

Estimating Subgraph Generation Models to Understand Large Network Formation

Laurens Bogaardt | Netherlands eScience Center | l.bogaardt@esciencecenter.nl

Frank W. Takes | University of Amsterdam | takes@uva.nl

Project time frame: January 2018 - October 2018

Introduction

Social scientists often aim to understand the incentives and mechanisms which result in large scale socio-economic structures. Key to this is network formation analysis. However, large datasets are not uncommon, leading to a computational challenge. For example, political scientists interested in global networks of corporate control may need to analyse millions of companies to answer their questions [1, 2].

The Exponential Random Graph Model (ERGM) is a frequently used network formation model. Unfortunately, it suffers from two fundamental flaws. Firstly, its parameter estimates are inconsistent [3, 4]. Secondly, it does not scale well [5]. Recently, an alternative network formation model was suggested: the Subgraph Generation Model (SUGM) [6, 7, 8].

This *Mathematica Notebook* examines SUGMs and a method of estimating their parameters using the subgraph census.

Initialisation

At the end of this *Mathematica Notebook*, a *Manipulate* will be created which allows the user to run simulations. The current work is meant to explore the ideas related to SUGMs and is not optimised for performance. Therefore, some of these simulations take a long time to evaluate. To accommodate this, the timeout threshold for *Dynamic* evaluations needs to be increased.

```
In[1]:= SetOptions[$FrontEnd, DynamicEvaluationTimeout -> 1200]
```

Subgraph Generation

A SUGM is defined by a set of q small subgraphs such as links, triangles or stars, each with corresponding probabilities. For each subgraph i of m_i nodes, the n nodes of

the entire network are grouped into all possible subsets of m_i nodes. Then, each of these subsets receives the subgraph i with probability p_i or remains empty with probability $1 - p_i$.

Input

This subsection defines 5 types of subgraphs, namely links, 3-paths, triangles, 4-stars and 4-cliques.

```
In[2]:= subgraphEdgeLists = Association["Links" → {1 ↔ 2}, "Paths" → {1 ↔ 2, 1 ↔ 3},
    "Triangles" → {1 ↔ 2, 1 ↔ 3, 2 ↔ 3}, "Stars" → {1 ↔ 2, 1 ↔ 3, 1 ↔ 4}, "Cliques" → {1 ↔ 2, 1 ↔ 3, 1 ↔ 4, 2 ↔ 3, 2 ↔ 4, 3 ↔ 4}];
```

Functions

This subsection defines a function which turns the subgraph edge lists defined above into a mapping function. It also defines a function which can generate graphs based on combinations of subgraphs. The more nodes a subgraph has, the more ways there are to group the entire network into subsets. To be able to interpret the probability p_i as the proportion of nodes participating in a certain subgraph, a function is defined which compensates for this larger number of subsets.

```
In[3]:= edgeListToEdgeMap[edgeList_] :=
    Function[{x}, Evaluate[Flatten[Map[Quiet[{Part[x, Part[#, 1]] ↔ Part[x, Part[#, 2]]}, {Part::partd}] &, edgeList]]]]

In[4]:= binomialCompensator[l_, n_] := Binomial[n, 2] / Binomial[n, l]

In[5]:= generateVertexSubsets[l_, n_, p_] :=
    RandomSample[Subsets[Range[n], {l}], RandomVariate[BinomialDistribution[Binomial[n, 1], binomialCompensator[l, n] p]]]

In[6]:= generateGraph[l_, n_, p_, edgeMap_] := Map[Sort, Map[edgeMap, Map[RandomSample, generateVertexSubsets[l, n, p]]], {2}]

In[7]:= generateGraphList[vertexCount_, edgeMap_, n_, pMap_] := Union[Flatten[KeyValueMap[generateGraph[vertexCount[#1], n, #2, edgeMap[#1]] &, pMap]]]
```

Derived Input

Some variables need not be set by hand and can be derived from previously defined input. This subsection defines variables related to the subgraph edge lists.

```
In[8]:= subgraphVertexCounts = Map[VertexCount, subgraphEdgeLists];

In[9]:= subgraphEdgeCounts = Map[EdgeCount, subgraphEdgeLists];

In[10]:= subgraphEdgeMaps = Map[edgeListToEdgeMap, subgraphEdgeLists];
```

Visualisation Functions

To be able to distinguish the generated subgraphs from each other in a visualisation, it helps to colour them differently. The following function does this.

```

In[11]:= colouredGraph[colorList_, edgeLists_] := With[{uniqueEdges = DeleteDuplicates[Flatten[edgeLists]]},
  With[{edgeStyles = AssociationMap[Function[{uniqueEdge}, Pick[colorList, Map[MemberQ[#, uniqueEdge] &, edgeLists]]], uniqueEdges]},
    Module[{edgeIndex = AssociationThread[uniqueEdges → ConstantArray[1, Length[uniqueEdges]]]},
      Fold[{SetProperty[{#1, #2}, EdgeShapeFunction → ({edgeStyles[#2][edgeIndex[#2]++]}, Line[#]) &}] &,
        Graph[Flatten[Map[Union, edgeLists]]], uniqueEdges]]]]]

```

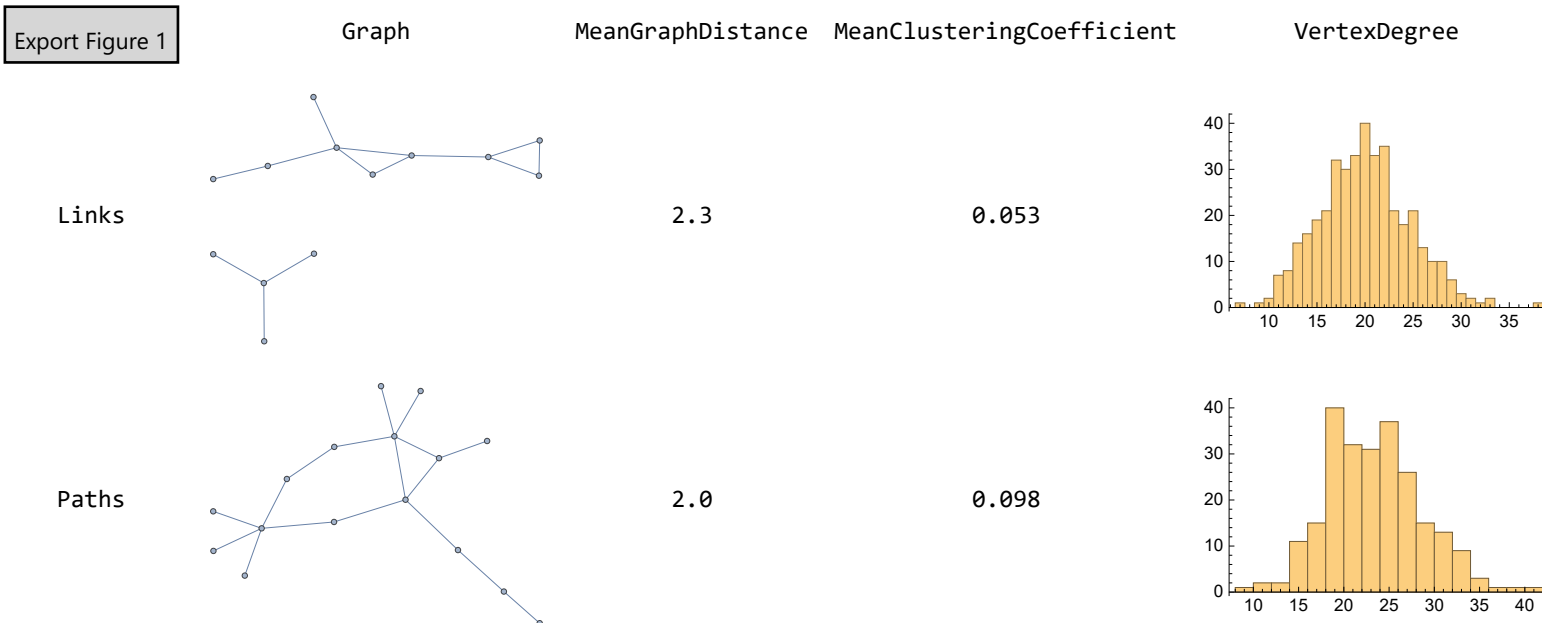
Examples

The first example in this subsection shows multiple graphs generated for each of the subgraphs. It also includes various relevant graph metrics. The second example shows a single graph combining multiple types of subgraphs. The observed network, left in the figure, is the union of all these subgraphs, right in figure, where the generated subgraphs may overlap. Multiple neighbouring subgraphs may incidentally form additional structures such as triangles or squares.

```

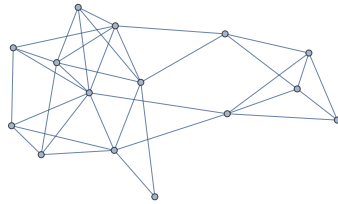
In[12]:= Module[{figureList = ConstantArray[Null, 10]},
  Grid[Join[{Button["Export Figure 1", Do[Export[ToString[NotebookDirectory[]] <> "\\Figure01_" <> ToString[i] <> ".pdf", figureList[[i]]],
    {i, 1, Length[figureList]}]], "Graph", "MeanGraphDistance", "MeanClusteringCoefficient", "VertexDegree"}],
  Table[With[{graph1 = Graph[generateGraphList[subgraphVertexCounts, subgraphEdgeMaps, iter[[3]], Association[iter[[2]] → iter[[4]]]]],
    graph2 = Graph[generateGraphList[subgraphVertexCounts, subgraphEdgeMaps, iter[[5]], Association[iter[[2]] → iter[[6]]]]],
    {iter[[2]], figureList[[iter[[1]]]] = graph1, N[MeanGraphDistance[graph2], 2], N[MeanClusteringCoefficient[graph2], 2],
      figureList[[iter[[1]] + 1]] = Histogram[VertexDegree[graph2]]}, {iter, {{1, "Links", 20, 0.1, 400, 0.05}, {3, "Paths", 20, 0.05, 240, 0.05},
    {5, "Triangles", 15, 0.1, 240, 0.05}, {7, "Stars", 15, 0.1, 170, 0.05}, {9, "Cliques", 15, 0.05, 170, 0.05}}]], Spacings → {1, 1}]]

```



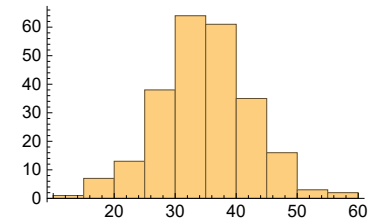
Out[12]=

Triangles

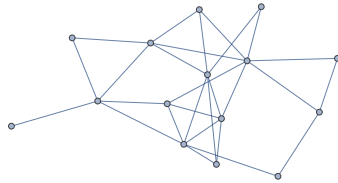


1.9

0.17

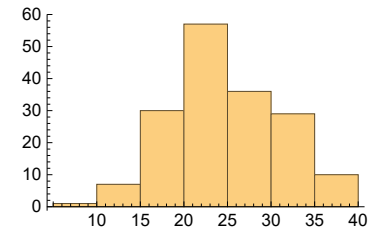


Stars

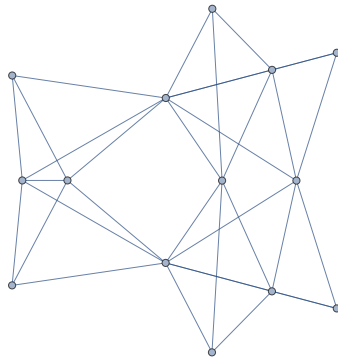


1.9

0.15

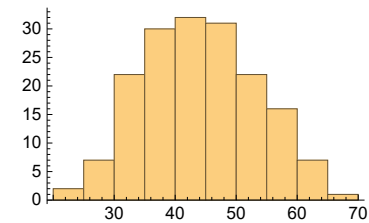


Cliques



1.7

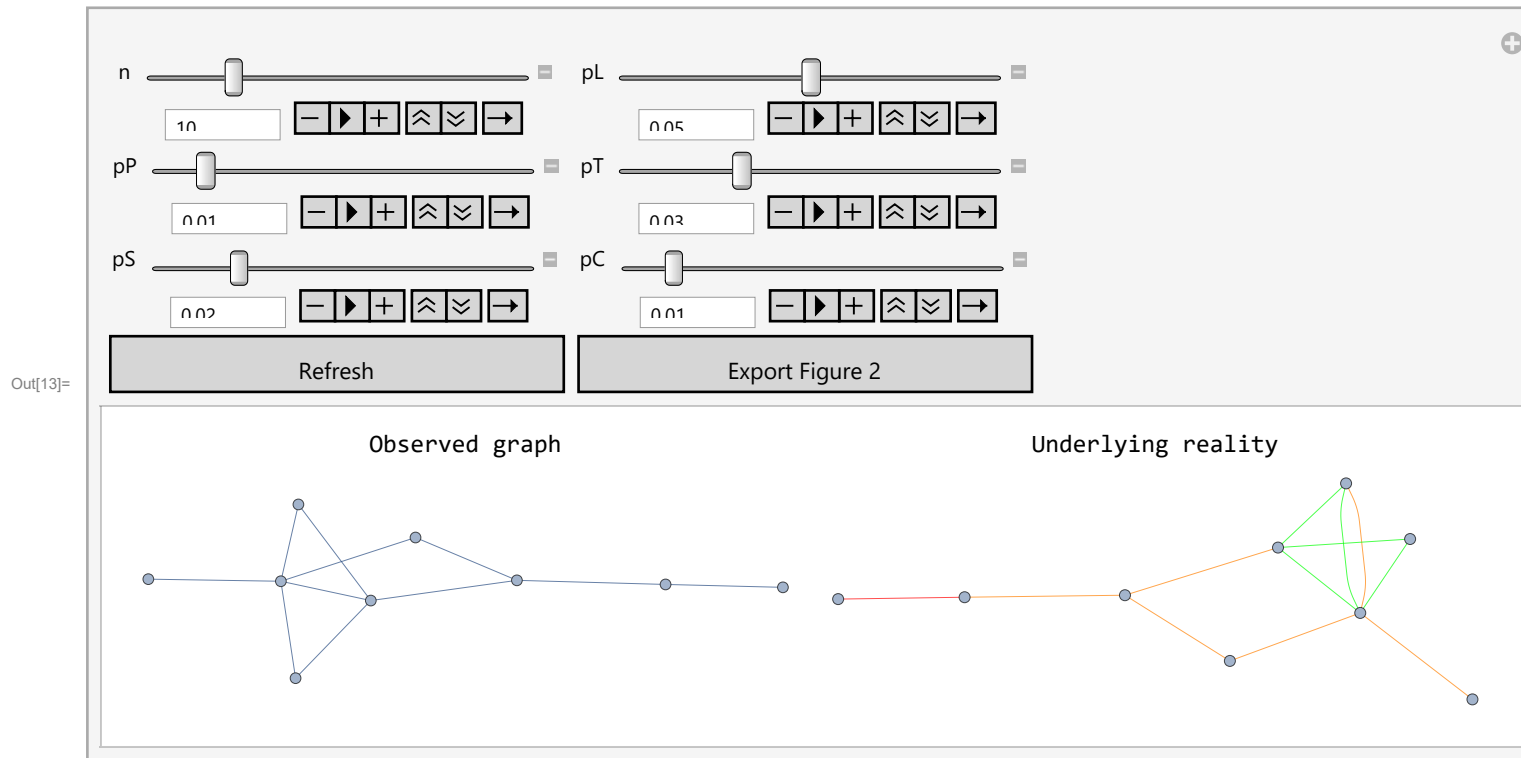
0.30



```

In[13]:= DynamicModule[{refreshDummy = 0, figureList = ConstantArray[Null, 2]}, Manipulate[refreshDummy;
  With[{colorList = {Red, Purple, Green, Orange, Blue}, edgeLists = KeyValueMap[Flatten[generateGraph[subgraphVertexCounts[#1], n, #2,
    subgraphEdgeMaps[#1]]] &, Association["Links" → pL, "Paths" → pP, "Triangles" → pT, "Stars" → pS, "Cliques" → pC]]], With[{edgeList =
    Union[Flatten[edgeLists]]}, Grid[{{"Observed graph", "Underlying reality"}, {figureList[[1]] = Show[Graph[edgeList], ImageSize → 350],
    figureList[[2]] = Show[colouredGraph[colorList, edgeLists], ImageSize → 350]}]], Grid[{{Control[{{n, 10}, 5, 30, 1, Appearance →
    "Open"}], Control[{{pL, 0.05}, 0.0, 0.1, 0.01, Appearance → "Open"}]], {Control[{{pP, 0.04}, 0.0, 0.1, 0.01, Appearance → "Open"}],
    Control[{{pT, 0.03}, 0.0, 0.1, 0.01, Appearance → "Open"}]], {Control[{{pS, 0.02}, 0.0, 0.1, 0.01, Appearance → "Open"}],
    Control[{{pC, 0.01}, 0.0, 0.1, 0.01, Appearance → "Open"}]], {Button["Refresh", refreshDummy = RandomReal[]], Button["Export Figure 2",
    Do[Export[ToString[NotebookDirectory[]] <> "\\Figure02_" <> ToString[i] <> ".pdf", figureList[[i]]]; {i, 1, Length[figureList]}], ""}}]]]

```



Subgraph Census

The goal of this *Mathematica Notebook* is to explore a method of estimating SUGMs parameters using the subgraph census. In a k -subgraph census, a network of n nodes is grouped into all possible subsets of k nodes, which are then tallied according to their isomorphism class [9, 10, 11].

Input

In the current work, the subgraph census is limited to size 4. One reason for this are the computational difficulties which arise with higher order censi. Another reason is the degree sequence trick discussed in the next subsection.

```
In[14]:= censusSizes = {2, 3, 4};
```

Functions

This subsection defines functions which generate all census subgraph types, including all their permutations. It also defines a function which performs the subgraph

census on a graph. For subgraphs up to size 4, the type can be identified uniquely by its degree sequence [12]. This allows for an optimisation trick in the count function which speeds up the determination of the census type. However, it limits the use of this function to census sizes of maximally 4.

```
In[15]:= edgeListPermutations[l_] := With[{edges = Map[#[[1]] ↔ #[[2]] &, Subsets[Range[1], {2}]]}, Subsets[edges, Binomial[1, 2]]]

In[16]:= canonicalEdgeList[edgeList_] := EdgeList[CanonicalGraph[Graph[edgeList]]]

In[17]:= generateCensusGraphs[l_] := DeleteDuplicates[Map[canonicalEdgeList, edgeListPermutations[l]]]

In[18]:= automorphismCount[n_, edgeList_] := n! / GroupOrder[GraphAutomorphismGroup[Graph[Range[n], edgeList]]]

In[19]:= countCensusGraphs[l_, n_, edgeList_] :=
  With[{adjacencyMatrix = AdjacencyMatrix[Graph[Range[n], edgeList]], variableList = Map[Symbol["i" <> ToString[#]] &, Range[1]],
    keyMap = Association[Map[PadLeft[Sort[VertexDegree[#]], 1] → # &, generateCensusGraphs[l]]]}, With[
    {iteratorSequence = Replace[Map[{variableList[[#]], Join[{0}, variableList][[#]] + 1, n - 1 + #} &, Range[1]], head_[arg_] => Sequence[arg]}],
    Module[{counts = Association[Map[PadLeft[Sort[VertexDegree[#]], 1] -> 0 &, generateCensusGraphs[l]]]},
      Do[counts[Sort[Total[adjacencyMatrix[[variableList, variableList]]]]] ++, iteratorSequence];
      counts = KeyMap[keyMap, counts];
      counts]]]
```

Derived Input

The total number of census types can be derived from the function which generates these types for each size.

```
In[20]:= censusGraphCounts = AssociationMap[Length[generateCensusGraphs[#]] &, censusSizes];
```

Examples

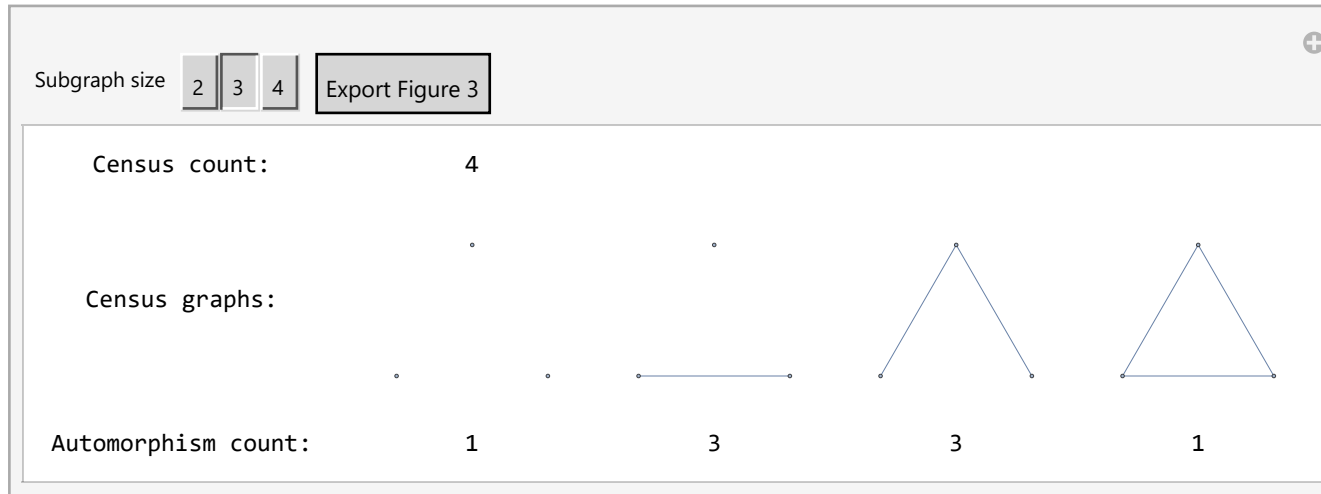
The following example shows the census types for each census size.

```

In[21]:= DynamicModule[{figureList},
  Manipulate[With[{graphs = generateCensusGraphs[1]}, Grid[{"Census count: ", censusGraphCounts[1]}, Join[{"Census graphs: "},
    figureList = Map[Graph[Range[1], #, GraphLayout -> "CircularEmbedding", ImagePadding -> {{8, 8}, {2, 8}}, ImageSize -> 100] &, graphs]],
    Join[{"Automorphism count: "}, Map[automorphismCount[1, #] &, graphs]]], Spacings -> {2, 2}]],
  Grid[{Control[{{1, 3, "Subgraph size"}, censusSizes}], Button["Export Figure 3",
    Do[Export[ToString[NotebookDirectory[]] <> "\\Figure03_" <> ToString[i] <> ".pdf", figureList[[i]]];, {i, 1, Length[figureList]}]]]}]]

```

Out[21]=



Metropolis-Hastings Sampling

In the “Parameter Estimation” section, the likelihood function of a model is determined based on the observed data. An easy way to work with likelihood functions is to sample from them using the Metropolis-Hastings algorithm. This algorithm is implemented in the current section.

Functions

The Metropolis-Hastings algorithm is a fairly simple algorithm, described in more detail elsewhere [13]. The implementation below takes as arguments the likelihood function, starting value(s), a distribution which influences the random walk, the number of iterations and the number of iterations to throw away from the start.

```

In[22]:= metropolisHastings[function_, start_, distribution_, iterations_, burnin_] :=
  Quiet[Module[{previous = start, proposal}, Table[proposal = Map[RandomVariate[distribution[#]] &, previous];
    If[function[proposal] > RandomReal[] function[previous], previous = proposal, previous], {i, 1, iterations}][[1 + burnin ;; -1]]]]

```

Visualisation Functions

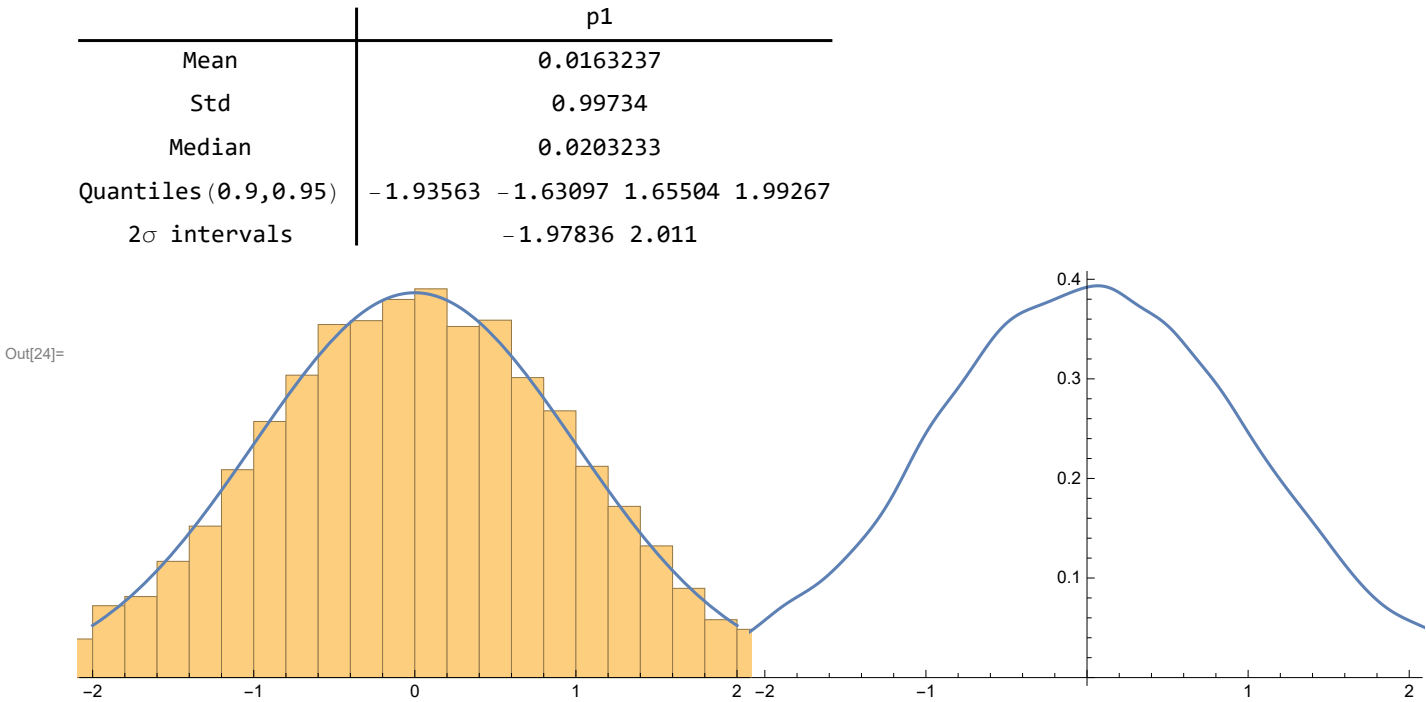
The following function helps visualise some metrics about the sample taken from the distribution.

```
In[23]:= samplingDescription[list_] := With[{mean = Mean[list], standardDeviation = StandardDeviation[list]},  
  Grid[{Join[{"", Map["p" <> ToString[#] &, Range[Length[mean]]]}, Join[{"Mean"}, N[mean]], Join[{"Std"}, N[standardDeviation]],  
    Join[{"Median"}, N[Median[list]]], Join[{"Quantiles(0.9,0.95)"}, Map[Row[], " "] &, N[Quantile[list, {0.025, 0.05, 0.95, 0.975}]]]],  
    Join[{"2σ intervals"}, MapThread[Row[{#1 - 2 #2, #1 + 2 #2}, " "] &, {mean, standardDeviation}]]],  
  Spacings → {1, 1}, Dividers → {{False, True}, {False, True}}]]
```

Examples

The examples in this subsection compare the results of the Metropolis-Hastings algorithm for some standard distributions with a more direct sampling technique.

```
In[24]:= With[{sampling = RandomVariate[MultinormalDistribution[{0}, {{1}}], 10000]},  
  Column[samplingDescription[sampling],  
    Row[{Show[{Histogram[Flatten[sampling], 30, "PDF"], Plot[PDF[NormalDistribution[0, 1], x], {x, -2, 2}], PlotRange -> {{-2, 2}, {0, 0.4}},  
      ImageSize → 350], Show[SmoothHistogram[Flatten[sampling]], PlotRange -> {{-2, 2}, {0, 0.4}}, ImageSize → 350}}]], Spacings → 1]]
```



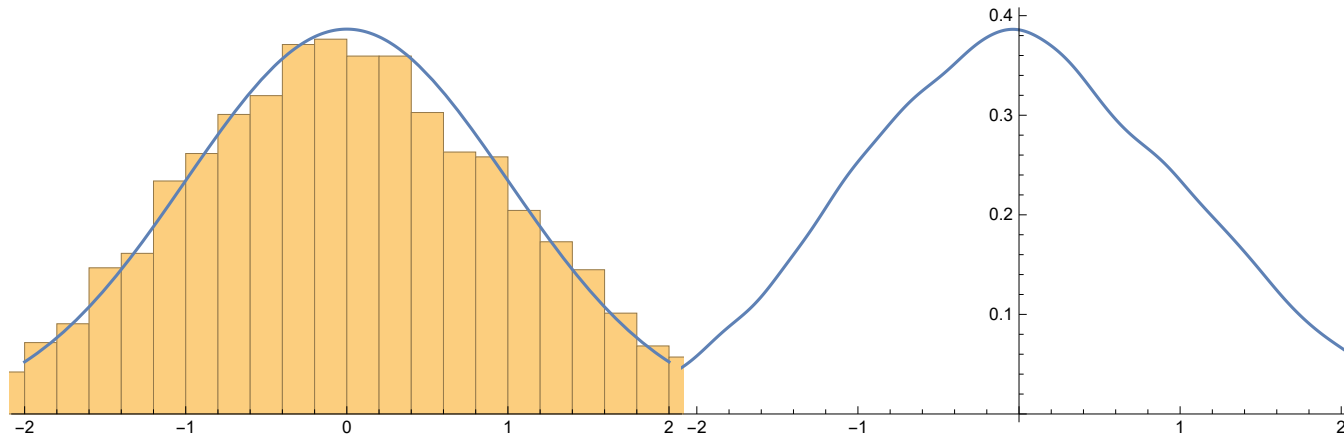

```

In[25]:= With[{sampling = metropolisHastings[PDF[MultinormalDistribution[{0}, {{1}}]], {0}, Function[{x}, NormalDistribution[x, 0.3]], 10000, 0]},
  Column[{samplingDescription[sampling],
    Row[{Show[{Histogram[Flatten[sampling], 30, "PDF"], Plot[PDF[NormalDistribution[0, 1], x], {x, -2, 2}]}], PlotRange -> {{-2, 2}, {0, 0.4}},
      ImageSize -> 350], Show[SmoothHistogram[Flatten[sampling]], PlotRange -> {{-2, 2}, {0, 0.4}}, ImageSize -> 350]}], Spacings -> 1]]

```

	p1
Mean	0.00344241
Std	1.05477
Median	-0.0209262
Quantiles (0.9,0.95)	-2.01617 -1.71317 1.75638 2.09353
2 σ intervals	-2.10609 2.11298

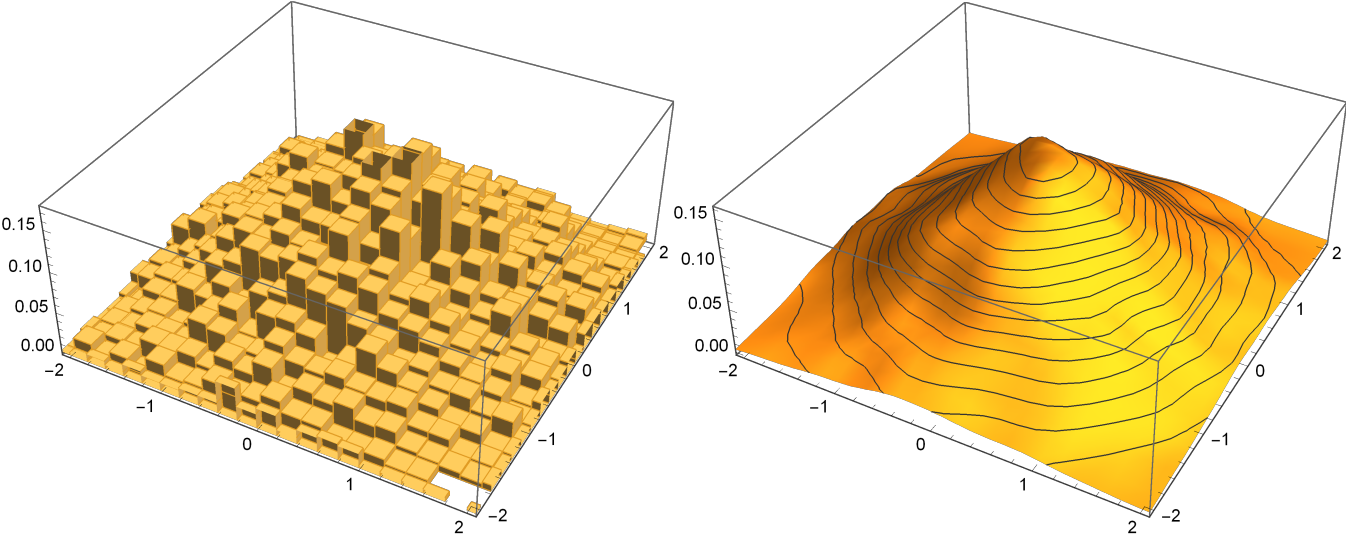
Out[25]=



```
In[26]:= With[{sampling = RandomVariate[MultinormalDistribution[{0, 0}, {{1, 0}, {0, 1}}], 10000]},
  Column[{samplingDescription[sampling], Row[{Show[Histogram3D[sampling, 30, "PDF"], PlotRange -> {{-2, 2}, {-2, 2}, {0, 0.16}}, ImageSize -> 350],
    Show[SmoothHistogram3D[sampling], PlotRange -> {{-2, 2}, {-2, 2}, {0, 0.16}}, ImageSize -> 350]}]], Spacings -> 1]]
```

	p1				p2			
Mean	-0.00562769				0.00424773			
Std	0.994314				1.00427			
Median	-0.0013348				0.00714657			
Quantiles (0.9,0.95)	-1.95541	-1.65796	1.62315	1.9262	-1.98075	-1.64643	1.64806	1.98866
2σ intervals	-1.99426 1.983				-2.0043 2.01279			

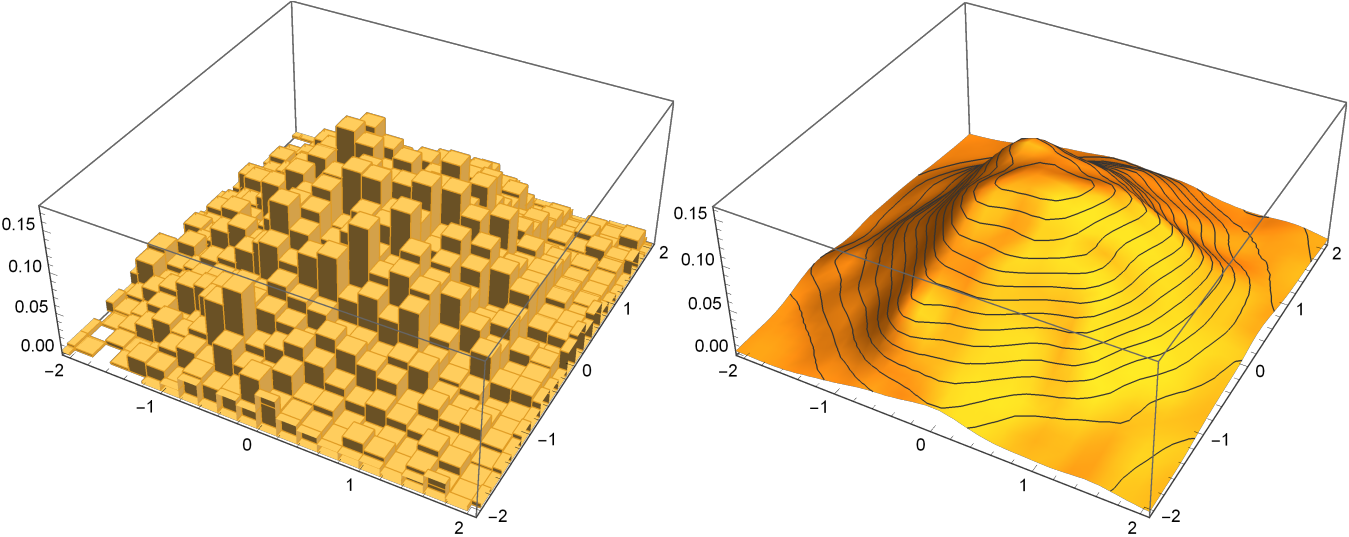
Out[26]=



```
In[27]:= With[{sampling =  
  metropolisHastings[PDF[MultinormalDistribution[{0, 0}, {{1, 0}, {0, 1}}]], {0, 0}, Function[{x}, NormalDistribution[x, 0.3]], 10000, 0]],  
  Column[{samplingDescription[sampling], Row[{Show[Histogram3D[sampling, 30, "PDF"], PlotRange -> {{-2, 2}, {-2, 2}, {0, 0.16}}, ImageSize -> 350],  
    Show[SmoothHistogram3D[sampling], PlotRange -> {{-2, 2}, {-2, 2}, {0, 0.16}}, ImageSize -> 350]}]], Spacings -> 1]]
```

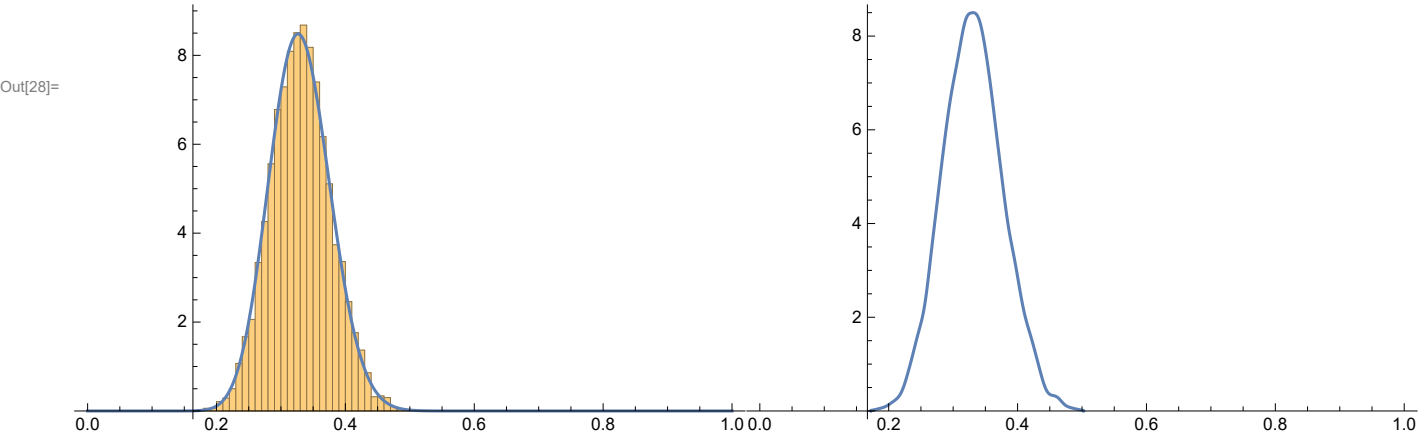
	p1				p2			
Mean	0.0436609				0.0566444			
Std	1.0097				1.05807			
Median	-0.00880543				0.0407781			
Quantiles(0.9,0.95)	-1.83103	-1.52435	1.78352	2.18326	-1.97127	-1.63765	1.83491	2.20139
2σ intervals	-1.97574 2.06306				-2.05949 2.17278			

Out[27]=



```
In[28]:= With[{sampling = RandomVariate[BetaDistribution[33, 67], {10000, 1}]},  
  Column[{samplingDescription[sampling], Row[  
    {Show[{Histogram[Flatten[sampling], 30, "PDF"], Plot[PDF[BetaDistribution[33, 67], x], {x, 0, 1}, PlotRange -> All]}, PlotRange -> {{0, 1}, All},  
    ImageSize -> 350], Show[SmoothHistogram[Flatten[sampling]], PlotRange -> {{0, 1}, All}, ImageSize -> 350]}]}, Spacings -> 1]]
```

	p1			
Mean	0.331083			
Std	0.0467674			
Median	0.330199			
Quantiles (0.9,0.95)	0.2422	0.255974	0.410696	0.425651
2σ intervals	0.237549 0.424618			

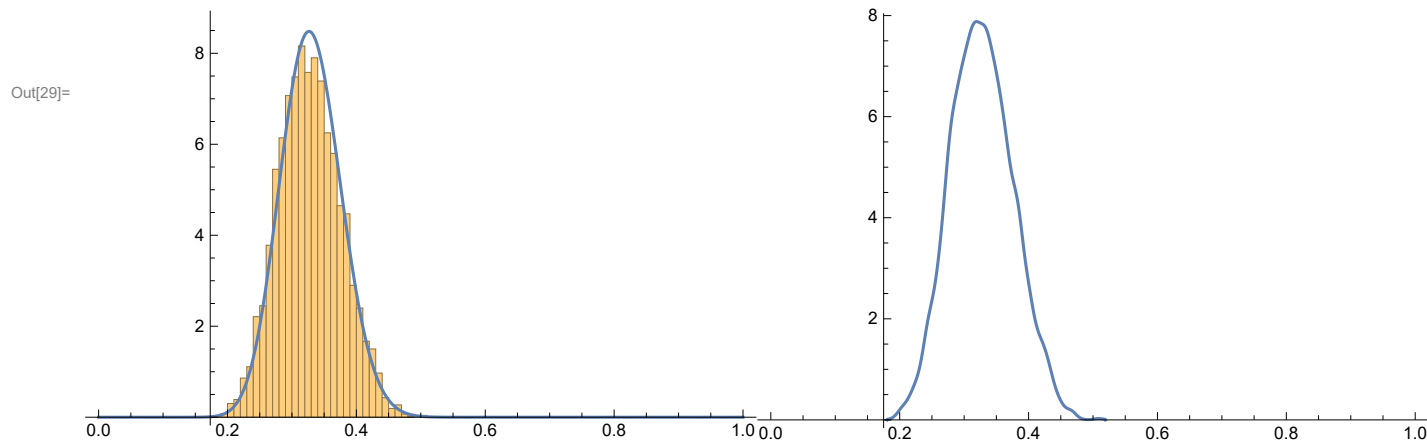


```

In[29]:= With[{sampling = metropolisHastings[Function[{x}, PDF[BetaDistribution[33, 67], x[[1]]],
  {0.5}, Function[{x}, BetaDistribution[10^-2 + 100 x, 10^-2 + 100 (1 - x)]], 10000, 0]},
  Column[{samplingDescription[sampling], Row[{Show[{Histogram[Flatten[sampling], 30, "PDF"],
    Plot[PDF[BetaDistribution[33, 67], x], {x, 0, 1}, PlotRange -> All]}, PlotRange -> {{0, 1}, All}, ImageSize -> 350],
    Show[SmoothHistogram[Flatten[sampling]], PlotRange -> {{0, 1}, All}, ImageSize -> 350]}]}, Spacings -> 1]}

```

	p1
Mean	0.327621
Std	0.0484697
Median	0.325706
Quantiles (0.9,0.95)	0.238228 0.250373 0.410977 0.426609
2σ intervals	0.230681 0.42456



Parameter Estimation

The original articles describing SUGMs contain two methods to estimate the parameters of the model. The current work suggests a third, more intuitive method based on the subgraph census, which also takes overlap and incidental subgraph formation into account.

Functions

This subsection defines various functions relevant for estimating the model's parameters. The ultimate goal is to define the correct likelihood function, which describes the distribution over the parameters depending on the observed data. The likelihood function is a combination of the census counts and the probability functions for the expected values for each census type. In general, each of the r counts of the census x_j , together with the probability functions $f_j(\hat{p}_1, \dots, \hat{p}_q)$, enter into the multinomial probability mass function to form the likelihood function.

When examining the probability functions, two patterns can be extracted. The first depends on the types of subgraphs contained in the model. The second depends on the size of the census. In this subsection, several functions are defined which generate permutations for subgraphs. To extract the first pattern, the permutations are used to determine in how many cases a subgraph can produce edges in a census subgraph. Then, the overlap of all permutations is used to determine the dependency on the census size. Combining both gives the probability functions and the likelihood functions for any model. These can be used to estimate the parameters of the model and their confidence intervals. The likelihood function also includes a degree-of-freedom compensator which approximately takes into account the correlations between generated edges.

Finally, a function is defined which determines the Mahalanobis distance between the maximum likelihood estimates and the true parameters. This allows for the verification of the standard errors of the estimates and of the degree-of-freedom compensator.

```
In[30]:= censusGraphPermutations[l_] := GroupBy[edgeListPermutations[l], canonicalEdgeList]

In[31]:= subgraphPermutations[l_] := GroupBy[Select[edgeListPermutations[l], And[Length[#] > 0, ConnectedGraphQ[Graph[#]]] &], canonicalEdgeList]

In[32]:= generateSubgraphs[l_] :=
  DeleteDuplicates[Map[canonicalEdgeList, Select[edgeListPermutations[l], And[Length[#] > 0, ConnectedGraphQ[Graph[#]]] &]]]

In[33]:= complementEdgeList[l_, edgeList_] := Complement[EdgeList[CompleteGraph[l]], edgeList]

In[34]:= generateCensusEquationCases[l_, edgeLists_, vertexCounts_] :=
  With[{relevantSubgraphs = Map[canonicalEdgeList, Values[subgraphEdgeLists[[Keys[Select[subgraphVertexCounts, # ≤ l &]]]]]],
    MapThread[Function[{edgeList, vertexCount}, With[{permutations = subgraphPermutations[vertexCount][canonicalEdgeList[edgeList]],
      AssociationMap[Function[{pattern}, Function[Evaluate[{n}], Evaluate[Count[permutations, permutation_ /; And[ContainsAll[permutation,
        pattern], ContainsNone[complementEdgeList[VertexCount[pattern], pattern], permutation]]] / Length[permutations] Binomial[n - 1,
        vertexCount - VertexCount[pattern]]]], Intersection[generateSubgraphs[l], relevantSubgraphs]]], {edgeLists, vertexCounts}]]

In[35]:= subgraphUnionReduceCount[l_, permutationList_, additionalSubgraph_, subsetSize_] :=
  Map[{Total[#][[All, 1]]], #[[1, 2]]] &, GatherBy[Flatten[Table[{permutationList[[i, 1]], Union[permutationList[[i, 2]], Flatten[subset]]],
    {i, 1, Length[permutationList]}, {subset, Subsets[additionalSubgraph, {subsetSize}]]], 1], canonicalEdgeList#[[2]]] &]]

In[36]:= subgraphUnionPermutations[l_, permutationCounts_, subgraphPermutations_] :=
  Module[{permutationList = {{1, {}}}}, Do[If[permutationCounts[[o]] > 0, permutationList =
    subgraphUnionReduceCount[l, permutationList, subgraphPermutations[[o]], permutationCounts[[o]]], {o, 1, Length[subgraphPermutations]}];
  permutationList]

In[37]:= countToCensusEquationPattern[permutationCount_, exponents_, maxExponents_] :=
  With[{variableList = Map[Symbol["v" <> ToString[#]] &, Range[Length[exponents]]]},
    permutationCount Apply[Times, MapThread[#1^#2 &, {variableList, (Values[maxExponents] - exponents)}]]]
  Apply[Times, MapThread[(1 - #1)^#2 &, {variableList, exponents}]]]
```

```

In[38]:= generateCensusEquationPatterns[l_] :=
  With[{relevantSubgraphsKeys = Map[Key[canonicalEdgeList[#]] &, Values[subgraphEdgeLists[[Keys[Select[subgraphVertexCounts, # ≤ 1 &]]]]]],
    With[{subgraphPermutations = subgraphPermutations[l][relevantSubgraphsKeys]],
      With[{subgraphPermutationLengths = Map[Length, subgraphPermutations]},
        With[{variableList = Map[Symbol["i" <> ToString[#]] &, Range[Length[subgraphPermutations]]]}, With[
          {iteratorSequence = Replace[MapThread[{#1, 0, #2} &, {variableList, Values[subgraphPermutationLengths]}], head_[arg_] => Sequence[arg]}],
          Module[{equationPattern = AssociationMap[0 &, generateCensusGraphs[l]]}, Do[equationPattern[canonicalEdgeList[permutations[[2]]]] +=
            countToCensusEquationPattern[permutations[[1]], variableList, subgraphPermutationLengths,
              iteratorSequence, {permutations, subgraphUnionPermutations[l, variableList, subgraphPermutations]}];
            Map[Function[Evaluate[Map[Symbol["v" <> ToString[#]] &, Range[Length[variableList]]], Evaluate[Simplify[#]]] &, equationPattern]]]]]]]]

In[39]:= exponentGenerator[n_, p_, l_, cases_] := Map[(1 - binomialCompensator[l, n] p) ^ # &, cases]

In[40]:= equationGenerator[l_, vertexCounts_, censusGraphCounts_, equationCases_, equationPatterns_] := With[{exponentiatedCases =
  MapThread[Function[{p}, Evaluate[exponentGenerator[n, p, #1, Map[# [n] &, #2]]]] &, {vertexCounts, Map[Values, equationCases[l]]}],
  Map[Function[{variableList}, Map[Function[Evaluate[{n, ps}], Evaluate[# [ReplaceAll[variableList, List -> Sequence]]]] &, equationPatterns[l]],
    AssociationMap[Product[exponentiatedCases[# [i]]] [Quiet[Part[ps, i], {Part::partd}]]], {i, 1, Length[#]}] &,
    Subsets[Keys[exponentiatedCases], {1, Min[Length[exponentiatedCases], censusGraphCounts[l] - 1]}]]]]

In[41]:= expectedValueGenerator[equations_] := Association[KeyValueMap[Function[{type, equationsPerType},
  type → Map[Function[Evaluate[{n, ps}], Evaluate[Binomial[n, type] # [n, ps]]] &, equationsPerType, {2}], equations]]

In[42]:= likelihoodGenerator[equations_] :=
  Association[KeyValueMap[Function[{type, equationsPerType}, type → Association[KeyValueMap[Function[{key, equation},
    key → Function[Evaluate[{n, ps, counts, d}], Evaluate[Apply[Times, MapThread[#1 [n, ps] ^ (d Quiet[Part[counts, #2], {Part::partd}]) &,
      {Values[equation], Range[Length[equation]]}]]]]], equationsPerType]]], equations]]

In[43]:= degreesOfFreedomCompensator[l_, s_, n_] := 1 / Binomial[l, 2] 1 / s Binomial[n, 2] / Binomial[n, 1]

In[44]:= estimateLikelihood[l_, s_, n_, counts_, equations_, likelihoods_, iterations_] :=
  With[{likelihoodsReduced = Function[{ps}, Evaluate[Quiet[likelihoods[n, ps, counts, degreesOfFreedomCompensator[l, s, n]]]]],
    With[{variableList = Variables[Last[Quiet[likelihoodsReduced[ps]]]]},
      With[{maxLikelihood = NArgMax[{likelihoodsReduced[variableList], 0 < variableList < 0.5}, variableList]},
        With[{metropolisHastings = metropolisHastings[likelihoodsReduced, maxLikelihood, Function[{x},
          BetaDistribution[10^-2 + 1000 x, 10^-2 + 1000 (1 - x)]]], iterations Length[variableList], 0]}, {maxLikelihood, metropolisHastings}]]]]

In[45]:= mahalanobisDistance[data_, point_] :=
  With[{mean = Mean[data], covariance = Covariance[data]}, Sqrt[Dot[mean - point, Inverse[covariance], mean - point]]]

In[46]:= verifyMahalanobisDistance[l_, s_, n_, equations_, likelihoods_, countsList_, ps_] :=
  Transpose[Table[With[{maxLikelihoodAndMetropolisHastings = estimateLikelihood[l, s, n, counts, equations, likelihoods, 6000]},
    {First[maxLikelihoodAndMetropolisHastings], mahalanobisDistance[Last[maxLikelihoodAndMetropolisHastings], ps]}], {counts, countsList}]]

```

Derived Input

In this subsection, the functions defined above are used to derive various inputs, such as the likelihood functions. These can be described analytically, which speeds up their use in the example below.

```
In[47]:= censusGraphCounts = AssociationMap[Length[generateCensusGraphs[#]] &, censusSizes];

In[48]:= censusEquationCases = AssociationMap[generateCensusEquationCases[#, subgraphEdgeLists, subgraphVertexCounts] &, censusSizes];

In[49]:= censusEquationPatterns = AssociationMap[generateCensusEquationPatterns, censusSizes];

In[50]:= censusEquations =
  AssociationMap[equationGenerator[#, subgraphVertexCounts, censusGraphCounts, censusEquationCases, censusEquationPatterns] &, censusSizes];

In[51]:= censusExpectedValue = expectedValueGenerator[censusEquations];

In[52]:= censusLikelihoods = likelihoodGenerator[censusEquations];
```

Visualisation Functions

The following functions help visualise the model, the simulations and the estimations.

```
In[53]:= keySort[keys_, vertexCounts_] := Sort[keys, First[First[Position[Keys[vertexCounts], #1]]] < First[First[Position[Keys[vertexCounts], #2]]] &]

In[54]:= visualiseLikelihood[l_, s_, n_, counts_, equations_, likelihoods_] :=
  With[{maxLikelihoodAndMetropolisHastings = estimateLikelihood[l, s, n, counts, equations, likelihoods, 10000]},
    With[{expectedCounts = Map[# [n, maxLikelihoodAndMetropolisHastings[[1]]] &, Values[equations]]},
      Column[{Grid[{Join[{"", Map["p" <> ToString[#] &, Range[Length[maxLikelihoodAndMetropolisHastings[[1]]]]}],
        Join[{"Max likelihood", maxLikelihoodAndMetropolisHastings[[1]]}], Spacings → {1, 1}, Dividers → {{False, True}, {False, True}}], Row[
          {"Goodness of fit: ", Mean[MapThread[If[#1 < #2, 2 (1 + #1) / (1 + #2) - 1, 2 (1 + #2) / (1 + #1) - 1] &, {counts, Total[counts] expectedCounts}]]}],
        samplingDescription[maxLikelihoodAndMetropolisHastings[[2]]], Spacings → 1.5]]]

In[55]:= visualiseHistogram[l_, n_, f_, counts_] := With[{expectedMean = Binomial[n, 1] f, expectedStandardDeviation = Sqrt[Binomial[n, 1] f (1 - f)]},
  Column[{Grid[{{"", "Observed"}, {"Mean", N[Mean[counts]]}, {"Std", N[StandardDeviation[counts]]}], Dividers → {{False, True}, {False, True}}],
    Show[Histogram[counts, Automatic, "PDF"], ImageSize → 200], Alignment → Center, Spacings → 1]}]

In[56]:= expectedMahalanobisDistances = Function[Evaluate[{d}], Evaluate[With[{distribution = x^ (d - 1) PDF[NormalDistribution[], x]},
  ProbabilityDistribution[distribution / Integrate[distribution, {x, 0, Infinity}, Assumptions → {d > 0}], {x, 0, Infinity}]]]]];
```



```

In[57]:= visualiseMahalanobisDistance[l_, s_, n_, equations_, likelihoods_, countsList_, ps_] :=
  With[{maxLikelihoodAndMahalanobisDistance = verifyMahalanobisDistance[l, s, n, equations, likelihoods, countsList, ps]},
    Column[{Grid[{Join[{"", Map["p" <> ToString[#] &, Range[Length[ps]]]},
      Join[{"Max likelihood", Mean[First[maxLikelihoodAndMahalanobisDistance]]}], Spacings → {1, 1}, Dividers → {{False, True}, {False, True}}}],
    Row[{"Cov:", MatrixForm[Covariance[First[maxLikelihoodAndMahalanobisDistance]]], " "], Grid[{"", "Observed", "Expected"},
      {"Mean", Mean[Last[maxLikelihoodAndMahalanobisDistance]], N[Mean[expectedMahalanobisDistances[Length[ps]]]}, {"Trimmed95",
        TrimmedMean[Last[maxLikelihoodAndMahalanobisDistance], {0, 0.05}], TrimmedMean[expectedMahalanobisDistances[Length[ps]], {0, 0.05}]},
      {"Trimmed85", TrimmedMean[Last[maxLikelihoodAndMahalanobisDistance], {0, 0.15}],
        TrimmedMean[expectedMahalanobisDistances[Length[ps]], {0, 0.15}]}, {"Median", Median[Last[maxLikelihoodAndMahalanobisDistance]],
        N[Median[expectedMahalanobisDistances[Length[ps]]]}], Spacings → {1, 1}, Dividers → {{False, True}, {False, True}}],
    Show[Histogram[TakeSmallest[Last[maxLikelihoodAndMahalanobisDistance], Ceiling[0.9 Length[Last[maxLikelihoodAndMahalanobisDistance]]]],
      Automatic, "PDF"], Plot[PDF[expectedMahalanobisDistances[Length[ps]], x],
      {x, 0, Max[Last[maxLikelihoodAndMahalanobisDistance]]}, PlotRange → All, ImageSize → 200]], Alignment → Center, Spacings → 1]]

In[58]:= visualiseEstimationGrid[l_, generationTypes_, modelTypes_, n_, pMap_, t_, equations_, counts_, countsList_, singleAnalysis_, multipleAnalysis_] :=
  Which[Length[generationTypes] < 1 || Length[modelTypes] < 1, "Model undetermined: please choose at least one generation and model type.",
    Length[generationTypes] >= censusGraphCounts[l] || Length[modelTypes] >= censusGraphCounts[l],
    "Model overfit: please choose fewer generation and model types, or select a higher subgraph size.",
    True, Quiet[With[{pList = Map["p" <> StringTake[#, 1] &, Keys[pMap]]},
      With[{equationList = Map[#["n", pList] &, Values[censusEquations[l][keySort[generationTypes, subgraphVertexCounts]]]}],
        Grid[{Join[{"Subgraph"}, Map[Graph[Range[l], #, GraphLayout → "CircularEmbedding"] &, generateCensusGraphs[l]], {"Total"}],
          Join[{Row[generationTypes, "&"]}, equationList, {Simplify[Total[equationList]]}],
          Join[{"Expected"}, Map[#["n", Values[pMap]] &, Values[censusExpectedValue[l][keySort[generationTypes, subgraphVertexCounts]]]],
            {Binomial[n, 1]}], Join[{"Single"}, counts, {singleAnalysis}], Quiet[Join[{"Multiple"},
              MapThread[visualiseHistogram[l, n, #1, #2] &, {Map[#["n", Values[pMap]] &, Values[equations[l][Keys[pMap]]]}, Transpose[countsList]}],
              {multipleAnalysis}]]], Frame → All, Spacings → {2, 3}]]], {Part::partw}]]

```

Examples

The example below contains the probabilities of observing any of the possible census types for various generation models. It also contain a button to generate a single graph and its census counts, as well as one to generate multiple graphs and counts. Both these simulations can be analysed using the maximum likelihood method.

```

In[59]:= With[{outerControls = Grid[
  {{Control[{{1, 2, "Subgraph size"}, censusSizes]], "", Control[{{generationTypes, {First[Keys[subgraphVertexCounts]]}, "Generation type(s)"},
    Keys[subgraphVertexCounts], ControlType → TogglerBar, Appearance → "Vertical"]}},
  Control[{{modelTypes, {First[Keys[subgraphVertexCounts]]}, "Model type(s)"}, Keys[subgraphVertexCounts],
    ControlType → TogglerBar, Appearance → "Vertical"]}}], Spacings → {1, 1}},
Manipulate[With[{pMap = Map[# -> Symbol["p" <> StringTake[#, 1]] &, keySort[generationTypes, subgraphVertexCounts]]},
  With[{innerControls = Grid[Join[{"General", Control[{{n, 50}, 2, 200, 1, Appearance → "Open"]},
    Control[{{t, 100}, 2, 400, 1, Appearance → "Open"}]], {"Single", Button["Generate",
      {counts = Values[countCensusGraphs[1, n, generateGraphList[subgraphVertexCounts, subgraphEdgeMaps, n, Association[pMap]]]}], Button[
        "Analyse", {singleAnalysis = visualiseLikelihood[1, Total[Values[subgraphVertexCounts[modelTypes]]], n, counts, censusEquations[
          1][keySort[modelTypes, subgraphVertexCounts]], censusLikelihoods[1][keySort[modelTypes, subgraphVertexCounts]]}],
      {"Multiple", Button["Generate", {countsList = Table[Values[countCensusGraphs[1, n, generateGraphList[subgraphVertexCounts,
        subgraphEdgeMaps, n, Association[pMap]]], {i, 1, t}]}], Button["Analyse", {multipleAnalysis = visualiseMahalanobisDistance[1,
        Total[Values[subgraphEdgeCounts[modelTypes]]], n, censusEquations[1][keySort[modelTypes, subgraphVertexCounts]],
        censusLikelihoods[1][keySort[generationTypes, subgraphVertexCounts]], countsList, pMap[All, 2]]}],
      KeyValueTypeMap[{{#1, Control[{{#2, 0.04}, 0, 0.1, 0.01, Appearance → "Open"}], ""} &, Association[pMap]]}],
DynamicModule[{counts = ConstantArray[0, censusGraphCounts[1]], countsList = ConstantArray[ConstantArray[0, censusGraphCounts[1]], 2],
  singleAnalysis = "Run Sim.", multipleAnalysis = "Run Sim."},
  Manipulate[visualiseEstimationGrid[1, generationTypes, modelTypes, n, Association[pMap], t, censusEquations,
    counts, countsList, singleAnalysis, multipleAnalysis], innerControls]]], outerControls]]

```

Subgraph size: 2 3 4

Generation type(s): Links Paths Triangles Stars Cliques

Model type(s): Links Paths Triangles Stars Cliques



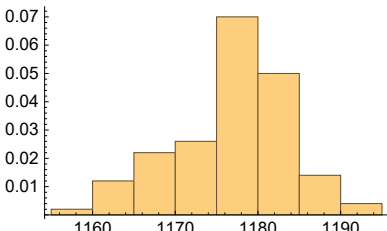
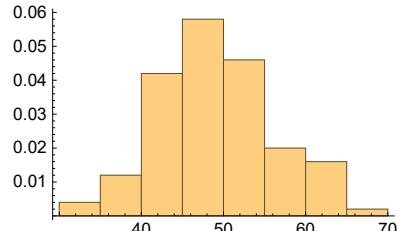
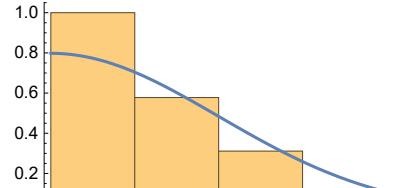
General: n = 50, t = 100

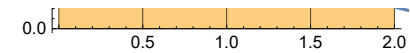
Single: Generate, Analyse

Multiple: Generate, Analyse

Links: pL = 0.04

Out[59]=

Subgraph			Total																												
Links	1 – pL	pL	1																												
Expected	1176.	49.	1225																												
Single	1175	50	<table><tr><td></td><td>p1</td></tr><tr><td>Max likelihood</td><td>0.0408163</td></tr><tr><td colspan="2">Goodness of fit: 1.</td></tr><tr><td></td><td>p1</td></tr><tr><td>Mean</td><td>0.040345</td></tr><tr><td>Std</td><td>0.00787127</td></tr><tr><td>Median</td><td>0.0398389</td></tr><tr><td>Quantiles (0.9,0.95)</td><td>0.026501 0.0282165 0.0539575 0.0569396</td></tr><tr><td>2σ intervals</td><td>0.0246024 0.0560875</td></tr></table>		p1	Max likelihood	0.0408163	Goodness of fit: 1.			p1	Mean	0.040345	Std	0.00787127	Median	0.0398389	Quantiles (0.9,0.95)	0.026501 0.0282165 0.0539575 0.0569396	2σ intervals	0.0246024 0.0560875										
	p1																														
Max likelihood	0.0408163																														
Goodness of fit: 1.																															
	p1																														
Mean	0.040345																														
Std	0.00787127																														
Median	0.0398389																														
Quantiles (0.9,0.95)	0.026501 0.0282165 0.0539575 0.0569396																														
2σ intervals	0.0246024 0.0560875																														
Multiple	<table><tr><td></td><td>Observed</td></tr><tr><td>Mean</td><td>1176.41</td></tr><tr><td>Std</td><td>6.69101</td></tr></table> 		Observed	Mean	1176.41	Std	6.69101	<table><tr><td></td><td>Observed</td></tr><tr><td>Mean</td><td>48.59</td></tr><tr><td>Std</td><td>6.69101</td></tr></table> 		Observed	Mean	48.59	Std	6.69101	<table><tr><td></td><td>p1</td></tr><tr><td>Max likelihood</td><td>0.0396653</td></tr><tr><td colspan="2">Cov: (0.000029834)</td></tr><tr><td></td><td>Observed Expected</td></tr><tr><td>Mean</td><td>0.76088 0.797885</td></tr><tr><td>Trimmed95</td><td>0.677283 0.716836</td></tr><tr><td>Trimmed85</td><td>0.557809 0.605616</td></tr><tr><td>Median</td><td>0.633382 0.67449</td></tr></table> 		p1	Max likelihood	0.0396653	Cov: (0.000029834)			Observed Expected	Mean	0.76088 0.797885	Trimmed95	0.677283 0.716836	Trimmed85	0.557809 0.605616	Median	0.633382 0.67449
	Observed																														
Mean	1176.41																														
Std	6.69101																														
	Observed																														
Mean	48.59																														
Std	6.69101																														
	p1																														
Max likelihood	0.0396653																														
Cov: (0.000029834)																															
	Observed Expected																														
Mean	0.76088 0.797885																														
Trimmed95	0.677283 0.716836																														
Trimmed85	0.557809 0.605616																														
Median	0.633382 0.67449																														



Future work should extend the list of possible subgraphs, deal with the correlations within the census, develop an *R*-package and apply the model to real-world data.

References

- [1] F. W. Takes and E. M. Heemskerk, "Centrality in the global network of corporate control," *Social Network Analysis and Mining*, vol. 6, pp. 1–18, 2016.
- [2] M. Fennema and E. M. Heemskerk, "When theory meets methods: the naissance of computer assisted corporate interlock research," *Global Networks*, vol. 1, pp. 81–104, 2018.
- [3] C. R. Shalizi and A. Rinaldo, "Consistency under sampling of exponential random graph models," *The Annals of Statistics*, vol. 41, pp. 508–535, 2013.
- [4] S. Chatterjee and P. Diaconis, "Estimating and understanding exponential random graph models," *The Annals of Statistics*, vol. 41, pp. 2428–2461, 2013.
- [5] S. Bhamidi, G. Bresler, and A. Sly, "Mixing time of exponential random graphs," *The Annals of Applied Probability*, vol. 21, pp. 2146–2170, 2011.
- [6] A. G. Chandrasekhar and M. O. Jackson, "Tractable and consistent random graph models," *ArXiv*, 2014. [Online]. Available: <https://arxiv.org/abs/1210.7375>.
- [7] A. G. Chandrasekhar and M. O. Jackson, "A network formation model based on subgraphs," *ArXiv*, 2016. [Online]. Available: <https://arxiv.org/abs/1611.07658>.
- [8] A. G. Chandrasekhar, "Econometrics of network formation," in *The Oxford Handbook of the Economics of Networks*, Y. Bramouille, A. Galeotti, and B. Rogers, Eds. Oxford University Press, 2016.
- [9] J. A. Davis and S. Leinhardt, "The structure of positive interpersonal relations in small groups," in *Sociological Theory in Progress*, J. Berger, M. Zelditch, and B. Anderson, Eds. Houghton-Mifflin, 1972.
- [10] P. W. Holland and S. Leinhardt, "A method for detecting structure in sociometric data," *American Journal of Sociology*, vol. 76, pp. 492–513, 1970.
- [11] P. W. Holland and S. Leinhardt, "Local structure in social networks," *Sociological Methodology*, vol. 7, pp. 1–45, 1976.
- [12] E. W. Weisstein, "Degree Sequence," *MathWorld-A Wolfram Web Resource*, 2018. [Online]. Available: <http://mathworld.wolfram.com/DegreeSequence.html>.
- [13] S. Chib and E. Greenberg, "Understanding the metropolis-hastings algorithm," *The American Statistician*, vol. 49, pp. 327–335, 1995.