3).

For the sake of simplicity, we replicate a basic model described in Snijders et˜al. (2010). In this model, the topology (i. e., the geometric shape) of the networks is determined by the following quantities (with the corresponding model terms of the **ergm** package given in brackets—see `help("ergm-terms")` for details):

- a baseline probability of establishing edges (`edges`),
- the indegree and outdegree of the nodes in the network (computed using the `degree` function and modeled using `nodeocov` and `nodeicov` terms),
- the sex of ego, the sex of alter, and sex match of ego and alter (`nodeofactor`, `nodeifactor`, and `nodematch`, respectively),
- primary school co-attendance (`edgecov`),

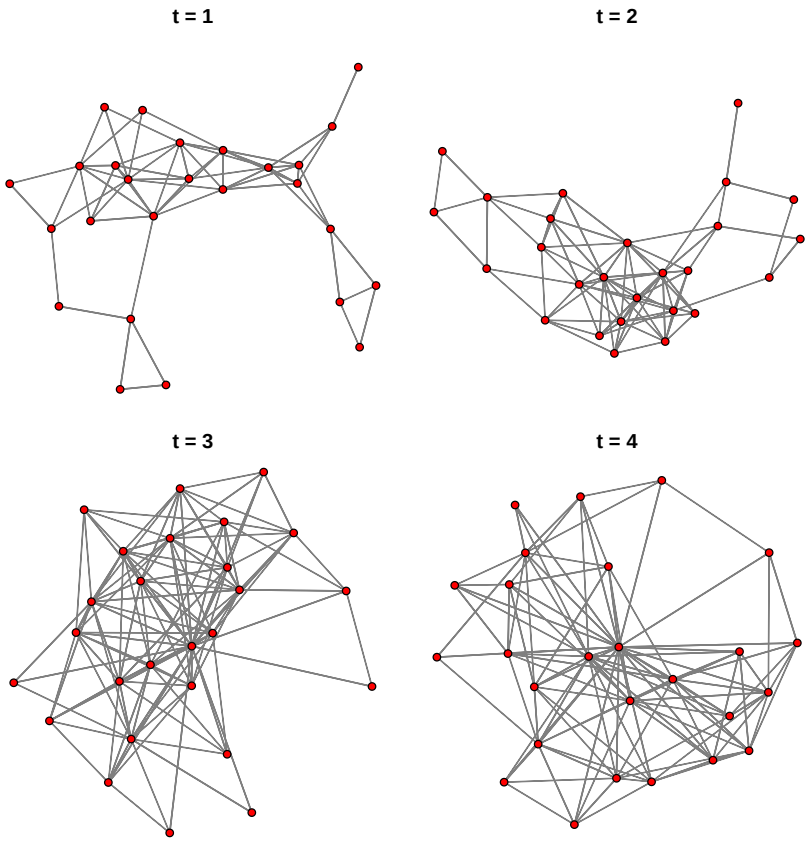

Figure 1: The four networks in the Knecht dataset.

- the tendency of edges to be reciprocated (`mutual`),
- and the number of cyclic triples, transitive triples, and transitive ties (`ctriple`, `ttriple`, and `transitiveties`, respectively).

1.3 Preprocessing the networks and covariates

Before we can estimate a model containing these terms, the data must be preprocessed. Preprocessing can take several different forms and is often necessary to ensure that the data matrices containing variable information are conformable at any given temporal wave and/or across temporal waves. Data matrices may not be conformable because of missing values, node entry, or node exit (known as composition change).

At each time step, all covariates and the dependent network should be composed of the same set of nodes. Because the `btergm` estimation function in the **xergm** package cannot handle any missing values, something must be done to address such values. One has several options. Perhapse most common is to replace `NA` values with the modal value (usually `0`—this approach is also taken by **RSiena**). This makes sense in situations where one can assume that all present edges will be observed, though we may not have a

specific recording of `0` for absent edges. Note, however, that the erronious introduction of `0`s where edges actually exist is a form of measurement error that will result in biased statistical models. A second option is to remove nodes with incomplete edge profiles from the dataset. This strategy, at best, is inefficient because removing a node—perhaps because of a single missing value in one of its edges—requires removing all incoming and outgoing ties from that node. In other words, the network nature of the data means that an entire row and column of the data matrix will be removed any time we remove a node. This compounds the inefficiency of the casewise deletion process compared to that same inefficiency in a normal, rectangular, data frame. At worst, casewise deletion of nodes with missing edges results in bias. Bias will occur from this proceedure any time the occurrence of missing edges is not completely random (e. g., if the occurrence of missing values is related to any attributes of the edge or node, observed or unobserved, bias will be the result). Lastly, missing edge values may be imputed with one of several techniques from a new and budding literature (Handcock and Gile 2010; Koskinen et˜al. 2013; Robins et˜al. 2004). With this cautionary note, the **xergm** package provides the `handleMissings` function to aid the user in removing or imputing missing data iteratively.

Moreover, when cross-temporal dependencies are modeled (such as delayed reciprocity, delayed triadic closure, a lagged dependent network, or edge stability over time), consecutive time steps should feature the same set of nodes. This is not necessarily the case because new nodes may join the network, there may be panel attrition, or respondents may be absent during some waves of data collection. The `xergm` package provides a function called `adjust` to adjust the dimensions of a matrix to the dimensions of another matrix by matching row and column labels and removing rows and columns from matrix A which are absent in matrix B as well as adding new `NA`-filled rows and columns to matrix A where matrix B has additional nodes.

A third function, `preprocess`, serves to automatize both the handling of missing data and the adjustment of matrix dimensions in a single call—it serves as an interface to both functions. Before we can use the `preprocess` function, we must ensure that the dependent networks and covariates have node labels because the networks are matched on the labels. The `preprocess` function requires consecutive networks to be saved as a list of matrices. The `friendship` networks are already provided as a list of matrices, but the nodes are not labeled. Hence we have to make sure first that the matrices have row and column names.

```
R> for (i in 1:length(friendship)) {
+    rownames(friendship[[i]]) <- 1:nrow(friendship[[i]])
+    colnames(friendship[[i]]) <- 1:ncol(friendship[[i]])
+  }
R> rownames(primary) <- rownames(friendship[[1]])
R> colnames(primary) <- colnames(friendship[[1]])
```

In the friendship matrices, missing data are marked as `NA` values, and the incoming and outgoing edges of absent nodes (so-called structural zeroes) are marked by entries

of 10. In the next step, the `preprocess` function takes the friendship networks, replaces missing entries by the modal value (here: 0), iteratively removes nodes that have rows or columns with structural zeroes, and adjusts the friendship matrices at each time step to the dimensions of the primary school network and the nodal sex covariate.

```
R> dep <- preprocess(friendship, primary, demographics$sex,
+      lag = FALSE, covariate = FALSE, na = NA,
+      na.method = "fillmode", structzero = 10,
+      structzero.method = "remove")
```

This command must be repeated for all covariates that have composition changes or missing data or structural zeroes. In this example, none of the covariates has missing data or composition change.

The first argument of the `preprocess` function is the list of matrices to be adjusted, followed by several other objects (matrices, vectors or lists) to which the dimensions of the first object shall be adjusted. If `lag = TRUE` is set, the object will be adjusted across time steps. If the object to be adjusted is a covariate, `covariate = TRUE` should be set. The combination of these arguments allows flexible preprocessing of the network matrices. For example, `lag = TRUE` and `covariate = FALSE` adjusts the dimensions of the second friendship matrix to the first primary matrix, the third to the second, and the fourth friendship matrix to the third primary matrix. If `lag = TRUE` and `covariate = TRUE` are specified, this will adjust the first friendship matrix to the second matrix and so on, effectively generating a lagged dependent network for use as an edge covariate to explain current network activity by previous network activity. Whenever the `lag = TRUE` argument is set, one of the time steps is effectively lost due to the temporal conditioning.

It should be noted that at least three time steps are required by the estimation function to compute meaningful measures of uncertainty when a model with cross-temporal effects is estimated. When no temporal conditioning is needed, two time steps are sufficient for the estimation.

The new object, `dep`, should still have four time steps because `lag = FALSE`. Moreover, Node 21 dropped out of the panel after the second time step, hence the dimensions of the `dep` object should be different from the dimensions of the `friendship` object. We can verify this by calling:

```
R> length(dep)
R> sapply(friendship, dim)
R> sapply(dep, dim)
R> rownames(dep[[3]])
```

Next, the dimensions of the constant covariates have to be adjusted. We would like to have three consecutive matrices per covariate, and these matrices should have the same dimensions as the dependent network at time steps two, three, and four:

```
R> primary.cov <- preprocess(primary, dep, demographics$sex,
+      lag = FALSE, covariate = TRUE)
R> sex.cov <- preprocess(demographics$sex, primary.cov, friendship,
+      lag = FALSE, covariate = TRUE)
```

The next step consists of converting the friendship matrices to `network` objects, computing the square root of the outdegree and indegree centralities for each node at every time step, and adding these centralities as well as the sex attribute as vertex attributes to the networks.

```
R> for (i in 1:length(dep)) {
+    dep[[i]] <- network(dep[[i]])
+    odegsqrt <- sqrt(degree(dep[[i]], cmode = "outdegree"))
+    idegsqrt <- sqrt(degree(dep[[i]], cmode = "indegree"))
+    dep[[i]] <- set.vertex.attribute(dep[[i]], "odegsqrt", odegsqrt)
+    dep[[i]] <- set.vertex.attribute(dep[[i]], "idegsqrt", idegsqrt)
+    dep[[i]] <- set.vertex.attribute(dep[[i]], "sex",
+        sex.cov[[i]])
+  }
```

1.4 Estimation

With these modifications, we are now able to estimate our first TERGM. As in cross-sectional ERGM estimation, nodal covariates are usually provided as vertex attributes of the dependent network, and dyadic covariates are provided as separate matrices or networks. In a temporal setting, dyadic covariates can be either constant or time-varying. Constant covariates can be provided as `network` or `matrix` objects. Time-varying covariates can be provided as lists of networks or matrices. In the Knecht example, `primary` is a constant covariate.

The TERGM is estimated with the `btergm` function, which implements the bootstrapping proceedure outlined in Desmarais and Cranmer (2010) and Desmarais and Cranmer (2012b) and stands for "bootstrapped TERGM." The first argument of the `btergm` function is a formula like in the `ergm` function, but accepting lists of networks or matrices instead of a single network or matrix as the dependent variable. In all other regards, the syntax of `btergm` and the syntax of `ergm` are identical. The second argument, `R`, is the number of bootstrapping replications used for estimation. The more, the better (but also slower). Options for parallel processing on multicore CPUs or HPC servers are available (see `help("btergm")`). As a rule of thumb, at least 100 replications are necessary for testing purposes, while something on the order of 1,000 replications should be used for publication purposes.

```
R> model1 <- btergm(dep ~ edges + mutual + ttriple + transitiveties +
+      ctriple + nodeicov("idegsqrt") + nodeicov("odegsqrt") +
+      nodeocov("odegsqrt") + nodeofactor("sex") +
```

```
+       nodeifactor("sex") + nodematch("sex") + edgecov(primary.cov),
+       R = 100)
```

A TERGM without cross-temporal dependencies is essentially a pooled ERGM. The model and the coefficients therefore describe a single data-generating process that applies to all four time steps independently. The resulting `btergm` object `model1` can be displayed as follows.

```
R> summary(model1, level = 0.95)

==========================
Summary of model fit
==========================

Formula:   dep ~ edges + mutual + ttriple + transitiveties + ctriple +
nodeicov("idegsqrt") + nodeicov("odegsqrt") + nodeocov("odegsqrt") +
nodeofactor("sex") + nodeifactor("sex") + nodematch("sex") +
edgecov(primary.cov)

Bootstrapping sample size: 100

Estimates and 95% confidence intervals:
                          Estimate     2.5%    97.5%
edges                     -9.17591 -10.3639  -8.7508
mutual                     2.96172   2.4417   3.6035
ttriple                    0.21316   0.1252   0.3151
transitiveties             0.36794   0.2097   0.4245
ctriple                   -0.67677  -0.8976  -0.5030
nodeicov.idegsqrt          1.17509   1.0262   1.4656
nodeicov.odegsqrt         -0.28432  -0.4917  -0.1686
nodeocov.odegsqrt          1.19846   1.1665   1.4125
nodeofactor.sex.2          0.60720   0.4753   0.7900
nodeifactor.sex.2          0.22578   0.1075   0.3598
nodematch.sex              1.76805   1.6078   2.1493
edgecov.primary.cov[[i]]   1.05123   0.7864   1.4567
```

By default, a 95 % confidence interval is reported around the estimates. This can be changed by modifying the `level` argument. The `primary` term, for example, has an estimate of 1.05, which means that going to primary school together increases the odds of being friends later on by $100 \cdot (\exp(1.05) - 1) \approx 186\%$ on average conditional on the rest of the network. The effect is significant because 0 is outside the confidence interval of $[0.79; 1.46]$. See, for example, Desmarais and Cranmer (2012a) for further details on interpretation of ERGMs and TERGMs.

1.5 Goodness-of-fit assessment

To assess the goodness of fit, `statnet`-style boxplots of simulated networks versus the observed network(s) or measures of classification performance can be employed. For either of these options, the `gof` function has to be called.

```
R> gof1 <- gof(model1, nsim = 25)
```

The `nsim = 25` argument causes the `gof` function to simulate a total of 100 networks (25 from each of the four time steps). Naturally, when estimating models for publication, more simulations are preferable. Here, however, exposition is accomplished with fewer simulations and less computing time. The resulting `gof1` object can be printed to the console in order to obtain comparison tables of the edge-wise shared partner, dyad-wise shared partner, geodesic distance, indegree, outdegree, instar and outstar distributions of the simulated versus the observed networks. Just like in the **ergm** package, this comparison can be done visually by using the `plot.btergmgof` method (for the result, see Figure˜2). Interpretation of these plots is straightforward; the model is said to fit better the closer the medians of the boxplots (based on the simulated networks) come to the line that plots the actual value of these statistics in the observed network.

```
R> gof1
R> plot(gof1)
```

The model fit looks acceptable. The theme of the next section will be an alternative model which takes into account cross-temporal dependencies.

2 Network evolution with temporal effects

It makes sense to conceive of consecutive measurements of a friendship network, and indeed many longitudinally observed networks, as a process over time rather than independent phenomena. We therefore want the ability to control for friendship choices at each previous time step ("memory"). Controlling for the history of the network is straightforward; a lagged outcome network may be included and functions as an autoregressive term does in regression analysis. The meaning of a "memory" term can be somewhat vague, and different types of memory terms may be preferred depending on the density of the network and the substantive interest of the analyst. For networks of medium density, a change statistic on a network that sums $(x_{ij}^{t-1} - x_{ij}^{t})$ over the dyad ij would be 1 if $x_{ij}^{t-1} = 1$ and -1 if $x_{ij}^{t-1} = 0$. This would be a memory term in which 1s and 0s have the same effect ("edge stability"). Alternatively, one can include an indicator for edge innovation (the creation of a new edge from $t-1$ to t). Lastly, a memory term that captures autoregression of edges (and ignoring non-edges) may be useful for modeling sparse networks, where connection is a fairly rare event. Each of

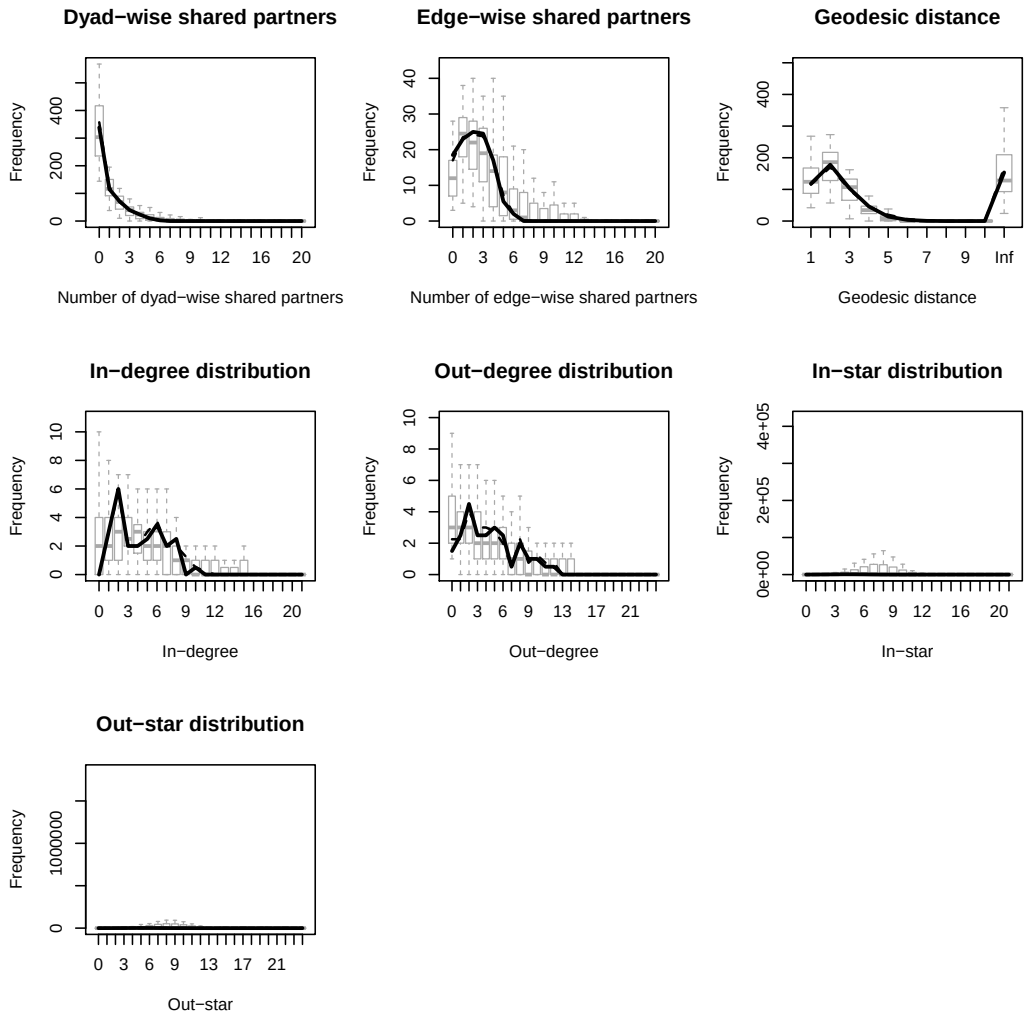

Figure 2: Goodness-of-fit assessment using `statnet`-style boxplots.

these memory terms capture somewhat different processes, and the choice of appropriate memory terms will usually depend on the application. Different types of temporal dependencies may also be used in conjunction with one another.

One may also theorize about other cross-temporal dependencies like single-period delayed reciprocity, in which a directed edge formed at $t-1$ is reciprocated in t. Opportunities for such temporally dependent network statistics are many, and a thorough review of one's options is beyond the scope of the present tutorial.

2.1 Preprocessing the data

As in the last section, the matrices have to be preprocessed. If cross-temporal dynamics are modeled, at least one time step is lost. One usually has to drop the first time step from the list of dependent networks. The reason is that the estimation is conditioned on the covariates at the previous time step each time, and for the first observed network, there is simply no previous time step. Therefore, estimation starts at $t = 2$ and ends at $t = 4$ in the friendship network example, and the covariates start at $t = 1$ and end at $t = 3$.

To achieve this, we first create the list of dependent networks. If `lag = TRUE` and `covariate = FALSE` are set, this will remove the first time step from the list of friendship matrices. Moreover, the following code will remove structural zeroes and adjust the dimensions of the matrices to the dimensions at the previous time steps.

```
R> dep <- preprocess(friendship, primary, demographics$sex,
+      lag = TRUE, covariate = FALSE, na = NA,
+      na.method = "fillmode", structzero = 10,
+      structzero.method = "remove")
```

Next, we create a lagged dependent network for use as a covariate because we can assume that current friendship ties are often the result of previous friendship ties. The following command will remove the last time step from the list of friendship networks and adjust the dimensions of the matrices to the next time step.

```
R> lag <- preprocess(friendship, primary, demographics$sex,
+      lag = TRUE, covariate = TRUE, na = NA,
+      na.method = "fillmode", structzero = 10,
+      structzero.method = "remove")
```

Alternatively, we might consider modeling dyadic stability using a memory term. An edge stability memory term is a matrix which contains cell entries of 1 where the previous matrix has a 1 and -1 where it has a 0. This captures the stability of dyads (not just ties as in the case of the autoregressive lag). We can generate a memory term by specifying `lag = TRUE`, `covariate = TRUE`, and additionally `memory = "stability"` in the `preprocess` function. There are several values the `memory` argument can take: `"stability"` (for dyad stability, irrespective of 0 or 1 values), `"autoregression"` (for positive autoregression, i. e., a lagged dependent network), and `"innovation"` (for edge innovation, i. e., the tendency to form new ties between time steps).

```
R> mem <- preprocess(friendship, primary, demographics$sex,
+      lag = TRUE, covariate = TRUE, memory = "stability",
+      na = NA, na.method = "fillmode", structzero = 10,
+      structzero.method = "remove")
```

We can confirm that there are three time steps for all objects and that there are 26, 25, and 25 nodes at these three time steps after the adjustment.

```
R> length(dep)
R> sapply(dep, dim)
R> sapply(lag, dim)
R> sapply(mem, dim)
```

Next, the dimensions of the covariates have to be adjusted. We would like to have three consecutive matrices per covariate, and these matrices should have the same dimensions as the dependent network at time steps two, three, and four:

```
R> primary.cov <- preprocess(primary, dep, demographics$sex,
+      lag = FALSE, covariate = TRUE)
R> sex.cov <- preprocess(demographics$sex, primary.cov, friendship,
+      lag = FALSE, covariate = TRUE)
```

Another plausible cross-temporal model term would be single-period delayed reciprocity: If there is a tie from v' to v at $t-1$, does this increase the odds that we see a tie from v to v' at t? In other words, are friendship choices reciprocated over time (possibly in addition to reciprocation that occurs within a time step)? To create such a term, one can simply transpose the friendship matrices and create a lagged covariate using the `preprocess` function:

```
R> delrecip <- lapply(friendship, t)
R> delrecip <- preprocess(delrecip, primary, friendship, lag = TRUE,
+      covariate = TRUE, na = NA, na.method = "fillmode",
+      structzero = 10, structzero.method = "remove")
```

Finally, the node attributes sex, indegree and outdegree must be added to the dependent network, as in the previous section.

```
R> for (i in 1:length(dep)) {
+    dep[[i]] <- network(dep[[i]])
+    odegsqrt <- sqrt(degree(dep[[i]], cmode = "outdegree"))
+    idegsqrt <- sqrt(degree(dep[[i]], cmode = "indegree"))
+    dep[[i]] <- set.vertex.attribute(dep[[i]], "odegsqrt", odegsqrt)
+    dep[[i]] <- set.vertex.attribute(dep[[i]], "idegsqrt", idegsqrt)
+    dep[[i]] <- set.vertex.attribute(dep[[i]], "sex", sex.cov[[i]])
+  }
```

2.2 Estimation

We are now ready to estimate the second model, this time with temporal dynamics. The syntax is the same as in section˜1. The lagged network, the memory term, and delayed reciprocity are added as edge covariates.

```
R> model2 <- btergm(dep ~ edges + mutual + ttriple + transitiveties +
+      ctriple + nodeicov("idegsqrt") + nodeicov("odegsqrt") +
+      nodeocov("odegsqrt") + nodeofactor("sex") +
+      nodeifactor("sex") + nodematch("sex") + edgecov(primary.cov) +
+      edgecov(delrecip) + edgecov(mem), R = 100)
```

As before, we can look at the results using `summary(model2)`. For a direct comparison of the first and the second model, we can employ the `screenreg`, `texreg`, and `htmlreg` functions from the **texreg** package:

```
R> screenreg(list(model1, model2))
```

```
============================================================
                           Model 1          Model 2
------------------------------------------------------------
edges                        -9.18 *          -9.54 *
                           [-10.36; -8.75]  [-10.75; -8.72]
mutual                        2.96 *           2.17 *
                           [  2.44;  3.60]  [  1.84;  2.85]
ttriple                       0.21 *           0.13 *
                           [  0.13;  0.32]  [  0.03;  0.24]
transitiveties                0.37 *           0.32 *
                           [  0.21;  0.42]  [  0.29;  0.39]
ctriple                      -0.68 *          -0.55 *
                           [ -0.90; -0.50]  [ -0.82; -0.42]
nodeicov.idegsqrt             1.18 *           1.28 *
                           [  1.03;  1.47]  [  1.12;  1.57]
nodeicov.odegsqrt            -0.28 *          -0.13 *
                           [ -0.49; -0.17]  [ -0.31; -0.04]
nodeocov.odegsqrt             1.20 *           1.50 *
                           [  1.17;  1.41]  [  1.38;  1.73]
nodeofactor.sex.2             0.61 *           0.53 *
                           [  0.48;  0.79]  [  0.38;  0.78]
nodeifactor.sex.2             0.23 *           0.29
                           [  0.11;  0.36]  [ -0.13;  0.51]
nodematch.sex                 1.77 *           1.49 *
                           [  1.61;  2.15]  [  1.30;  1.75]
edgecov.primary.cov[[i]]      1.05 *           0.42
```

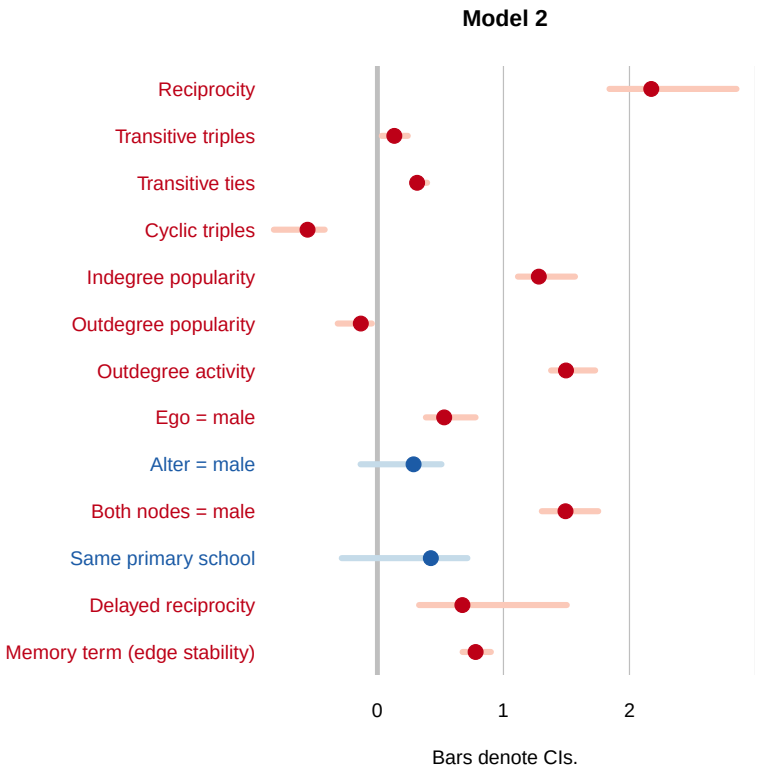

Figure 3: Estimates and confidence intervals of the TERGM with temporal dynamics.

```
                          [  0.79;  1.46]  [ -0.28;  0.71]
edgecov.delrecip[[i]]                        0.67 *
                                            [  0.33;  1.50]
edgecov.mem[[i]]                             0.78 *
                                            [  0.68;  0.90]
============================================================
* 0 outside the confidence interval
```

The coefficients and confidence intervals can also be explored visually using the `plotreg` function from the **texreg** package. The resulting forest plot is shown in Figure˜3.

```
R> plotreg(model2, custom.model.names = "Model 2", custom.coef.names =
+      c("Edges", "Reciprocity", "Transitive triples",
+      "Transitive ties", "Cyclic triples", "Indegree popularity",
+      "Outdegree popularity", "Outdegree activity", "Ego = male",
+      "Alter = male", "Both nodes = male", "Same primary school",
```

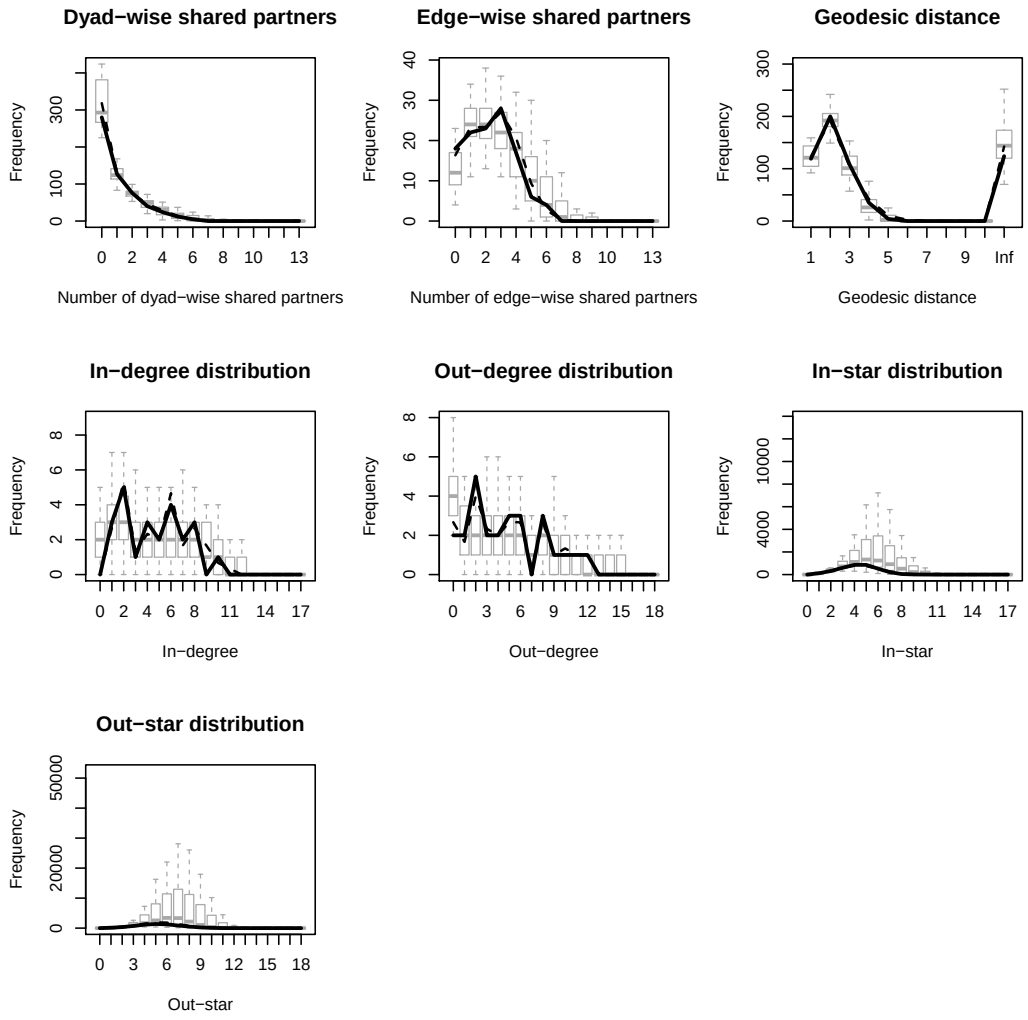

Figure 4: Goodness-of-fit assessment of the second TERGM.

```
+       "Delayed reciprocity", "Memory term (edge stability)"),
+       omit.coef = "Edges")
```

With the inclusion of the three significant temporal effects, transitive triples and one of the outdegree terms are no longer significant.

2.3 Goodness of fit

As in section˜1, we assess the goodness of fit as follows.

```
R> gof2 <- gof(model2, nsim = 25)
R> plot(gof2)
```

The boxplot diagrams (Figure˜4) show a similar goodness of fit as in the first model (shown in Figure˜2). We seem to capture the data-generating process reasonably well with both specifications. In the second specification, the k-star distributions are captured more accurately.

3 Out-of-sample prediction with TERGMs

For predicting the network at future time points and to gain more confidence in the robustness of the results, we may consider out-of-sample prediction based on our model specification. As an illustration, we will try to predict the network at $t = 4$ using a model estimated from the three previous networks and based on the covariates at $t = 3$ and $t = 4$.

We can recycle the lists of objects prepared in the previous section. Particularly, the first two out of the three elements of the `dep` object contain the friendship network at time steps two and three while the first two items of the covariate lists contain time steps one and two because we modeled cross-temporal dynamics (i. e., there is a lag). Within the formula of the `btergm` estimation function, it is possible to index the lists to work only with the first two items in each list because we want to reserve the last item for comparison with the results. To verify that the coefficients are substantively similar as before, we can print a table with the three models using the `screenreg` function from the **texreg** package (Leifeld 2013).

```
R> model3 <- btergm(dep[1:2] ~ edges + mutual + ttriple +
+      transitiveties + ctriple + nodeicov("idegsqrt") +
+      nodeicov("odegsqrt") + nodeocov("odegsqrt") +
+      nodeofactor("sex") + nodeifactor("sex") + nodematch("sex") +
+      edgecov(primary.cov[1:2]) + edgecov(delrecip[1:2]) +
+      edgecov(mem[1:2]), R = 100)
R> screenreg(list(model1, model2, model3))
```

Next, we can employ the `gof` function to simulate 100 networks from the model and compare them to the observed network at $t = 4$. Remember that the model was estimated based on time steps two to three for the dependent networks and one to two for the lagged covariates (hence we used index `[1:2]` above). The dependent network at $t = 4$ and the covariates at $t = 3$ can therefore be accessed by using the index `[[3]]` when we use the `gof` function.

```
R> gof3 <- gof(model3, nsim = 100, target = dep[[3]], formula =
+      dep[[3]] ~ edges + mutual + ttriple + transitiveties +
+      ctriple + nodeicov("idegsqrt") + nodeicov("odegsqrt") +
+      nodeocov("odegsqrt") + nodeofactor("sex") +
+      nodeifactor("sex") + nodematch("sex") +
+      edgecov(primary.cov[[3]]) + edgecov(delrecip[[3]]) +
+      edgecov(mem[[3]]))
```

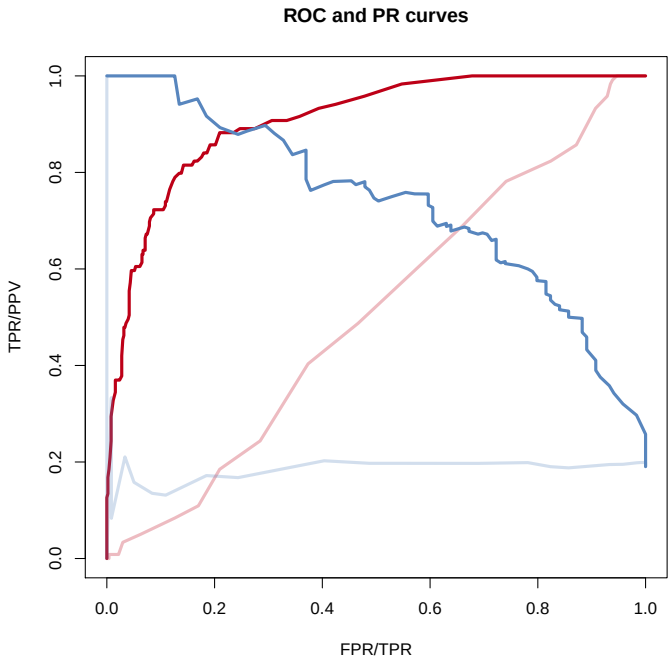

Figure 5: Out-of-sample predictive fit of model~3 versus a null model.

This function call simulates 100 new networks based on model 3 (i. e., without the last time step) and based on the covariates at $t = 3$ and compares the simulations to the observed network at $t = 4$. The `target` argument specifies the network to which we are comparing the simulations. The coefficients are taken from the `model3` object. We specify an explicit `formula` where we tell the `gof` function to simulate from the covariates at $t = 3$.

Now we have four options to assess the predictive performance: we can display boxplot diagrams as above (results not reported here), we can print the output of the `gof3` object to the **R** console (results not reported here), we can plot a receiver operating characteristics (ROC) curve of the out-of-sample prediction of network ties, and we can plot a precision-recall (PR) curve of the prediction.

```
R> plot(gof3, roc = FALSE, pr = FALSE)
R> gof3
R> plot(gof3, boxplot = FALSE, roc = TRUE, pr = FALSE,
+      roc.random = TRUE, ylab = "TPR/PPV",
+      xlab = "FPR/TPR", roc.main = "ROC and PR curves")
R> plot(gof3, boxplot = FALSE, roc = FALSE, pr = TRUE,
+      pr.random = TRUE, rocpr.add = TRUE)
```

The two curves are displayed in Figure~5. The dark red curve shows the ROC curve for model~3 while the light red curve is the ROC curve of a random graph of the same

size with the same density (i. e., the same model but only with an `edges` term). Model~3 clearly has a much better predictive performance than the null model. The dark blue curve shows the PR curve for model~3 while the light blue curve is the PR curve of the null model. PR curves are better suited for sparse networks because absent ties are not taken into account. For the Knecht data, however, this is not an issue. The PR curve also shows that the predictive fit is good.

ROC and PR curves can be used to compare different model specifications, also for within-sample goodness of it. To condense the performance into a single measure, the area under the curve (AUC) can be reported for both curves. AUC values are stored in the `btergmgof` objects and are printed along with other goodness-of-fit measures when the object (here: `gof3`) is called. They can be accessed directly by calling `gof3$auc.roc` and `gof3$auc.pr`.

4 Other TERGM-related functions

The **xergm** package provides several other TERGM-related functions.

Checking for model degeneracy is possible using the `gof` function and specifying the argument `checkdegeneracy = TRUE` (which is the default value). When the resulting `btergmgof` object is printed, a table indicates which network statistics are problematic. Note that at least 1,000 simulations should be created for the degeneracy check (argument `nsim = 1000`).

The `gof` function thus serves to assess degeneracy, goodness of fit (via `statnet`-like boxplot diagrams), and (out-of-sample or within-sample) predictive performance using ROC and PR curves. For easy comparison of different types of models, there are `gof` methods for `btergm`, `ergm`, and `sienaAlgorithm` objects (the latter produced by **RSiena**) (Leifeld and Cranmer 2014).

The `simulate.btergm` method serves to create new networks given a `btergm` model and a set of covariates. The `confint.btergm` method can be used to recompute confidence intervals at arbitrary confidence levels. The same result can be achieved by using the `level` argument of the `summary.btergm` method. The `btergm.se` function computes standard errors and p values for the estimates. `btergm.timesteps` extracts the number of distinct time steps from a `btergm` object. There is also a `coef.btergm` and a `nobs.btergm` method for extracting the estimates and the number of observations, respectively. The formula can be extracted from a `btergm` object (or from an `ergm` object) using the `getformula` function.

Finally, the generic `interpret` function accepts `ergm` and `btergm` models and facilitates micro-level interpretation of the effects, as suggested by Desmarais and Cranmer (2012a).

References

Butts, C.˜T. (2008). **network**: A package for managing relational data in **R**. *Journal of Statistical Software*, 24(2):1–36.

Cranmer, S.˜J. and Desmarais, B.˜A. (2011). Inferential network analysis with exponential random graph models. *Political Analysis*, 19(1):66–86.

Desmarais, B.˜A. and Cranmer, S.˜J. (2010). Consistent confidence intervals for maximum pseudolikelihood estimators. In *Proceedings of the Neural Information Processing Systems 2010 Workshop on Computational Social Science and the Wisdom of Crowds*.

Desmarais, B.˜A. and Cranmer, S.˜J. (2012a). Micro-level interpretation of exponential random graph models with application to estuary networks. *The Policy Studies Journal*, 40(3):402–434.

Desmarais, B.˜A. and Cranmer, S.˜J. (2012b). Statistical mechanics of networks: Estimation and uncertainty. *Physica A: Statistical Mechanics and its Applications*, 391(4):1865–1876.

Goodreau, S.˜M., Handcock, M.˜S., Hunter, D.˜R., Butts, C.˜T., and Morris, M. (2008). A **statnet** tutorial. *Journal of Statistical Software*, 24(9):1–26.

Handcock, M.˜S. and Gile, K.˜J. (2010). Modeling social networks from sampled data. *The Annals of Applied Statistics*, 4(1):5–25.

Handcock, M.˜S., Hunter, D.˜R., Butts, C.˜T., Goodreau, S.˜M., and Morris, M. (2008). **statnet**: Software tools for the representation, visualization, analysis and simulation of network data. *Journal of Statistical Software*, 24(1):1–11.

Hanneke, S., Fu, W., and Xing, E.˜P. (2010). Discrete temporal models of social networks. *Electronic Journal of Statistics*, 4:585–605.

Hunter, D.˜R., Handcock, M.˜S., Butts, C.˜T., Goodreau, S.˜M., and Morris, M. (2008). **ergm**: A package to fit, simulate and diagnose exponential-family models for networks. *Journal of Statistical Software*, 24(3):1–29.

Knecht, A. (2006). Networks and actor attributes in early adolescence. ICS Codebook˜61, The Netherlands Research School ICS, Department of Sociology, Utrecht University, Utrecht.

Knecht, A. (2008). *Friendship Selection and Friends' Influence. Dynamics of Networks and Actor Attributes in Early Adolescence*. Phd dissertation, University of Utrecht, Utrecht.

Knecht, A., Snijders, T. A.˜B., Baerveldt, C., Steglich, C. E.˜G., and Raub, W. (2010). Friendship and delinquency: Selection and influence processes in early adolescence. *Social Development*, 19(3).

Koskinen, J.˜H., Robins, G.˜L., Wang, P., and Pattison, P.˜E. (2013). Bayesian analysis for partially observed network data, missing ties, attributes and actors. *Social Networks*, 35(4):514–527.

Leifeld, P. (2013). **texreg**: Conversion of statistical model output in R to LaTeX and HTML tables. *Journal of Statistical Software*, 55(8):1–24.

Leifeld, P. and Cranmer, S.˜J. (2014). Comparing the temporal exponential random graph model (TERGM) and the stochastic actor-oriented model (SAOM). Paper presented at the 2014 Annual Meeting of the Swiss Political Science Association (SVPW), Bern, Switzerland, January 30.

Leifeld, P., Cranmer, S.˜J., and Desmarais, B.˜A. (2014). ***xergm**: Extensions of Exponential Random Graph Models*. R package version 0.51.

Morris, M., Handcock, M.˜S., and Hunter, D.˜R. (2008). Specification of exponential-family random graph models: Terms and computational aspects. *Journal of Statistical Software*, 24(4):1–24.

R Core Team (2014). ***R**: A Language and Environment for Statistical Computing*. **R** Foundation for Statistical Computing, Vienna, Austria.

Ripley, R.˜M., Snijders, T. A.˜B., and Preciado, P. (2011). *Manual for RSiena*. University of Oxford: Department of Statistics, Nuffield College, Oxford.

Robins, G., Pattison, P., and Woolcock, J. (2004). Missing data in networks: Exponential random graph (p*) models for networks with non-respondents. *Social Networks*, 26(3):257–283.

Snijders, T. A.˜B., Steglich, C. E.˜G., and van˜de Bunt, G.˜G. (2010). Introduction to actor-based models for network dynamics. *Social Networks*, 32.

Steglich, C. E.˜G. and Knecht, A. (2009). Die statistische Analyse dynamischer Netzwerkdaten. In Stegbauer, C. and Häussling, R., editors, *Handbuch der Netzwerkforschung*. VS Verlag für Sozialwissenschaften, Wiesbaden.