Nonlinear spectral clustering with C++ GraphBLAS

D. Pasadakis & O. Schenk – V. Vlacic & A.-J. Yzelman

Universitá della Svizzera Italiana, Institute of Computing, Lugano, Switzerland.
Computing Systems Lab, Huawei Zurich Research Center, Switzerland.

•D. Pasadakis, C. L. Alappat, O. Schenk, and G. Wellein, "Multiway p-spectral graph cuts on Grassmann manifolds," Machine Learning, vol. 111, Feb 2022, **DOI: 10.1007/s10994-021-06108-1**.

•A. N. Yzelman, D. Di Nardo, J. M. Nash, and W. J. Suijlen, "A C++ GraphBLAS: specification, implementation, parallelisation, and evaluation," 2020, **http://albert-jan.yzelman.net/PDFs/yzelman20.pdf**.

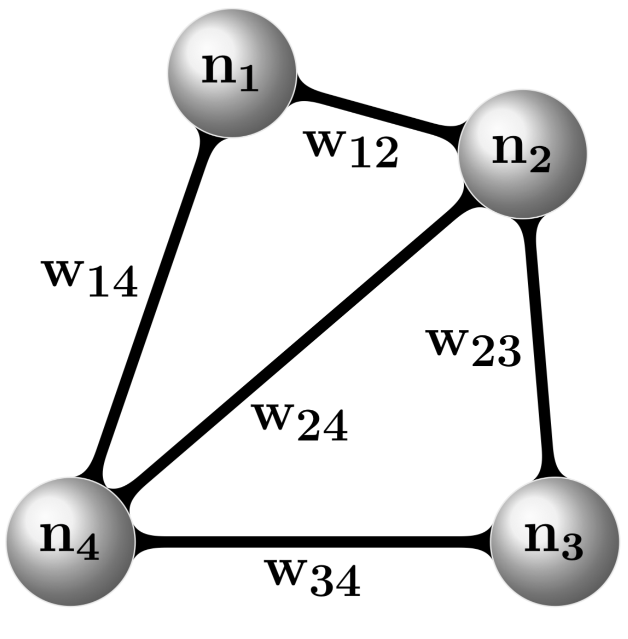
➜ We combine their benefits and furnish the first p-norm spectral clustering algorithm applicable to large-scale data for shared-memory machines.

Nonlinear spectral clustering

For a graph $\mathcal{G}(V, E, \mathbf{W})$ with n vertices and m edges:

- Scalar function $\phi_p(x) = |x|^{p-1}\text{sign}(x)$, and p-norm $\|\boldsymbol{u}\|_p = \sqrt[p]{\sum_{i=1}^{n} |u_i|^p}$ for $p \in (1, 2]$.
- p-Laplacian operator $(\boldsymbol{\Delta}_p \boldsymbol{u})_i = \sum_{j \in V} \boldsymbol{W}_{ij} \phi_p (u_i - u_j)$

$$\boldsymbol{W} = \begin{bmatrix} 0 & w_{12} & 0 & w_{14} \\ w_{12} & 0 & w_{23} & w_{24} \\ 0 & w_{23} & 0 & w_{34} \\ w_{14} & w_{24} & w_{34} & 0 \end{bmatrix}, \quad d_{ii} = \begin{bmatrix} \sum_j w_{1j} \\ \sum_j w_{2j} \\ \sum_j w_{3j} \\ \sum_j w_{4j} \end{bmatrix}$$



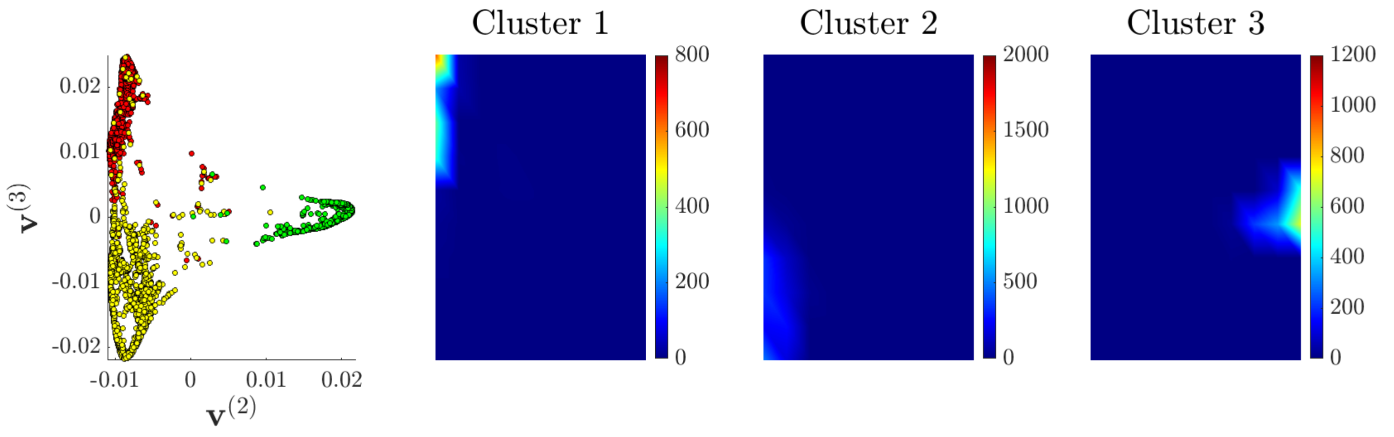
Calculate a multiway partition using k eigenvectors of $\boldsymbol{\Delta}_p \in \mathbf{R}^{n \times n}$.

2-spectral clustering

$$\min_{\boldsymbol{U} \in \mathbb{R}^{n \times k}} F_2(\boldsymbol{U}) = \text{Tr}\left(\boldsymbol{U}^\intercal \boldsymbol{L} \boldsymbol{U}\right), \quad \text{s.t.} \ \boldsymbol{U}^\intercal \boldsymbol{U} = \boldsymbol{I}.$$

Mutually orthogonal eigenvectors

	$\boldsymbol{u}_1$	…	$\boldsymbol{u}_k$
$\boldsymbol{U}_1$	u_{11}	…	u_{1k}
⋮	⋮	…	⋮
$\boldsymbol{U}_n$	u_{n1}	…	u_{nk}

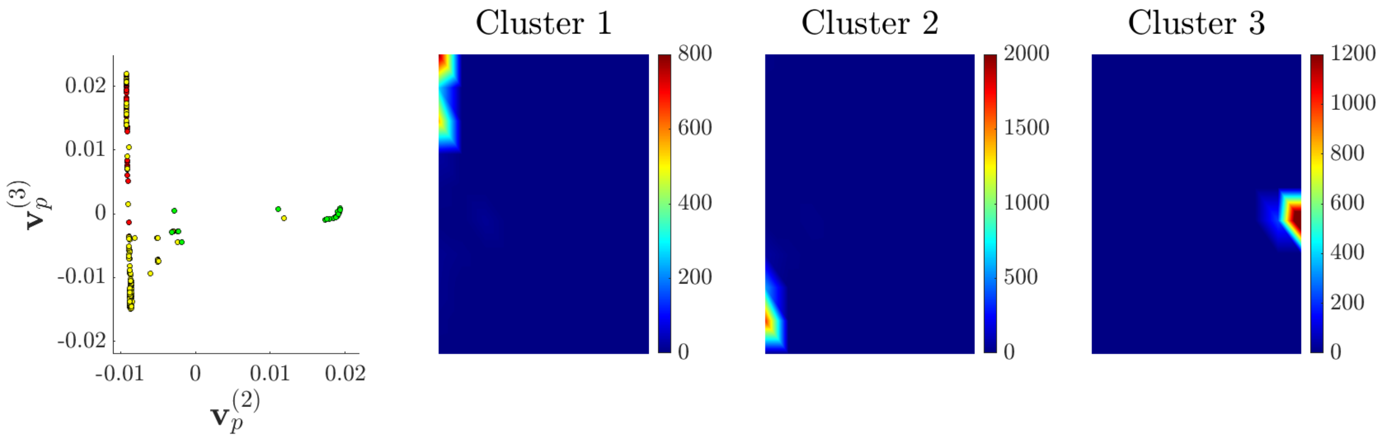


p-spectral clustering

$$\underset{\boldsymbol{U} \in \mathcal{G}r(k,n)}{\text{minimize}} \ F_p(\boldsymbol{U}) = \sum_{l}^{k} \sum_{i,j=1}^{n} \frac{w_{ij} |u_i^l - u_j^l|^p}{2 \|\boldsymbol{u}^l\|_p^p}.$$

Optimization on the manifold

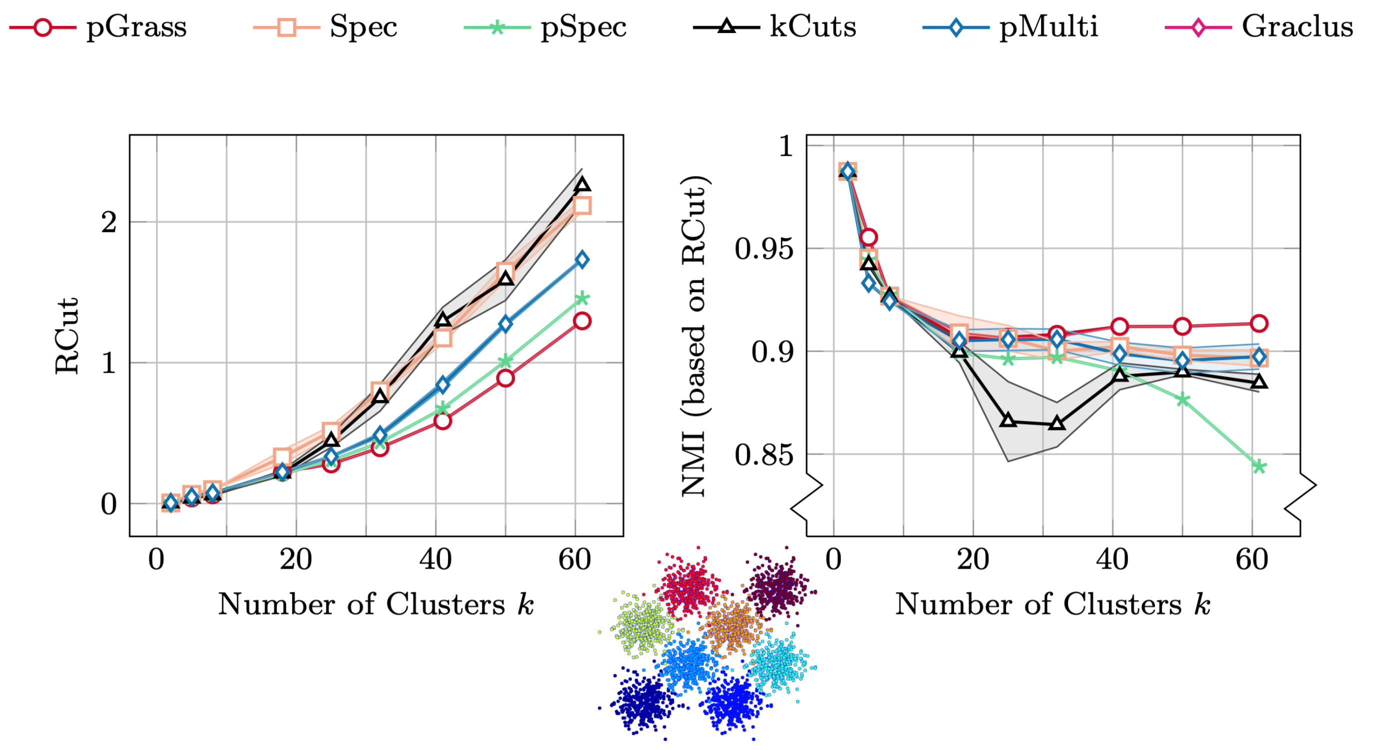
- Software package ROPTLIB. **github.com/whuang08/ROPTLIB**
- Newton's method on the Grassmann.



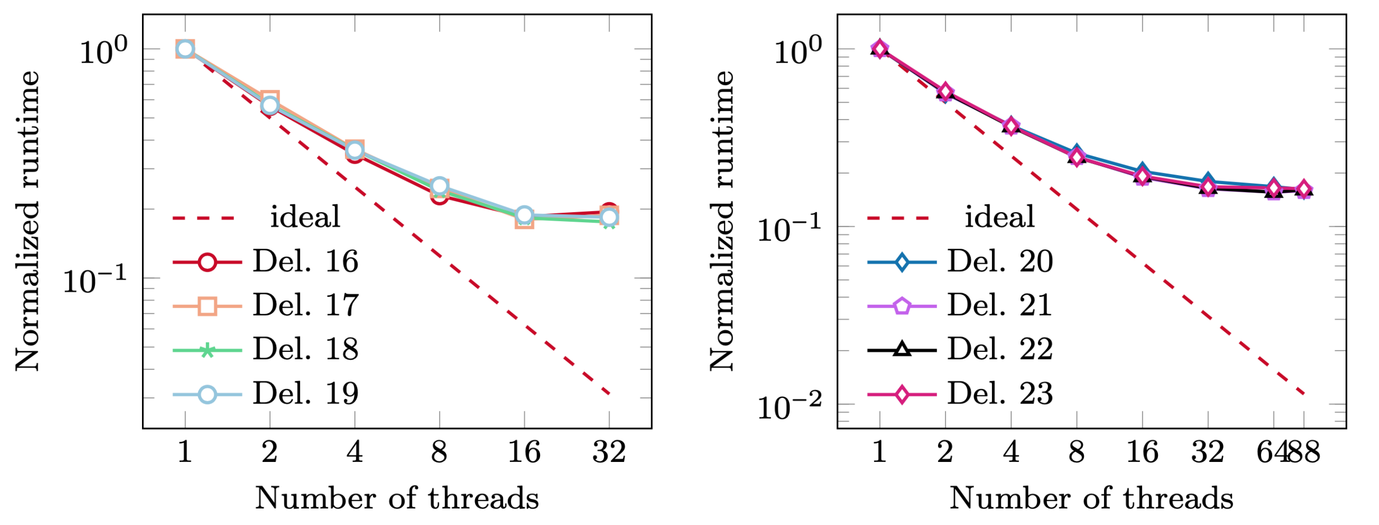
Numerical results

Gaussian datasets with an increasing number of clusters $k \in \{2, \ldots, 61\}$, nodes $n = 400 * k$, and edges $m \in \{4902, \ldots, 160312\}$.

✔ Superior results over competing spectral methods in terms of balanced graph cut metrics (RCut), and accuracy of classification (NMI).



Scalability studies in Delaunay graphs after expressing the objective, gradient, Hessian, and the k-means step, in C++ GraphBLAS terms.



➜ Up to 32 threads for the mid-scale cases of node size $n \in [2^{16}, 2^{19}]$, up to 88 threads for large-scale cases with $n \in [2^{20}, 2^{23}]$. Edge distribution is $m \approx 6 * n$.

✔ On average, the parallel execution of the algorithm is 5.5× faster than its sequential variant for the mid-scale tests, and 6.4× faster for the large-scale cases.

✔ Run-time of smallest case (Del. 16) was ~ 300 sec, and of the largest one (Del. 23) ~ 20 hours. Only the GraphBLAS components of the algorithm exhibit excellent weak scalability for the large-scale graphs.

A C++ GraphBLAS algorithm

Design

https://github.com/DmsPas/Multiway-p-spectral-clustering

➤ C++11 implementation.

➤ Algebraic containers for sparse matrices and vectors (grb::Vector).

➤ Subroutines for I/O from and to the ROPTLIB data structures.

➤ Algebraic structure of the ring of real numbers to parallelize the SpMV operations (grb::vxm).

➤ Leveraging the auto-parallelisation shared-memory capabilities.

Implementation

Compute $\boldsymbol{\eta} \mapsto \mathcal{H}\boldsymbol{\eta} = (\mathcal{H}^\ell \eta^\ell)_{\ell=1}^{k}$ for arbitrary $\boldsymbol{\eta} \in \mathbb{R}^{k \times n}$.

Algorithm 1 Hessian evaluation

```
Input:   $\boldsymbol{\eta}$, a $k \times n$ matrix
         $(D[\ell])_{\ell=1}^{k}$, where each $D[\ell] = \text{diag}(\mathcal{H}^\ell)$
         $(H[\ell])_{\ell=1}^{k}$, where each $H[\ell] = \text{diag}(\mathcal{H}^\ell) - \mathcal{H}^\ell$
Output:  $\boldsymbol{r}$, the result of $\boldsymbol{\eta} \mapsto \mathcal{H}\boldsymbol{\eta}$
std::vector<grb::Vector<double>> grb_eta, grb_res
grb::Vector<double> v, w
ROPTLIBtoGRB($\boldsymbol{\eta}$, grb_eta)
for $\ell = 1$ to $k$ do
    grb::set(v, 0)
    grb::vxm(v, grb_eta[$\ell$], H[$\ell$], reals_ring)
    grb::eWiseApply(w, grb_eta[$\ell$]),D[$\ell$]
    grb::operators::mul<double>())
    grb::eWiseApply(grb_res[$\ell$], w, v,
    grb::operators::subtract<double>())
end for
GRBtoROPTLIB(grb_res, $\boldsymbol{r}$);
return $\boldsymbol{r}$
```

Open source library

https://github.com/Algebraic-Programming/ALP

① Sequential programs.

② Nonblocking shared-memory auto-parallelised programs.

③ Sequential programs with HyperDAG representations.

④ Distributed-memory auto-parallelised implementations.

⑤ Hybrid shared- and distributed-memory auto-parallelised programs.