Generalized Network Dismantling (GND): Response to the blog post by Petter Holme

June 21, 2019

In this short report, we respond to the following blog post [1], written by Petter Holme (P.H.).

Summary

- The results from the PNAS paper on Generalized Network Dismantling[2] are reproducible with our code[3].
- The algorithm for eigenvector approximation that we used in the PNAS paper produces good partitions with the optimization settings of our paper.
- We clarify the relation between network dismantling and generalized network dismantling in more details.

We think that it is very inadequate to present our paper as an example of "non-reproducible research," and even more so as the *only* example discussed.

Yours sincerely,

Xiao-Long Ren and Nino Antulov-Fantulin

[1] https://petterhol.me/2019/05/17/reproducing-computational-studies-in-general-and-general-network-dismantling-in-particular/

[2] https://doi.org/10.1073/pnas.1806108116

[3] https://github.com/renxiaolong/Generalized-Network-Dismantling

1 Random numbers

P.H.'s comment: "as I looked at the code I noticed that they initialized the random number generator to the default seed (1) at every call of the function partitioning the graph. In other words, they did not really using random numbers, at least not as the praxis is in the field. After fixing that, the output started depending on the seed. Here is the figure (Fig. S5) that I aimed to reproduce. The (GND) line of the paper is (in black) and 10 runs of the my code with the handling of random numbers fixed (in color)."

Our reply: We have used pseudorandom numbers with C++11 according to the standard Mersenne Twister generator[4] (periodicity of $2^{19937} - 1$) with a default seed. This is our implementation of the proposed GND method from our paper, which is reproducible with our seed. Re-implementing our method in a different way (e.g. different seeds, random number generators) may lead to some variability in results, but this is not to be misinterpreted as a sign of non-reproducibility.

The choice of seed does not really matter, because in our paper we present convergence proofs, which are not dependent on the initial seed (see Figure 2(A)-(C) for an illustration). Finally, Fig. 2(D) shows that we do not necessarily have to wait for full convergence because we are getting excellent results even before that point.

PH's implementation[5] uses the routine ARPACK and, with this, a different random number generator to determine v_2. If one looks at the code (line 194) of ARPACK, one can find the default seed that is being used[6]. For symmetric matrices, ARPACK is using an algorithmic variant of the Lanczos process called the Implicitly Restarted Lanczos Method (IRLM). Interestingly, if one tries to re-implement ARPACK library or plays with its seed, "non-deterministic" (and, therefore, apparently non-reproducible) outputs may result. One can find dozens of posts on the Internet about such "non-deterministic" behaviour of ARPACK. For example, we have found that P.H.'s ARPACK implementation produces "non-deterministic" results with the Python3 version, see Fig 1.

Results from our paper were obtained by our PNAS GitHub code, which was compiled with the MCVC [7] C+11 compiler, and now we have additionally reproduced results with GCC compiler. Note that the MSVC compiler C+11 uses the Mersenne Twister generator and the GCC compiler uses multiplicative congruential pseudo-random number generator as a default one. Therefore, in order to have reproducible results across different platforms, one just needs to specify that "std::default_random_engine" should be "std::mt19937". We have tested our public GitHub code across different platforms and we are getting the same results if only "std::mt19937" is specified. We will add this detail in our description on GitHub.

[4] Matsumoto, M.; Nishimura, T. (1998). "Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator" (PDF). ACM Transactions on Modeling and Computer Simulation. 8 (1): 3–30

[5] `https://github.com/pholme/gnd`

[6] `https://github.com/scipy/scipy/blob/v0.15.1/scipy/sparse/linalg/eigen/arpack/ARPACK/SRC/cgetv0.f`

[7] `https://marketplace.visualstudio.com/items?itemName=ms-vscode.cpptools`

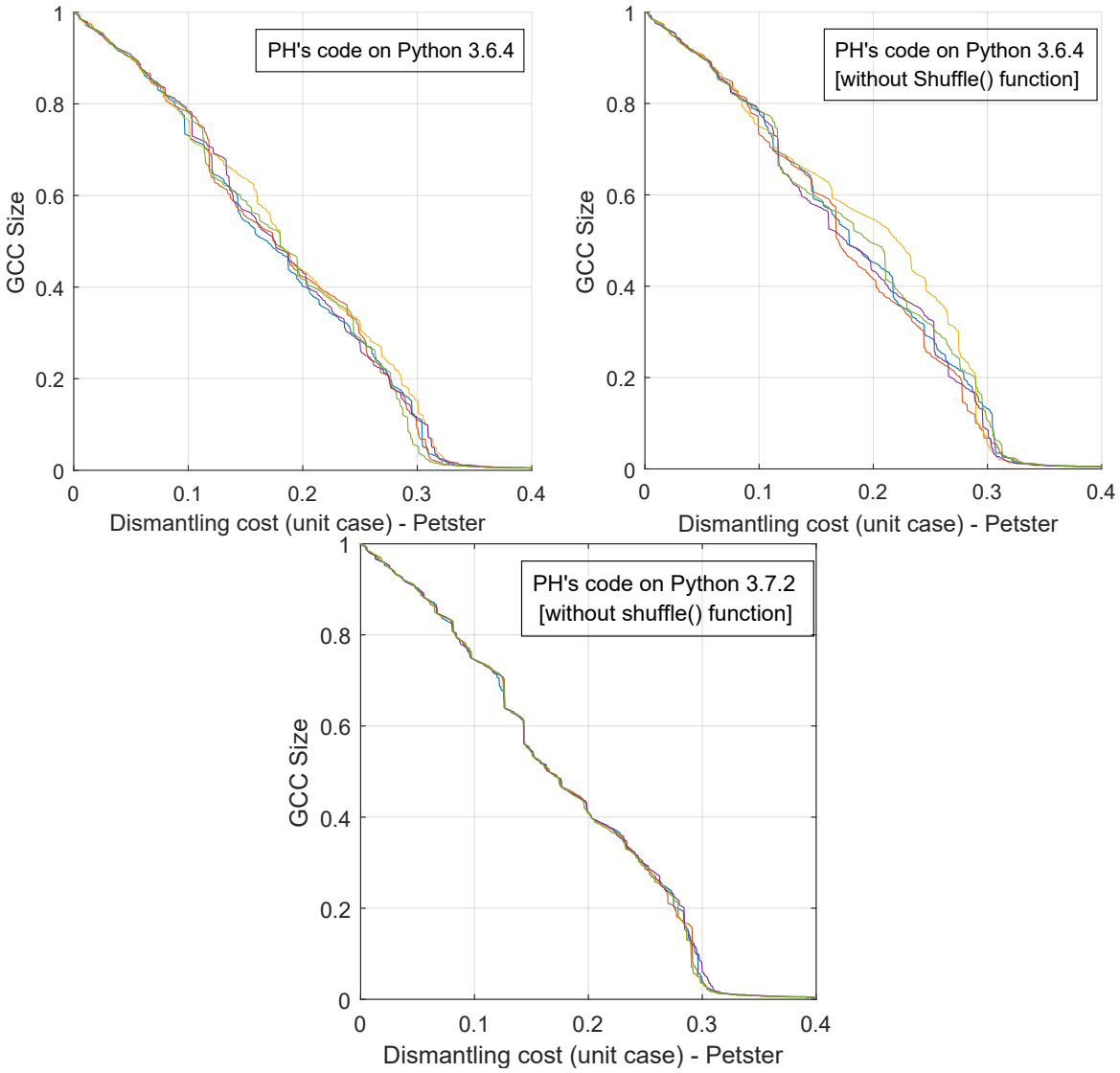

Figure 1: **Non-deterministic behaviour of P.H.'s code** on different python versions. On Python2 it produces deterministic results. However, it produces non-deterministic results on different versions of Python3.

2 Eigenvectors

P.H.'s comment: "It turns out that the authors solve the eigenvector problem not by calling a linear-algebra library routine, but their own power iteration. The core of linear algebra libraries (BLACS) is optimized to an incredible level."

Our reply: This is correct, however it is not a bug, but a feature. It is not necessary to determine exactly an eigenvector of the second smallest eigenvalue. Determining it is unnecessarily time-consuming and it is numerically inefficient. In our PNAS Appendix,

section 3, we provide a detailed analysis, why a linear combination of eigenvectors can (also) be close to the minimum of our objective function (Rayleigh quotient), which is easier to find. Variants of these statements are known for a long time [8]. In case of interest in any further details on this, see Appendix 5.1. of this response.

2.0.1 Convergence

In P.H.'s re-implementation (with different seeds) the largest variability occurs in the Petster network, so here we will study the performance of dismantling results with different seeds and iterations. In our paper, we have shown that, one can expect asymptotically good partitions with $O(\log(n)^{1+\epsilon})$, where $\epsilon > 0$. In our Github code, we have used $M = a * \log(n) * \sqrt{log(n)}$ iterations, where $a = 30$. In Fig. 2(A), we show the curves for $a = 30$ and 10 different seeds. Figs. 2(B) and (C) are for $200 * M$ and $500 * M$ iterations. We find convergence for $500 * M$. However, simulating for a longer time does not mean that the dismantling performance is better. To demonstrate this, let us measure the difference of dismantling performances

$$GCC_{500*M}(c) - GCC_M(c),$$

where $GCC_x(c)$ denotes the size of the Giant Connected Component obtained by GND algorithm with x spectral approximation iterations for the cost c. If this difference is positive on average, it means that dismantling with M iterations performs better. In order to assess the situation, we create a histogram of dismantling differences over all possible costs and different seeds. In 2(D), we observe that in the majority of cases there is no benefit of using 500 times more iterations for dismantling, on the contrary.

2.0.2 Comparison of ARPACK and GND partitions

In Fig. 3, we compare ARPACK [P.H. version] and GND [PNAS version] algorithms. We can observe that the ARPACK version is not outperforming, but takes approximately 25 times more time to compute.

Why there is a small difference in results w.r.t. code from P.H. that uses the ARPACK library? The difference comes from the fact that we can also take a linear combination of eigenvectors (rather than the eigenvector corresponding to the second smallest eigenvalue) as long as they are close to the minimum of the objective function (as we have explained in the Appendix of the main PNAS paper). This can be important when there is no clear spectral gap between second and third eigenvalue. Furthermore, due to the possibility of having an algebraic/geometric multiplicity of the Laplacian eigenvalues greater than 1, there is no surprise in having different eigenvectors that are considered equivalent in a partition of graphs w.r.t the graph-cutting objective. See Appendix 5.2. of this response for more experimental details.

[8] Alpert et. al. (1999), "Spectral partitioning with multiple eigenvectors", Discrete Applied Mathematics 90 1-3.

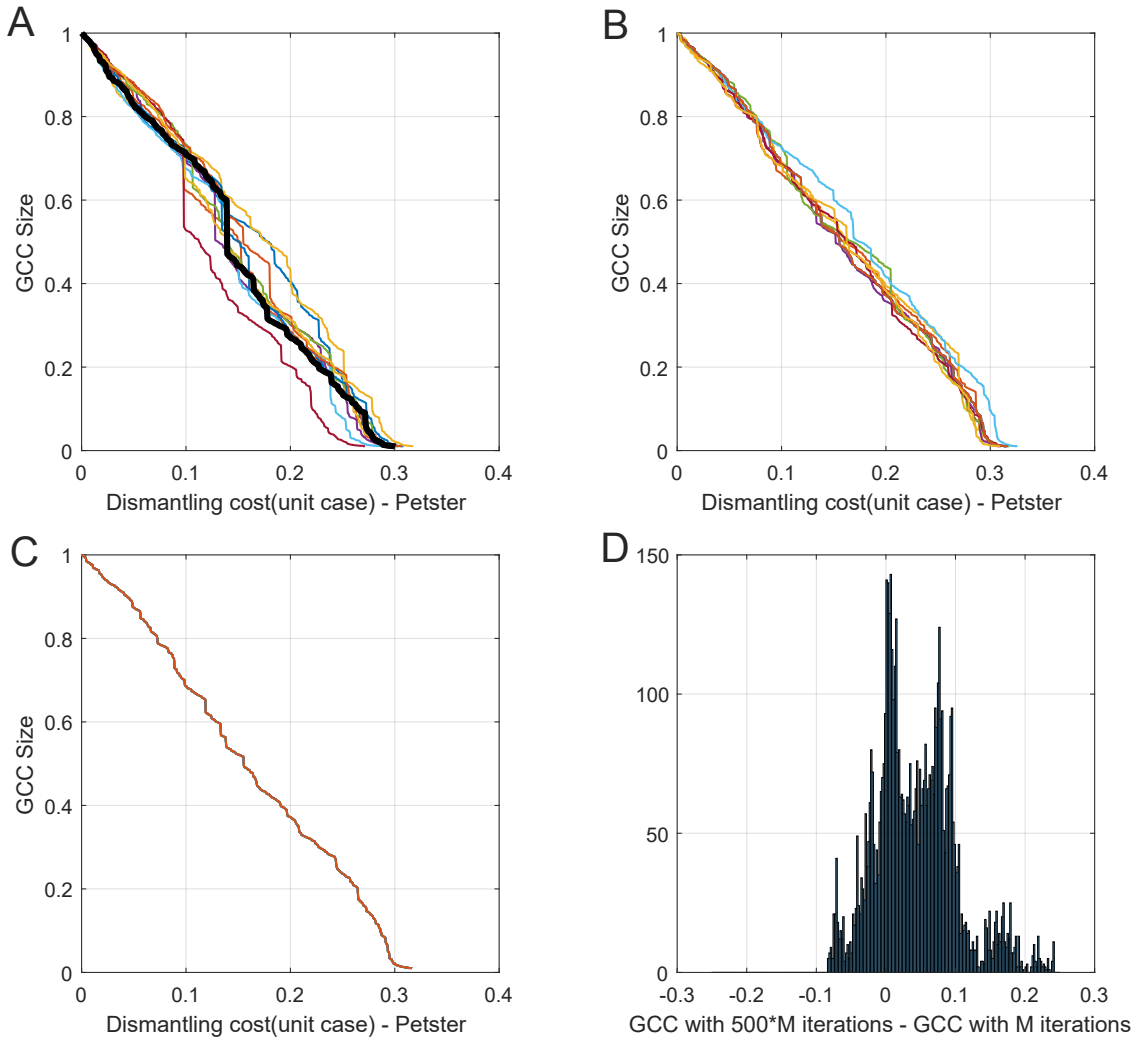

Figure 2: **Convergence of dismantling curves.** (A)-(C) show numerical results for 10 different seeds: (A) for M iterations, (B) for $200 * M$ iterations, (C) for $500 * M$ iterations, where $M = 30 * \log(n) * \sqrt{log(n)}$ (as in our PNAS paper). Even when seeds are chosen differently, all curves have converged to the same curve for $500 * M$, as seen in (C). (D) We measure the difference in the dismantling performance $GCC_{500*M}(c) - GCC_M(c)$, where $GCC_x(c)$ denotes the GND algorithm with x spectral approximation iterations for cost c. The graphic shows the histogram of differences in GCC over all possible costs for different seeds. We observe that the majority of differences is positive, which implies having a smaller GCC for the same cost for our settings, i.e. a better performance of the algorithm.

3 On the generalization

P. H.'s comments: "The authors code output nodes of the vertex cover set (step 2 above) in order of decreasing degree (for the degree-weight case) or increasing degree (for the unit-weight case). This is a bit curious because one can no longer get the unit-degree case by

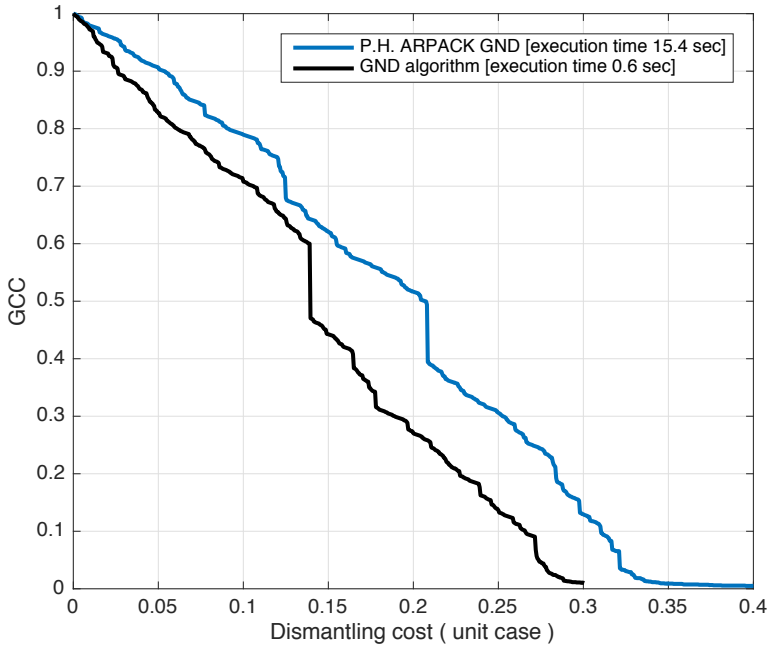

Figure 3: **Comparison of dismantling curve of P.H. ARPACK version [Python2 version] (in blue) and GND PNAS version (in black)** on the Petster network. The GND [PNAS variant] gets smaller GCC (better dismantling) over all costs and is 25 times faster. Results are reproducible over different platforms.

replacing degrees by one somewhere in the algorithm (so it is no longer generalized network dismantling)."

Our reply: Note that the term Generalized Network Dismantling refers to the novel problem formulation with non-unit costs. In our proposed method when $W = I$, the weighted partitioning problem becomes equal to the standard spectral partitioning problem and fine-tuning mechanism turns from weighted vertex cover to a normal vertex cover. Note, that the fine-tuning mechanism produces a set of nodes, with no ordering. However, it is allowed to use some topological properties for ordering the nodes from this set in the **network dismantling** problem, as long as the costs are still unit.

4 Conclusion

We have rechecked our code and can confirm that it reproduces the results reported in our PNAS paper. Insofar, it is inadequate to call our results non-reproducible.

However, we would like to stress that, P.H.'s code, may produce "non-deterministic" (i.e. platform-dependent) outputs. Our code can only produce "non-deterministic" results if one is changing seeds or random number generators. We have presented a simple way how our

code can become cross-platform reproducible. Most importantly, however, it does not matter for sufficiently many iterations. In our paper, we have presented proofs that the algorithm converges to some vector that minimises the cut objective function independently of the initial seed (as long as the sequence of pseudorandom numbers preserves certain statistical properties). Furthermore, we have also shown that even a linear combination of eigenvectors $v_2, ..., v_k$ is a good approximation if the $\lambda_2 \approx \lambda_3 \approx ... \approx \lambda_k < \lambda_{k+1}$. In such cases, our method requires much less iterations than it would take to get the accurate estimation of the v_2 (which P.H.s ARPACK method is trying to do). In other words, our algorithm may be more difficult to understand, but it is more efficient (25 times more faster on the Petster network).

Finally, the second eigenvector is the analytical solution for the relaxed formulation of the true dismantling objective function, which is NP-hard (see our PNAS Appendix, section 7 for more results). Furthermore, in our PNAS paper, we are not claiming that one can not produce better results then our proposed method for the generalized network dismantling problem. Actually, we think that there are multiple open research frontiers. If one can find better method, we encourage writing a new publication.

We would like to thank P.H. for the feedback on our paper, which allowed us to clarify a number of interesting points. However, we would like to ask to add a disclaimer and add a link to our response in the blog post or make some other measure which is going to reduce the potential unfair negative effects on our work.

5 Appendix

5.1 Theoretical analysis around eigenvector optimization

Recall that the idea behind spectral clustering is this: For a vector v with entries drawn from $\{-1, 1\}$, the quantity $\frac{v^T L v}{v^T v}$ corresponds to the cut-size. So we search the balanced vector x that minimizes this quantity in the relaxed case (that is, unit vectors that are orthogonal to $\vec{1}$ but have entries that are not necessarily in $\{-1, 1\}$) and this happens to be the normalized eigenvector corresponding to the second smallest eigenvalue (or some of those eigenvectors, if this eigenvalue has multiplicity > 1). Then we replace the positive entries by 1 and the negative entries by -1 and hope that this newly obtained vector (which I will now denote $\tilde{x}$), gives us a value $\frac{\tilde{x}^T L \tilde{x}}{\tilde{x}^T \tilde{x}}$ which is small too and hence gives us a small cut-size. This is a standard relaxation of the NP-hard problem. However, there is no guarantee [9] that this heuristics will preserve the optimality when passing from x to $\tilde{x}$. Furthermore, since any iterative method gives us only an approximation to the exact second eigenvector, we will get a value of $\frac{x^T L x}{x^T x}$, which is only ϵ-close to the minimum. Therefore it is just as good to find any balanced vector x' for which $\frac{x'^T L x'}{x'^T x'}$ is ϵ-close to the minimum (of course this will give us different results, sometimes better sometimes worse, but there is no reason why the so obtained partition would be generally better or worse). How far this vector x' is from the exact second eigenvector in any particular vector norm (or how large the hamming distance between $\tilde{x}$ and $\tilde{x'}$ is) is completely irrelevant. Now the important thing here is, that computing such a vector x' is computationally much less demanding than finding an approximation of the exact second eigenvector. And that is precisely the reason why computing the second eigenvector with any linear algebra library (no matter how optimized it is) would be inefficient. It is "too much". For example if L had several eigenvectors with eigenvalues close (but not equal) to the second, any iterative method would need to do many iterations to distinguish between the different eigenvectors and get to a vector that is close to the exact second eigenvector with respect to any particular norm. However, already after few iterations we would have a vector x for which $\frac{x^T L x}{x^T x}$ is ϵ-close to optimal.

5.2 ARPACK and GND spectral approximation

We will exemplify the source of the difference between ARPACK and GND approximations by using only one spectral partition without a fine-tuning mechanism. In Fig. 4, we calculate only the effect of spectral partitioning (without fine-tuning effect). In this scenario, we remove all the nodes i whose corresponding value in the estimated eigenvector is non-negative ($v_i^{(2)} \geq 0$) and has a neighbour j with a negative entry ($v_j^{(2)} < 0$). The ordering of the removed nodes follows their fixed index in the network. In Fig. 4(A), we show the spectral partitioning based on this strategy for ARPACK and GND with M iterations. Here we observe that the ARPACK partition is not better than the GND spectral partition.

Alternatively, we remove all the nodes i whose corresponding value in the estimated eigenvector is negative ($v_i^{(2)} < 0$) and has a neighbour j with a non-negative entry ($v_j^{(2)} \geq 0$). In

[9] Guattery et. al. (1998), "On the quality of spectral separators", SIAM J. Matrix Anal. Appl. 19-3

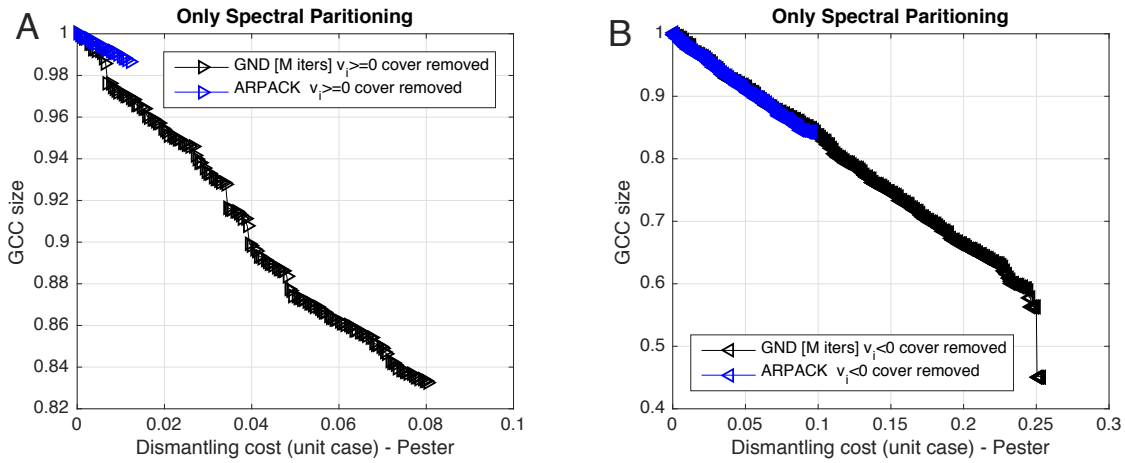

Figure 4: **GND and ARPACK spectral partition comparison.** Here we show the unit case dismantling results, where only eigenvectors were used for partitioning. Instead of calling vertex cover, we only remove nodes that are covering edges on the boundary between spectral partitions $v_i \geq 0$ and $v_i < 0$. Here we observe that the ARPACK partition is not better than the GND spectral approximation with M iterations.

Fig. 4(B), we show the spectral partitioning based on this strategy for ARPACK and GND with M iterations. Here we also observe that the ARPACK partition is not better than the GND spectral partition.