

Socnet.se: An Open-Source C# Console Application for Direct Blockmodeling

Carl Nordlund^{1¶}, José Luis Estévez^{2,3}, Kristian Gade Kjellmann⁴,
Jesper Lindmarker¹, and Chandreyee Roy⁵

¹ Linköping University, Sweden ² University of Helsinki, Finland ³ Population Research Institute, Västtöliitto, Finland ⁴ Aalborg University, Denmark ⁵ Aalto University, Finland ¶ Corresponding author

DOI: [10.xxxxxx/draft](https://doi.org/10.xxxxxx/draft)

Software

- [Review](#)
- [Repository](#)
- [Archive](#)

Editor: [✉](#)

Submitted: 17 December 2025

Published: unpublished

License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](#))

Summary

The Socnet.se client is an open-source, cross-platform console application for direct blockmodeling of one-mode network data. Developed in C#, it unifies multiple blockmodeling approaches — structural, regular, and generalized equivalence — into a single tool, supporting both binary and valued networks through classical Hamming distance and correlation-based goodness-of-fit measures. The software includes specialized functionality for detecting various core-periphery structures and provides integrated isomorphism detection for blockmodels. Socnet.se operates via a simple scripting language and can be used as a standalone command-line tool, through script files, or as an external process called from R, Python, or similar environments. Given sufficient knowledge in C#, Socnet.se can easily be modified and extended with additional types of equivalences, ideal blocks, goodness-of-fit measures, and search heuristics, as well as additional data structures if needed, and the code may also be scavenged for the further development of other existing or future software solutions for direct blockmodeling. Socnet.se is available as pre-compiled binaries for Windows, Linux, and macOS at www.socnet.se, and can also be compiled directly from source.

Statement of Need

Building on the foundational ideas of Harrison White (Lorrain & White, 1971) on how to capture the notion of social roles in relational terms, direct blockmodeling is a pivotal method in network analysis for identifying how actors occupy similar structural positions in a network and how these positions relate to one another (Doreian et al., 2005). Unlike community detection or core-periphery identification, blockmodeling maintains neutrality toward the structures that might emerge, making it valuable for exploratory analysis of social, organizational, and other relational data. Starting off with a meaningful notion of actor equivalence, direct blockmodeling involves the systematic identification and grouping of actors that share such relational similarities, while simultaneously mapping out the overarching relational patterns within and between such subsets of equivalent actors (aka 'positions'). A well-fitted blockmodel, characterized by emerging blocks of relations within and between positions mirroring certain ideal counterparts, enables the effective reduction of a potentially intricate and complex network to a blockimage, capturing its underlying functional anatomy (Nordlund, 2020), allocating the actors to these various roles according to such relational similarities.

The current software ecosystem for direct blockmodeling is fragmented across multiple tools, each implementing different subsets of methods:

- Pajek (Batagelj & Mrvar, 2004) is the most established tool but only supports binary networks and is closed-source Windows software

- **The blockmodeling R package** (Žiberna, 2007) implements valued blockmodeling with novel ideal blocks but lacks correlation-based approaches
- **Specialized tools** for core-periphery detection (Borgatti & Everett, 2000) and various extensions to this exist as separate implementations in various languages

This fragmentation means researchers must learn multiple tools, convert data between formats, and often cannot replicate certain approaches. Moreover, recent methodological advances, such as dichotomization-free valued blockmodeling analysis (Nordlund, 2020), power-relational core-periphery models (Nordlund, 2018), and extensions with p-cores (Estévez & Nordlund, 2025), have so far lacked any accessible implementations.

Socnet.se addresses these gaps by providing:

1. **Unified approach:** Provides both binary and valued blockmodeling, using either Hamming distances or correlation-based measures
2. **Comprehensive ideal blocks:** Implements all classical blocks (null, complete, regular, row/column-regular, row/column-functional) plus several specialized blocks for core-periphery structures
3. **Multiple equivalence types:** Supports structural, regular, and generalized equivalence in a single tool
4. **Open and extensible:** MIT-licensed with object-oriented architecture designed for adding new ideal blocks, goodness-of-fit measures, and search algorithms
5. **Cross-platform:** Available as pre-compiled binaries for Windows, Linux, and macOS, as well as build scripts for respective platform

Table 1 below provides a comparison of features available in Pajek, Žiberna's blockmodeling R package, and Socnet.se.

Feature	Pajek	blockmodeling R package	Socnet.se
Equivalence types	Structural, regular, generalized	Structural, regular, generalized, homogeneity	Structural, regular, generalized
Binary networks	Yes (Hamming only)	Yes (Hamming + homogeneity)	Yes (Hamming + correlation)
Valued networks	No (must dichotomize)	Yes (novel valued blocks)	Yes (correlation-based)
Borgatti-Everett-style Core-periphery models	No	No	Yes (multiple variants)
Open source	No	Yes (GPL-2/3)	Yes (MIT)
Platforms	Windows only	Win/Linux/macOS	Win/Linux/macOS

Table 1: Comparison of direct blockmodeling software features for one-mode networks.

The target audience includes social network researchers, computational social scientists, and anyone analyzing relational data who needs flexible, theory-driven methods for uncovering latent structural patterns.

Key Features

Socnet.se provides comprehensive functionality for direct blockmodeling:

Blockmodeling Approaches:

71 ▪ Structural equivalence: Actors with identical tie patterns
 72 ▪ Regular equivalence: Actors with ties to equivalent others
 73 ▪ Generalized equivalence: Flexible combinations of ideal blocks
 74 ▪ Pre-specified structure search: Test networks against hypothesized models
 75 ▪ Pre-specified partitions: Test models against hypothesized partitions

76 **Ideal Block Types:**

77 ▪ Classical blocks: null, complete, regular, row/column-regular, row/column-functional
 78 ▪ Density blocks: exact density, minimum density, Ucinet-style density (Borgatti & Everett,
 79 2000)
 80 ▪ Core-periphery blocks: p-cores, peripheral dependency and/or core dominance
 81 ▪ Do-not-care blocks: Allow any pattern without penalty

82 **Goodness-of-Fit Measures:**

83 ▪ Hamming distance (inconsistency counts) for binary networks (Doreian et al., 2005)
 84 ▪ Weighted point-biserial correlation (Nordlund, 2020) for valued networks (also applicable
 85 to binary)

86 **Search Algorithms:**

87 ▪ Exhaustive search for small networks
 88 ▪ localopt: Local optimization with optional actor switching
 89 ▪ ljubljana: Semi-stochastic depth/width hybrid search
 90 ▪ Configurable parameters: number of random tests for starting partitions, number of
 91 restarts, iteration limits, minimum cluster sizes, timeout settings

92 **Core-Periphery Detection:**

93 ▪ Built-in coreperi() function with shortcut access to multiple core-periphery models
 94 ▪ Classical Borgatti-Everett model (Borgatti & Everett, 2000)
 95 ▪ Power-relational varieties (Nordlund, 2018): peripheral dependency and/or core domi-
 96 nance
 97 ▪ Extensions with p-cores and adjusted density blocks (Estévez & Nordlund, 2025)

98 **Additional Functionality:**

99 ▪ Automatic detection and filtering of isomorphic solutions and blockimages
 100 ▪ Partially constrained blockimages (mixing single or multiple ideal blocks in a blockimage)
 101 ▪ Incremental expansion of blockimage templates based on previous optimal findings (using
 102 the biextend() command; see online documentation)
 103 ▪ Data transformations: dichotomization, symmetrization, rescaling
 104 ▪ Density table generation for partitions
 105 ▪ Matrix and edgelist file formats
 106 ▪ Integrated file handling and data management

107 Full documentation on how to use the above features, all available Socnet CLI commands,
 108 and a comprehensive quick-start guide are available at <https://socnet.se/>.

109 Research impact

110 Developed within the scope of the Network Dynamics of Ethnic integration project, a large
 111 Nordic research project funded by the NordForsk program on interdisciplinary research, the Soc-
 112 net.se client was first introduced in Estévez & Nordlund (2025), providing the implementations
 113 for the extensions to the traditional Borgatti-Everett approach to identifying core-periphery
 114 structures. Socnet.se has also been used on course modules in Social network analysis within
 115 the MSc program in Computational Social Science at Linköping university, Sweden, as well as
 116 the Research School in Computational Social Science. In ongoing research on the structure of

117 the contemporary world-system, the features of partially constrained blockimages are proven
118 to be particularly useful in an ongoing study on semiperipheral patterns.

119 Usage Example

120 Below exemplifies a typical workflow for doing structural equivalence blockmodeling of the
121 Little League TI data (Fine, 1987), as provided in the /example_data folder, when working
122 on the command-line interface of the Socnet.se client. A full specification of the scripting
123 language used by Socnet.se is available on the project website as well a quick-start user guide
124 exemplifying various use patterns. There is also an extensively commented replication script
125 (cli_script.txt) available in the /example_data folder of the repository.

```
126 # Make sure that the Socnet.se working directory is where the data file is:
127 > setwd("[path_to_example_data_folder]")
128
129 # Load network data
130 > loadmatrix(little_league_ti.txt, name=llti)
131 Stored structure 'llti_actors' (Actorset)
132 Stored structure 'llti' (Matrix)
133 Loading data structure: OK
134
135 # Create a 4-position blockimage with complete and null blocks (i.e. structural
136 # equivalence)
137 > bi4se = blockimage(size = 4, type = structural)
138 Stored structure 'bi4se' (BlockImage)
139
140 # Initialize search using ljubljana algorithm with Hamming distance
141 > bminit(network=llti, blockimage=bi4se, searchtype=ljubljana, method=hamming)
142 Initializing search...
143 Network: llti
144 Method: hamming
145 nbrrestarts: 50
146 maxiterations: 100
147 nbrrandomstart: 50
148 minnbrbetter: 5
149 Search heuristic: ljubljana
150 Blockimage: bi4se
151 minclustersize: 1
152 maxtime: 300000ms (timeout active)
153 Initialization seems to have gone ok!
154
155 # Execute search
156 > bmstart()
157 Execution time (ms):229
158 Nbr tests done:25853
159 Stored structure 'bm_llti_bi4se_0' (BlockModel)
160 Goodness-of-fit (1st BlockModel): 20 (hamming)
161
162 # In this example, using the Little League TI data, we arrive at a solution
163 # named 'bm_llti_bi4se_0'
164
165 # View results
166 > bmview(blockmodel = bm_llti_bi4se_0)
167 Blockmodel:
168 +-----+
```

```

169 |\ XX|  X| | |      0_John_6
170 |X\  |X  | | |      0_Jerry_10
171 | X\  |X  |X| |      0_Darrin_11
172 |XX \ |X  | | |      0_Ben_12
173 |X   \|  X| | |      0_Arnie_13
174 +-----+
175 |      |\XXX| | |      1_Ron__1
176 |  X  |X\X | | |      1_Frank_3
177 |      |XX\ |X| |      1_Boyd_4
178 |      |XXX\| | |      1_Tim__5
179 +-----+
180 |  X  |XX  |\| |      2_Tom__2
181 +-----+
182 |      |      |X|\XX|      3_Jeff_7
183 |      |      |X|X\X|      3_Jay__8
184 |      |      |X|XX\|      3_Sandy_9
185 +-----+
186 Blockimage:
187         P0      P1      P2      P3
188 P0      nul     nul     nul     nul
189 P1      nul     com     nul     nul
190 P2      nul     nul     nul     nul
191 P3      nul     nul     com     com
192 Goodness-of-fit: 20 (hamming)
193
194 # Extract blockmodel matrix, naming it 'bm_llti_se4'
195 > bmextract(bm_llti_bi4se_0, type = matrix, outname = bm_llti_se4)
196 Stored structure 'bm_llti_se4_actors' (Actorset)
197 Stored structure 'bm_llti_se4' (Matrix)
198 Stored structure 'bm_llti_se4_ideal' (Matrix)
199
200 # Save the blockmodel matrix to file
201 > save(name = bm_llti_se4, file = bm_llti_se4.txt)
202 Matrix 'bm_llti_se4' saved: bm_llti_se4.txt
203
204 For core-periphery detection, the process is simplified:
205
206 > loadmatrix(zachary.txt)
207 > coreperi(zachary, ljubljana)
208 > bmview(bm_zachary_cp_0)
209
210 # For non-optional arguments, argument names can be omitted if stated in the
211 # specified order.
212
213 Additional examples including valued network analysis and power-relational core-periphery
214 detection are provided in the comprehensive documentation at https://socnet.se/. The
215 /example_data/ folder also contains an extensively commented script - cli_script.txt - that
216 describes how typical analytical workflows look like.

```

Software Design

Developed explicitly with compactness and self-sufficiency in mind, with no other dependencies than the standard libraries of .NET 8.0, the Socnet.se client builds on standard object-oriented principles to allow for future extensions. Specifically, the codebase has been prepared with three types of extensions in mind: implementing new ideal blocks, new goodness-of-fit measures, and new search heuristics. While a short introduction to extending Socnet.se follows below,

220 more detailed technical instructions are provided in the CONTRIBUTING.txt file that is available
221 in the Socnet.se repository.

222 Ideal blocks

223 Each ideal block is implemented as a separate class that inherits from a shared abstract class
224 (`_Block.cs`), the latter specifying the necessary properties and virtual methods of all ideal
225 blocks. To add a new ideal block, a new class is created (in the `/DataLibrary/Blocks/` folder)
226 that implements the block-specific properties and goodness-of-fit method(s). To make a new
227 ideal block available when constructing blockimages, the ideal block is finally registered in the
228 `BlockmodelingConstants.cs` class. An extensively commented template (`_exampleBlock.cs`)
229 is provided in the `/DataLibrary/Blocks/` folder to facilitate this process.

230 Goodness-of-fit measures

231 Socnet.se currently includes two measures (Hamming and weighted correlation-based), but
232 additional measures can be implemented as well. Each measure is defined as a method in
233 `Blockmodeling.cs` with a specific method signature: a new goodness-of-fit method should
234 thus follow the same signature as the `Blockmodeling.binaryHamming(...)` method. The new
235 measure must then be implemented for all ideal block classes that should support it, which also
236 must be declared in `BlockmodelingConstants.cs` and added as an option in the `bminit()`
237 command and the `InitializeSearch()` method.

238 Search heuristics

239 New heuristics and algorithms for finding optimal partitions can also be implemented in
240 Socnet.se. Each search approach is implemented as a separate method in `Blockmodeling.cs`
241 and initiated as a method delegate. All these search methods must be registered in
242 `BlockmodelingConstants.cs` and exposed as an option in the `bminit()` command.

243 Integration with Other Software

244 Socnet.se can be integrated into existing workflows:

- 245 ■ **R integration:** The GitHub repository includes `client_socnetse.R`, a library of wrapper
246 functions for calling Socnet.se from R. Make sure that the Socnet client is first installed
247 properly. See the `demo_socnetse.R` file for a demonstration on how to use the R wrapper.
- 248 ■ **Python integration:** Also possible to invoke as an external process from Python
- 249 ■ **Script files:** Batch processing via script files containing Socnet.se commands
- 250 ■ **Standalone use:** Direct command-line interaction for exploratory analysis

251 The software reads and writes standard text-based formats (tab-separated matrices, edgelist),
252 simplifying compatibility with common network analysis tools like igraph, networkx, Pajek, and
253 Ucinet.

254 Related Software

255 Socnet.se complements existing blockmodeling tools:

- 256 ■ **Pajek** (Batagelj & Mrvar, 2004): Comprehensive network analysis software with binary
257 blockmodeling; Socnet.se extends this with valued network support and correlation-based
258 methods
- 259 ■ **blockmodeling R package** (Žiberna, 2007): Implements valued blockmodeling with
260 novel ideal blocks; Socnet.se provides correlation-based alternatives and core-periphery
261 specialization

- **Core-periphery tools:** Various implementations (Kojaku & Masuda, 2018; Schoch, 2023) focus specifically on the traditional core-periphery model of Borgatti-Everett (Borgatti & Everett, 2000); Socnet.se integrates this within a broader blockmodeling framework and provides a variety of extended core-periphery models

Acknowledgments

This research was supported by NordForsk through funding to the Network Dynamics of Ethnic Integration (project number 105147). We thank colleagues in the research project and students in the Masters programme in Computational Social Science at Linköping University, Sweden, for valuable feedback.

AI Usage Disclosure

No generative AI tools were used in the development of the software code. AI assistance (Claude) was sporadically used for proofreading and improving readability of the manuscript text.

References

- Batagelj, V., & Mrvar, A. (2004). Pajek — analysis and visualization of large networks. In M. Jünger & P. Mutzel (Eds.), *Graph drawing software* (pp. 77–103). Springer. https://doi.org/10.1007/978-3-642-18638-7_4
- Borgatti, S. P., & Everett, M. G. (2000). Models of core/periphery structures. *Social Networks*, 21(4), 375–395. [https://doi.org/10.1016/S0378-8733\(99\)00019-2](https://doi.org/10.1016/S0378-8733(99)00019-2)
- Doreian, P., Batagelj, V., & Ferligoj, A. (2005). *Generalized blockmodeling*. Cambridge University Press. <https://doi.org/10.1017/CBO9780511584176>
- Estévez, J. L., & Nordlund, C. (2025). Revising the borgatti-everett core-periphery model: Inter-categorical density blocks and partially connected cores. *Social Networks*, 81, 31–51. <https://doi.org/10.1016/j.socnet.2024.11.002>
- Fine, G. A. (1987). *With the boys: Little league baseball and preadolescent culture*. University of Chicago Press.
- Kojaku, S., & Masuda, N. (2018). Core-periphery structure requires something else in the network. *New Journal of Physics*, 20(4), 043012. <https://doi.org/10.1088/1367-2630/aab547>
- Lorrain, F., & White, H. C. (1971). Structural equivalence of individuals in social networks. *The Journal of Mathematical Sociology*, 1(1), 49–80. <https://doi.org/10.1080/0022250X.1971.9989788>
- Nordlund, C. (2018). Power-relational core-periphery structures: Peripheral dependency and core dominance in binary and valued networks. *Network Science*, 6(3), 348–369. <https://doi.org/10.1017/nws.2018.15>
- Nordlund, C. (2020). Direct blockmodeling of valued and binary networks: A dichotomization-free approach. *Social Networks*, 61, 128–143. <https://doi.org/10.1016/j.socnet.2019.10.004>
- Schoch, D. (2023). *netUtils: A collection of tools for network analysis*. <https://doi.org/10.32614/CRAN.package.netUtils>
- Žiberna, A. (2007). Generalized blockmodeling of valued networks. *Social Networks*, 29(1), 105–126. <https://doi.org/10.1016/j.socnet.2006.04.002>