Semidefinite Relaxation for the Clique Partitioning and Clustering Problems

Brendan Ames$^1$  Stephen Vavasis$^1$

$^1$Department of Combinatorics and Optimization
University of Waterloo

MS144: Semidefinite Approaches to Combinatorial Problems - Part I of III

ICIAM 2011, Vancouver, July 18-22
Clusters, cliques, and low-rank matrices

The maximum node KDC problem

The weighted KDC problem

Numerical results

Outline

1. Clusters, cliques, and low-rank matrices

2. The maximum node KDC problem

3. The weighted KDC problem

4. Numerical results
Average vs Planted Case analysis

- **Classical idea**: study the behaviour of problems for random program inputs.
- Algorithms or heuristics that perform poorly in general may still work for most program instances (eg. simplex).
- For many problems, a truly random program instance is not a good model for a generic instance.
  - Matrix completion: matrix to be recovered may be necessarily of low-rank (eg. partially observed EDM in low-dimensional space, matrix of customer preferences in recommendation systems).
  - Image processing: natural images tend to have large regions of similar pixels separated by sharp edges.
- A better model for a generic instance for these problems is the **planted case**: generate a program instance where the desired hidden structure has been obscured by random noise.
Rank minimization

- **Affine rank minimization problem (RMP):** find matrix with minimum rank satisfying linear constraints:

  \[
  \min \{ \text{rank}(X) : \mathcal{A}(X) = \mathbf{b} \}
  \]

  (here \( \mathcal{A} : \mathbb{R}^{m \times n} \rightarrow \mathbb{R}^p \) linear, \( \mathbf{b} \in \mathbb{R}^p \)).

- NP-hard but can be solved in the case that \( \mathcal{A} \) is random and a low-rank solution exists by relaxing to a convex program.

- Replace \( \text{rank}(X) \) with nuclear norm

  \[
  \|X\|_* = \sigma_1(X) + \cdots + \sigma_m(X). \quad (\text{NNM})
  \]

The clustering problem

- **Clustering**: Want to partition a given data set so that items in each cluster are similar to each other and items not in the same cluster are dissimilar.
- Intractible in general.
- If data is actually clustered, can we identify the clusters efficiently?
Clustering as graph partitioning

- Can model data as graph $G = (V, E)$.
- $V$ is the set of items in the data.
- Similarity is indicated by $E$:
  - Either by adjacency ($ij \in E$ iff $i, j$ are similar), or
  - By assigning edge-weights to each edge of complete graph.
- A set of nodes $C \subseteq V$ is a clique of $G$ if $ij \in E$ for all $i, j \in C$.
- Every clique of $G$ (with char. vector $v$) defines a rank-one block of $A_G + I$ by $vv^T$.
- Optimal partitioning of the graph into cliques gives an optimal clustering of the data.
The k-disjoint-clique problem

- Given graph $G = (V, E)$ and integer $k \in [1, |V|]$, a k-disjoint-clique-subgraph is a subgraph of $G$ composed of $k$ disjoint cliques.
- We consider the problem of identifying the largest $k$-disjoint-clique subgraph in a given graph $G$.
- **binary case**: maximize number of nodes.
- **weighted case**: maximize normalized edge-weights.
The maximum node KDC problem

The maximum node k-disjoint-clique problem (KDC):
Find a $k$-disjoint-clique subgraph containing the maximum number of nodes of input graph $G$. 

[A-V 09] if input graph contains a sufficiently large clique ($\Omega(\sqrt{N})$), can solve the maximum clique problem exactly by formulating as rank minimization and relaxing to nuclear norm minimization (even though $A$ does not satisfy RIP or incoherence).
The maximum node KDC problem

The maximum node $k$-disjoint-clique problem ($\text{KDC}$):
Find a $k$-disjoint-clique subgraph containing the maximum number of nodes of input graph $G$.

**Special case:** $k = 1$ The maximum clique problem

[A-V 09] if input graph contains a sufficiently large clique ($\Omega(\sqrt{N})$), can solve the maximum clique problem exactly by formulating as rank minimization and relaxing to nuclear norm minimization (even though $\mathcal{A}$ does not satisfy RIP or incoherence).
What if we want to find many cliques?

- In general, can’t formulate $k$-disjoint-clique as rank minimization.
- $k$ may be very large, don’t want to recover a low-rank matrix.
- Can’t generate series of cuts using the maximum clique because not in the planted case.
- **Solution**: Relax rank constraints with combination of nuclear norm constraint and semidefinite constraint.
Relaxation as SDP

- We relax KDC as the SDP

\[
\begin{align*}
\max & \quad \sum \sum X_{ij} \\
\text{s.t.} & \quad Xe \leq e, \\
& \quad X_{ij} = 0, \quad \forall (i, j) \notin E, i \neq j \\
& \quad tr(X) = k, \\
& \quad X \succeq 0.
\end{align*}
\]

- \( Xe \leq e \Rightarrow \|X\|_* \leq rank(X) \) for all feasible \( X \succeq 0 \).
- \( X \succeq 0 \Rightarrow \|X\|_* = tr(X) \) for all feasible \( X \).
The planted case

- Consider graphs constructed as follows:
- Start with disjoint cliques $C_1, \ldots, C_k$ of size $r_1, \ldots, r_k$.
- Noise: add set $C_{k+1}$ containing $r_{k+1}$ additional nodes and additional edges either deterministically by an adversary or at random independently with fixed prob $p$.
- $C_1, \ldots, C_k$ induce a feasible solution of $(\ast)$:

$$X^* = \sum_{i=1}^{k} \frac{1}{r_i} v_i v_i^T$$

where $v_i \in \mathbb{R}^V$ is the characteristic vector of $C_i$. 
Results

• if not too much extra noise is added then, we can recover \( X^* \) (and hence \( \{C_1, \ldots, C_k\} \)) by solving (\( \ast \)).

• For both formulations, can add at most \( r_{k+1} = O(\hat{r}^2) \) extra nodes, where \( \hat{r} = \min_{i=1,\ldots,k} r_i \).

• **Adv case:** Can add up to \( O(\hat{r}^2) \) additional edges, provided there is at most \( O(\min\{r_q, r_{cl(v)}\}) \) edges from \( v \) to \( C_q \).

• **Random case:** “too many” noise edges quantified by number of cliques \( k \) and the discrepancy between their sizes: need

\[
\left( \sum_{s=1}^{k} r_s^2 \right)^{1/2} \left( \sum_{q=1}^{k} \frac{1}{r_q} \right)^{1/2} \leq O(\hat{r}).
\]
The maximum mean weight KDC problem

Mean weight $k$-disjoint-clique problem (WKDC):

Given a complete graph $K_N$ and edge-weights $W$, find a $k$-disjoint-clique subgraph of $K_N$ that maximizes the sum of average weights covered by each clique.

We relax WKDC as the SDP

$$\begin{align*}
\max & \quad tr(WX) \\
\text{s.t.} & \quad X e \leq e, \\
& \quad X \succeq 0, \\
& \quad tr(X) = k, \\
& \quad X \succeq 0
\end{align*}$$
Weighted planted case

- Unlike **KDC**, planted case does not correspond to any particular structure in input graph.
- Instead, is induced by edge-weight matrix $W$: entries of $W$ corresponding to the planted $k$-disjoint-clique subgraph are larger than the rest.
- Let $\{C_1, \ldots, C_k\} \subseteq V$ define a $k$-disjoint-clique subgraph of the complete graph $K_N = (V, E)$ on $N$ vertices.
- Randomly sample entries of $W$ from two distributions $\Omega_1, \Omega_2$ such that if $u, v$ in same clique $C_i$ then $E[W_{uv}] = \alpha$; otherwise $E[W_{uv}] = \beta$ for $\alpha > \beta$. 
Results: weighted case

- If not too much noise, the relaxation \((w^*)\) of WKDC is exact with extremely high probability for sufficiently large \(\hat{r}\).
- Can tolerate up to \(O(\hat{r})\) additional nodes.
- As before, too much edge noise is quantified by number of cliques \(k\) and the discrepancy between their sizes:
  \[
  \left( \frac{k+1}{\sum_{s=1}^{k} r_s} \right)^{1/2} \leq O(\hat{r}).
  \]
Rehnquist Supreme Court

- Data set is the set of U.S. Supreme Court Justices (serving from 1994-95 to 2003-04).
- Assign edge-weights corresponding to fraction of decisions on which Justices agreed:

<table>
<thead>
<tr>
<th></th>
<th>St</th>
<th>Br</th>
<th>Gi</th>
<th>So</th>
<th>Oc</th>
<th>Ke</th>
<th>Re</th>
<th>Sc</th>
<th>Th</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 St</td>
<td>1</td>
<td>0.62</td>
<td>0.66</td>
<td>0.63</td>
<td>0.33</td>
<td>0.36</td>
<td>0.25</td>
<td>0.14</td>
<td>0.15</td>
</tr>
<tr>
<td>2 Br</td>
<td>0.62</td>
<td>1</td>
<td>0.72</td>
<td>0.71</td>
<td>0.55</td>
<td>0.47</td>
<td>0.43</td>
<td>0.25</td>
<td>0.24</td>
</tr>
<tr>
<td>3 Gi</td>
<td>0.66</td>
<td>0.72</td>
<td>1</td>
<td>0.78</td>
<td>0.47</td>
<td>0.49</td>
<td>0.43</td>
<td>0.28</td>
<td>0.26</td>
</tr>
<tr>
<td>4 So</td>
<td>0.63</td>
<td>0.71</td>
<td>0.78</td>
<td>1</td>
<td>0.55</td>
<td>0.5</td>
<td>0.44</td>
<td>0.31</td>
<td>0.29</td>
</tr>
<tr>
<td>5 Oc</td>
<td>0.33</td>
<td>0.55</td>
<td>0.47</td>
<td>0.55</td>
<td>1</td>
<td>0.67</td>
<td>0.71</td>
<td>0.54</td>
<td>0.54</td>
</tr>
<tr>
<td>6 Ke</td>
<td>0.36</td>
<td>0.47</td>
<td>0.49</td>
<td>0.5</td>
<td>0.67</td>
<td>1</td>
<td>0.77</td>
<td>0.58</td>
<td>0.59</td>
</tr>
<tr>
<td>7 Re</td>
<td>0.25</td>
<td>0.43</td>
<td>0.43</td>
<td>0.44</td>
<td>0.71</td>
<td>0.77</td>
<td>1</td>
<td>0.66</td>
<td>0.68</td>
</tr>
<tr>
<td>8 Sc</td>
<td>0.14</td>
<td>0.25</td>
<td>0.28</td>
<td>0.31</td>
<td>0.54</td>
<td>0.58</td>
<td>0.66</td>
<td>1</td>
<td>0.79</td>
</tr>
<tr>
<td>9 Th</td>
<td>0.15</td>
<td>0.24</td>
<td>0.26</td>
<td>0.29</td>
<td>0.54</td>
<td>0.59</td>
<td>0.68</td>
<td>0.79</td>
<td>1</td>
</tr>
</tbody>
</table>
Rehnquist Supreme Court: results

- We solved WKDC with $k = 2$ using SeDuMi. We obtained the following partition of the supreme court:

<table>
<thead>
<tr>
<th>1: “Liberal”</th>
<th>2: “Conservative”</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stevens (St)</td>
<td>O’Connor (Oc)</td>
</tr>
<tr>
<td>Breyer (Br)</td>
<td>Kennedy (Ke)</td>
</tr>
<tr>
<td>Ginsberg (Gi)</td>
<td>Rehnquist (Re)</td>
</tr>
<tr>
<td>Souter (So)</td>
<td>Scalia (Sc)</td>
</tr>
<tr>
<td></td>
<td>Thomas (Th)</td>
</tr>
</tbody>
</table>
Rehnquist Supreme Court: results

- Algorithm is sensitive to choice of $k$.
- Solve with $k = 3$:

<table>
<thead>
<tr>
<th>1: “Most Conservative”</th>
<th>2: “Moderate Conservative”</th>
<th>3: “Liberal”</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thomas (Th)</td>
<td>O’Connor (Oc)</td>
<td>Stevens (St)</td>
</tr>
<tr>
<td>Scalia (Sc)</td>
<td>Kennedy (Ke)</td>
<td>Breyer (Br)</td>
</tr>
<tr>
<td></td>
<td>Rehnquist (Re)</td>
<td>Ginsberg (Gi)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Souter (So)</td>
</tr>
</tbody>
</table>
Image Segmentation

- Want to partition pixels of a given image into similar “segments” (wrt some property, eg. intensity).
- Essentially want to cluster the pixels of the image.

- Solve **WKDC** using with edge weights given by

\[
W_{ij} = \exp(-\|F_i - F_j\|^2/\sigma_F - \|X_i - X_j\|^2/\sigma_X)
\]

where \(F_i\) is the feature value, \(X_i\) is the position in image of the \(i\)th pixel, parameters \(\sigma_F, \sigma_X\) chosen by user.
• Example 238011 from the *Berkeley Segmentation Data Set*.
• Original image $300 \times 300$; work with resized image $30 \times 30$.
  Takes $4321s$ to solve.
• $F_i$ is RGB value of $i$th pixel, $\sigma_F = 0.3$, $\sigma_X = 10000$.  

![Sample Image](image1)

![Resized Image](image2)
• Example 135069 from the *Berkeley Segmentation Data Set*.
• Original image 175x230; resized image 25x33. Takes 3430s to solve.
• \( F_i \) is RGB value of \( i \)th pixel, \( \sigma_F = 0.3, \sigma_X = 10000 \).
Conclusions

- The clique and clustering problems be relaxed as SDPs.
- We obtain the exact solution in polynomial time if the data contains the desired hidden structure.

Future work:
- extension to other problems: eg. other graph-partitioning objectives (e.g. normalized cut).
- faster algorithms: more efficient solvers for SDP/doubly nonnegative programs and/or theoretical guarantees for spectral clustering heuristics.