

AN EFFICIENT MATLAB ALGORITHM FOR GRAPH PARTITIONING

TECHNICAL REPORT

João P. Hespanha

October 8, 2004

Abstract

This report describes a graph partitioning algorithm based on spectral factorization that can be implemented very efficiently with just a hand full of MATLAB commands. The algorithm is closely related to the one proposed by Phillips and Kokotović [3] for state-aggregation in Markov chains. The appendix contains a MATLAB script that implements the algorithm.

1 Graph partitioning

Consider an undirected graph $G = (V, E)$ with vertex set V and edge set E , together with a positive edge-cost function $c : E \rightarrow [0, \infty)$. A k -partition of V is a collection $\mathcal{P} = \{V_1, V_2, \dots, V_k\}$ of k disjoint subsets of V , whose union equals V . The *cost associated with \mathcal{P}* is defined by

$$C(\mathcal{P}) := \sum_{i \neq j} \sum_{\substack{(v, \bar{v}) \in E \\ v \in V_i, \bar{v} \in V_j}} \bar{c}(v, \bar{v}).$$

The ℓ -bounded Graph Partitioning (ℓ -GP) problem consists of finding a k -partition \mathcal{P} that minimizes $C(\mathcal{P})$, with no element in partition having more than ℓ vertices.

The smallest value of ℓ for which this problem is solvable is $\ell = \lceil n/k \rceil$, where n denotes the number of vertices in V . When ℓ takes this value we say that the partition is *perfectly balanced*. For $\ell = \lceil (1 + \epsilon)n/k \rceil$ we say that the partition has an ϵ *percentage of imbalance*. Several algorithms to solve this problem for the special case of $\bar{c}(\cdot)$ constant and equal to one are compared in <http://staffweb.cms.gre.ac.uk/~c.walshaw/partition/>.

Without loss of generality we assume that the graph is fully connected and set $c(v, \bar{v}) = 0$ for every edge not present in the original graph. In this case, we can simply write

$$C(\mathcal{P}) := \sum_{i \neq j} \sum_{v \in V_i, \bar{v} \in V_j} c(v, \bar{v}).$$

Note that since the graph is undirected, $c(v, \bar{v}) = c(\bar{v}, v)$, $\forall v, \bar{v} \in V$.

This problem is related to the *MAX k -CUT problem* in [2], which consists of finding a partition for V that maximizes the reward¹

$$R(\mathcal{P}) := \sum_{i \neq j} \sum_{v \in V_i, \bar{v} \in V_j} r(v, \bar{v}).$$

¹The reward in [2] is actually half of the one presented here because in that reference the outer summation was only over $i < j$.

for a given edge-reward $r : V \times V \rightarrow [0, \infty)$ with the property that $r(v, \bar{v}) = r(\bar{v}, v)$, $\forall v, \bar{v} \in V$. Ageev and Sviridenko [1] considered a variation of the MAX k -CUT problem, called the *Hypergraph MAX k -CUT problem with given sizes of parts (HM k C)*, which adds the constraint that $|V_i| = s_i$, $\forall i$ for a given set of k integers $\{s_1, s_2, \dots, s_k\}$.

1.1 Matrix formulation of the ℓ -GP problem

We are pursuing spectral techniques to solve this problem, i.e., techniques based on eigenvector/eigenvalue decomposition. To this effect, it is convenient to redefine the problem in matrix form.

We say that $n \times k$ matrix $\Pi = [\pi_{vj}]$ is a *k -partition matrix* if $\pi_{vj} \in \{0, 1\}$, $\forall v, j$, Π is an orthogonal matrix (i.e., $\Pi' \Pi$ is diagonal), and $\|\Pi\|_F = \sqrt{n}$. The notation $\|\cdot\|_F$ denotes the Frobenius norm, i.e., $\|\Pi\|_F := \sqrt{\text{trace}(\Pi' \Pi)}$. One can show that a $n \times k$ matrix Π is a k -partition matrix if and only if each row of Π is a vector of the canonical basis of \mathbb{R}^k (cf. Proposition 1 in the Appendix). Therefore a k -partition matrix Π is completely specified by a n -vector whose v th entry contains the index within the v th row of Π of the entry equal to one. We call this vector the *partition vector associated with Π* . The following two lemmas, which are proved in the appendix, will be needed.

Lemma 1. *There is a one-to-one correspondence between the set of k -partitions of $V := \{1, 2, \dots, n\}$ and the set of k -partition matrices Π . The two correspondences can be defined by*

1. *Given a k -partition $\mathcal{P} = \{V_1, V_2, \dots, V_k\}$ of V , a k -partition matrix $\Pi = [\pi_{vj}]$ can be defined by*

$$\pi_{vj} = \begin{cases} 1 & v \in V_j \\ 0 & v \notin V_j \end{cases} \quad \forall v \in V, j \in \{1, 2, \dots, k\}. \quad (1)$$

2. *Given a k -partition matrix $\Pi = [\pi_{vj}]$, a k -partitions $\mathcal{P} = \{V_1, V_2, \dots, V_k\}$ of V can be defined by*

$$V_j := \{v \in V : \pi_{vj} = 1\}, \quad \forall j \in \{1, 2, \dots, k\}. \quad (2)$$

For these correspondences

$$\Pi' \Pi = \text{diag}[|V_1|, |V_2|, \dots, |V_k|]$$

and the v th entry p_v of the partition vector p associated with Π specifies to which V_i does the v th vertex belong, i.e., $v \in V_{p_v}$, $\forall v \in V$. \square

It turns out that it is easy to express the cost associated with a k -partition of V in terms of the corresponding k -partition matrix:

Lemma 2. *Let $\mathcal{P} = \{V_1, V_2, \dots, V_k\}$ be a k -partition of $V := \{1, 2, \dots, n\}$ and $\Pi = [\pi_{vj}]$ the corresponding k -partition matrix. Then*

$$C(\mathcal{P}) = \mathbf{1}'_n A \mathbf{1}_n - \text{trace}(\Pi' A \Pi),$$

where A is an $n \times n$ matrix whose $v\bar{v}$ th entry is equal to $c(v, \bar{v})$ and $\mathbf{1}_n$ a n -vector with all entries equal to one. \square

These two lemmas allow us to reformulate the ℓ -GP problem as follows

$$\text{maximize} \quad \text{trace}(\Pi' A \Pi) \quad (3a)$$

$$\text{subject to} \quad \pi_{vj} \in \{0, 1\}, \quad \forall v, j \quad (3b)$$

$$\Pi \text{ orthogonal} \quad (3c)$$

$$\|\Pi\|_F = \sqrt{n} \quad (3d)$$

$$\Pi' \Pi \leq \ell I_{k \times k}. \quad (3e)$$

Since all entries of Π belong to $\{0, 1\}$, this optimization could also be formulated as follows

$$\begin{aligned}
& \text{maximize} && \text{trace}(\Pi' A \Pi) \\
& \text{subject to} && \pi_{vj} \in \{0, 1\}, \forall v, j \\
& && \Pi \text{ orthogonal} \\
& && \mathbf{1}_n \Pi \mathbf{1}_k = n \\
& && \mathbf{1}_n \Pi e_i \leq \ell,
\end{aligned}$$

where e_i denotes the i th vector in the canonical basis of \mathbb{R}^k .

In this paper, we restrict our attention to the case where the matrix A is (doubly) stochastic, i.e.,

$$\sum_{v \in \mathcal{V}} c(v, \bar{v}) = 1, \quad \forall v \in \mathcal{V}.$$

This corresponds to a situation where the costs associated with each node are “normalized” so that when the degree of a node is high then the costs associated with its edges must be low (on the average). By *degree* of a node v , we mean the number of nodes \bar{v} for which $c(v, \bar{v}) > 0$. The following MATLAB commands can be used to normalize a nonnegative symmetric matrix A , making it doubly stochastic:

```

A=A/(max(sum(A)));
A=A+sparse(1:n,1:n,1-sum(A));

```

2 Spectral partition

For a doubly stochastic matrix A , all its eigenvalues are real and smaller than or equal to one, with one of them exactly equal to one. In this case the optimization (3) is always smaller than or equal to n . This is because, for an arbitrary k -partition matrix Π that satisfies $\text{trace} \Pi' \Pi = n$, we must have

$$\text{trace}(\Pi' A \Pi) = \sum_{i=1}^k \pi_i' A \pi_i \leq \sum_{i=1}^k \pi_i' \pi_i = \text{trace} \Pi' \Pi = n \tag{5}$$

where π_i denotes Π 's i th column. Here we used the fact that the largest eigenvalue of A is no larger than one and therefore $\pi_i' A \pi_i \leq \pi_i' \pi_i$.

Let $\lambda_1 = 1 \geq \lambda_2 \geq \dots \geq \lambda_k$ be the largest eigenvalues of A and u_1, u_2, \dots, u_k the corresponding orthonormal eigenvectors. It is convenient to define

$$U := [u_1 \quad u_2 \quad \dots \quad u_k], \quad D := \text{diag}[\lambda_1, \lambda_2, \dots, \lambda_k],$$

for which $U'U = I_{k \times k}$ and $AU = UD$. The computation of U and D can be done very efficiently in MATLAB for large sparse matrices using the following commands:

```

options.issym=1;           % matrix is symmetric
options.isreal=1;         % matrix is real
options.tol=1e-6;         % decrease tolerance
options.maxit=500;        % increase maximum number of iterations
[U,D]=eigs(A,k,'la',options); % only compute 'k' largest eigenvalues/vectors

```

Suppose that we set

$$\Pi := UZ,$$

for some matrix $Z \in \mathbb{R}^{k \times k}$. In this case,

$$\Pi' \Pi = Z' U' U Z = Z' Z$$

and

$$\text{trace}(\Pi' A \Pi) = \text{trace} Z' U' A U Z = \text{trace} Z' D Z.$$

From this we conclude that the upper bound in (5) can be achieved exactly if the following conditions hold

1. $D = I$ (i.e., there are k independent eigenvectors associated with the eigenvalue equal to one)
2. UZ is a k -partition matrix.
3. $Z' Z \leq \ell I_{k \times k}$.

The algorithm that we propose to approximately solve the ℓ -GP problem is applicable when 1 holds approximately. It consists of setting

$$\Pi := \arg \min_{\bar{\Pi}} \|\bar{\Pi} - UZ\|_F, \quad (6)$$

where the minimization is carried out over the set of k -partition matrices and Z is a $k \times k$ matrix obtained by an Heuristic algorithm aimed at making the conditions 2–3 approximately hold. For reasons that will become clear shortly, we call this algorithm the *clustering algorithm*. Given a matrix Z , we call the computation of optimal Π in (6) the *projection algorithm*. We now describe both algorithms.

2.1 Projection algorithm

Given a matrix Z the computation of Π that minimizes (6) is straightforward: Since

$$(\bar{\Pi}' - Z' U')(\bar{\Pi} - ZU) = \bar{\Pi}' \bar{\Pi} + Z' Z - Z' U' \bar{\Pi} - \bar{\Pi} U Z,$$

denoting by $\bar{\pi}_{vi}$ and by \bar{u}_{vi} the v th entries of the $n \times k$ matrices $\bar{\Pi}$ and $\bar{U} := UZ$, respectively, we conclude that

$$\|\bar{\Pi} - UZ\|_F^2 = n + \|Z\|_F^2 + 2 \text{trace}(Z' U' \bar{\Pi}) = n + \|Z\|_F^2 - 2 \sum_{i=1}^k \sum_{v=1}^n \bar{u}_{iv} \bar{\pi}_{vi}$$

Denoting by i_v the index of the (only) entry in the v th row of $\bar{\Pi}$ whose entry is equal to one, we further conclude that

$$\|\bar{\Pi} - UZ\|_F^2 = n + \|Z\|_F^2 - 2 \sum_{v=1}^n \bar{u}_{i_v v} \bar{\pi}_{v i_v}$$

Therefore, to minimize $\|\bar{\Pi} - UZ\|_F$ one should simply chose i_v to be equal to the index of the largest element in the v th row of UZ . The computation of the optimal (sparse) matrix Π and the associated partition vector p can be done very efficiently in MATLAB using the following commands:

```
[dummy, p]=max(U*Z, [], 2);
Pi=sparse(1:length(p), p, 1);
```

2.2 Clustering algorithm

Suppose that the n rows of the $n \times k$ matrix U are clustered around k orthogonal row vectors z_1, z_2, \dots, z_k and let n_i denote the number of row clustered around z_i . Defining

$$Z := \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_k \end{bmatrix}^{-1} = \begin{bmatrix} \frac{z'_1}{\|z_1\|^2} & \frac{z'_2}{\|z_2\|^2} & \cdots & \frac{z'_k}{\|z_k\|^2} \end{bmatrix},$$

the v th row of $\bar{U} := UZ$ is given by

$$\bar{u}_v = \begin{bmatrix} \frac{\langle u_v, z_1 \rangle}{\|z_1\|^2} & \frac{\langle u_v, z_2 \rangle}{\|z_2\|^2} & \cdots & \frac{\langle u_v, z_k \rangle}{\|z_k\|^2} \end{bmatrix},$$

where u_v denotes the v th row of U . Since u_v is close to one of the v_i and all z_i are orthogonal, we conclude that \bar{u}_v will be close to one of the vectors in the canonical base of \mathbb{R}^k . In practice, this means that UZ is close to a k -partition matrix. Moreover, since there are n_k rows close to z_k , the number of entries close to one in the j th column of $\bar{U} := UZ$ is approximately n_k . Therefore

$$\Pi' \Pi \approx \text{diag}[n_1, n_2, \dots, n_k].$$

The computation of Z can then be viewed as a clustering algorithm whose goal is to determine orthogonal vectors z_1, z_2, \dots, z_k around which the rows of U are clustered, with no cluster larger than ℓ .

The simple case When the vectors z_i defined above exist and the clustering is tight, these vectors can be easily found because they are orthogonal and therefore far apart. In practice, one simply needs to select k -rows of U that are far apart from each other so that we can be sure that no two belong to the same cluster and take them as “cluster representatives.” Find such rows is essentially a pivoting problem like the one that arises in Gauss elimination. The following algorithm can be used to select such rows: Let (Q, R, E) be a QR -factorization with pivoting of U' , i.e., $Q^{k \times k}$ is an orthonormal matrix, $R \in \mathbb{R}^{k \times n}$ is an upper triangular matrix with elements in the main diagonal of decreasing magnitude, and $E \in \mathbb{R}^{n \times n}$ a permutation matrix such that

$$U' E = QR \quad \Leftrightarrow \quad E' U = R' Q'.$$

Partitioning E as

$$E = \begin{bmatrix} E_1 & E_2 \end{bmatrix} \in \mathbb{R}^{n \times n}, \quad E_1 \in \mathbb{R}^{n \times k}, E_2 \in \mathbb{R}^{n \times (n-k)},$$

the $k \times k$ matrix Z in (6) should be selected by

$$Z := (E'_1 U)^{-1},$$

This algorithm essentially select the z_i 's to be the top k rows of the permuted matrix $E'U$ and Z maps these rows to the vectors of the canonical basis of \mathbb{R}^k . The computation of Z can be done efficiently by noting that if we partition R as

$$R = \begin{bmatrix} R_1 & R_2 \end{bmatrix} \in \mathbb{R}^{k \times n}, \quad R_1 \in \mathbb{R}^{k \times k}, R_2 \in \mathbb{R}^{k \times (n-k)},$$

we obtain

$$V' \begin{bmatrix} E_1 & E_2 \end{bmatrix} = Q \begin{bmatrix} R_1 & R_2 \end{bmatrix} \Rightarrow E'_1 V = R'_1 Q'$$

and therefore Z can be obtained by the following product of $k \times k$ matrices

$$Z = Q(R'_1)^{-1}.$$

This can be done efficiently in MATLAB using the following commands:

```
[Q,R,E]=qr(U',0);
Z=Q*(R(:,1:k)')^(-1);
```

However, this algorithm provides no guarantee that the cluster sizes will not exceed ℓ .

The tough case When the rows of U are not tightly clustered more sophisticated algorithms are needed. In this case, one can use a general purpose clustering algorithm such as k -means or the Expectation Maximization (EM) algorithm. Note that the matrix Π can be obtained directly from the clustering procedure by associating each cluster with one of the vectors in the canonical basis of \mathbb{R}^k and replacing each row of U by the basis vector associated with its cluster.

Since the k -mean clustering algorithm is implemented in MATLAB, we can use it to obtain the matrix Π using the following commands:

```
[p,zs]=kmeans(U,k,'distance','cosine');
Pi=sparse(1:length(p),p,1);
```

This algorithm also does not guarantee that the cluster sizes will not exceed ℓ , but it can be adapted to do so [4].

A Appendix

A.1 Technical Results

Proposition 1. *A $n \times k$ matrix Π is a k -partition matrix if and only if each row of Π is a vector of the canonical basis of \mathbb{R}^k .* \square

Proof of Proposition 1. Let Π be a k -partition matrix with columns $\pi_1, \pi_2, \dots, \pi_k \in \mathbb{R}^n$. Each row of Π must have at most a single one because otherwise, if there were two ones in columns π_i and π_j , then $\pi_i' \pi_j \geq 1$ and Π would not be orthogonal. On the other hand, every row of Π must have at least a one because for a matrix whose entries are in $\{0, 1\}$, its squared Frobenius norm $\|\Pi\|_F^2$ is exactly the total number of entries equal to one. Since each row can have at most a one, every row must actually have a one to get $\|\Pi\|_F^2 = n$. This proves that if Π is a k -partition matrix then each row of Π is a vector of the canonical basis of \mathbb{R}^k .

Conversely, suppose that each row of Π is a vector of the canonical basis of \mathbb{R}^k . Then all entries of Π belong to the set $\{0, 1\}$ and $\|\Pi\|_F^2 = n$ since its squared Frobenius norm is equal to the number of entries equal to one (one per row). Moreover, Π is orthogonal because if π_i and π_j are two distinct columns of Π then each row of these vector must have at least one zero and therefore $\pi_i' \pi_j = 0, \forall i \neq j$. \blacksquare

Proof of Lemma 1. To show that (1) defines a k -partition matrix, we must show that $\Pi' \Pi$ is diagonal and that its trace equal n . To this effect, let π_i and π_j be two distinct columns of Π . For every row v , at least one of these column vectors must have a zero, since otherwise the corresponding node v would have to belong to both V_i and V_j . This means that $\pi_i' \pi_j = 0, \forall i \neq j$ and therefore Π is orthogonal. On the other hand, π_i has an entry equal to one for each node than belongs to V_i so $\pi_i' \pi_i = |V_i|, \forall i$. Therefore $\text{trace } \Pi' \Pi = \sum_j |V_j| = |V| = n$ because the V_i are a k -partition of V .

To show that the sets defined by (2) define a k partition of V , we must show that they are disjoint and their union is equal to V . These sets are indeed disjoint because otherwise if $v \in V_i$ and $v \in V_j$, then $\pi_{vi} = \pi_{vj} = 1$ and therefore the inner product of the corresponding columns would be nonzero. Moreover, the i th element of the main diagonal of $\Pi' \Pi$ is equal to the number of ones in V_i and therefore $\sum_j |V_j| = \text{trace } \Pi' \Pi = n$ because Π is a k -partition matrix. \square

Proof of Lemma 2. From the definition of A , we have that

$$A\Pi = \left[\sum_{\bar{v} \in V} c(v, \bar{v}) \pi_{\bar{v}j} \right] = \left[\sum_{\bar{v} \in V_j} c(v, \bar{v}) \right].$$

Therefore

$$C(\mathcal{P}) := \sum_{i \neq j} \sum_{v \in V_i} \sum_{\bar{v} \in V_j} c(v, \bar{v}) = \sum_{i \neq j} \sum_{v \in V_i} (A\Pi)_{vj}$$

where $(A\Pi)_{vj}$ denotes the v th entry of the matrix $A\Pi$. We can re-write the above summation as

$$C(\mathcal{P}) = \sum_{i \neq j} \sum_{v \in V} \pi_{vi} (A\Pi)_{vj} = \sum_{i \neq j} (\Pi' A\Pi)_{ij}$$

where now $(\Pi' A\Pi)_{ij}$ denotes the ij th entry of the matrix $\Pi' A\Pi$. The result then follows by decomposition the last summation as

$$\begin{aligned} C(\mathcal{P}) &= \sum_{i,j} (\Pi' A\Pi)_{ij} - \sum_i (\Pi' A\Pi)_{ii} \\ &= \mathbf{1}'_k \Pi' A \Pi \mathbf{1}_k - \text{trace}(\Pi' A \Pi) \\ &= \mathbf{1}'_n A \mathbf{1}_n - \text{trace}(\Pi' A \Pi) \end{aligned}$$

where we used the property of k -partition matrices that $\Pi \mathbf{1}_k = \mathbf{1}_n$. ■

A.2 MATLAB script

The algorithm described in this paper can be implemented using the following MATLAB script. A more complete version of this script is available online at <http://www.ece.ucsb.edu/~hespanha/techrep.html>

```
function [ndx,Pi,cost]= grPartition(C,k,nrep);
% Inputs:
% C - n by n edge-weights matrix.
% k - desired number of partitions
% nrep - number of repetition for the clustering algorithm
% Outputs:
% ndx - n-vector with the cluster index for every node
% Pi - Projection matrix
% cost - cost of the partition (sum of broken edges)
%
% By Joao Pedro Hespanha, Copyright 2004

% Make C double stochastic
C=C/(max(sum(C)));
C=C+sparse(1:n,1:n,1-sum(C));

% Spectral partition
options.issym=1; % matrix is symmetric
options.isreal=1; % matrix is real
options.tol=1e-6; % decrease tolerance
options.maxit=500; % increase maximum number of iterations
options.disp=0;
[U,D]=eigs(C,k,'la',options); % only compute 'k' largest eigenvalues/vectors

% Clustering
[ndx,zs]=kmeans(U,k,'Distance','cosine','Start','sample','Replicates',nrep);

Pi=sparse(1:length(ndx),ndx,1);
cost=full(sum(sum(C))-trace(Pi'*C*Pi));
```

References

- [1] A. A. Ageev and M. I. Sviridenko. An approximation algorithm for hypergraph max k -cut with given sizes of parts. BRICS Report Series RS-99-49, BRICS, Dept. of Computer Science, University of Aarhus, Denmark, Dec. 1999.
- [2] A. Frieze and M. Jerrum. Improved approximation algorithms for max k -cut and max bisection. *Algorithmica*, 18(1):67–81, 1997.
- [3] R. Phillips and P. Kokotović. A singular perturbation approach to modeling and control of Markov chains. *IEEE Trans. on Automat. Contr.*, 26(5):1087–1094, Oct. 1981.
- [4] S. Zhong and J. Ghosh. Scalable, balanced model-based clustering. In *Proc. 3rd SIAM Int. Conf. Data Mining*, pages 71–82, May 2003.