

¹ MatrixBandwidth.jl: Fast algorithms for matrix bandwidth minimization and recognition

³ Luis M. B. Varona  ^{1,2,3}

⁴ Department of Politics and International Relations, Mount Allison University ² Department of Mathematics and Computer Science, Mount Allison University ³ Department of Economics, Mount Allison University

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

Software

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

Editor: [Open Journals](#) 

Reviewers:

- [@openjournals](#)

Submitted: 01 January 1970

Published: unpublished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](#)).
¹⁸
¹⁹
²⁰
²¹

⁷ Summary

⁸ The *bandwidth* of an $n \times n$ matrix A is the minimum non-negative integer $k \in \{0, 1, \dots, n-1\}$ such that $A_{i,j} = 0$ whenever $|i - j| > k$. Reordering the rows and columns of a matrix to reduce its bandwidth has many practical applications in engineering and scientific computing: it can improve performance when solving linear systems, approximating partial differential equations, optimizing circuit layout, and more ([Mafteiu-Scai, 2014](#)). There are two variants of this problem: *minimization*, which involves finding a permutation matrix P such that the bandwidth of PAP^T is minimized, and *recognition*, which entails determining whether there exists a permutation matrix P such that the bandwidth of PAP^T is less than or equal to some fixed non-negative integer (an optimal permutation that fully minimizes the bandwidth of A is not required). Accordingly, [MatrixBandwidth.jl](#) offers fast algorithms for matrix bandwidth minimization and recognition. Julia's ([Bezanson et al., 2017](#)) combination of easy syntax and high performance, along with its rapidly growing ecosystem for scientific computing, made it the ideal language of choice for this project.
¹⁰
¹¹
¹²
¹³
¹⁴
¹⁵
¹⁶
¹⁷
¹⁸
¹⁹
²⁰
²¹

²¹ Example

²² Consider the following 60×60 sparse matrix with initial bandwidth 51:

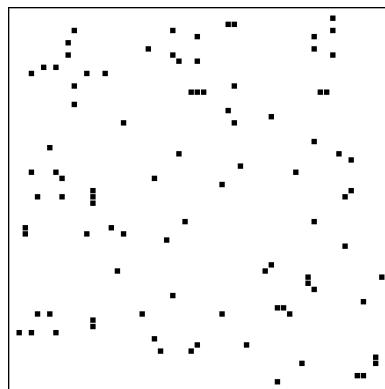


Figure 1: Original 60×60 matrix with bandwidth 51

²³ MatrixBandwidth.jl can both recognize whether the minimum bandwidth of A is less than or equal to some fixed integer ([Figure 2](#)) and actually minimize the bandwidth of A ([Figure 3](#)):
²⁴

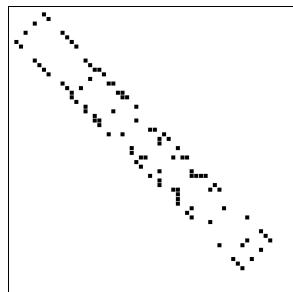


Figure 2: The matrix with bandwidth recognized as ≤ 6 via the Del Corso–Manzini algorithm

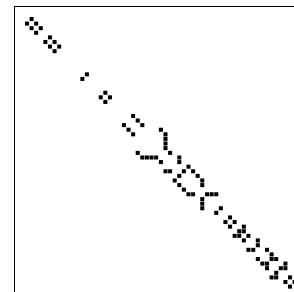


Figure 3: The matrix with bandwidth minimized to 5 via the Gibbs–Poole–Stockmeyer algorithm

25 Note that since Gibbs–Poole–Stockmeyer is a heuristic algorithm, 5 may not be the *true*
 26 minimum bandwidth of A , but it is likely close.

27 Algorithms

28 The following matrix bandwidth reduction algorithms are currently available:

- 29 ▪ Minimization
 - 30 – Exact
 - 31 * Caprara–Salazar–González ([Caprara & Salazar–González, 2005](#))
 - 32 * Del Corso–Manzini ([Del Corso & Manzini, 1999](#))
 - 33 * Del Corso–Manzini with perimeter search ([Del Corso & Manzini, 1999](#))
 - 34 * Saxe–Gurari–Sudborough ([Gurari & Sudborough, 1984; Saxe, 1980](#))
 - 35 * Brute-force search
 - 36 – Heuristic
 - 37 * Gibbs–Poole–Stockmeyer ([Gibbs et al., 1976](#))
 - 38 * Cuthill–McKee ([Cuthill & McKee, 1969](#))
 - 39 * Reverse Cuthill–McKee ([Cuthill & McKee, 1969; George, 1971](#))
 - 40 ▪ Recognition
 - 41 – Caprara–Salazar–González ([Caprara & Salazar–González, 2005](#))
 - 42 – Del Corso–Manzini ([Del Corso & Manzini, 1999](#))
 - 43 – Del Corso–Manzini with perimeter search ([Del Corso & Manzini, 1999](#))
 - 44 – Saxe–Gurari–Sudborough ([Gurari & Sudborough, 1984; Saxe, 1980](#))
 - 45 – Brute-force search

46 Recognition algorithms determine whether any row-and-column permutation of a matrix induces
 47 bandwidth less than or equal to some fixed integer. Exact minimization algorithms always
 48 guarantee optimal orderings to minimize bandwidth, while heuristic minimization algorithms
 49 produce near-optimal solutions more quickly. Metaheuristic minimization algorithms employ
 50 iterative search frameworks to find better solutions than heuristic methods (albeit more slowly);
 51 no such algorithms are already implemented, but several (e.g., simulated annealing) are
 52 currently under development.

53 Thus far, the Caprara–Salazar–González algorithms are the only ones implemented that require
 54 integer linear programming; it is for these that the `JuMP.jl` package ([Lubin et al., 2023](#)) is
 55 included as a dependency.

56 Statement of need

57 Many matrix bandwidth reduction algorithms exist in the literature, but implementations in
 58 the open-source ecosystem are scarce, with those that do exist primarily tackling older, less
 59 efficient algorithms. The `Boost` libraries in C++ ([Lumsdaine et al., 2001](#)), the `NetworkX`

library in Python ([NetworkX Developers, 2025](#)), and the MATLAB standard library ([MATLAB Developers, 2025](#)) all only implement the aforementioned reverse Cuthill–McKee algorithm from 1971. In Julia, the only other relevant packages identified by the author are `BandedMatrices.jl` ([JuliaLinearAlgebra Developers, 2016](#)) and `SymRCM.jl` ([Krysl, 2020](#)), both of which also only implement reverse Cuthill–McKee as their sole bandwidth reduction algorithm.

Furthermore, not enough attention is given to recognition algorithms or exact minimization algorithms. Although more performant modern alternatives are often neglected, at least reverse Cuthill–McKee is a widely implemented method of approximating a minimal bandwidth ordering (as noted above). However, no such functionality for recognition or exact minimization is widely available, requiring researchers with such needs to fully re-implement these algorithms themselves.

These two gaps in the ecosystem not only make it difficult for researchers to benchmark and compare new proposed algorithms but also preclude the application of the most performant modern algorithms in real-life industry settings. `MatrixBandwidth.jl` aims to bridge this gap by presenting a unified interface for matrix bandwidth reduction algorithms in Julia.

Research applications

The author either has used or is using `MatrixBandwidth.jl` to do the following:

- Develop a new polynomial-time algorithm for “bandwidth $\leq k$ ” recognition efficient for both small and large k , and benchmarking it against other approaches ([Gurari & Sudborough, 1984](#); [Saxe, 1980](#))
- Speed up k -coherence checks of quantum states in many cases by confirming that the density matrix’s minimum bandwidth is greater than k ([Johnston et al., 2025](#))
- Compute the spectral graph property of “ S -bandwidth” ([Johnston & Plosker, 2025](#)) via the `SDiagonalizability.jl` package ([Varona et al., 2025](#)), which depends critically on `MatrixBandwidth.jl` for bandwidth recognition
- Investigate the precise performance benefits of reducing the propagation graph’s bandwidth when training a recurrent neural network, building on Balog et al. ([2019](#))

The first three use cases rely on the recognition and exact minimization functionality unique to `MatrixBandwidth.jl` (indeed, they largely motivated the package’s development). The last (ongoing) research project *could* be facilitated by `SymRCM.jl` instead, but the author intends to use more performant metaheuristic minimization algorithms currently under development when producing the final computational results, as well as use recognition algorithms to minimize bandwidth to various target levels when quantifying performance improvements.

Limitations

Currently, `MatrixBandwidth.jl`’s core functions generically accept any input of the type `AbstractMatrix{<:Number}`, not behaving any differently when given sparsely stored matrices (e.g., from the `SparseArrays.jl` standard library package). Capabilities for directly handling graph inputs (aiming to reduce the matrix bandwidth of a graph’s adjacency) are also not available. Given that bandwidth reduction is often applied to sparse matrices and graphs, this will be addressed in future releases.

Moreover, many of the algorithms only apply to structurally symmetric matrices (i.e., those whose nonzero pattern is symmetric). However, this is a limitation of the algorithms themselves, not the package’s implementation. Future releases with metaheuristic algorithms will include more methods that accept structurally asymmetric inputs.

104 Conflict of interests

105 The author declares no conflict of interest.

106 Acknowledgements

107 I owe much to my research supervisors—Nathaniel Johnston, Sarah Plosker, and Craig
108 Brett—for supporting and guiding me in my work. I would also like to thank Liam Keliher, Peter
109 Lelièvre, and Marco Cognetta for useful discussions. Finally, credit for MatrixBandwidth.jl's
110 telepathic-cat-and-turtle logo goes to Rebekka Jonasson.

111 References

- 112 Balog, M., Merriënboer, B. van, Moitra, S., Li, Y., & Tarlow, D. (2019). *Fast Training of Sparse*
113 *Graph Neural Networks on Dense Hardware*. <https://doi.org/10.48550/arXiv.1906.11786>
- 114 Bezanson, J., Edelman, A., Karpinski, S., & Shah, V. B. (2017). Julia: A Fresh Approach to
115 Numerical Computing. *SIAM Review*, 59(1), 65–98. <https://doi.org/10.1137/141000671>
- 116 Caprara, A., & Salazar-González, J.-J. (2005). Laying Out Sparse Graphs with Provably
117 Minimum Bandwidth. *INFORMS Journal on Computing*, 17(3), 356–373. <https://doi.org/10.1287/ijoc.1040.0083>
- 118 Cuthill, E., & McKee, J. (1969). Reducing the bandwidth of sparse symmetric matrices.
119 *Proceedings of the 24th National Conference of the ACM*, 157–172. <https://doi.org/10.1145/800195.805928>
- 120 Del Corso, G. M., & Manzini, G. (1999). Finding Exact Solutions to the Bandwidth Minimiza-
121 tion Problem. *Computing*, 62, 189–203. <https://doi.org/10.1007/s006070050002>
- 122 George, J. A. (1971). *Computer Implementation of the Finite Element Method* [PhD thesis,
123 Department of Computer Science, Stanford University]. <https://apps.dtic.mil/sti/tr/pdf/AD0726171.pdf>
- 124 Gibbs, N. E., Poole, W. G., Jr., & Stockmeyer, P. K. (1976). An Algorithm for Reducing the
125 Bandwidth and Profile of a Sparse Matrix. *SIAM Journal on Numerical Analysis*, 13(2),
126 236–250. <https://doi.org/10.1137/0713023>
- 127 Gurari, E. M., & Sudborough, I. H. (1984). Improved dynamic programming algorithms
128 for bandwidth minimization and the MinCut Linear Arrangement problem. *Journal of*
129 *Algorithms*, 5(4), 531–546. [https://doi.org/10.1016/0196-6774\(84\)90006-3](https://doi.org/10.1016/0196-6774(84)90006-3)
- 130 Johnston, N., Moein, S., & Plosker, S. (2025). The factor width rank of a matrix. *Linear*
131 *Algebra and Its Applications*, 716, 32–59. <https://doi.org/10.1016/j.laa.2025.03.016>
- 132 Johnston, N., & Plosker, S. (2025). Laplacian $\{-1,0,1\}$ - and $\{-1,1\}$ -diagonalizable graphs.
133 *Linear Algebra and Its Applications*, 704, 309–339. <https://doi.org/10.1016/j.laa.2024.10.016>
- 134 JuliaLinearAlgebra Developers. (2016). *BandedMatrices.jl: A Julia package for representing*
- 135 banded matrices. GitHub. <https://github.com/JuliaLinearAlgebra/BandedMatrices.jl>
- 136 Krysl, P. (2020). *SymRCM: Reverse Cuthill-McKee node-renumbering algorithm for sparse*
137 *matrices*. GitHub. <https://github.com/PetrKryslUCSD/SymRCM.jl>
- 138 Lubin, M., Dowson, O., Dias Garcia, J., Huchette, J., Legat, B., & Vielma, J. P. (2023).
139 JuMP 1.0: recent improvements to a modeling language for mathematical optimiza-
140 tion. *Mathematical Programming Computation*, 15, 581–589. <https://doi.org/10.1007/s12532-023-00239-3>

- 146 Lumsdaine, A., Lee, L.-Q., Siek, J. G., Gregor, D., & McGrath, K. D. (2001). *example/cuthill_mckee_ordering.cpp*. Boost v1.37.0 documentation. https://www.boost.org/doc/libs/1_37_0/libs/graph/example/cuthill_mckee_ordering.cpp
- 147
- 148
- 149 Mafteiu-Scai, L. O. (2014). The Bandwidths of a Matrix. A Survey of Algorithms. *Annals of West University of Timisoara - Mathematics and Computer Science*, 52(2), 183–223. <https://doi.org/10.2478/awutm-2014-0019>
- 150
- 151
- 152 MATLAB Developers. (2025). *symrcm - Sparse reverse Cuthill-McKee ordering - MATLAB*. MATLAB R2025b documentation. <https://www.mathworks.com/help/matlab/ref/symrcm.html>
- 153
- 154
- 155 NetworkX Developers. (2025). *Source code for networkx.utils.rcm*. NetworkX v3.5 documentation. https://networkx.org/documentation/stable/_modules/networkx/utils/rcm.html
- 156
- 157 Saxe, J. B. (1980). Dynamic-Programming Algorithms for Recognizing Small-Bandwidth Graphs in Polynomial Time. *SIAM Journal on Algebraic and Discrete Methods*, 1(4), 363–369. <https://doi.org/10.1137/0601042>
- 158
- 159
- 160 Varona, L. M. B., Johnston, N., & Plosker, S. (2025). *SDiagonalizability: A dynamic algorithm to minimize or recognize the S-bandwidth of an undirected graph*. GitHub. <https://github.com/GraphQuantum/SDiagonalizability.jl>
- 161
- 162