CS 267 Applications of Parallel Computers

Lecture 17:

Graph Partitioning - III

James Demmel

http://www.cs.berkeley.edu/~demmel/cs267_Spr99

Outline of Graph Partitioning Lectures

° Review of last lectures

° Multilevel Acceleration

  • BIG IDEA, will appear often in course

° Available Software

  • good sequential and parallel software availble

° Comparison of Methods

° Application to DNA sequencing

Review Definition of Graph Partitioning

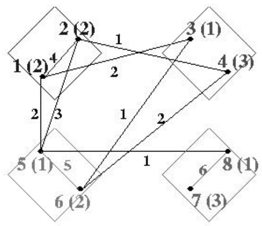
° **Given a graph G = (N, E, W_N, W_E)**
  - **N = nodes (or vertices), E = edges**
  - **W_N = node weights, W_E = edge weights**

° **Ex: N = {tasks}, W_N = {task costs}, edge (j,k) in E means task j sends W_E(j,k) words to task k**

° **Choose a partition N = N_1 U N_2 U … U N_P such that**
  - **The sum of the node weights in each N_j is "about the same"**
  - **The sum of all edge weights of edges connecting all different pairs N_j and N_k is minimized**

° **Ex: balance the work load, while minimizing communication**

° **Special case of N = N_1 U N_2: Graph Bisection**



Review of last 2 lectures

° **Partitioning with nodal coordinates**
  - **Rely on graphs having nodes connected (mostly) to "nearest neighbors" in space**
  - **Common when graph arises from physical model**
  - **Finds a circle or line that splits nodes into two equal-sized groups**
  - **Algorithm very efficient, does not depend on edges**

° **Partitioning without nodal coordinates**
  - **Depends on edges**
  - **Breadth First Search (BFS)**
  - **Kernighan/Lin - iteratively improve an existing partition**
  - **Spectral Bisection - partition using signs of components of second eigenvector of L(G), the Laplacian of G**

Introduction to Multilevel Partitioning

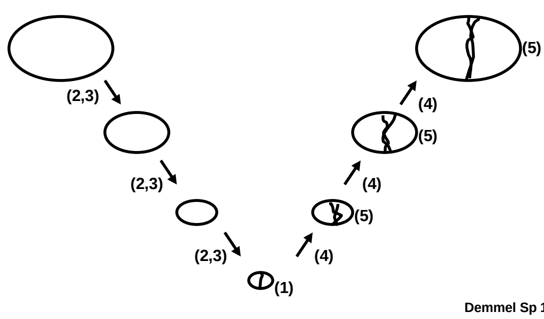
° **If we want to partition G(N,E), but it is too big to do efficiently, what can we do?**
  - **1) Replace G(N,E) by a coarse approximation $G_C(N_C,E_C)$, and partition G_C instead**
  - **2) Use partition of G_C to get a rough partitioning of G, and then iteratively improve it**

° **What if G_C still too big?**
  - **Apply same idea recursively**

Multilevel Partitioning - High Level Algorithm

(N+,N-) = Multilevel_Partition(N, E)
… recursive partitioning routine returns N+ and N- where N = N+ U N-
if |N| is small
(1) Partition G = (N,E) directly to get N = N+ U N-
Return (N+, N-)
else
(2) Coarsen G to get an approximation $G_C = (N_C, E_C)$
(3) (N_C+ , N_C-) = Multilevel_Partition(N_C, E_C)
(4) Expand (N_C+ , N_C-) to a partition (N+ , N-) of N
(5) Improve the partition (N+ , N-)
Return (N+ , N-)
endif

"V - cycle:"



How do we
Coarsen?
Expand?
Improve?

Multilevel Kernighan-Lin

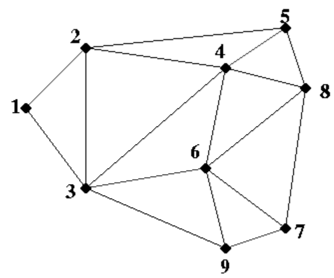
- Coarsen graph and expand partition using maximal matchings
- Improve partition using Kernighan-Lin

Maximal Matching

- *Definition*: A matching of a graph G(N,E) is a subset E_m of E such that no two edges in E_m share an endpoint
- *Definition:* A maximal matching of a graph G(N,E) is a matching E_m to which no more edges can be added and remain a matching
- A simple greedy algorithm computes a maximal matching:

```
let E_m be empty
mark all nodes in N as unmatched
for i = 1 to |N|      … visit the nodes in any order
    if i has not been matched
        if there is an edge e=(i,j)  where j is also unmatched,
            add e to E_m
            mark i and j as matched
        endif
    endif
endfor
```

Maximal Matching - Example



Coarsening using a maximal matching

Construct a maximal matching E_m of G(N,E)

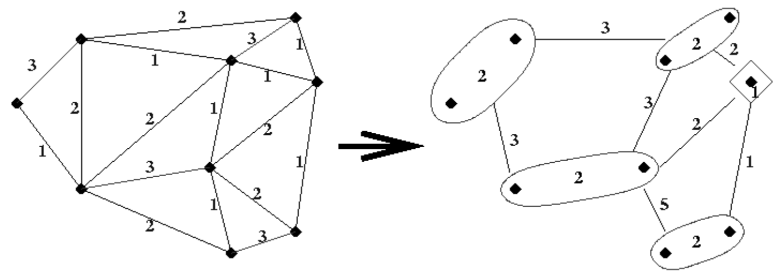
for all edges e=(j,k) in E_m
Put node n(e) in N_c
W(n(e)) = W(j) + W(k) … gray statements update node/edge weights
for all nodes n in N not incident on an edge in E_m
Put n in N_c … do not change W(n)
… Now each node r in N is "inside" a unique node n(r) in N_c

… Connect two nodes in Nc if nodes inside them are connected in E
for all edges e=(j,k) in E_m
for each other edge e'=(j,r) in E incident on j
Put edge ee = (n(e),n(r)) in E_c
W(ee) = W(e')
for each other edge e'=(r,k) in E incident on k
Put edge ee = (n(r),n(e)) in E_c
W(ee) = W(e')

If there are multiple edges connecting two nodes in N_c, collapse them, adding edge weights

Example of Coarsening

How to coarsen a graph using a maximal matching



$G = (N, E)$

E_m is shown in red

Edge weights shown in blue

Node weights are all one

$G_c = (N_c, E_c)$

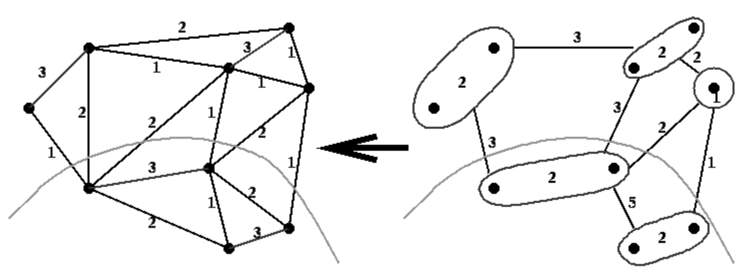
N_c is shown in red

Edge weights shown in blue

Node weights shown in black

Expanding a partition of G_c to a partition of G

Converting a coarse partition to a fine partition



Partition shown in green

Multilevel Spectral Bisection

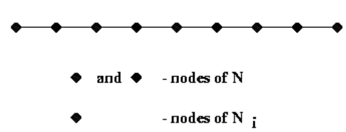
- Coarsen graph and expand partition using maximal independent sets
- Improve partition using Rayleigh Quotient Iteration

Maximal Independent Sets

- *Definition*: An independent set of a graph G(N,E) is a subset N_i of N such that no two nodes in N_i are connected by an edge
- *Definition:* A maximal independent set of a graph G(N,E) is an independent set N_i to which no more nodes can be added and remain an independent set
- A simple greedy algorithm computes a maximal independent set:

```
let Ni be empty
for i = 1 to |N|      … visit the nodes in any order
    if  node i is not adjacent to any node already in Ni
        add i to Ni
    endif
endfor
```

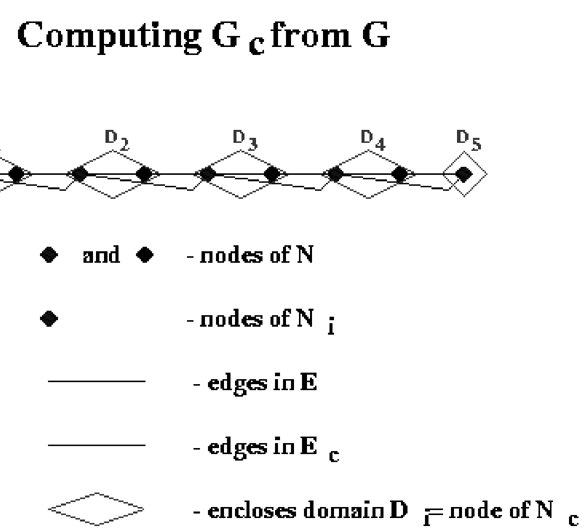
Maximal Independent Subset N_i of N



Coarsening using Maximal Independent Sets

```
… Build "domains" D(i) around each node i in N_i to get nodes in N_c
… Add an edge to E_c whenever it would connect two such domains
E_c = empty set
for all nodes i in N_i
    D(i) = ( {i}, empty set )
    … first set contains nodes in D(i), second set contains edges in D(i)
unmark all edges in E
repeat
    choose an unmarked edge e = (i,j) from E
    if exactly one of i and j (say i) is in some D(k)
        mark e
        add j and e to D(k)
    else if i and j are in two different D(k)'s (say D(ki) and D(kj))
        mark e
        add edge (ki, kj) to E_c
    else if both i and j are in the same D(k)
        mark e
        add e to D(k)
    else
        leave e unmarked
    endif
until no unmarked edges
```


Example of Coarsening



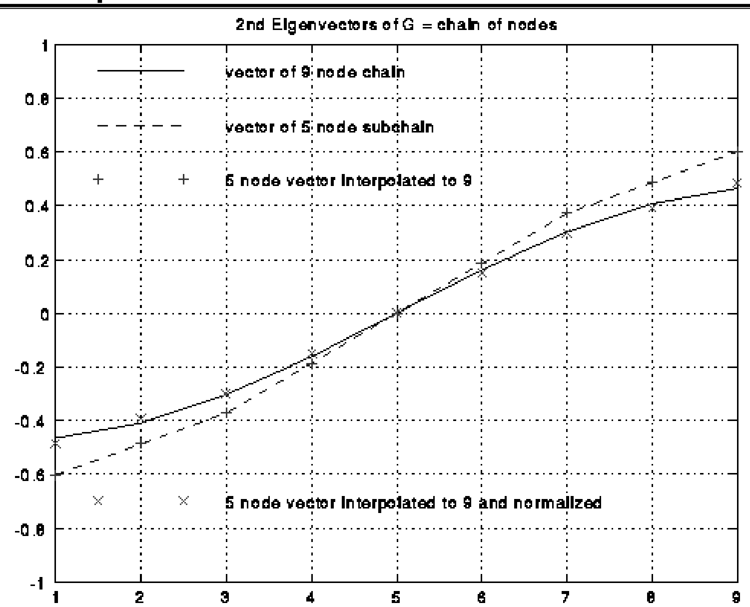
Expanding a partition of G_c to a partition of G

° Need to convert an eigenvector v_c of $L(G_c)$ to an approximate eigenvector v of L(G)

° Use interpolation:

```
For each node j in N
   if j is also a node in N_c, then
      v(j) = v_c(j)    … use same eigenvector component
   else
      v(j) = average of v_c(k) for all neighbors k of j in N_c
   end if
endif
```

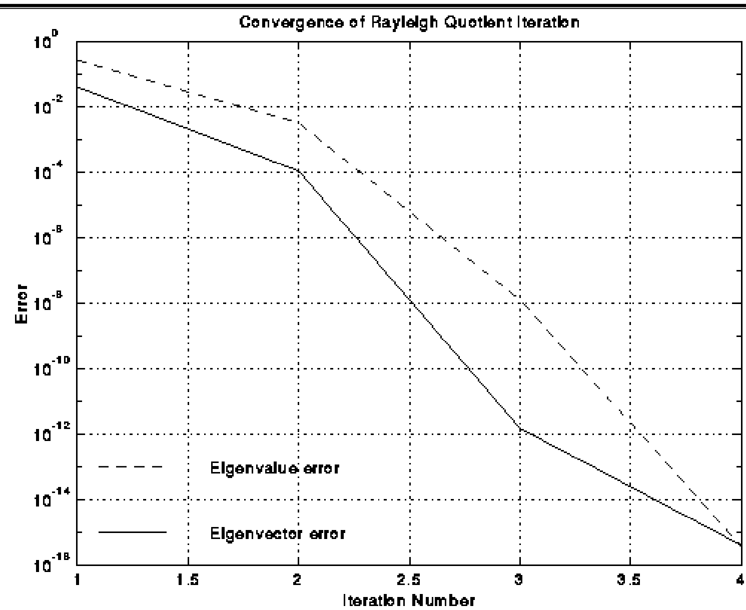

Example: 1D mesh of 9 nodes



Improve eigenvector v using Rayleigh Quotient Iteration

j = 0
pick starting vector v(0) … from expanding v_C
repeat
  j=j+1
  r(j) = v^T(j-1) * L(G) * v(j-1)
  … r(j) = *Rayleigh Quotient* of v(j-1)
  … = good approximate eigenvalue
  v(j) = (L(G) - r(j)*I)$^{-1}$ * v(j-1)
  … expensive to do exactly, so solve approximately
  … using an iteration called SYMMLQ,
  … which uses matrix-vector multiply (no surprise)
  v(j) = v(j) / || v(j) || … normalize v(j)
until v(j) converges
… Convergence is very fast: cubic

Example of convergence for 1D mesh



Available Implementations

° **Multilevel Kernighan/Lin**

- METIS (www.cs.umn.edu/~metis)
- ParMETIS - parallel version

° **Multilevel Spectral Bisection**

- S. Barnard and H. Simon, “A fast multilevel implementation of recursive spectral bisection …”, Proc. 6th SIAM Conf. On Parallel Processing, 1993
- Chaco (www.cs.sandia.gov/CRF/papers_chaco.html)

° **Hybrids possible**

- Ex: Using Kernighan/Lin to improve a partition from spectral bisection

Comparison of methods

° **Compare only methods that use edges, not nodal coordinates**

- CS267 webpage and KK95a (see below) have other comparisons

° **Metrics**

- Speed of partitioning
- Number of edge cuts
- Other application dependent metrics

° **Summary**

- No one method best
- Multi-level Kernighan/Lin fastest by far, comparable to Spectral in the number of edge cuts
  - www-users.cs.umn.edu/~karypis/metis/publications/mail.html
  - see publications KK95a and KK95b
- Spectral give much better cuts for some applications
  - Ex: image segmentation
  - www.cs.berkeley.edu/~jshi/Grouping/overview.html
  - see “Normalized Cuts and Image Segmentation”

Test matrices, and number of edges cut for a 64-way partition

For Multilevel Kernighan/Lin, as implemented in METIS (see KK95a)

Graph	# of Nodes	# of Edges	# Edges cut for 64-way partition	Expected # cuts for 2D mesh	Expected # cuts for 3D mesh	Description
144	144649	1074393	88806	6427	31805	3D FE Mesh
4ELT	15606	45878	2965	2111	7208	2D FE Mesh
ADD32	4960	9462	675	1190	3357	32 bit adder
AUTO	448695	3314611	194436	11320	67647	3D FE Mesh
BBMAT	38744	993481	55753	3326	13215	2D Stiffness M.
FINAN512	74752	261120	11388	4620	20481	Lin. Prog.
LHR10	10672	209093	58784	1746	5595	Chem. Eng.
MAP1	267241	334931	1388	8736	47887	Highway Net.
MEMPLUS	17758	54196	17894	2252	7856	Memory circuit
SHYY161	76480	152002	4365	4674	20796	Navier-Stokes
TORSO	201142	1479989	117997	7579	39623	3D FE Mesh

Expected # cuts for 64-way partition of 2D mesh of n nodes

$$n^{1/2} + 2*(n/2)^{1/2} + 4*(n/4)^{1/2} + \ldots + 32*(n/32)^{1/2} \sim 17 * n^{1/2}$$

Expected # cuts for 64-way partition of 3D mesh of n nodes =

$$n^{2/3} + 2*(n/2)^{2/3} + 4*(n/4)^{2/3} + \ldots + 32*(n/32)^{2/3} \sim 11.5 * n^{2/3}$$

Speed of 256-way partitioning (from KK95a)

Partitioning time in seconds

Graph	# of Nodes	# of Edges	Multilevel Spectral Bisection	Multilevel Kernighan/ Lin	Description
144	144649	1074393	607.3	48.1	3D FE Mesh
4ELT	15606	45878	25.0	3.1	2D FE Mesh
ADD32	4960	9462	18.7	1.6	32 bit adder
AUTO	448695	3314611	2214.2	179.2	3D FE Mesh
BBMAT	38744	993481	474.2	25.5	2D Stiffness M.
FINAN512	74752	261120	311.0	18.0	Lin. Prog.
LHR10	10672	209093	142.6	8.1	Chem. Eng.
MAP1	267241	334931	850.2	44.8	Highway Net.
MEMPLUS	17758	54196	117.9	4.3	Memory circuit
SHYY161	76480	152002	130.0	10.1	Navier-Stokes
TORSO	201142	1479989	1053.4	63.9	3D FE Mesh

Kernighan/Lin much faster than Spectral Bisection!

Application to DNA Sequencing

- "A spectral algorithm for seriation and the consecutive ones problem", J. Atkins, E. Boman and B. Hendrickson, SIAM J. Computing, 1995
  - www-sccm.stanford.edu/~boman/seriation.ps.gz
- DNA is a very long string of 4 letters: ACCTGATCTGACT…
- To sequence, we have a large set of short fragments Fi, whose sequences (ACCT…) we know
- Fragments can to attach to the original DNA at places where their sequences are complementary
- In the lab, we can determine which fragments attach to the DNA at certain locations called probes Pj
- If we knew the order the probes appeared in the DNA, we would know its sequence, as a concatenation of fragment sequences
- We get information from the fact that multiple fragments may attach to the DNA at multiple probes, since they are similar

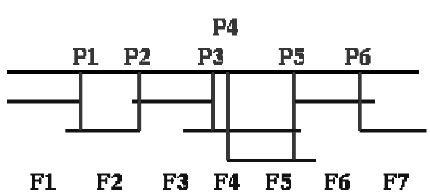
Probes and Fragments

- Record which fragments Fi attach to which probes Pj in a matrix B:

B(Fi,Pj) = 1 if Fi attaches at Pj, and 0 otherwise

- When fragments and probes are sorted in the order they appear in the DNA, and there is no experimental error, then B is a band-matrix, or consecutive-ones matrix

DNA Sequencing



$$
B = \begin{array}{c} \begin{array}{cccccc} P1 & P2 & P3 & P4 & P5 & P6 \end{array} \\ \left[\begin{array}{cccccc} 1 & & & & & \\ 1 & 1 & & & & \\ & 1 & 1 & & & \\ & & 1 & 1 & 1 & \\ & & & 1 & 1 & \\ & & & & 1 & 1 \\ & & & & & 1 \end{array}\right] \end{array} \begin{array}{c} \\ F1 \\ F2 \\ F3 \\ F4 \\ F5 \\ F6 \\ F7 \end{array}
$$

Fragments shown in blue
Probes shown in red

Actual B not sorted this way, so we want to sort it

- Since we don't know the correct order of probes and fragments, B is not a consecutive-ones matrix
- Instead, we get $B_P = P_F*B*P_P$ where P_P and P_F are unknown permutation matrices, i.e. B_P = B with rows and columns scrambled
- Goal of DNA sequencing is to reconstruct P_P and P_F from B_P

DNA Unsequenced

$$
Bp = \begin{array}{c}
\begin{array}{cccccc} P5 & P3 & P2 & P4 & P6 & P1 \end{array} \\
\left[\begin{array}{cccccc}
 & 1 & 1 & & & \\
1 & & & 1 & & \\
 & & 1 & & & 1 \\
1 & 1 & & 1 & & \\
 & & & & 1 & \\
 & & & & & 1 \\
1 & & & & 1 &
\end{array}\right]
\end{array}
\begin{array}{c} \\ F3 \\ F5 \\ F2 \\ F4 \\ F7 \\ F1 \\ F6 \end{array}
$$

Relation to Graph Partitioning

- Let G(N,E) be graph, L(G) its Laplacian
- Recall:

minimum # edge cuts to bisect G

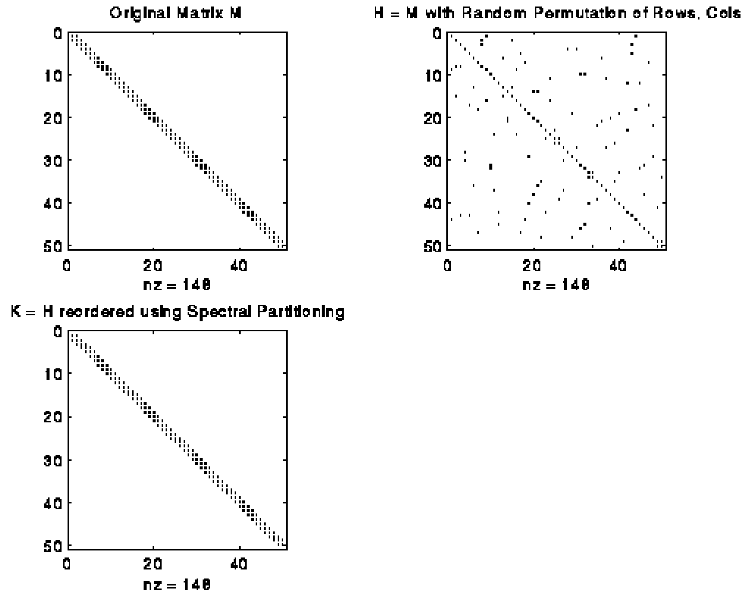
$$= \min_{\pm 1 \text{ vectors } x,\ \Sigma_i x(i) = 0} \quad .25 * \Sigma_{e=(i,k)} (x(i) - x(k))^2$$

$.25 * |N| * \lambda_2$

$$= \min_{\Sigma_i v(i)^2 = |N|,\ \Sigma_i v(i) = 0} \quad .25 * \Sigma_{e=(i,k)} (v(i) - v(k))^2$$

- Think of each node i in N embedded in real axis at v(i), and each edge e=(i,j) as line segment from v(i) to v(j)
  - Sum of squares of line segment lengths are minimized over all possible embeddings v such that $||v|| = |N|^{1/2}$, $\Sigma_i v(i) = 0$
- If we permute nodes so that v(i) <= v(i+1), then renumbered nodes will tend to be connected to those with nearby numbers
- Let P be a permutation so that P*v is sorted; thus $P*L(G)*P^T$ will look banded

Example: recovering a symmetric band matrix via graph partitioning



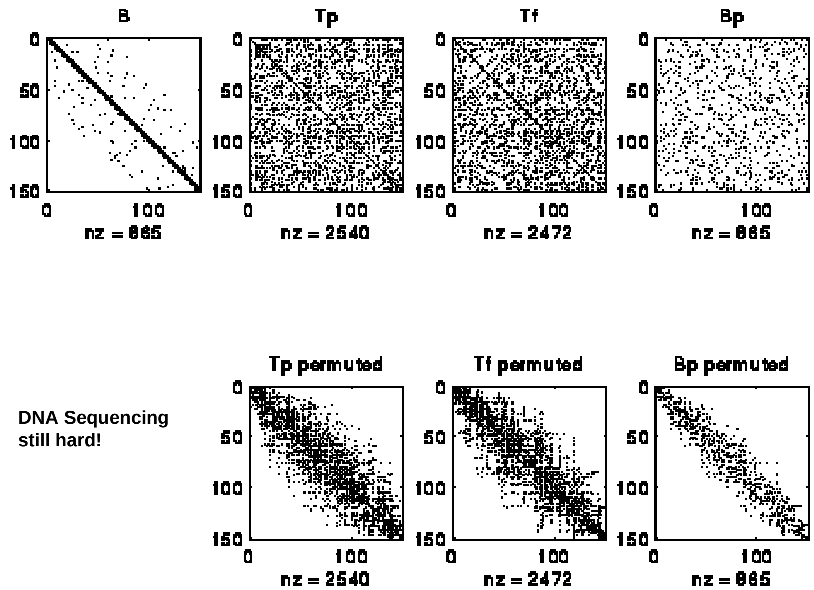
Unscrambling the rows and columns of Bp

- We need to recover two permutations to get B from $B_P = P_F * B * P_P$, not just one, since B nonsymmetric
- Consider

$$T_P = B_P^T * B_P = P_P^T * (B^T * B) * P_P$$
$$T_F = B_P * B_P^T = P_F * (B * B^T) * P_F^T$$

- Both T_P and T_F are symmetric
  - Compute second eigenvector of both
  - Recover P_P by making T_P banded
  - Recover P_F by making T_F banded

Example of effectiveness in the presence of error



DNA Sequencing
still hard!