Lehrstuhl Algorithmik, Prof. Dr. Ulrik Brandes
Fachbereich Informatik und Informationswissenschaft
Universität Konstanz
Wintersemester 2009

Exponentielle Zufallsgraphen-Modelle

Bericht zum Master-Projekt „Netzwerkmodelle“

von Christian Ehinger

Inhaltsverzeichnis

1 Einführung **2**

1.1 Exponentielle Zufallsgraphen-Modelle (ERGM) 2

1.2 Die Spiegelnetzwerk-Methode 3

1.3 Testumgebung 3

2 Verbesserungsansätze **5**

2.1 Modellierung der Vererbung 5

2.1.1 Introduce 5

2.1.2 Attractiveness 6

2.2 Parametrisierung 6

2.2.1 Einfache Suche 7

2.2.2 Parameterschätzung 7

2.3 Neue Statistiken 8

3 Durchführung **9**

3.1 Die Tests 9

3.1.1 Ausgangstests 9

3.1.2 Andere Parameter 9

3.1.3 Vererbung 9

3.1.4 Neue Statistiken 10

3.2 Ergebnisse 11

4 Zusammenfassung und Ausblick **12**

A R-Code **13**

A.1 Ausgangstests 13

A.2 Andere Parameter 17

A.3 Vererbung 20

A.4 Neue Statistiken 26

1 Einführung

Gegenstand des Projekts ist die Veröffentlichung von Fowler et al. [1], in der die Autoren ein Modell für soziale Netzwerke vorstellen, das die Vererbung bestimmter Attribute zur Bildung von Beziehungen zwischen den Akteuren berücksichtigt. Dieses „Attract&Introduce"-Modell (A&I) wird u.a. durch die von den Autoren entwickelte Spiegelnetzwerk-Methode mit alternativen Netzwerkmodellen verglichen. Dabei wird der Schluss gezogen, dass das A&I-Modell bestimmte Effekte, die in realen Netzwerken vorkommen (wie etwa Transtitivitäten), besser modelliert. Ziel des Projektes ist, Aussagen über die Qualität des Vergleichs von Fowler et al. zu treffen. Dieser Bericht schildert die Vorgehensweise und die Ergebnisse des Projektteils, in dem untersucht wurde, ob speziell für das Exponentielle Zufallsgraphen-Modell (ERGM) alternative Implementierungen für die Beurteilung mit Hilfe des Spiegelnetzwerks angemessener sind.

1.1 Exponentielle Zufallsgraphen-Modelle (ERGM)

Die Klasse der ERGM (*Exponential Random Graph Models*) ist gegeben durch eine Familie von Wahrscheinlichkeitsfunktionen der Gestalt

$$P(Y = y) = \frac{1}{\kappa} \exp\{\eta^T g(y)\} = \frac{1}{\kappa} \exp\{\sum_i \eta_i g_i(y)\},$$

wobei Y eine Matrix von Zufallsvariablen $Y_{ij} \in \{0, 1\}$ darstellt und y eine bestimmte Matrix, d.h. eine konkrete Ausprägung des Netzwerkes, ist. Im Exponenten der Exponentialfunktion steht das Skalarprodukt zweier Vektoren. Der Vektor $g(y)$ enthält Statistiken. Diese Statistiken können bestimmte Konfigurationen des Netzwerkes zählen (z.B. Kanten, Dreiecke, k-Sterne, usw.) oder sogenannte Kovariate sein, also Statistiken, die nicht nur die Netzwerkstruktur, sondern auch Knotenattribute betrachten. Die Statistiken werden gewichtet durch einen Vektor η mit Parametern. Die Konstante $\kappa = \sum_y \exp\{\eta^T g(y)\}$ dient der Normalisierung, damit $\sum_y P(Y = y) = 1$.

Bestimmte Spezifikationen können zu entarteten Modellen führen. Ein Modell wird als nahezu entartet (*near degenerate*) bezeichnet, wenn es nur wenige Graphen mit einer merklich von Null verschiedenen Wahrscheinlichkeit erzeugt. Dies

kann besonders bei Markov-Zufallsgraphen der Fall sein, die gegegen sind durch

$$P(Y = y) = \frac{1}{\kappa} \exp\{\sum_{k=1}^{n-1} \sigma_k S_k(y) + \tau T(y)\}$$

1.2 Die Spiegelnetzwerk-Methode

Fowler et al. entwickelten die Spiegelnetzwerk-Methode, um festzustellen, ob Netzwerkmodelle die Modellierung von Vererbung bezüglich Knotengrad, Transitivität und Betweenness-Zentralität erlauben. Dabei wird zunächst eine Menge von Knoten erzeugt, die mit bestimmten Attributen versehen sind. Dann wird nach den Vorschriften eines gegebenes Netzwerkmodells ein Netzwerk bestimmter Größe aufgebaut (simuliert). Anschließend wird eine zweite Knotenmenge erzeugt, wobei die Attribute eines zufällig gewählten Knotens des ersten Netzwerkes auf einen Knoten der zweite Menge kopiert werden. Der Knoten besitzt dann einen Zwilling in der zweiten Menge. Über der neuen Knotenmenge wird nun ein zweites Netzwerk nach den Vorschriften des Modells simuliert. Dies alles wird 10.000 Mal wiederholt und jedes Mal werden der Eingangsgrad, der Ausgangsgrad, die Transtitivität (gemessen durch den lokalen Clustering-Koeffizienten) sowie die Betweenness-Zentralität der Zwillinge miteinander verglichen. Abschließend wird über alle Durchgänge der lineare Zusammenhang berechnet (Pearson Korrelation).

1.3 Testumgebung

Ursprünglich sollten die Modell-Simulationen und -Fits auf den Titan-Rechnern des Rechenzentrums der Universität Konstanz durchgeführt werden, die speziell für aufwändige Berechnungen zur Verfügung stehen. Da jedoch Probleme bei der Installation der benötigten R-Bibliotheken **statnet** und **igraph** auftraten, mussten die Tests auf einem privaten Rechner gestartet werden. Die im Originalartikel verwendete Netzwerkgröße von 750 Knoten und 3150 Kanten sowie 10000 Iterationen der Spiegelnetzwerk-Methode war daher für das Projekt aus Effizienzgründen nicht übertragbar. Damit sich die Testzeiten in einem einigermaßen erträglichen Rahmen von mehreren Stunden bewegen konnten, wurden die Netzwerke auf $n = 100$ Knoten sowie $m = \frac{3150}{750}n$ Kanten reduziert, wobei das durchschnittliche Kanten-pro-Knoten-Verhältnis gleich blieb. Darüberhinaus wurde die Anzahl der Iterationen auf 1000 gesenkt.

Diese Einschränkung birgt jedoch Probleme für die Interpretierbarkeit der Testergebnisse. Die Implementierungen der Modelle von Fowler et al. verwendeten bestimmte, im Sinne der Autoren optimale Modell-Parameter. Selbst wenn diese Parameter optimal sind für Netzwerke mit der oben beschriebenen Größe, so ist diese Optimalität für Netzwerke mit reduzierter Größe nicht mehr gewährleistet. Da jedoch die Herkunft der Werte für diese Parameter teilweise nicht nachzuvollziehen ist, wurden sie zumindest für einige Tests übernommen bzw. als Grundlage für weitere Tests verwendet. Trotz dieses Mankos wurde versucht, Tendenzen für die Verbesserbarkeit des Abschneidens der ERGM-Modelle bei der Spiegelnetzwerk-Methode festzustellen.

2 Verbesserungsansätze

Der Fokus der Untersuchungen liegt auf der Umsetzung des ERGM-Modells in der Spiegelnetzwerk-Methode. Diese wird von zwei Seiten beleuchtet: einerseits bezüglich der Aktualität der Spezifikationen und andererseits bezüglich der Fairness, mit der das Modell gegenüber dem A&I-Ansatz behandelt wurde.

Zunächst wurde der Originalcode sowohl für das A&I- als auch für das ERGM-Modell modifiziert, indem die Netzwerkgröße sowie die Anzahl der Iterationen wie oben beschrieben verringert wurden. Die Ergebnisse der Korrelationen beider Modelle dienen als Grundlage für die weiteren Tests. Dabei wird versucht, die Werte durch Modifikation des ERGM-Modells zu verbessern bzw. denen des A&I-Modells anzugleichen oder sie zu übertreffen.

2.1 Modellierung der Vererbung

Im A&I-Ansatz werden jedem Knoten zwei vererbbare Attribute mitgegeben. Das Attractiveness-Attribut gibt die Anziehungskraft eines Knotens auf die restlichen Knoten an. Je höher er ist, desto wahrscheinlicher gehen sie Verbindungen mit ihm ein. Das Introduce-Attribut reflektiert das Bestreben eines Knotens, befreundete Knoten einander vorzustellen, sodass sich zwischen ihnen Kanten bilden. Im Gegensatz dazu wird im ERGM-Modell nur ein Attribut für die Attractiveness vererbt, welches jedoch nicht gleich definiert ist wie für A&I. Daraus ergeben sich zwei Ansätze.

2.1.1 Introduce

Da im **statnet**-Paket bereits sehr viele Statistiken umgesetzt sind, wird versucht, mit ihrer Hilfe eine Art Introduce-Attribut zu modellieren, bevor eine neue, eventuell sehr aufwändige Statistik implementiert werden muss. Dazu bietet sich die ttriangle-Statistik an, die transitive Dreiecke zählt und der ein Knotenattribut übergeben werden kann. Wenn ein Attribut übergeben wird, zählt diese Statistik nur noch die transitiven Dreiecke, bei denen alle Knoten denselben Attributwert besitzen. Dies ist keine genaue Abbildung des Introduce-Attributs aus dem A&I-Modell, stellt aber eine einfache Möglichkeit dar, ein zweites Attribut an das Spiegelnetzwerk des ERGM-Modells zu übergeben. Die n Knoten erhalten für das Attribut zunächst die Werte von 1 bis n, damit sie paarweise verschie-

den sind. Dann wird wie im A&I-Modell für jeden Knoten ermittelt, ob ihm eine „Introduce“-Eigenschaft zugewiesen wird oder nicht. Ist dies für einen Knoten der Fall, erhält er den Wert 1 für das Attribut. Die ttriangle-Statistik zählt eine der unterschiedlichen Dreieckskonfigurationen, die von der triadcencus-Statistik gezählt werden. Daher wird letztere durch sie ersetzt und für den Anfang der Parameterwert übernommen.

2.1.2 Attractiveness

Das A&I-Modell enthält zwei Parameter $\alpha = 0.9$ und $\beta = 0.3$. Sie dienen der Zuweisung der Knotenattribute und ihre Werte werden zurückgeführt auf die Beobachtung, dass in den Netzwerken aus der Add-Health-Studie etwa 10% der Schüler einen Attractiveness-Wert besitzen, der nahe Null ist, während der Rest einer Gleichverteilung folgt. Außerdem sollen etwa 30% der Schüler die Eigenschaft besitzen, ihre Freunde einander vorzustellen. Im ERGM-Modell wird für jeden Knoten eine Zufallszahl gleichverteilt ermittelt und ihm als Attractiveness-Wert zugewiesen. Als Kovariate für dieses Attribut wird der Haupteffekt verwendet. Im A&I-Modell wird im Unterschied dazu jeder Wert unter 0.3 als Null angesehen. Dies auch für das ERGM-Modell zu übernehmen, ist ein erster Ansatz. Um die Auswirkungen der angenommenen Verteilung zu untersuchen, wird in einer weiteren Simulation statt der Gleich- eine Normalverteilung angenommen.

2.2 Parametrisierung

Das ursprüngliche ERGM-Modell von Fowler et al. enthält insgesamt sieben Statistiken (einen Haupteffekt für die Attractiveness und sechs Dreiecksstatistiken). Diese Statistiken werden jedoch nicht alle unterschiedlich parametrisiert. Während der Haupteffekt mit 2.5 parametrisiert wird, besitzen alle Dreiecksstatistiken denselben Parameterwert von 2.7. Wie die Autoren zu diesen Werten kommen, ist im Artikel nicht nachzuvollziehen. Dass die Parameter der Dreiecksstatistiken alle genau gleich sind, spricht für die Vermutung, dass diese Parameter nicht durch eine der gängigen Methoden für ERGM, wie etwa MCMC-Algorithmen zur ML-Schätzung [3], erhalten wurden. Daher ist ein weiterer Ansatz, durch das Auffinden alternativer Parameterwerte das Abschneiden des ERGM-Modells im Spiegelnetzwerk zu verbessern.

2.2.1 Einfache Suche

Der einfachste Ansatz zum Auffinden neuer Parameter wäre, jeweils einen der Werte festzuhalten und den bzw. die anderen mit einer bestimmten Schrittweite (etwa 0.1) für jeden Durchlauf im Spiegelnetzwerk zu verändern. Dadurch könnten evtl. auch Aussagen über den Einfluss bestimmter Wertebereiche der Parameter auf die Entartung des von Fowler et al. verwendeten ERGM-Modells gewonnen werden, welches im Prinzip nur ein Markov-Modell darstellt, das um eine Kovariate erweitert wurde. Da das Abtasten des Parameterraumes jedoch keine effiziente Methode darstellt und der folgende Ansatz eine gezieltere Möglichkeit bietet, auch für zusätzliche Statistiken einen geeigneteren Satz von Parameterwerten zu erhalten, wurde daher auf die Durchführung verzichtet.

2.2.2 Parameterschätzung

In den Tests werden neue Parameter mit Hilfe von ML-Schätzungen durch MCMC-Algorithmen gefunden, für die in der **ergm**-Bibliothek des **statnet**-Packages entsprechende Methoden implementiert sind. Diese Methoden benötigen ein konkretes Netzwerk und finden zu einem gegebenen Modell diejenigen Parameterwerte, welche die Wahrscheinlichkeit des Netzwerkes in diesem Modell maximieren (was nicht heißt, dass andere Netzwerke keine höhere Wahrscheinlichkeit besitzen können). Leider standen bis zum Abschluss des Projekts die Add-Health-Netzwerke nicht zur Verfügung. Man hätte nun irgendwelche Netzwerke, etwa Beispielnetzwerke des **statnet**-Pakets, fitten können, um alternative Parameterkombinationen zu erhalten. Sinnvoller erscheint es jedoch, Netzwerke mit der oben genannten Größe zu verwenden. Um ein solches zu erhalten, wurden Simulationen mit Hilfe des A&I-Modells durchgeführt, das in der Spiegelnetzwerk-Methode so gut abgeschnitten hat. Die Methode *simulateAttract(nodes=100)*, definiert in der Datei „attract.R“ (siehe Anhang), erzeugt mit A&I ein Netzwerk mit 100 Knoten. Auf diese Weise entstandene Netzwerke werden auch bei den nächsten Tests verwendet, bei denen alternative bzw. zusätzliche Statistiken ausprobiert werden, für die noch keine Ausgangsparameter existierten. Das **ergm**-Paket stellt eine Methode *ergm(network* $\sim$ *formula)* bereit, mit deren Hilfe man ein gegebenes Netzwerk (*network*) hinsichtlich eines Modells (gegeben durch *formula*, einer Kombination von Statistiken) fitten kann. In den Tests wird sowohl bei den Simulationen, als auch bei den Fits stets die Netzwerkgröße auf die gegebene Anzahl an Kanten beschränkt und bei den Fits die Verwendung von ML-Schätzmethoden

(statt PL) erzwungen.

2.3 Neue Statistiken

Unter anderem existieren Vorschläge von Snijders et al. [4] und Hunter [2] für Spezifiakationen von ERGM-Modellen, die sich besser für die Modellierung sozialer Netzwerke eignen, als die ursprünglichen Markov-Modelle. Diese alternativen Statistiken wurden von Fowler et al. in ihrer Umsetzung nicht berücksichtigt. Dort werden lediglich Dreiecke als einzige strukturelle Netzwerkeigenschaft betrachtet. Daher wird das ERGM-Modell um die neuen Statistiken erweitert. Genauer gesagt, werden die von Snijders et al. vorgeschlagenen Statistiken, d.h. die Alternierenden k-Sterne, die Alternierenden k-Dreiecke sowie die Alternierenden k-Zwei-Pfade bzw. ihre von Hunter formulierten Pendants verwendet, die im **statnet**-Paket implementiert sind. Dort heißen sie Geometrically Weighted Degrees (gwidegree und gwodegree), Geometrically Weighted Edgewise Shared Partners (gwesp) und Geometrically Weighted Dyadic Shared Partners (gwdsp). Die Parameter für diese Statistiken werden mittels der in Abschnitt 2.2.2 beschriebenen Methode geschätzt.

3 Durchführung

Die in Abschnitt 2 vorgestellten Ansätze wurden in verschiedenen Tests teilweise miteinander kombiniert. Im Folgenden werden die Tests näher erläutert und anschließend die Beobachtungen aufgeführt, die sich aus ihnen ergeben haben.

3.1 Die Tests

3.1.1 Ausgangstests

Als Grundlage für die Bewertung der Tests dient der Vergleich mit den Ergebnissen, welche die Simulationen des ursprünglichen ERGM-Modells und des A&I-Modells mit reduzierter Netzwerkgröße ergeben haben. Die Tests sollten zeigen, ob das Abschneiden der ERGM bei veränderter Spezifikation in der Spiegelnetzwerk-Methode besser abschneiden können, als das ursprüngliche Modell.

3.1.2 Andere Parameter

Das ursprüngliche ERGM-Modell wurde zunächst lediglich dahingehend verändert, dass die Parameter nicht fest gewählt, sondern wie in Abschnitt 2.2.2 beschrieben geschätzt wurden. Tabelle 1 zeigt die neuen Parameter, wobei auffällt, dass für die Dreiecksstatistiken völlig unterschiedliche Werte geschätzt wurden.

Statistik	**Parameterwert**
nodeicov.attract	-6.5656
triadcensus.030T	-0.2222
triadcensus.120D	2.0403
triadcensus.120U	0.3178
triadcensus.120C	-0.6335
triadcensus.210	1.4634
triadcensus.300	1.3315

Tabelle 1: Parameterschätzung für das ERGM-Modell von Fowler et al.

3.1.3 Vererbung

Wie in Abschnitt 2.1 festgehalten, wurden drei Tests durchgeführt, um die Vererbung alternativ zu modellieren. Im ersten Test wurde das Introduce-Attribut hin-

zugefügt, wobei die Dreiecksstatistiken ersetzt und die Berechnung des Attractiveness-Attributs beibehalten wurden. Im zweiten Test wurde das Modell aus dem ersten übernommen und die Attractiveness wie im A&I-Modell berechnet. Im dritten Test ebenso, nur wurde stattdessen die Attractiveness als normalverteilt angenommen.

3.1.4 Neue Statistiken

Die Statistiken von Snijders et al. (bzw Hunter) wurden zunächst in einem Test verwendet, ohne Kovariate zu berückstichtigen. Dies sollte zeigen, ob es bereits ausreicht, lediglich die empfohlenen Statistiken zu verwenden, um ein besseres Ergebnis zu erhalten. Im nächsten Test wurde anschließend das ursprüngliche Attractiveness-Attribut wieder hinzugefügt und in einem weiteren noch das Introduce-Attribut. Beim Hinzufügen des Introduce-Attributs wird die gwesp-Statistik durch die Dreiecksstatistik ttriangle ersetzt. Der Grund ist die Vermeidung zu vieler Statistiken, die dieselben Konfigurationen zählen. In allen drei Tests wurden die Parameter geschätzt, wobei dies auch die zusätzlichen Parameter mancher Statistiken mit einschließt, die z.B. die Abschwächung der geometrischen Gewichtung angeben (wie etwa gwesp.alpha und gwdsp.alpha). Tabelle 2 zeigt die geschätzten Werte:

Statistik	**Snijders o. Kov.**	**zus. Attr.**	**zus. Intr.**
nodeicov.attract	-	-2.7300197	-2.5107121
gwidegree	-3.7888527	-3.7717353	-4.4521364
gwidegree.decay	-0.0005352	0.0166413	0.0038171
gwodegree	-1.2852646	-0.0379860	-0.9885236
gwodegree.decay	0.0003672	0.0000693	0.0008575
ttriple.introduce	-	-	0.4212148
gwesp	0.4325034	0.8782464	-
gwesp.alpha	0.0021760	0.0023192	-
gwdsp	-0.4632346	-0.2395671	-0.2649714
gwdsp.alpha	-0.0077918	-0.0028124	0.0023786

Tabelle 2: Parameterschätzung für die neuen Statistiken

3.2 Ergebnisse

In Tabelle 3 werden die Ergebnisse aller oben angeführten Tests aufgelistet. Insgesamt zeigen viele der Tests Verbesserungen bei den Korrelationen der strukturellen Eigenschaften, die von der Spiegelnetzwerk-Methode als Vergleich herangezogen wurden. Die besten Ergebnisse beim Eingangsgrad (In-Deg.) und beim Ausgangsgrad (Out-Deg.) zeigt das Modell, bei dem lediglich das Introduce-Attribut hinzugefügt wurde. Die Betweenness-Zentralität (Betw.) erhält den höchsten Wert bei der Kombination der Statistiken von Snijders mit beiden Kovariaten. Dass hier auch gleichzeitig die Transitivität (Trans.) am schlechtesten abschneidet, liegt wahrscheinlich daran, dass die gwesp-Statistik ersetzt wurde. Das beste Ergebnis für die Transitivität wird erreicht durch die Normalverteilung des Attractiveness-Attributs. Generell ist festzustellen, dass sich die Transitivität bei allen Änderungen (bis auf die genannte Ausnahme) verbessert hat.

Modell	**In-Deg.**	**Out-Deg.**	**Trans.**	**Betw.**
A&I	0.5176	0.1252	0.3273	0.3319
ERGM Fowler	0.1643	0.0204	-0.1025	0.0006
ERGM Fowler geschätzt	0.0713	0.0270	0.3072	0.1031
ERGM Fowler + Introduce	0.7326	0.7073	0.1064	0.0261
ERGM Fowler + Introduce + Attr. von A&I	0.7073	0.6924	0.0038	0.0253
ERGM Fowler + Introduce + Attr. normalverteilt	0.7160	0.7070	0.7354	0.0172
ERGM Snijders	0.0404	0.0766	0.0240	0.0297
ERGM Snijders + Attractiveness	0.3152	-0.0108	0.1046	0.1408
ERGM Snijders + Attr. + Intr.	0.0336	0.0216	-0.2047	0.0444

Tabelle 3: Ergebnisse der Spiegelnetzwerk-Methode

4 Zusammenfassung und Ausblick

Die durchgeführten Tests haben gezeigt, dass durchaus eine Tendenz besteht, das von Fowler et al. verwendete ERGM-Modell hinsichtlich ihrer Vergleichsmethode auf vielerlei Arten verbessern zu können. Um die Ergebnisse zu bestätigen, müssten dieselben Tests noch einmal auf Netzwerken der ursprünglichen Größe durchgeführt werden. Dabei könnte auch versucht werden, eine Statistik zu finden, die das Introduce-Attribut korrekt abbildet. Dazu wäre wahrscheinlich eine Statistik ähnlich der gwesp geeignet, wobei die Bildung der Basiskante $\{i, j\}$ dieser Konfiguration davon abhängt, wie viele der direkten Nachbarn von i und j ein positives Introduce-Attribut erhalten haben.

Abgesehen davon, dass von Fowler et al. sehr wahrscheinlich nicht die bestmögliche ERGM-Spezifikation verwendet wurde, wäre ein weiterer Untersuchungsaspekt, die Vergleichsmethode selbst in Zweifel zu ziehen. Ob die Korrelationsmaße, die dort verwendet wurden, tatsächlich aussagekräftig sind bezüglich der ERGM-Modelle, lässt sich nur prüfen, wenn man die Gestalt der simulierten Netzwerke konkret betrachtet. Dabei spielt auch das bei den ERGM bekannte Problem der Entartung eine Rolle. Wenn etwa durch eine bestimmte Spezifikation des Modells fast nur vollständige Graphen simuliert werden, besitzen solche Korrelationsmaße keine Aussagekraft mehr.

A R-Code

A.1 Ausgangstests

```
# R code for attract and introduce model
# using a mirror network method to measure
# the extent to which this model generates
# heritability of network characteristics
#
# NOTE: Modified version of the original code
# of Fowler et al. with reduced network size.

rm(list=ls())

library(igraph) # Thanks to Gabor Csardi for the igraph package!

m<-3150/750 # average number of edges per node (degree)
n<-100 # number of nodes
e<-m*n # number of edges

alpha<-0.9
beta<-0.3

T<-1000 # number of simulations

twindeg1 <- rep(0, times=T)
twoutdeg1 <- rep(0, times=T)
twtr1 <- rep(0, times=T)
twbe1 <- rep(0, times=T)

twindeg2 <- rep(0, times=T)
twoutdeg2 <- rep(0, times=T)
twtr2 <- rep(0, times=T)
twbe2 <- rep(0, times=T)

for(t in 1:T) { # do simulation run

 #if(t%%100==0) print(t)
 print(t)

 # randomly choose one individual in each network to be a twin
 twin1 <- sample(n,1)
 twin2 <- sample(n,1)

 # create first set of individual traits
 d1<-as.numeric(runif(n)<alpha)*runif(n) # distribution of attractiveness
 c1<-as.numeric(runif(n)<beta) # distribution of prob. of introducing friends

 # create original network
 g1<-graph.empty(n) # create empty graph of size n
 while(ecount(g1)<e) { # loop until enough edges are produced
  dyad<-sample(n,2) # choose a random pair
  if(runif(1)<d1[dyad[2]]) { # check if person 1 nominates person 2 as a friend
   if(runif(1)<c1[dyad[1]]) { # check if person 1 introduces person 2 to friends
    ne<-neighbors(g1,dyad[1]-1,mode="out") # get person 1ís neighbors
    if(length(ne)>0) { # check if there is at least one existing neighbor

     for(i in 1:length(ne)) { # loop through neighbors
      if(runif(1)<d1[ne[i]+1]) g1<-add.edges(g1,c(dyad[2]-1,ne[i]))    # introduce friends
       if(runif(1)<d1[dyad[2]]) g1<-add.edges(g1,c(ne[i],dyad[2]-1))# and add edges if
                 # they become friends
    }
   }
  }
  g1<-add.edges(g1,dyad-1) # add random pair edge at end
 }
 g1<-simplify(g1) # remove duplicate edges and loops
}

 # create second set of individual traits
 d2<-as.numeric(runif(n)<alpha)*runif(n) # distribution of attractiveness
```

```
 cl2<-as.numeric(runif(n)<beta) # distribution of prob. of introducing friends

 # copy genes
 d2[twin2] <- d1[twin1]
 cl2[twin2] <- cl1[twin1]

 # create mirror network
 g2<-graph.empty(n)
 while(ecount(g2)<e) {

  dyad<-sample(n,2)
  if(runif(1)<d2[dyad[2]]) {
   if(runif(1)<cl2[dyad[1]]) {
    ne<-neighbors(g2,dyad[1]-1,mode="out")
    if(length(ne)>0) {
     for(i in 1:length(ne)) {
      if(runif(1)<d2[ne[i]+1]) g2<-add.edges(g2,c(dyad[2]-1,ne[i]))
       if(runif(1)<d2[dyad[2]]) g2<-add.edges(g2,c(ne[i],dyad[2]-1))
     }
    }
   }
   g2<-add.edges(g2,dyad-1)
  }
  g2<-simplify(g2)
 }

 # generate/store node statistics for original network
 twindeg1[t] <- degree(g1,v=twin1-1,mode="in")
 twoutdeg1[t] <- degree(g1,v=twin1-1,mode="out")
 twtr1[t] <- transitivity(g1,v=twin1-1,type="local")
 twbe1[t] <- betweenness(g1,v=twin1-1,directed=F)

 # generate/store node statistics for mirror network
 twindeg2[t] <- degree(g2,v=twin2-1,mode="in")
 twoutdeg2[t] <- degree(g2,v=twin2-1,mode="out")
 twtr2[t] <- transitivity(g2,v=twin2-1,type="local")
 twbe2[t] <- betweenness(g2,v=twin2-1,directed=F)
}

corindeg <- cor.test(twindeg1, twindeg2)
coroutdeg <- cor.test(twoutdeg1, twoutdeg2)
cortr <- cor.test(twtr1, twtr2, use="complete.obs")
corbe <- cor.test(twbe1, twbe2)

# Results obtained after 10000 simulations:
corindeg # 0.46 (0.44,0.48)
coroutdeg # 0.12 (0.10,0.14)
cortr # 0.48 (0.46,0.50)
corbe # 0.29 (0.27,0.31)
```

```
# R code for ERGM
# using a mirror network method to measure
# the extent to which this model generates
# heritability of network characteristics
#
# NOTE: Modified version of the original code
# of Fowler et al. with reduced network size.

rm(list=ls())

library(ergm)
library(network)

m<-3150/750 # average number of edges per node (degree)
n<-100  # number of nodes
e<-m*n # number of edges

T<-1000 # number of simulations

twindeg1 <- rep(0, times=T)
twoutdeg1 <- rep(0, times=T)
twtr1 <- rep(0, times=T)
twbe1 <- rep(0, times=T)

twindeg2 <- rep(0, times=T)
twoutdeg2 <- rep(0, times=T)
twtr2 <- rep(0, times=T)
twbe2 <- rep(0, times=T)

for(t in 1:T) { # do simulation run

 #if(t%%100==0) print(t)
 print(t)

 # randomly choose one individual in each network to be a twin
 twin1 <- sample(n,1)
 twin2 <- sample(n,1)

 # create first set of individual traits
 d1<-runif(n) # distribution of attractiveness

 # create original network
 g1.use <- network(as.matrix(cbind(sample(n,e,replace=T),sample(n,e,replace=T))))
 g1.use %v% "attract" <- d1 # assign intrinsic characteristic to each node
 g1 <- simulate(~ nodeicov("attract")+ # in degree proportional to intrinsic characteristics
  triadcensus(c(8,11:15)), # count number of transitive triplets
  theta0=c(2.5,rep(2.7,6)), # coefficients 2.5 (for in-degree) and 2.7 (for triplets)
  constraints = ~ edges, # constrain the network to yield e edges
  basis=g1.use, # use basis network to get total number of edges and nodes
  burnin=100000) # let Monte Carlo Chain run a long time before sampling a network

 # create second set of individual traits
 d2<-runif(n) # distribution of attractiveness

 # copy genes to one individual in new network
 d2[twin2] <- d1[twin1]

 # create mirror network
 g2.use <- network(as.matrix(cbind(sample(n,e,replace=T),sample(n,e,replace=T))))
 g2.use %v% "attract" <- d2 # assign intrinsic characteristic to each node
 g2 <- simulate(~ nodeicov("attract")+ # in degree proportional to intrinsic characteristics
  triadcensus(c(8,11:15)), # count number of transitive triplets
  theta0=c(2.5,rep(2.7,6)), # coefficients 2.5 (for in-degree) and 2.7 (for triplets)
  constraints = ~ edges, # constrain the network to yield e edges
  basis=g2.use, # use basis network to get total number of edges and nodes
  burnin=100000) # let Monte Carlo Chain run a long time before sampling a network

 # convert networks to igraph graph objects
 m1 = g1[,]
 m2 = g2[,]

 require(igraph)
```

```
 g1i <- graph.adjacency(m1)
 g2i <- graph.adjacency(m2)

 # generate/store node statistics for original network
 twindeg1[t] <- degree(g1i,v=twin1-1,mode="in")
 twoutdeg1[t] <- degree(g1i,v=twin1-1,mode="out")
 twtr1[t] <- transitivity(g1i,v=twin1-1,type="local")
 twbe1[t] <- betweenness(g1i,v=twin1-1,directed=F)

 # generate/store node statistics for mirror network
 twindeg2[t] <- degree(g2i,v=twin2-1,mode="in")
 twoutdeg2[t] <- degree(g2i,v=twin2-1,mode="out")
 twtr2[t] <- transitivity(g2i,v=twin2-1,type="local")
 twbe2[t] <- betweenness(g2i,v=twin2-1,directed=F)

 detach(package:igraph)
}

corindeg <- cor.test(twindeg1, twindeg2)
coroutdeg <- cor.test(twoutdeg1, twoutdeg2)
cortr <- cor.test(twtr1, twtr2, use="complete.obs")
corbe <- cor.test(twbe1, twbe2)

# Results obtained after 1000 simulations:
corindeg # 0.50 (0.48,0.52)
coroutdeg # 0.12 (0.10,0.14)
cortr # 0.02 (0.00,0.04)
corbe # 0.34 (0.32,0.36)
```

A.2 Andere Parameter

```
# R code for ERGM
# using a mirror network method to measure
# the extent to which this model generates
# heritability of network characteristics
#
# NOTE: Modified version of the original code
# of Fowler et al. with reduced network size
# and parameters estimated by fitting a network
# simulated with the attract and introduce model.


rm(list=ls())

# Fit the attract&introduce model
source("attract.R")      # Load attract.R to simulate Attract & Introduce model
attr <- simulateAttract(nodes=100)   # Simulate Attract & Introduce with 100 nodes
library(ergm)
model<- ergm(attr ~ nodeicov("attract") +
  triadcensus(c(8,11:15)),
  constraints= ~ edges, # constrain on number of edges in the network
  MLestimate=TRUE)  # use MCMC for ML estimation

library(network)

m<-3150/750 # average number of edges per node (degree)
n<-100 # number of nodes
e<-m*n # number of edges

T<-1000 # number of simulations

twindeg1 <- rep(0, times=T)
twoutdeg1 <- rep(0, times=T)
twtr1 <- rep(0, times=T)
twbe1 <- rep(0, times=T)

twindeg2 <- rep(0, times=T)
twoutdeg2 <- rep(0, times=T)
twtr2 <- rep(0, times=T)
twbe2 <- rep(0, times=T)

for(t in 1:T) { # do simulation run

 #if(t%%100==0) print(t)
 print(t)

 # randomly choose one individual in each network to be a twin
 twin1 <- sample(n,1)
 twin2 <- sample(n,1)

 # create first set of individual traits
 d1<-runif(n) # distribution of attractiveness

 # create original network
 g1.use <- network(as.matrix(cbind(sample(n,e,replace=T),sample(n,e,replace=T))))
 g1.use %v% "attract" <- d1 # assign intrinsic characteristic to each node
 g1 <- simulate(model,
  constraints = ~ edges, # constrain the network to yield e edges
  basis=g1.use, # use basis network to get total number of edges and nodes
  burnin=100000) # let Monte Carlo Chain run a long time before sampling a network

 # create second set of individual traits
 d2<-runif(n) # distribution of attractiveness

 # copy genes to one individual in new network
 d2[twin2] <- d1[twin1]

 # create mirror network
 g2.use <- network(as.matrix(cbind(sample(n,e,replace=T),sample(n,e,replace=T))))
 g2.use %v% "attract" <- d2 # assign intrinsic characteristic to each node
 g2 <- simulate(model,
  constraints = ~ edges, # constrain the network to yield e edges
```

```
  basis=g2.use, # use basis network to get total number of edges and nodes
  burnin=100000) # let Monte Carlo Chain run a long time before sampling a network

 # convert networks to igraph graph objects
 m1 = g1[,]
 m2 = g2[,]

 require(igraph)

 g1i <- graph.adjacency(m1)
 g2i <- graph.adjacency(m2)

 # generate/store node statistics for original network
 twindeg1[t] <- degree(g1i,v=twin1-1,mode="in")
 twoutdeg1[t] <- degree(g1i,v=twin1-1,mode="out")
 twtr1[t] <- transitivity(g1i,v=twin1-1,type="local")
 twbe1[t] <- betweenness(g1i,v=twin1-1,directed=F)

 # generate/store node statistics for mirror network
 twindeg2[t] <- degree(g2i,v=twin2-1,mode="in")
 twoutdeg2[t] <- degree(g2i,v=twin2-1,mode="out")
 twtr2[t] <- transitivity(g2i,v=twin2-1,type="local")
 twbe2[t] <- betweenness(g2i,v=twin2-1,directed=F)

 detach(package:igraph)
}

corindeg <- cor.test(twindeg1, twindeg2)
coroutdeg <- cor.test(twoutdeg1, twoutdeg2)
cortr <- cor.test(twtr1, twtr2, use="complete.obs")
corbe <- cor.test(twbe1, twbe2)

# Results obtained after 1000 simulations:
corindeg # 0.50 (0.48,0.52)
coroutdeg # 0.12 (0.10,0.14)
cortr # 0.02 (0.00,0.04)
corbe # 0.34 (0.32,0.36)
```

```
# Function simulateAttract that
# simulates a network of the given
# network size with the attract
# and introduce model.

simulateAttract <- function(nodes, m=3150/750, alpha=0.9, beta=0.3, directed=TRUE)
{
 library(igraph)  # igraph packge needed for working with graphs

 n<-nodes  # number of nodes
 e<-m*n # number of edges, depends on n and m (the number of edges per node)
 a<-alpha
 b<-beta

 # create first set of individual traits
 d1<-as.numeric(runif(n)<a)*runif(n) # distribution of attractiveness
 cl1<-as.numeric(runif(n)<b)   # distribution of prob. of introducing friends

 # create original network
 g1<-graph.empty(n)  # create empty graph of size n
 while(ecount(g1)<e)  # loop until enough edges are produced
 {
  dyad<-sample(n,2) # choose a random pair

  if(runif(1)<d1[dyad[2]])  # check if person 1 nominates person 2 as a friend
  {
   if(runif(1)<cl1[dyad[1]])   # check if person 1 introduces person 2 to friends
   {
    ne<-neighbors(g1,dyad[1]-1,mode="out")  # get person 1ís neighbors
    if(length(ne)>0)   # check if there is at least one existing neighbor
    {
     for(i in 1:length(ne))  # loop through neighbors
     {
      if(runif(1)<d1[ne[i]+1]) g1<-add.edges(g1,c(dyad[2]-1,ne[i]))  # introduce friends
      if(runif(1)<d1[dyad[2]]) g1<-add.edges(g1,c(ne[i],dyad[2]-1))  # and add edges if
                # they become friends
     }
    }
   }
   g1<-add.edges(g1,dyad-1) # add random pair edge at end
  }
  g1<-simplify(g1) # remove duplicate edges and loops
 }


 # Convert the igraph object to a network object via the adjacency matrix
 library(network) # load network package at this time, otherwise add.edges function may cause problems
 m1<-get.adjacency(g1)
 net<-network(m1,directed=directed)

 # Store attract and introduce as node attributes
 net %v% "attract"<- d1
 net %v% "introduce"<-cl1

 # Unload the igraph package
 detach(package:igraph)

 # Return the network object
 net
}
```

A.3 Vererbung

```
# R code for ERGM
# using a mirror network method to measure
# the extent to which this model generates
# heritability of network characteristics
#
# NOTE: Modified version of the original code
# of Fowler et al. with reduced network size
# and a new statistic replacing the triandcencus
# statistic and including a new introduce node
# attribute

rm(list=ls())

library(ergm)
library(network)

m<-3150/750 # average number of edges per node (degree)
n<-100 # number of nodes
e<-m*n # number of edges
alpha<-0.9
beta<-0.3

T<-1000 # number of simulations

twindeg1 <- rep(0, times=T)
twoutdeg1 <- rep(0, times=T)
twtr1 <- rep(0, times=T)
twbe1 <- rep(0, times=T)

twindeg2 <- rep(0, times=T)
twoutdeg2 <- rep(0, times=T)
twtr2 <- rep(0, times=T)
twbe2 <- rep(0, times=T)

for(t in 1:T) { # do simulation run

 #if(t%%100==0) print(t)
 print(t)

 # randomly choose one individual in each network to be a twin
 twin1 <- sample(n,1)
 twin2 <- sample(n,1)

 # create first set of individual traits
 d1<-runif(n) # distribution of attractiveness
 cl1<-2:(n+1) # introduce parameter
 cl1<-ifelse(as.numeric(runif(n)<beta)==1,1,cl1)

 # create original network
 g1.use <- network(as.matrix(cbind(sample(n,e,replace=T),sample(n,e,replace=T))))
 g1.use %v% "attract" <- d1 # assign intrinsic characteristic to each node
 g1.use %v% "introduce" <- cl1
 g1 <- simulate(~ nodeicov("attract")+ # in degree proportional to intrinsic characteristics
  ttriple(attrname="introduce"), # transitive triplets with an "introduce"-parameter
  theta0=c(2.5, 2.7), # coefficients 2.5 (for in-degree) and 2.7 (for triplets)
  constraints = ~ edges, # constrain the network to yield e edges
  basis=g1.use, # use basis network to get total number of edges and nodes
  burnin=100000) # let Monte Carlo Chain run a long time before sampling a network

 # create second set of individual traits
 d2<-runif(n) # distribution of attractiveness
 cl2<-2:(n+1) # introduce parameter
 cl2<-ifelse(as.numeric(runif(n)<beta)==1,1,cl2)

 # copy genes to one individual in new network
 d2[twin2] <- d1[twin1]
 cl2[twin2]<- cl1[twin1]

 # create mirror network
 g2.use <- network(as.matrix(cbind(sample(n,e,replace=T),sample(n,e,replace=T))))
 g2.use %v% "attract" <- d2 # assign intrinsic characteristic to each node
```

```
 g2.use %v% "introduce" <- cl2
 g2 <- simulate(~ nodeicov("attract")+ # in degree proportional to intrinsic characteristics
  ttriple(attrname="introduce"), # transitive triplets with an "introduce"-parameter
  theta0=c(2.5, 2.7), # coefficients 2.5 (for in-degree) and 2.7 (for triplets)
  constraints = ~ edges, # constrain the network to yield e edges
  basis=g2.use, # use basis network to get total number of edges and nodes
  burnin=100000) # let Monte Carlo Chain run a long time before sampling a network

 # convert networks to igraph graph objects
 m1 = g1[,]
 m2 = g2[,]

 require(igraph)

 g1i <- graph.adjacency(m1)
 g2i <- graph.adjacency(m2)

 # generate/store node statistics for original network
 twindeg1[t] <- degree(g1i,v=twin1-1,mode="in")
 twoutdeg1[t] <- degree(g1i,v=twin1-1,mode="out")
 twtr1[t] <- transitivity(g1i,v=twin1-1,type="local")
 twbe1[t] <- betweenness(g1i,v=twin1-1,directed=F)

 # generate/store node statistics for mirror network
 twindeg2[t] <- degree(g2i,v=twin2-1,mode="in")
 twoutdeg2[t] <- degree(g2i,v=twin2-1,mode="out")
 twtr2[t] <- transitivity(g2i,v=twin2-1,type="local")
 twbe2[t] <- betweenness(g2i,v=twin2-1,directed=F)

 detach(package:igraph)
}

corindeg <- cor.test(twindeg1, twindeg2)
coroutdeg <- cor.test(twoutdeg1, twoutdeg2)
cortr <- cor.test(twtr1, twtr2, use="complete.obs")
corbe <- cor.test(twbe1, twbe2)

# Results obtained after 1000 simulations:
corindeg # 0.50 (0.48,0.52)
coroutdeg # 0.12 (0.10,0.14)
cortr # 0.02 (0.00,0.04)
corbe # 0.34 (0.32,0.36)
```

```
# R code for ERGM
# using a mirror network method to measure
# the extent to which this model generates
# heritability of network characteristics
#
# NOTE: Modified version of the original code
# of Fowler et al. with reduced network size
# and a new statistic replacing the triandcencus
# statistic and including a new introduce node
# attribute. The attractiveness attribute is
# defined as for the attract and introduce model.

rm(list=ls())

library(ergm)
library(network)

m<-3150/750 # average number of edges per node (degree)
n<-100 # number of nodes
e<-m*n # number of edges
alpha<-0.9
beta<-0.3

T<-1000 # number of simulations

twindeg1 <- rep(0, times=T)
twoutdeg1 <- rep(0, times=T)
twtr1 <- rep(0, times=T)
twbe1 <- rep(0, times=T)

twindeg2 <- rep(0, times=T)
twoutdeg2 <- rep(0, times=T)
twtr2 <- rep(0, times=T)
twbe2 <- rep(0, times=T)

for(t in 1:T) { # do simulation run

 #if(t%%100==0) print(t)
 print(t)

 # randomly choose one individual in each network to be a twin
 twin1 <- sample(n,1)
 twin2 <- sample(n,1)

 # create first set of individual traits
 d1<-as.numeric(runif(n)<alpha)*runif(n)  # distribution of attractiveness
 cl1<-2:(n+1) # introduce parameter
 cl1<-ifelse(as.numeric(runif(n)<beta)==1,1,cl1)

 # create original network
 g1.use <- network(as.matrix(cbind(sample(n,e,replace=T),sample(n,e,replace=T))))
 g1.use %v% "attract" <- d1 # assign intrinsic characteristic to each node
 g1.use %v% "introduce" <- cl1
 g1 <- simulate(~ nodeicov("attract")+ # in degree proportional to intrinsic characteristics
  ttriple(attrname="introduce"), # transitive triplets with an "introduce"-parameter
  theta0=c(2.5, 2.7), # coefficients 2.5 (for in-degree) and 2.7 (for triplets)
  constraints = ~ edges, # constrain the network to yield e edges
  basis=g1.use, # use basis network to get total number of edges and nodes
  burnin=100000) # let Monte Carlo Chain run a long time before sampling a network

 # create second set of individual traits
 d2<-as.numeric(runif(n)<alpha)*runif(n)  # distribution of attractiveness
 cl2<-2:(n+1) # introduce parameter
 cl2<-ifelse(as.numeric(runif(n)<beta)==1,1,cl2)

 # copy genes to one individual in new network
 d2[twin2] <- d1[twin1]
 cl2[twin2]<- cl1[twin1]

 # create mirror network
 g2.use <- network(as.matrix(cbind(sample(n,e,replace=T),sample(n,e,replace=T))))
 g2.use %v% "attract" <- d2 # assign intrinsic characteristic to each node
 g2.use %v% "introduce" <- cl2
```

```
 g2 <- simulate(~ nodeicov("attract")+ # in degree proportional to intrinsic characteristics
  ttriple(attrname="introduce"), # transitive triplets with an "introduce"-parameter
  theta0=c(2.5, 2.7), # coefficients 2.5 (for in-degree) and 2.7 (for triplets)
  constraints = ~ edges, # constrain the network to yield e edges
  basis=g2.use, # use basis network to get total number of edges and nodes
  burnin=100000) # let Monte Carlo Chain run a long time before sampling a network

 # convert networks to igraph graph objects
 m1 = g1[,]
 m2 = g2[,]

 require(igraph)

 g1i <- graph.adjacency(m1)
 g2i <- graph.adjacency(m2)

 # generate/store node statistics for original network
 twindeg1[t] <- degree(g1i,v=twin1-1,mode="in")
 twoutdeg1[t] <- degree(g1i,v=twin1-1,mode="out")
 twtr1[t] <- transitivity(g1i,v=twin1-1,type="local")
 twbe1[t] <- betweenness(g1i,v=twin1-1,directed=F)

 # generate/store node statistics for mirror network
 twindeg2[t] <- degree(g2i,v=twin2-1,mode="in")
 twoutdeg2[t] <- degree(g2i,v=twin2-1,mode="out")
 twtr2[t] <- transitivity(g2i,v=twin2-1,type="local")
 twbe2[t] <- betweenness(g2i,v=twin2-1,directed=F)

 detach(package:igraph)
}

corindeg <- cor.test(twindeg1, twindeg2)
coroutdeg <- cor.test(twoutdeg1, twoutdeg2)
cortr <- cor.test(twtr1, twtr2, use="complete.obs")
corbe <- cor.test(twbe1, twbe2)

# Results obtained after 1000 simulations:
corindeg # 0.50 (0.48,0.52)
coroutdeg # 0.12 (0.10,0.14)
cortr # 0.02 (0.00,0.04)
corbe # 0.34 (0.32,0.36)
```

```
# R code for ERGM
# using a mirror network method to measure
# the extent to which this model generates
# heritability of network characteristics
#
# NOTE: Modified version of the original code
# of Fowler et al. with reduced network size
# and a new statistic replacing the triandcencus
# statistic and including a new introduce node
# attribute. The attractiveness attribute now
# follows a normal distribution.

rm(list=ls())

library(ergm)
library(network)

m<-3150/750 # average number of edges per node (degree)
n<-100 # number of nodes
e<-m*n # number of edges
alpha<-0.9
beta<-0.3

T<-1000 # number of simulations

twindeg1 <- rep(0, times=T)
twoutdeg1 <- rep(0, times=T)
twtr1 <- rep(0, times=T)
twbe1 <- rep(0, times=T)

twindeg2 <- rep(0, times=T)
twoutdeg2 <- rep(0, times=T)
twtr2 <- rep(0, times=T)
twbe2 <- rep(0, times=T)

for(t in 1:T) { # do simulation run

 #if(t%%100==0) print(t)
 print(t)

 # randomly choose one individual in each network to be a twin
 twin1 <- sample(n,1)
 twin2 <- sample(n,1)

 # create first set of individual traits
 d1<-rnorm(n)  # normal distribution of attractiveness
 cl1<-2:(n+1) # introduce parameter
 cl1<-ifelse(as.numeric(runif(n)<beta)==1,1,cl1)

 # create original network
 g1.use <- network(as.matrix(cbind(sample(n,e,replace=T),sample(n,e,replace=T))))
 g1.use %v% "attract" <- d1 # assign intrinsic characteristic to each node
 g1.use %v% "introduce" <- cl1
 g1 <- simulate(~ nodeicov("attract")+ # in degree proportional to intrinsic characteristics
  ttriple(attrname="introduce"), # transitive triplets with an "introduce"-parameter
  theta0=c(2.5, 2.7), # coefficients 2.5 (for in-degree) and 2.7 (for triplets)
  constraints = ~ edges, # constrain the network to yield e edges
  basis=g1.use, # use basis network to get total number of edges and nodes
  burnin=100000) # let Monte Carlo Chain run a long time before sampling a network

 # create second set of individual traits
 d2<-rnorm(n)  # distribution of attractiveness normal distribution
 cl2<-2:(n+1) # introduce parameter
 cl2<-ifelse(as.numeric(runif(n)<beta)==1,1,cl2)

 # copy genes to one individual in new network
 d2[twin2] <- d1[twin1]
 cl2[twin2]<- cl1[twin1]

 # create mirror network
 g2.use <- network(as.matrix(cbind(sample(n,e,replace=T),sample(n,e,replace=T))))
 g2.use %v% "attract" <- d2 # assign intrinsic characteristic to each node
 g2.use %v% "introduce" <- cl2
```

```
 g2 <- simulate(~ nodeicov("attract")+ # in degree proportional to intrinsic characteristics
  ttriple(attrname="introduce"), # transitive triplets with an "introduce"-parameter
  theta0=c(2.5, 2.7), # coefficients 2.5 (for in-degree) and 2.7 (for triplets)
  constraints = ~ edges, # constrain the network to yield e edges
  basis=g2.use, # use basis network to get total number of edges and nodes
  burnin=100000) # let Monte Carlo Chain run a long time before sampling a network

 # convert networks to igraph graph objects
 m1 = g1[,]
 m2 = g2[,]

 require(igraph)

 g1i <- graph.adjacency(m1)
 g2i <- graph.adjacency(m2)

 # generate/store node statistics for original network
 twindeg1[t] <- degree(g1i,v=twin1-1,mode="in")
 twoutdeg1[t] <- degree(g1i,v=twin1-1,mode="out")
 twtr1[t] <- transitivity(g1i,v=twin1-1,type="local")
 twbe1[t] <- betweenness(g1i,v=twin1-1,directed=F)

 # generate/store node statistics for mirror network
 twindeg2[t] <- degree(g2i,v=twin2-1,mode="in")
 twoutdeg2[t] <- degree(g2i,v=twin2-1,mode="out")
 twtr2[t] <- transitivity(g2i,v=twin2-1,type="local")
 twbe2[t] <- betweenness(g2i,v=twin2-1,directed=F)

 detach(package:igraph)
}

corindeg <- cor.test(twindeg1, twindeg2)
coroutdeg <- cor.test(twoutdeg1, twoutdeg2)
cortr <- cor.test(twtr1, twtr2, use="complete.obs")
corbe <- cor.test(twbe1, twbe2)

# Results obtained after 1000 simulations:
corindeg # 0.50 (0.48,0.52)
coroutdeg # 0.12 (0.10,0.14)
cortr # 0.02 (0.00,0.04)
corbe # 0.34 (0.32,0.36)
```

A.4 Neue Statistiken

```
# R code for ERGM
# using a mirror network method to measure
# the extent to which this model generates
# heritability of network characteristics
#
# NOTE: Modified version of the original code
# of Fowler et al. with reduced network size
# and parameters estimated by fitting a network
# simulated with the attract and introduce model.
# The statistics are the ones recommended by
# Snijders et al. respectively Hunter et al.
# without covariates.

rm(list=ls())

# Fit the attract&introduce model
source("attract.R")      # Load attract.R to simulate Attract & Introduce model
attr <- simulateAttract(nodes=100)   # Simulate Attract & Introduce with 100 nodes
library(ergm)
model<- ergm(attr ~ gwidegree(fixed=FALSE) +
  gwodegree(fixed=FALSE) +
  gwesp +
  gwdsp,
  constraints= ~ edges, # constrain on number of edges in the network
  MLestimate=TRUE)  # use MCMC for ML estimation

library(network)

#n<-750 # number of nodes
#e<-3150 # number of edges
m<-3150/750 # average number of edges per node (degree)
n<-100
e<-m*n

#T<-10000 # number of simulations
T<-1000

twindeg1 <- rep(0, times=T)
twoutdeg1 <- rep(0, times=T)
twtr1 <- rep(0, times=T)
twbe1 <- rep(0, times=T)

twindeg2 <- rep(0, times=T)
twoutdeg2 <- rep(0, times=T)
twtr2 <- rep(0, times=T)
twbe2 <- rep(0, times=T)


for(t in 1:T) { # do simulation run

 #if(t%%100==0) print(t)
 print(t)

 # randomly choose one individual in each network to be a twin
 twin1 <- sample(n,1)
 twin2 <- sample(n,1)

 # create first set of individual traits
 d1<-runif(n) # distribution of attractiveness

 # create original network
 g1.use <- network(as.matrix(cbind(sample(n,e,replace=T),sample(n,e,replace=T))))
 g1.use %v% "attract" <- d1 # assign intrinsic characteristic to each node
 g1 <- simulate(model,
  constraints = ~ edges, # constrain the network to yield e edges
  basis=g1.use, # use basis network to get total number of edges and nodes
  burnin=100000) # let Monte Carlo Chain run a long time before sampling a network

 # create second set of individual traits
 d2<-runif(n) # distribution of attractiveness
```

```
 # copy genes to one individual in new network
 d2[twin2] <- d1[twin1]

 # create mirror network
 g2.use <- network(as.matrix(cbind(sample(n,e,replace=T),sample(n,e,replace=T))))
 g2.use %v% "attract" <- d2 # assign intrinsic characteristic to each node
 g2 <- simulate(model,
  constraints = ~ edges, # constrain the network to yield e edges
  basis=g2.use, # use basis network to get total number of edges and nodes
  burnin=100000) # let Monte Carlo Chain run a long time before sampling a network

 # convert networks to igraph graph objects
 m1 = g1[,]
 m2 = g2[,]

 require(igraph)

 g1i <- graph.adjacency(m1)
 g2i <- graph.adjacency(m2)

 # generate/store node statistics for original network
 twindeg1[t] <- degree(g1i,v=twin1-1,mode="in")
 twoutdeg1[t] <- degree(g1i,v=twin1-1,mode="out")
 twtr1[t] <- transitivity(g1i,v=twin1-1,type="local")
 twbe1[t] <- betweenness(g1i,v=twin1-1,directed=F)

 # generate/store node statistics for mirror network
 twindeg2[t] <- degree(g2i,v=twin2-1,mode="in")
 twoutdeg2[t] <- degree(g2i,v=twin2-1,mode="out")
 twtr2[t] <- transitivity(g2i,v=twin2-1,type="local")
 twbe2[t] <- betweenness(g2i,v=twin2-1,directed=F)

 detach(package:igraph)
}

corindeg <- cor.test(twindeg1, twindeg2)
coroutdeg <- cor.test(twoutdeg1, twoutdeg2)
cortr <- cor.test(twtr1, twtr2, use="complete.obs")
corbe <- cor.test(twbe1, twbe2)

# Results obtained after 10000 simulations:
corindeg # 0.50 (0.48,0.52)
coroutdeg # 0.12 (0.10,0.14)
cortr # 0.02 (0.00,0.04)
corbe # 0.34 (0.32,0.36)

# The model parameters
model
```

```
# R code for ERGM
# using a mirror network method to measure
# the extent to which this model generates
# heritability of network characteristics
#
# NOTE: Modified version of the original code
# of Fowler et al. with reduced network size
# and parameters estimated by fitting a network
# simulated with the attract and introduce model.
# The statistics are the ones recommended by
# Snijders et al. respectively Hunter et al.
# with attractiveness covariate.

rm(list=ls())

# Fit the attract&introduce model
source("attract.R")      # Load attract.R to simulate Attract & Introduce model
attr <- simulateAttract(nodes=100)   # Simulate Attract & Introduce with 100 nodes
library(ergm)
model<- ergm(attr ~ nodeicov("attract") +
  gwidegree(fixed=FALSE) +
  gwodegree(fixed=FALSE) +
  gwesp +
  gwdsp,
  constraints= ~ edges, # constrain on number of edges in the network
  MLestimate=TRUE)  # use MCMC for ML estimation

library(network)

m<-3150/750 # average number of edges per node (degree)
n<-100 # number of nodes
e<-m*n # number of edges
alpha<-0.9
beta<-0.3

T<-1000 # number of simulations

twindeg1 <- rep(0, times=T)
twoutdeg1 <- rep(0, times=T)
twtr1 <- rep(0, times=T)
twbe1 <- rep(0, times=T)

twindeg2 <- rep(0, times=T)
twoutdeg2 <- rep(0, times=T)
twtr2 <- rep(0, times=T)
twbe2 <- rep(0, times=T)


for(t in 1:T) { # do simulation run

 #if(t%%100==0) print(t)
 print(t)

 # randomly choose one individual in each network to be a twin
 twin1 <- sample(n,1)
 twin2 <- sample(n,1)

 # create first set of individual traits
 d1<-runif(n) # distribution of attractiveness
 cl1<-2:(n+1) # introduce parameter
 cl1<-ifelse(as.numeric(runif(n)<beta)==1,1,cl1)

 # create original network
 g1.use <- network(as.matrix(cbind(sample(n,e,replace=T),sample(n,e,replace=T))))
 g1.use %v% "attract" <- d1 # assign intrinsic characteristic to each node
 g1.use %v% "introduce" <- cl1
 g1 <- simulate(model,
  constraints = ~ edges, # constrain the network to yield e edges
  basis=g1.use, # use basis network to get total number of edges and nodes
  burnin=100000) # let Monte Carlo Chain run a long time before sampling a network

 # create second set of individual traits
 d2<-runif(n) # distribution of attractiveness
```

```
 cl2<-2:(n+1) # introduce parameter
 cl2<-ifelse(as.numeric(runif(n)<beta)==1,1,cl2)

 # copy genes to one individual in new network
 d2[twin2] <- d1[twin1]
 cl2[twin2]<- cl1[twin1]

 # create mirror network
 g2.use <- network(as.matrix(cbind(sample(n,e,replace=T),sample(n,e,replace=T))))
 g2.use %v% "attract" <- d2 # assign intrinsic characteristic to each node
 g2.use %v% "introduce" <- cl2
 g2 <- simulate(model,
  constraints = ~ edges, # constrain the network to yield e edges
  basis=g2.use, # use basis network to get total number of edges and nodes
  burnin=100000) # let Monte Carlo Chain run a long time before sampling a network

 # convert networks to igraph graph objects
 m1 = g1[,]
 m2 = g2[,]

 require(igraph)

 g1i <- graph.adjacency(m1)
 g2i <- graph.adjacency(m2)

 # generate/store node statistics for original network
 twindeg1[t] <- degree(g1i,v=twin1-1,mode="in")
 twoutdeg1[t] <- degree(g1i,v=twin1-1,mode="out")
 twtr1[t] <- transitivity(g1i,v=twin1-1,type="local")
 twbe1[t] <- betweenness(g1i,v=twin1-1,directed=F)

 # generate/store node statistics for mirror network
 twindeg2[t] <- degree(g2i,v=twin2-1,mode="in")
 twoutdeg2[t] <- degree(g2i,v=twin2-1,mode="out")
 twtr2[t] <- transitivity(g2i,v=twin2-1,type="local")
 twbe2[t] <- betweenness(g2i,v=twin2-1,directed=F)

 detach(package:igraph)
}

corindeg <- cor.test(twindeg1, twindeg2)
coroutdeg <- cor.test(twoutdeg1, twoutdeg2)
cortr <- cor.test(twtr1, twtr2, use="complete.obs")
corbe <- cor.test(twbe1, twbe2)

# Results obtained after 1000 simulations:
corindeg # 0.50 (0.48,0.52)
coroutdeg # 0.12 (0.10,0.14)
cortr # 0.02 (0.00,0.04)
corbe # 0.34 (0.32,0.36)

# The model parameters
model
```

```
# R code for ERGM
# using a mirror network method to measure
# the extent to which this model generates
# heritability of network characteristics
#
# NOTE: Modified version of the original code
# of Fowler et al. with reduced network size
# and parameters estimated by fitting a network
# simulated with the attract and introduce model.
# The statistics are the ones recommended by
# Snijders et al. respectively Hunter et al.
# with attractiveness and introduce covariate
# where ttriangle statistic replaces gwesp
# statistic.

rm(list=ls())

# Fit the attract&introduce model
source("attract.R")      # Load attract.R to simulate Attract & Introduce model
attr <- simulateAttract(nodes=100)   # Simulate Attract & Introduce with 100 nodes
library(ergm)
model<- ergm(attr ~ nodeicov("attract") +
  gwidegree(fixed=FALSE) +
  gwodegree(fixed=FALSE) +
  ttriple("introduce") +
  gwdsp,
  constraints= ~ edges, # constrain on number of edges in the network
  MLestimate=TRUE)  # use MCMC for ML estimation

library(network)

m<-3150/750 # average number of edges per node (degree)
n<-100 # number of nodes
e<-m*n # number of edges
alpha<-0.9
beta<-0.3


#T<-10000 # number of simulations
T<-1000

twindeg1 <- rep(0, times=T)
twoutdeg1 <- rep(0, times=T)
twtr1 <- rep(0, times=T)
twbe1 <- rep(0, times=T)

twindeg2 <- rep(0, times=T)
twoutdeg2 <- rep(0, times=T)
twtr2 <- rep(0, times=T)
twbe2 <- rep(0, times=T)


for(t in 1:T) { # do simulation run

 #if(t%%100==0) print(t)
 print(t)

 # randomly choose one individual in each network to be a twin
 twin1 <- sample(n,1)
 twin2 <- sample(n,1)

 # create first set of individual traits
 d1<-runif(n) # distribution of attractiveness
 cl1<-2:(n+1) # introduce parameter
 cl1<-ifelse(as.numeric(runif(n)<beta)==1,1,cl1)

 # create original network
 g1.use <- network(as.matrix(cbind(sample(n,e,replace=T),sample(n,e,replace=T))))
 g1.use %v% "attract" <- d1 # assign intrinsic characteristic to each node
 g1.use %v% "introduce" <- cl1
 g1 <- simulate(model,
  constraints = ~ edges, # constrain the network to yield e edges
  basis=g1.use, # use basis network to get total number of edges and nodes
```

```
  burnin=100000) # let Monte Carlo Chain run a long time before sampling a network

 # create second set of individual traits
 d2<-runif(n) # distribution of attractiveness
 cl2<-2:(n+1) # introduce parameter
 cl2<-ifelse(as.numeric(runif(n)<beta)==1,1,cl2)

 # copy genes to one individual in new network
 d2[twin2] <- d1[twin1]
 cl2[twin2]<- cl1[twin1]

 # create mirror network
 g2.use <- network(as.matrix(cbind(sample(n,e,replace=T),sample(n,e,replace=T))))
 g2.use %v% "attract" <- d2 # assign intrinsic characteristic to each node
 g2.use %v% "introduce" <- cl2
 g2 <- simulate(model,
  constraints = ~ edges, # constrain the network to yield e edges
  basis=g2.use, # use basis network to get total number of edges and nodes
  burnin=100000) # let Monte Carlo Chain run a long time before sampling a network

 # convert networks to igraph graph objects
 m1 = g1[,]
 m2 = g2[,]

 require(igraph)

 g1i <- graph.adjacency(m1)
 g2i <- graph.adjacency(m2)

 # generate/store node statistics for original network
 twindeg1[t] <- degree(g1i,v=twin1-1,mode="in")
 twoutdeg1[t] <- degree(g1i,v=twin1-1,mode="out")
 twtr1[t] <- transitivity(g1i,v=twin1-1,type="local")
 twbe1[t] <- betweenness(g1i,v=twin1-1,directed=F)

 # generate/store node statistics for mirror network
 twindeg2[t] <- degree(g2i,v=twin2-1,mode="in")
 twoutdeg2[t] <- degree(g2i,v=twin2-1,mode="out")
 twtr2[t] <- transitivity(g2i,v=twin2-1,type="local")
 twbe2[t] <- betweenness(g2i,v=twin2-1,directed=F)

 detach(package:igraph)
}

corindeg <- cor.test(twindeg1, twindeg2)
coroutdeg <- cor.test(twoutdeg1, twoutdeg2)
cortr <- cor.test(twtr1, twtr2, use="complete.obs")
corbe <- cor.test(twbe1, twbe2)

# Results obtained after 1000 simulations:
corindeg # 0.50 (0.48,0.52)
coroutdeg # 0.12 (0.10,0.14)
cortr # 0.02 (0.00,0.04)
corbe # 0.34 (0.32,0.36)

# The model parameters
model
```

Literatur

[1] Fowler, James H., Christopher T. Dawes und Nicholas A. Christakis: *Model of Genetic Variation in Human Social Networks.* PNAS - Volume 106, 2009.

[2] Hunter, David R.: *Curved Exponential Family Models for Social Networks.* Social Networks - Volume 29, 2007.

[3] Snijders, Tom A. B.: *Markov Chain Monte Carlo Estimation of Exponential Random Graph Models.* Journal of Social Structure - Volume 3, 2002.

[4] Snijders, Tom A.B., Philippa E. Pattison, Garry L. Robins und Mark S. Handcock: *New Specifications for Exponential Random Graph Models.* Sociological Methodology, 2006.