Convex Optimization Theory and Applications

Topic 9 - Convex Relaxation

Li Li

Department of Automation,
Tsinghua University,

Fall, 2009-2021.

9.0. Outline

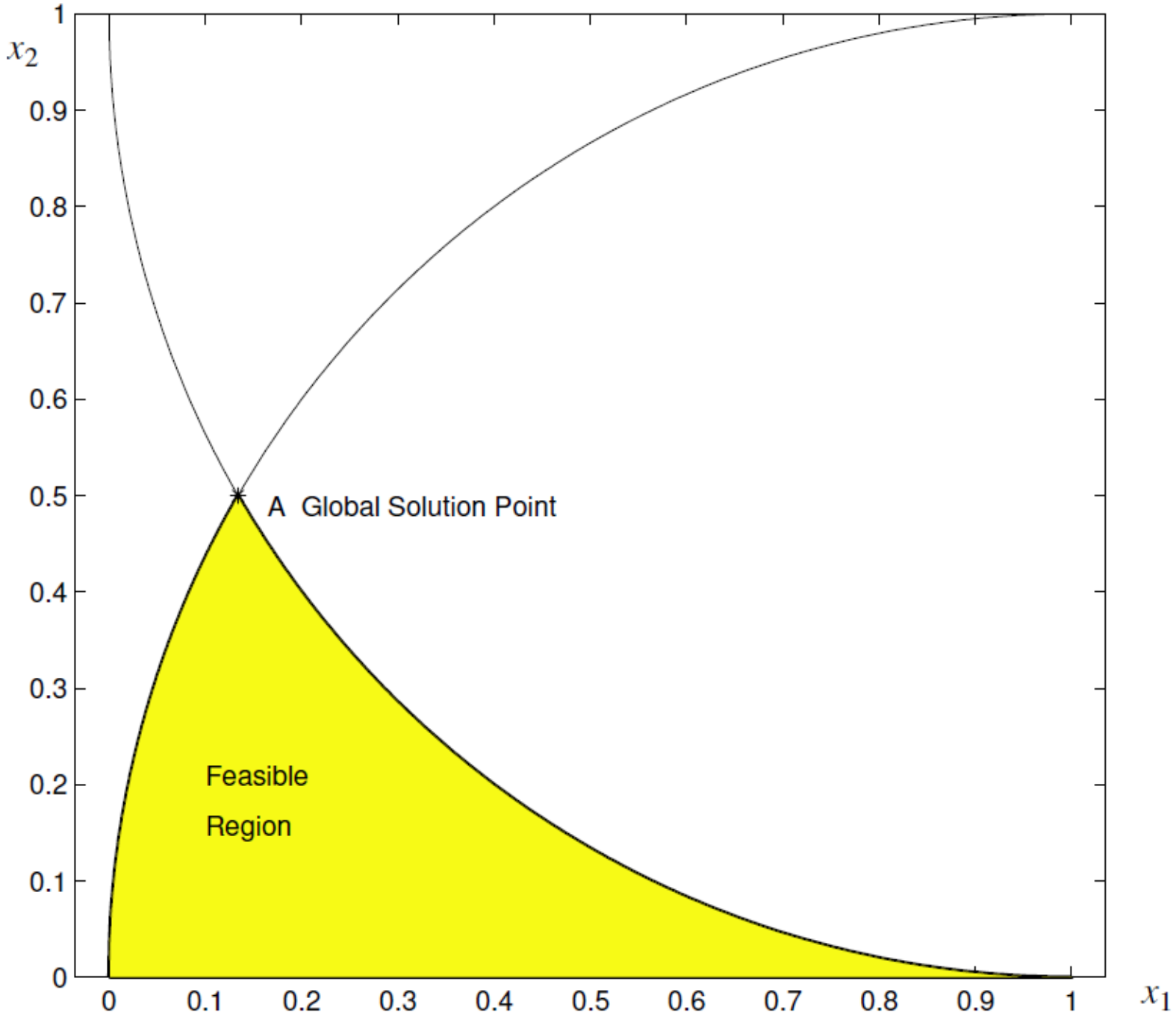
9.1. Max-Cut Problems

9.1.1 Semidefinite Relaxation

$$\text{minimize} \quad x_1 - 2x_2 \tag{6.1}$$

s.t. $x_1 \geq 0$, $x_2 \geq 0$, $(x_1 - 1)^2 + x_2^2 \leq 1$, $(x_1 - 1)^2 + (x_2 - 1)^2 \geq 1$.

This is not a convex optimization problem. The best solution is $x^* = [0.1340, 0.5]^T$


x_2
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
A Global Solution Point
Feasible
Region
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
x_1

9.1. Max-Cut Problems

9.1.1 Semidefinite Relaxation

Definte $y_1 = x_1$, $y_2 = x_2$, $y_3 = x_1^2$, $y_4 = x_2^2$, we have

$$\text{minimize} \quad y_1 - 2y_2 \tag{6.2}$$

s.t. $y_1 \geq 0$, $y_2 \geq 0$, $y_3 \geq 0$, $y_4 \geq 0$.
$-y_3 + 2y_1 - y_4 \geq 0$, $y_3 - 2y_1 + y_4 - 2y_2 + 1 \geq 0$

This is a linear programming problem. Its global optimal solution is $y' = [0, 0.5, 0, 0]^T$, which gives $x' = [0, 0.5]^T$.

Now further define $y_5 = x_1 x_2$, we have

9.1. Max-Cut Problems

9.1.1 Semidefinite Relaxation

$$\text{minimize} \quad y_1 - 2y_2 \tag{6.3}$$

s.t. $y_1 \geq 0$, $y_2 \geq 0$, $y_3 \geq 0$, $y_4 \geq 0$.

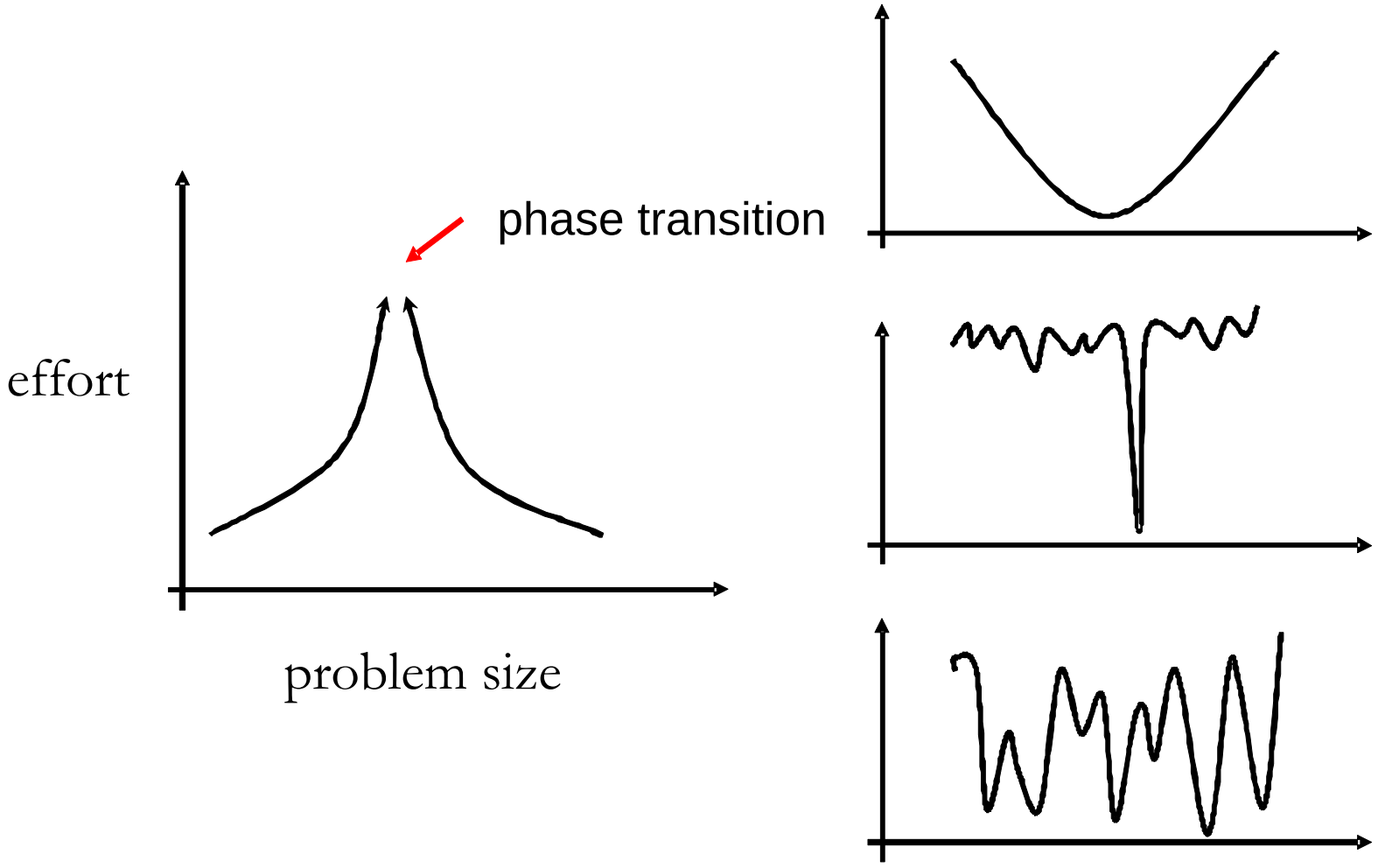
$-y_3 + 2y_1 - y_4 \geq 0$, $y_3 - 2y_1 + y_4 - 2y_2 + 1 \geq 0$

$$\begin{bmatrix} 1 & y_1 & y_2 \\ y_1 & y_3 & y_5 \\ y_2 & y_5 & y_4 \end{bmatrix} = \begin{bmatrix} 1 & x_1 & x_2 \\ x_1 & x_1^2 & x_1x_2 \\ x_2 & x_1x_2 & x_2^2 \end{bmatrix} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix} \begin{bmatrix} 1 & x_1 & x_2 \end{bmatrix} \geq 0$$

This is a SDP probem, whose global optimal solution is $y'' = [0.1340, 0.5, 0.0179, 0.25, 0.0670]^T$, and thus $x'' = [0.1340, 0.5]^T$.

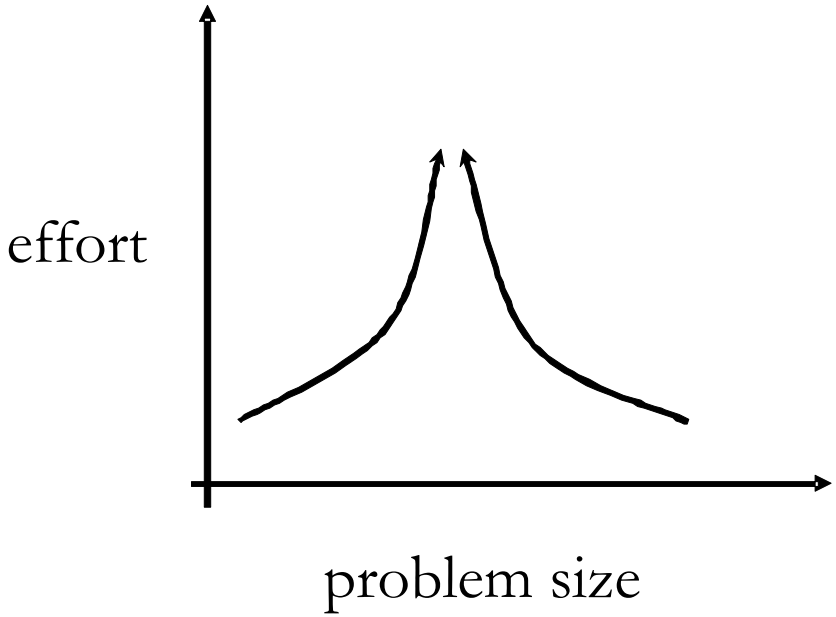
9.1. Max-Cut Problems

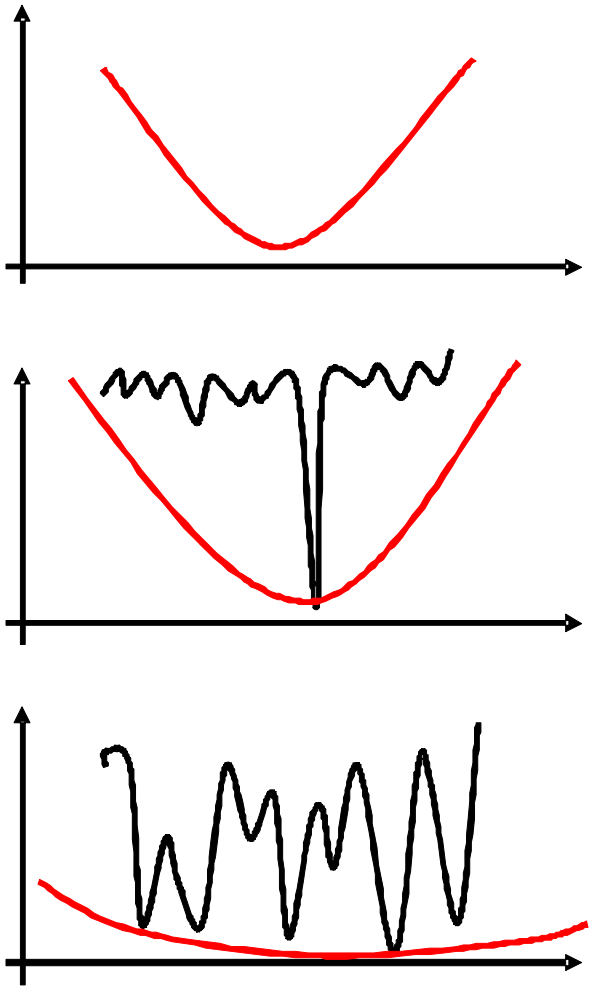
9.1.1 Semidefinite Relaxation



9.1. Max-Cut Problems

9.1.1 Semidefinite Relaxation

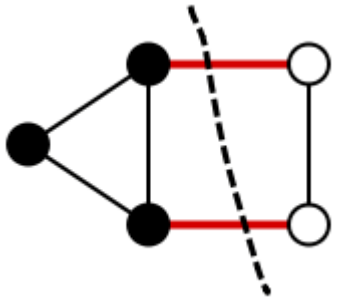




9.1. Max-Cut Problems

9.1.2 Cut in Graph Theory and Max/Min-Cut Problems

A cut is a partition of the vertices of a graph into two disjoint subsets.

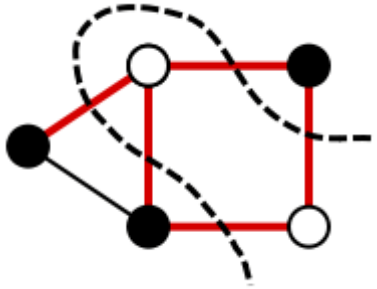


The cut-set of the cut is the set of edges whose end points are in different subsets of the partition. Edges are said to be crossing the cut if they are in its cut-set.

9.1. Max-Cut Problems

9.1.2 Cut in Graph Theory and Max/Min-Cut Problems

A cut is minimum if the size of the cut is not larger than the size of any other cut. There are polynomial-time methods to solve the min-cut problem, e.g. the Ford-Fulkerson algorithm.



A cut is maximum if the size of the cut is not smaller than the size of any other cut. The max-cut problem is one of Karp's 21 NP-complete problems.

9.1. Max-Cut Problems

9.1.2 Cut in Graph Theory and Max/Min-Cut Problems

The max-flow min-cut theorem states that in a flow network, the maximum amount of flow passing from the source to the sink is equal to the minimum capacity that needs to be removed from the network so that no flow can pass from the source to the sink.

Recall that the input to the maximum $(s,t)-flow$ problem consists of a weighted directed graph $G=(V,E)$, two special vertices s and t, and a function assigning a non-negative capacity c_e to each edge e.

9.1. Max-Cut Problems

9.1.2 Cut in Graph Theory and Max/Min-Cut Problems

Our task is to choose the flow f_e across each edge e so that

$$\text{maximize} \quad \sum_w f_{s\to w} - \sum_u f_{u\to s} \tag{6.4}$$

subject to $\sum_w f_{v\to w} - \sum_u f_{u\to v} = 0$ for every vertex $v \neq s$, t

$0 \le f_{u\to v} \le c_{u\to v}$ for every edge $u \to v$

Similarly, the minimum cut problem can be formulated using 'indicator' variables similarly to the shortest path problem.

9.1. Max-Cut Problems

9.1.2 Cut in Graph Theory and Max/Min-Cut Problems

We have a variable S_v for each vertex v, indicating whether $v \in S$ or $v \in T$, and a variable $X_{u \to v}$ for each edge $u \to v$, indicating whether $u \in S$ and $v \in T$, where (S,T) is some $(s,t)-cut$.

$$\text{minimize} \quad \sum_{u \to v} c_{u \to v} \cdot X_{u \to v} \qquad (6.5)$$

subject to $X_{u \to v} + S_v - S_u \geq 0$ for every edge $u \to v$

$X_{u \to v} \geq 0$ for every edge $u \to v$

$S_v = 1, \quad S_t = 0$

9.1. Max-Cut Problems

9.1.2 Cut in Graph Theory and Max/Min-Cut Problems

The above are prime and dual problems and can be sovled via linear programming. The equality in the max-flow min-cut theorem follows from the strong duality theorem.

9.1. Max-Cut Problems

9.1.2 Cut in Graph Theory and Max/Min-Cut Problems

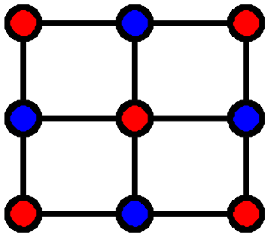
The max-cut problem can be found in different cases

Given n activities, m persons. Suppose each activity can be scheduled either in the morning or in the afternoon. Each person is interested in two activities.

Our task is to schedule the activities to maximize the number of persons that can enjoy both activities. If exactly $n/2$ of the activities have to be held in the morning, we get the so called MAX Bisection.

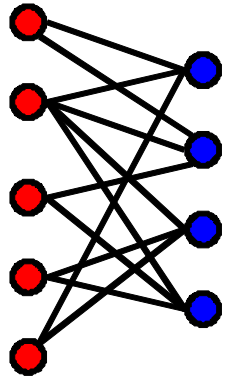
9.1. Max-Cut Problems

9.1.2 Cut in Graph Theory and Max/Min-Cut Problems



9.1. Max-Cut Problems

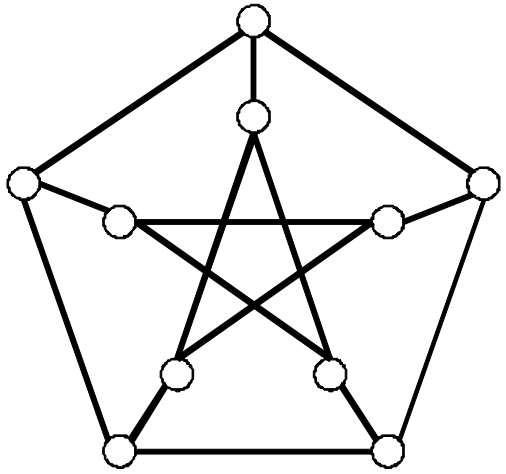
9.1.2 Cut in Graph Theory and Max/Min-Cut Problems



bipartite graphs are easy to solve

9.1. Max-Cut Problems

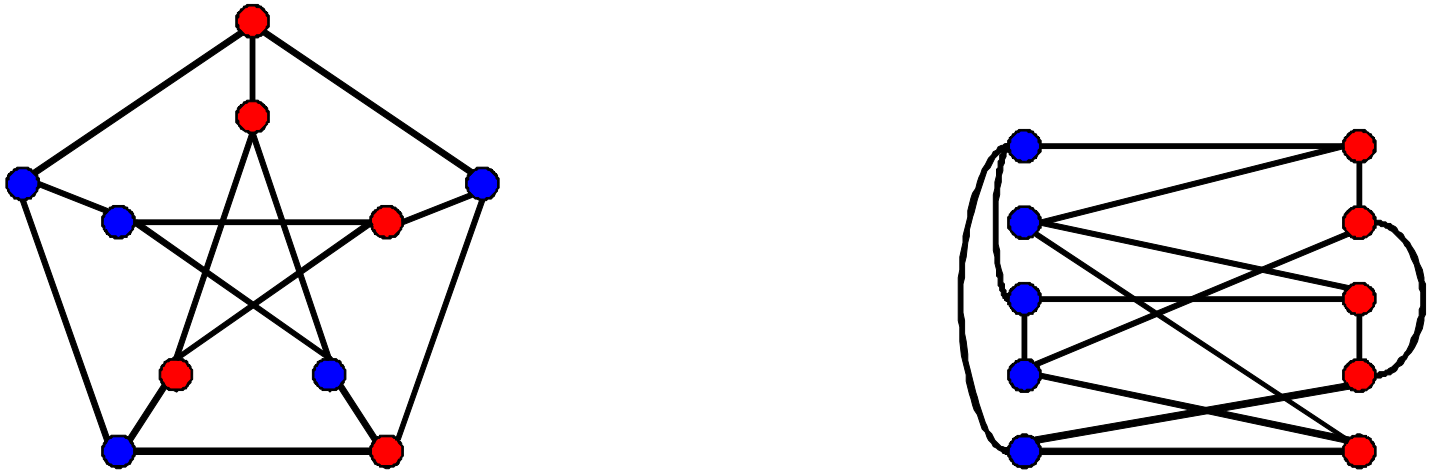
9.1.2 Cut in Graph Theory and Max/Min-Cut Problems



Petersen graph is classic counterexample with Max-Cut = 12

9.1. Max-Cut Problems

9.1.2 Cut in Graph Theory and Max/Min-Cut Problems



Let us color every vertex arbitrary blue or red with equal probability. Suppose we get the above colored graph, the cut is 9. Error is 25%.

Is sampling a good way? Is expectation a better way?

9.1. Max-Cut Problems

9.1.3 Max-Cut via Semidefinite Relaxation

Given $G=(N,E)$ (we can assume G is complete by adding edges with zero weight), $|N|=n$, $w: E \to R_+$, we want to find a cut $\delta(S)=\{ij \in E : i \in S, j \in N \setminus S\}$ with the maximum weight $w(\delta(S)) = \sum_{ij \in \delta(S)} w_{ij}$.

To reach the QP Formulation of the max-cut problem, let

$$x_j = \begin{cases} 1 & j \in S \\ -1 & j \notin S \end{cases} \qquad (6.6)$$

9.1. Max-Cut Problems

9.1.2 Max-Cut via Semidefinite Relaxation

$$x_i x_j = \begin{cases} -1 & i \in S,\ j \notin S \quad \text{or} \quad \textit{viceversa} \\ 1 & \textit{otherwise} \end{cases} \tag{6.7}$$

This gives $w(\delta(S)) = \frac{1}{2}\sum_{i,j} w_{ij}(1 - x_i x_j)$ and our (non-convex, quadratically constrained) QP is

$$\text{maximize} \quad w(\delta(S)) = \frac{1}{2}\sum_{i,j} w_{ij}(1 - x_i x_j) \tag{6.8}$$

subject to $x_i^2 = 1, \quad i = 1, \ldots, \quad n$

9.1. Max-Cut Problems

9.1.3 Max-Cut via Semidefinite Relaxation

If we define $L = L(G)$ to be the Laplacian of G with

$$l_{ij} = \begin{cases} -\frac{1}{2} w_{ij} & i \neq j \\ \frac{1}{2} \sum_{k \neq i} w_{ik} & i = j \end{cases}$$

, our QP becomes

$$\text{maximize} \quad \sum_i \sum_j l_{ij} x_i x_j \qquad (6.9)$$

subject to $x_i^2 = 1, \quad i = 1, \ldots, \quad n$

9.1. Max-Cut Problems

9.1.3 Max-Cut via Semidefinite Relaxation

A basic idea for semidefinite relaxation in such situations is: For each i replace non-convex constraint $x_i^2 = 1$ with relaxed convex constraint $x_i^2 \leq 1$, which is an LMI constraint

$$\begin{bmatrix} 1 & x_i \\ x_i & 1 \end{bmatrix} \geq 0 \qquad (6.10)$$

9.1. Max-Cut Problems

9.1.3 Max-Cut via Semidefinite Relaxation

We can also replace all non-convex constraints $x_i^2 = 1$ for $i = 1, \ldots, n$ with relaxed LMI constraint

$$X = \begin{bmatrix} 1 & x_1 & x_2 & \cdots & x_n \\ x_1 & 1 & x_{12} & & x_{1n} \\ x_2 & x_{12} & & & x_{2n} \\ \vdots & & & \ddots & \vdots \\ x_n & x_{1n} & x_{2n} & \cdots & 1 \end{bmatrix} \geq 0 \qquad (6.11)$$

where x_{ij} are additional variables = liftings, it is always less conservative than previous relaxation. (6.8) indicates (6.7)!

9.1. Max-Cut Problems

9.1.3 Max-Cut via Semidefinite Relaxation

The above QP is equivalent to

$$\text{maximize} \quad trace(Lxx^T) = x^T Lx \tag{6.12}$$

subject to $diag(xx^T) = e$

Letting $X = xx^T$ and we have $trace(Lxx^T) = trace(LX)$

$$\{xx^T : x \in \{-1,+1\}^n\} = \{X \in SR^{n\times n};\ diag(X) = e, X \geq 0, rank(X) = 1\} \tag{6.13}$$

9.1. Max-Cut Problems

9.1.3 Max-Cut via Semidefinite Relaxation

Thus, the following rank-constrained SDP is equivalent

$$\max_{X \in SR^{n\times n}} \quad trace(LX) \tag{6.14}$$

subject to $diag(X) = e$, $X \geq 0$, $rank(X) = 1$.

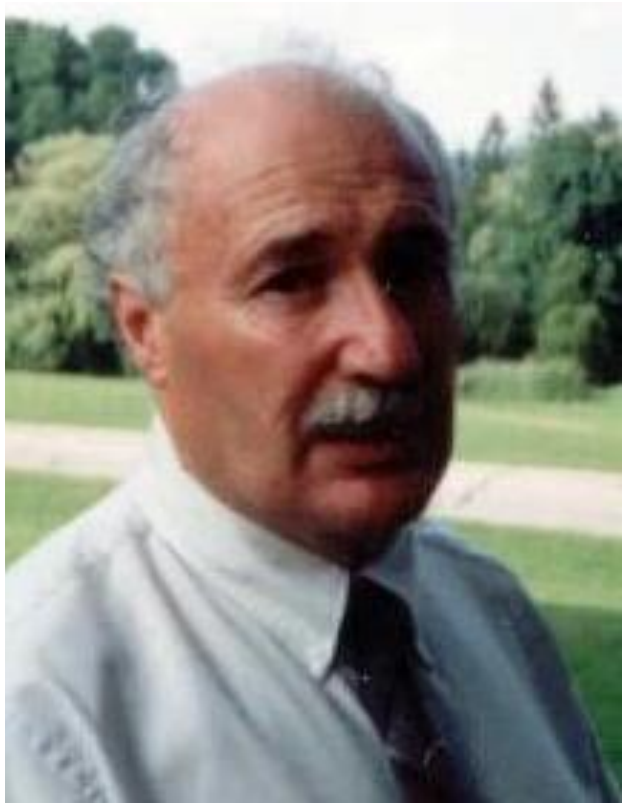
Now, we relax the rank constraint to get a relaxation

$$\max_{X \in SR^{n\times n}} \quad trace(LX) \tag{6.15}$$

subject to $diag(X) = e$, $X \geq 0$.

9.1. Max-Cut Problems

9.1.3 Max-Cut via Semidefinite Relaxation



Naum Zuselevich Shor
January 1, 1937 Kiev - February 26, 2006

9.1. Max-Cut Problems

9.1.3 Max-Cut via Semidefinite Relaxation

Using the decomposition $X = B^T B$, one can see that a positive semidefinite X with $y_{ii} = 1$ corresponds precisely to a set of unit vectors v_1, …, $v_n \in S_m$, where we simply correspond the vector v_i to the ith column of B. Then $y_{ij} = v_i v_j$. The matrix Y is known as the Gram matrix of $\{v_1, v_2, \cdots, v_n\}$. By this equivalence, we can reformulate (P) as a semidefinite program:

$$\text{maximize} \quad \frac{1}{2}\sum w_{ij}\left(1 - v_i \cdot v_j\right) \tag{6.16}$$

Subjecto $v_i \in R^n$, $\| v_i \| = 1$. This is a relaxation of (6.14).

9.1. Max-Cut Problems

9.1.4 Bounds of Semidefinite Relaxation

Consider the plane spanned by v_i and v_j, and let θ_{ij} be the angle between these two vectors. We have

$$v_i \cdot v_j = \cos\theta_{ij} \tag{6.17}$$

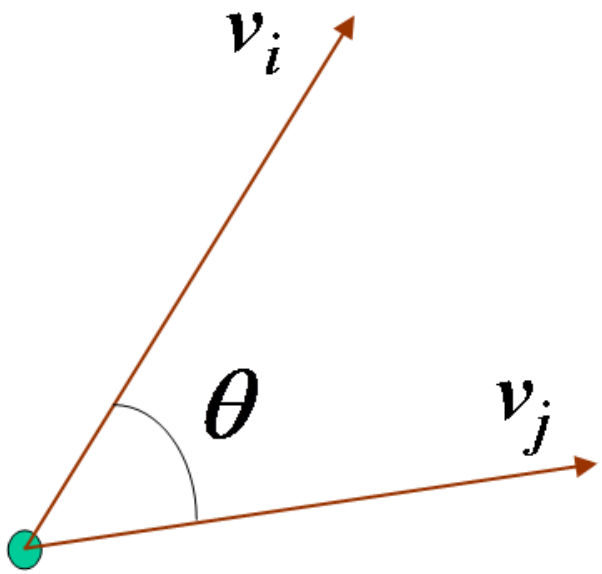
Therefore, the maximum problem becomes

$$\text{maximize} \quad \sum w_{ij}\left(1-\cos\theta_{ij}\right) \tag{6.18}$$

Subjecto $\theta_{ij} \in R$.

9.1. Max-Cut Problems

9.1.4 Bounds of Semidefinite Relaxation

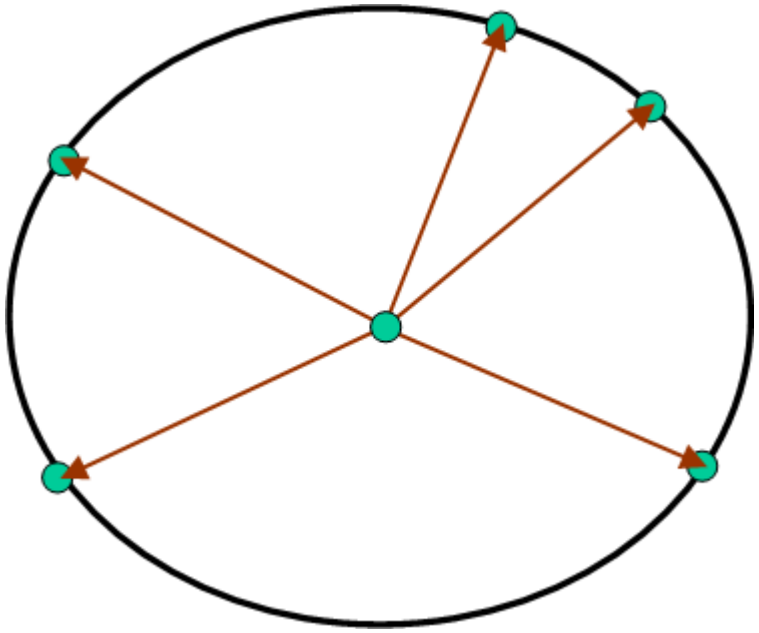


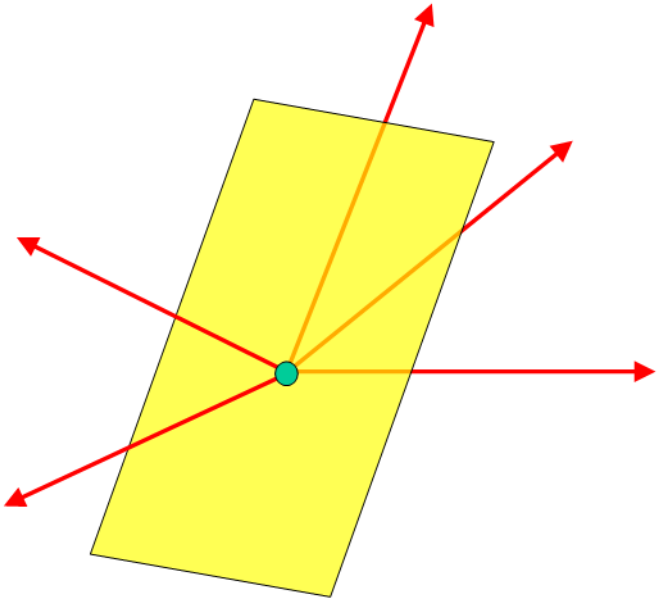
Goemans and Williamson 1994 proved that LMI relaxations of MAXCUT provide solutions at least 87% the optimal value. They formulate a slightly different relaxation form.

9.1. Max-Cut Problems

9.1.4 Bounds of Semidefinite Relaxation

The geometric intuition behind is to embed the vertices of the graph on the unit sphere such that vertices that are joined by edges are far apart. To choose a random hyperplane, we can choose a random normal vector

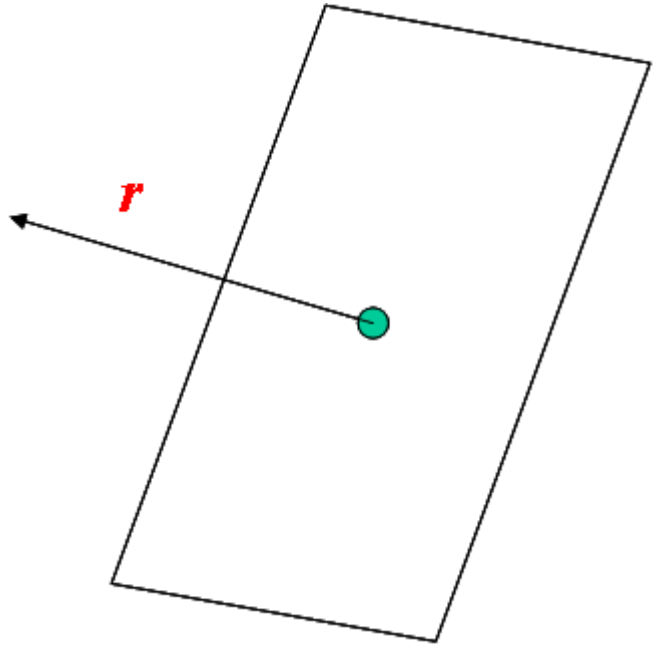


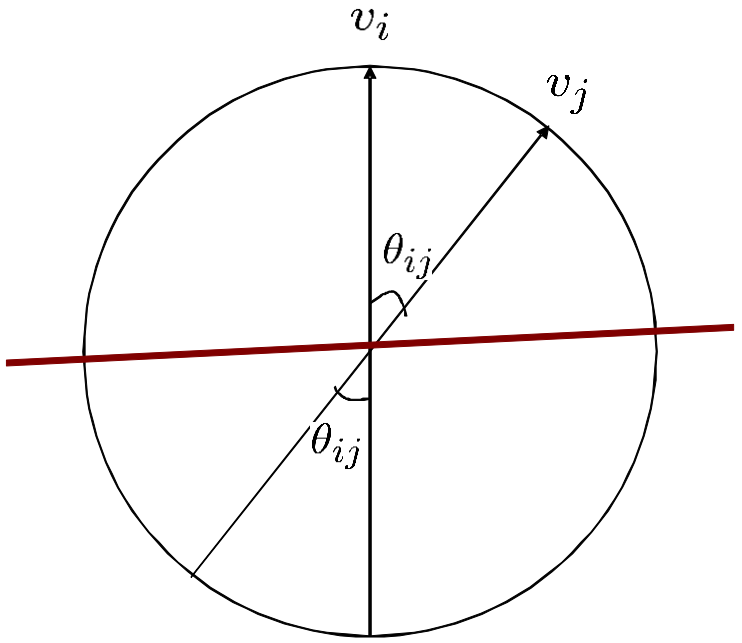


9.1. Max-Cut Problems

9.1.4 Bounds of Semidefinite Relaxation

If $r = (r_1, ..., r_n)$, and $r_1, \ldots, r_n \sim N(0,1)$, then the direction of r is uniformly distributed over the n-dimensional unit sphere. The probability that two vectors are separated by a random hyperplane is θ / π.





9.1. Max-Cut Problems

9.1.4 Bounds of Semidefinite Relaxation

The random hyperplane can be characterized by its normal vector p, which is chosen to be uniformly distributed on the unit sphere (e.g., by suitably normalizing a standard multivariate Gaussian random variable). Then, according to the description above, the rounded solution is given by $x_i = sign(p^T v_i)$. The expected value of this solution can then be written as

$$c_{sdp-\exp ected} = \sum_{ij} w_{ij}[sign(p^T v_i) \neq sign(p^T v_j)]$$

$$= \sum_{ij} w_{ij} \frac{\theta_{ij}}{\pi} = \frac{1}{\pi} \sum_{ij} w_{ij} \arccos(v_i \cdot v_j)$$

(6.19)

9.1. Max-Cut Problems

9.1.4 Bounds of Semidefinite Relaxation

On the other hand, the solution of the SDP gives an upper bound on the cut capacity equal to

$$c_{sdp-upper-bound} = \max \frac{1}{2} \sum_{ij} w_{ij}(1 - v_i \cdot v_j) \tag{6.20}$$

So we have

$$c_{sdp-\text{exp}ected} \leq c_{true} \leq c_{sdp-upper-bound} \tag{6.21}$$

9.1. Max-Cut Problems

9.1.4 Bounds of Semidefinite Relaxation

Moreover, the upper and lower bound are related, because we can always find a constant α such that

$$\alpha\left[\frac{1}{2}(1-t)\right] \leq \frac{1}{\pi}\arccos(t) \qquad (6.22)$$

for all $t \in [-1,+1]$, which indicates

$$c_{sdp-upper-bound} \leq \frac{1}{\alpha} c_{sdp-expected} \qquad (6.23)$$

(Hint: substitue $t = v_i \cdot v_j$ for each pair in (6.19) and (6.20))

9.1. Max-Cut Problems

9.1.4 Bounds of Semidefinite Relaxation

Thus, we have

$$c_{sdp-\exp ected} \le c_{true} \le c_{sdp-upper-bound} \le \frac{1}{\alpha} c_{sdp-\exp ected} \qquad (6.24)$$

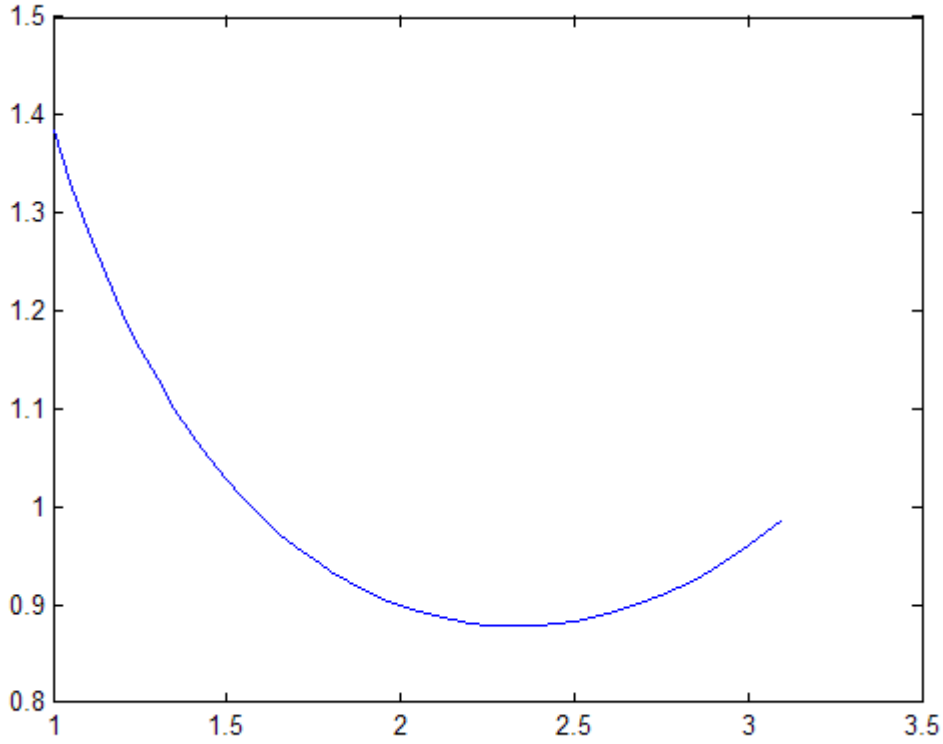
This means LMI relaxations of MAXCUT provide solutions at least α the optimal value.

$$\alpha c_{true} \le \alpha c_{sdp-upper-bound} \le c_{sdp-\exp ected} \qquad (6.25)$$

We know that the best possible (i.e., largest) such constant is $\alpha = 0.878$. But, how to solve α from (6.22)?

9.1. Max-Cut Problems

9.1.4 Bounds of Semidefinite Relaxation



Actually, a numerical solution is OK

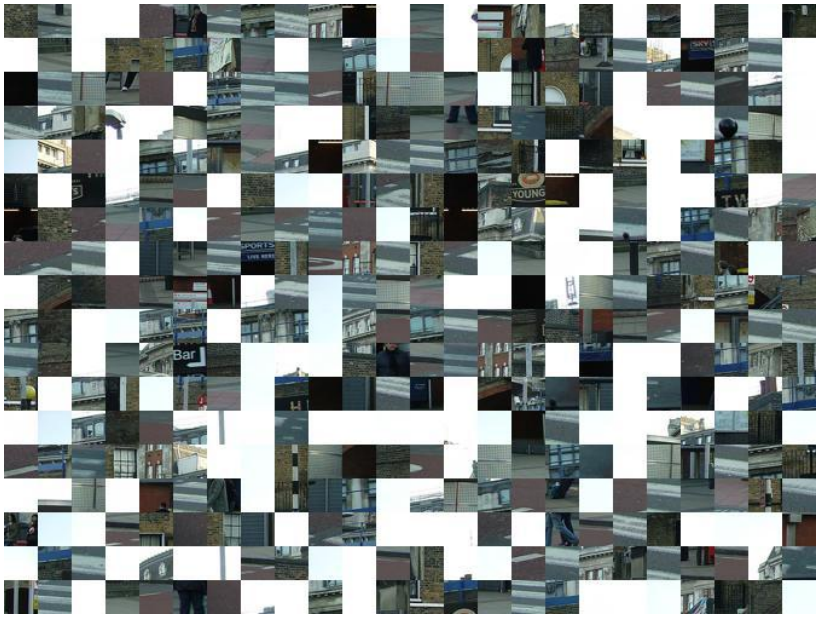
9.2. Jigsaw Puzzle

Jigsaw Puzzle 最初是在 1760 年左右由英国地图制作者作为娱乐方式设计的，目标是从给定的一组图像碎片中重建原始图像。

该问题吸引了很多计算机科学家，解决这个问题的第一个计算方法可以追溯到 1964 年，后来证明该组合问题是 NP-Hard 问题。

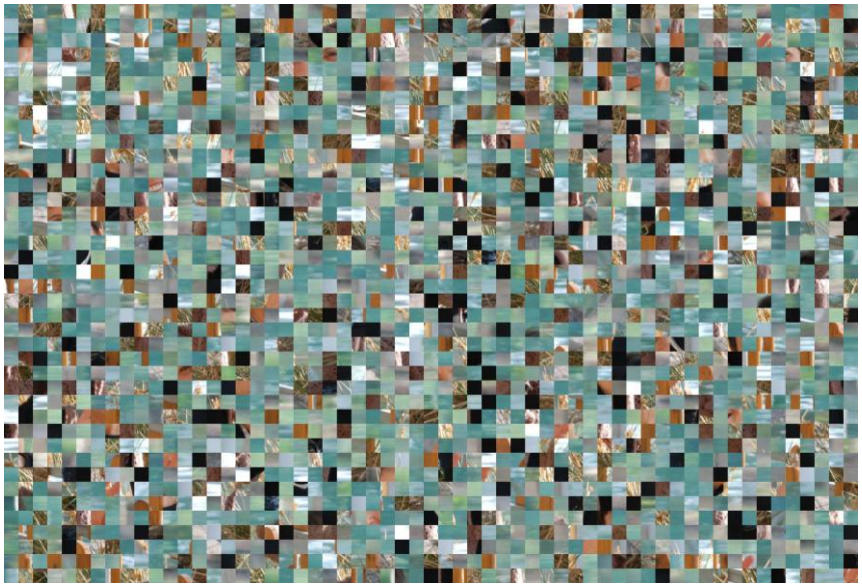
使用计算机解决拼图问题可以应用于解决一些现实问题。例如，在计算机图形中可应用于解决自动修复壁画或三维重建裂缝物体等问题。

9.2. Jigsaw Puzzle



MIT 数据集，28×28 小块，共 432 块

9.2. Jigsaw Puzzle



McGill 数据集，28×28 小块，共 2360 块

9.2. Jigsaw Puzzle

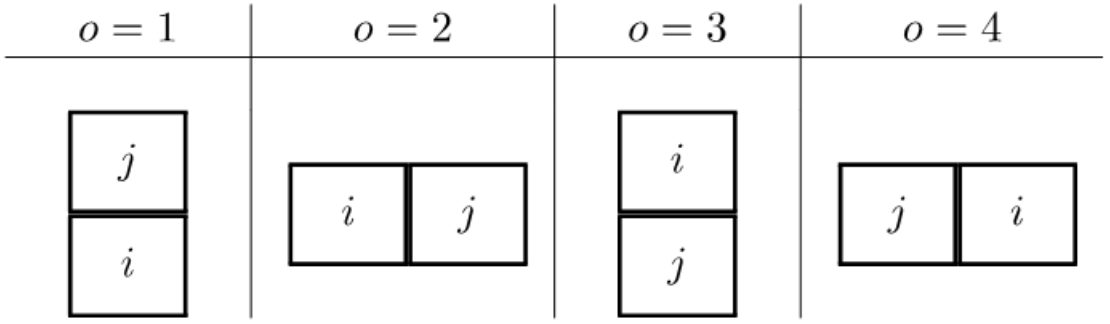
对拼图问题的研究过程中产生了自动拼图解法。自动拼图解法通常由两个部分组成：估计不同碎片之间的匹配度及组合策略。

现有的方法倾向于使用贪心算法或启发式全局求解法方法，例如遗传算法。但是已知的贪心算法对局部极小值敏感，遗传算法的性能在很大程度上依赖于拟合函数和交叉运算的选择。

这里关注的算法引入了一系列 LP 松弛，采用的连续逼近是全局的和凸的，所以对局部极小值的敏感度较低。同时使用加权的 L_1 惩罚，增加了对成对匹配中失配的鲁棒性。

9.2. Jigsaw Puzzle

给定两个图片块 i,j，它们的空间方位有下述 **4** 种情况：



其中字母 o 用来表示这四种空间位置。
我们用一个三元组 (i,j,o)来表示两个图片块和它们的位置。
我们假设图片$+x$ 轴方向向右，$+y$ 轴方向向下，每个图片块有坐标 $(x_i,y_i)\,(x_j,y_j)$

9.2. Jigsaw Puzzle

对两个方位摆放正确的图片块，定义期望位移（desired offset）

$$\delta_o^x = \begin{cases} 0 & \text{if } o = 1 \\ -1 & \text{if } o = 2 \\ 0 & \text{if } o = 3 \\ 1 & \text{if } o = 4 \end{cases}$$

$$\delta_o^y = \begin{cases} 1 & \text{if } o = 1 \\ 0 & \text{if } o = 2 \\ -1 & \text{if } o = 3 \\ 0 & \text{if } o = 4 \end{cases}$$

相当于 $\delta_o^x = x_i - x_j$，$\delta_o^y = y_i - y_j$

给定一个图片块组合三元组 (i, j, o)，计算两个图片块之间的 Mahalanobis Gradient Compatibility (MGC) 距离 D_{ijo}，并求匹配权重 ω_{ijo}。

9.2. Jigsaw Puzzle

匹配权重 ω_{ijo} 定义如下：

$$w_{ijo} = \frac{\min(\min_{k\neq i}(D_{kjo}), \min_{k\neq j}(D_{iko}))}{D_{ijo}},$$

相当于 D_{ijo} 和其他最优块匹配在相同空间位置下的反比，当两个图像块匹配距离较小时，该权重值较大。

下面，给定所有图像块集合 $V = \{i | i = 1 \ldots N \times M\}$，其中 M 和 N 是拼图的维度（横向纵向拼图块的个数）。并定义所有可能的拼图块三元组集合：

$$U = \{(i, j, o) | i \in V, j \in V, o = 1, 2, 3, 4\}$$

每一个三元组都有一个匹配权重 ω_{ijo} 与之关联。

9.2. Jigsaw Puzzle

使用零范数定义目标函数（拼图 cost）如下：

$$C(\mathbf{x}) = \sum_{(i,j,o)\in U} w_{ijo}|x_i - x_j - \delta_o^x|_0,$$

$$C(\mathbf{y}) = \sum_{(i,j,o)\in U} w_{ijo}|y_i - y_j - \delta_o^y|_0,$$

拼图问题可以表达为下面的组合优化问题：

$$\begin{aligned}
&\text{minimize: } C(\mathbf{x}) + C(\mathbf{y}) \\
&\text{subject to: } \forall i, x_i \in \mathbb{N}, 1 \le x_i \le M \\
&\qquad\qquad \forall j, y_j \in \mathbb{N}, 1 \le y_j \le N \\
&\qquad\qquad \forall i, \forall j, |x_i - x_j| + |y_i - y_j| > 0
\end{aligned}$$

优化目标函数是 x 和 y 两个方向的拼图 cost，前两个约束是保证每个图片块都处在“整数位置”上，第三个约束保证图片块不会重叠。然而，可行域和目标都是离散非凸的，这是个 NP 难问题。

9.2. Jigsaw Puzzle

凸松弛：$L_0\ norm \ \rightarrow \ L_1\ norm$

集合 A：在方向 o 和子块 x_i 的已知时，找到最接近的子块 x_j

$$\begin{aligned} C^+(\mathbf{x},\mathbf{y}) =& C^+(\mathbf{x}) + C^+(\mathbf{y}) \\ =& \sum_{(i,j,o)\in A} w_{ijo}|x_i - x_j - \delta_o^x|_1 \\ +& \sum_{(i,j,o)\in A} w_{ijo}|y_i - y_j - \delta_o^y|_1 \end{aligned}$$

A 采用迭代的方法获得：

$$A^{(k)} = \{(i,j,o) \in U^{(k)} : j = \underset{j:(i,j,o)\in U^{(k)}}{\arg\min} D_{ijo}\},$$

9.2. Jigsaw Puzzle

在已知 A^k 的情况下，求解各子块的相对坐标 x, y：

$$\arg\min_{\mathbf{x}} \sum_{(i,j,o)\in A^{(k)}} w_{ijo}|x_i - x_j - \delta_o^x|_1.$$

转化为线性规划 (LP) 问题：

$$\begin{aligned}
\arg\min_{\mathbf{x},\mathbf{h}} \quad & \sum_{(i,j,o)\in A^{(k)}} w_{ijo}h_{ijo} \\
\text{subject to} \quad & h_{ijo} \geq x_i - x_j - \delta_o^x, \qquad (i,j,o) \in A^{(k)} \\
& h_{ijo} \geq -x_i + x_j + \delta_o^x, \quad (i,j,o) \in A^{(k)}.
\end{aligned}$$

9.2. Jigsaw Puzzle

LP 求解出的相对坐标 x, y 非常接近整数解（$< 10e-6$）

$$R^{(k)} = \{\forall (i,j,o) \in A^{(k)} : |x_i - x_j - \delta_o^x| > 10^{-5}\}.$$

将不符合整数解或临近的部分剔除掉：

$$U^{(k)} = U^{(k-1)} - R^{(k)},$$

9.2. Jigsaw Puzzle

采用 MGC (Mahalanobis Gradient Compatibility)衡量块与块之间的兼容性。

MGC 描述的是图像块边界上的亮度梯度，其惩罚边界上亮度的梯度变化而不是亮度变化。另外不同于不同于计算欧氏距离上的梯度，利用不同颜色之间的协方差矩阵，来计算马氏距离。两张相邻的图片能够拼接，需要其边界处有着相近的梯度。

9.2. Jigsaw Puzzle

计算左边图像最后两列梯度

$$G_{iL}(p,c) = x_i(p,P,c) - x_i(p,P-1,c)$$

取均值

$$\mu_{iL}(c) = \frac{1}{P}\sum_{p=1}^{P} G_{iL}(p,c)$$

计算从 x_i 到 x_j 的兼容性

$$D_{LR}(x_i,x_j) = \sum_{p=1}^{P}(G_{ijLR}(p) - \mu_{iL})S_{iL}^{-1}(G_{ijLR}(p) - \mu_{iL})^T$$

计算从 x_i 到 x_j 的兼容性，其中 $G_{ijLR}(p,c) = x_j(p,1,c) - x_i(p,P,c)$ 为第 p 行 x_i 右侧到 x_j 左侧的梯度。S_{iL} 为边界上梯度不同颜色之间的协方差矩阵。为了保证 S_{iL} 的非奇异，可以在 G_{iL} 上加上九个虚拟梯度以保证 G_{iL} 列满秩

9.2. Jigsaw Puzzle

MATLAB 数据存储结构：

- 拼图块：$p_1 \times p_2 \times 3 \times n$ 四维数组（n 个 RGB 小块）
- MGC 距离 D: $n \times n \times 4$ 三维数组（D_{ijo}）
- 匹配权重 w: $n \times n \times 4$ 三维数组（w_{ijo}）
- 可行拼接集合 $U^{(k)}$: $n \times n \times 4$ 三维逻辑数组（$U_{ijo} = 1$ 代表未被剔除）
- 本次迭代拼接集合 $A^{(k)}$: $n \times 3$ $[i, j, o]$ 三元组

9.2. Jigsaw Puzzle

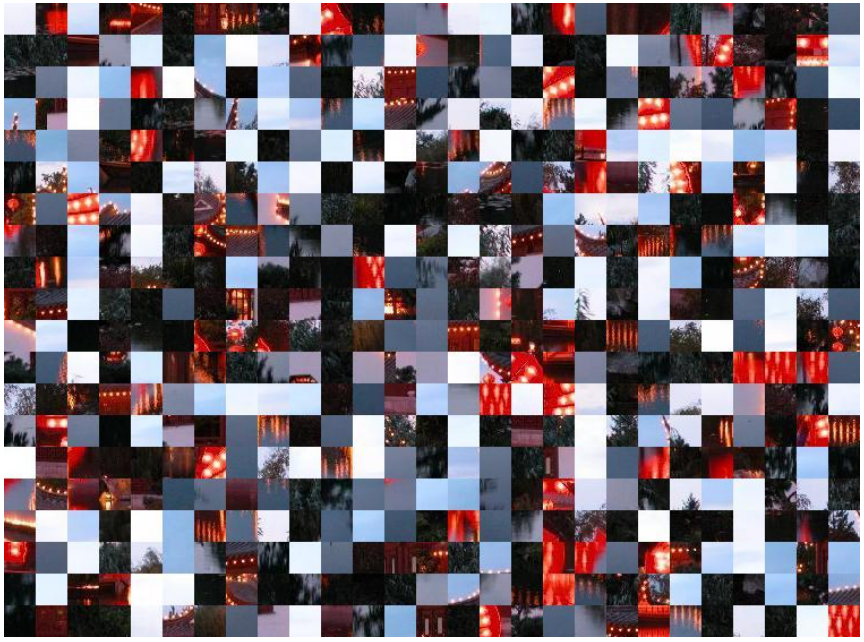
使用 CVX 工具箱求解线性规划问题（以 x 坐标为例）：

$$\begin{aligned}
\arg\min_{\mathbf{x},\mathbf{h}} \quad & \sum_{(i,j,o)\in A^{(k)}} w_{ijo} h_{ijo} \\
\text{subject to} \quad & h_{ijo} \geq x_i - x_j - \delta_o^x, \qquad (i,j,o) \in A^{(k)} \\
& h_{ijo} \geq -x_i + x_j + \delta_o^x, \quad (i,j,o) \in A^{(k)}.
\end{aligned}$$

```
indA = sub2ind([n,n,4], i, j, o);
cvx_begin
  variables x(n) hx(4*n);
  minimize ( w(indA).' * hx );
  subject to
    -hx <= x(i) - x(j) - delta_x(o) <= hx;
    1 <= x <= M;
cvx_end
```

对于 y 坐标也采用同样方法求解。

9.2. Jigsaw Puzzle

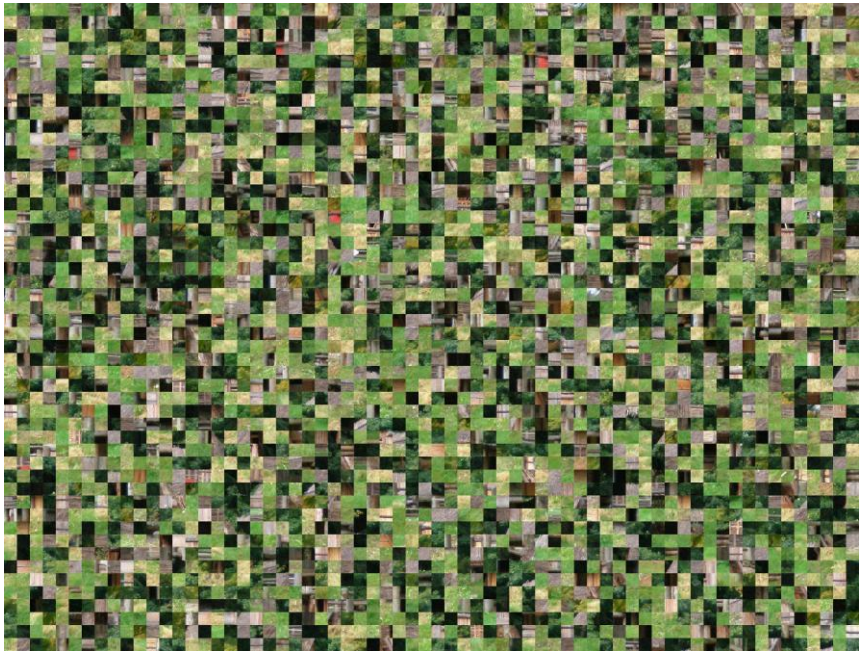


McGill 数据集，28×28 小块，共 540 块

9.2. Jigsaw Puzzle

McGill 数据集，28×28 小块，共 850 块

9.2. Jigsaw Puzzle



McGill 数据集，28×28 小块，共 3300 块

9.3. MM Algorithm and Reweighted l_1

The *Majorization-Minimization (MM) algorithm* is a speical convex relaxation algorithm. It first finds a surrogate function that majorizes the objective function and then optimizes this surrogate functions until a local optimal solution is reached [3]-[7]. This local optimal solution will be taken as the local optimal solution to the original problem, if some conditions could be satisfied.

Let us consider a minimization problem, say

$$\min_{\boldsymbol{v}} \ f(\boldsymbol{v}) \ \text{ s.t. } \boldsymbol{v} \in \Omega \tag{1.23}$$

with Ω is a convex set.

Let $\boldsymbol{v}^{(j)}$ represent a fixed value of $\boldsymbol{v}$ and $g\left(\boldsymbol{v} \mid \boldsymbol{v}^{(j)}\right)$ denote a real-valued function that depends on $\boldsymbol{v}^{(j)}$. The surrogate function $g\left(\boldsymbol{v} \mid \boldsymbol{v}^{(j)}\right)$ is said to be majorize function $f(\boldsymbol{v})$ at the point $\boldsymbol{v}^{(j)}$ provided

$$g\left(\boldsymbol{v} \mid \boldsymbol{v}^{(j)}\right) \geq f(\boldsymbol{v}) \ \text{ for all } \ \boldsymbol{v} \tag{1.24}$$

$$g\left(\boldsymbol{v}^{(j)} \mid \boldsymbol{v}^{(j)}\right) = f\left(\boldsymbol{v}^{(j)}\right) \tag{1.25}$$

That is, $g\left(\boldsymbol{v} \mid \boldsymbol{v}^{(j)}\right)$ lies above $f(\boldsymbol{v})$ and is tangent to it at the point $\boldsymbol{v}^{(j)}$. We can minimize the surrogate function $g(\boldsymbol{v} \mid \boldsymbol{v}^{(j)})$ rather than the actual function $f(\boldsymbol{v})$.

If $f(\boldsymbol{v} \mid \boldsymbol{v}^{(m)})$ is a concave function, therefore, below its tangent. Starting with a feasible solution $\boldsymbol{v}^{(0)} \in \Omega$, we can iteratively solve the following simple function to force the given objective function $f(\boldsymbol{v})$ downhill

$$\boldsymbol{v}^{(j+1)} = \arg\min_{\boldsymbol{v} \in \Omega} \ f\left(\boldsymbol{v}^{(j)}\right) + \nabla f\left(\boldsymbol{v}^{(j)}\right)^T \left(\boldsymbol{v} - \boldsymbol{v}^{(j)}\right) \tag{1.26}$$

until convergence. Each iteration then gives the solution to a convex optimization problem that can be easily calculated.

9.3. MM Algorithm and Reweighted l_1

To reconstruct $\boldsymbol{x}$, we usually seek the sparsest solution can be obtained via solving the following constrained 0-norm minimization problem

$$(P_0') \quad \min_{\boldsymbol{x}} |\boldsymbol{x}|_0 \quad \text{s.t. } A\boldsymbol{x} = \boldsymbol{b} \tag{1.28}$$

The starting point behind any re-weighted algorithm [9]-[10] is the one-to-one correspondence between the optimal solution of Problem (P_0) and the optimal solution of the following optimization problem

$$(P_{\log}) \quad \min_{\boldsymbol{x}} \sum_i \log |x_i| \quad \text{s.t. } A\boldsymbol{x} = \boldsymbol{b} \tag{1.29}$$

because $|\boldsymbol{x}|_0 = \lim_{p\to 0} \sum_i |x_i|^p$ and $\lim_{p\to 0} \frac{1}{p} \sum_i (|x_i|^p - 1) = \sum_i \log |x_i|$ [11]-[13].

To avoid calculating $\log |x_i|$ when $x_i = 0$, we consider the following problem instead

$$(P_{\log}') \quad \min_{\boldsymbol{x}} \sum_i \log (|x_i| + \epsilon) \quad \text{s.t. } A\boldsymbol{x} = \boldsymbol{b} \tag{1.30}$$

where ϵ is a small positive constant.

Since $f(z) = |z|$ is not smooth at $z = 0$, we turn to study an equivalent form of Problem $(P_{\log}')$. As demonstrated in [10], Problem $(P_{\log}')$ is equivalent to the following optimization problem

$$(P_{\log}'') \quad \min_{\boldsymbol{x},\boldsymbol{u}} \sum_i \log (u_i + \epsilon) \quad \text{s.t. } A\boldsymbol{x} = \boldsymbol{b}, \quad |x_i| \le u_i \tag{1.31}$$

9.3. MM Algorithm and Reweighted l_1

The key idea of classical re-weighted algorithm [9]-[13] is to iteratively solve Problem $(P''_{\log})$ via a special Majorization-Minimization algorithm.

Taking the partial derivative of the surrogate function $\sum_i \log(u_i + \epsilon)$, we have the optimization problem for the jth iteration as

$$\left(\boldsymbol{x}^{(j+1)}, \boldsymbol{u}^{(j+1)}\right) = \arg\min_{\boldsymbol{x},\ \boldsymbol{u}} \quad \sum_i \frac{u_i}{u_i^{(j)} + \epsilon} \tag{1.32}$$

$$\text{s.t.} \quad A\boldsymbol{x} = \boldsymbol{b},\ \ |x_i| \le u_i \tag{1.33}$$

Eliminating $\boldsymbol{u}$, this optimization problem is equivalent to

$$\boldsymbol{x}^{(j+1)} = \arg\min_{\boldsymbol{x}} \quad \sum_i \frac{|x_i|}{\left|x_i^{(j)}\right| + \epsilon} \ \text{ s.t. } A\boldsymbol{x} = \boldsymbol{b} \tag{1.34}$$

This is a constrained 1-norm minimization problem that can be quickly solved by many techniques, including simplex algorithm [9]-[10].

The re-weighted algorithms were named after the inversely proportional weighting coefficient $\frac{1}{\left|x_i^{(j)}\right|+\epsilon}$ shown in the objective function of Problem (1.34). The intuitive explanation for this re-weighted trick is to force a potential solution $\boldsymbol{x}$ to shrink on the indices where $|x_i|$ is small.

The constant ϵ is introduced to guarantee iteration stability and to ensure that a zero-valued component x_i in $\boldsymbol{x}$ does not strictly prohibit a nonzero x_i in the next round of iteration. However, ϵ is usually set as a very small value at the beginning of iterations. If the ith entry $x_i^{(j)}$ becomes about zero after the jth iteration, the corresponding weighting coefficient will then become a very large number satisfying $\frac{1}{\epsilon} \gg 0$. This generally prevents us to reconsider the possibility to include ith entry $x_i^{(j)}$ as a part of the non-zero solution after the jth iteration.

9.3. MM Algorithm and Reweighted l_1

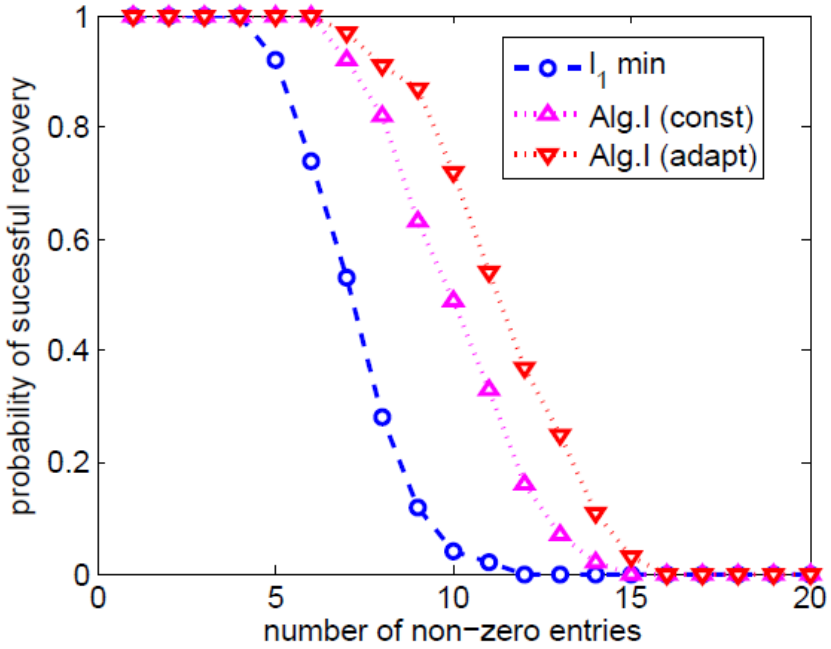


Fig. 1.1 The (experimentally determined) probability of successful recovery for classical and new re-weighted algorithms vs. the number of non-zero entries. l_1-min refers to directly solving Problem (P_1) described as (??)-(??); Alg.I (const) refers to Algorithm I with constant thresholding scalar $\epsilon = 0.05$. Alg.I (adap) refers to Algorithm I with adaptive thresholding scalar sequence $\epsilon^{(j)} = \max\{0.5 - 0.05 \cdot j, 0.05\}$.

9.4. Joint Estimation

Suppose we have n testing tasks and m testees, with m, $n \in \mathrm{N}$. The testing result for the ith testee on the jth testing task is denoted by a binary variable as $x_{i,j} \in \{0, 1\}$, $i = 1, \ldots, m$ and $j = 1, \ldots, n$. If the ith testee successfully finishes the jth testing task, we set $x_{i,j} = 1$; otherwise, we set $x_{i,j} = 0$. This leads a $m \times n$ binary observation matrix $X = (x_{i,j}) \in \{0, 1\}^{m \times n}$.

If all the testing tasks have generally equal difficulties, the ranks of the testees can be easily calculated in terms of $\sum_{i=1}^{m} x_{i,j}$, the number of tasks that had been successfully finished for every testee. Similarly, if all the testees have generally equal capabilities, the difficulty levels of the testing tasks can be directly calculated in terms of $\sum_{j=1}^{n} x_{i,j}$; the number of testees had been successfully finished for each task. However, when the testing tasks have different difficulties; meanwhile, the testees have different capabilities, and the above calculations become meaningless.

9.4. Joint Estimation

To solve this problem, we assume in this paper that the relative difficulty level of the jth testing task can be denoted by a weighting coefficient $q_j \in [0, 1]$, $j = 1, \ldots, n$; also, the relative capability of the ith testee can be denoted by a weighting coefficient $p_i \in [0, 1], i = 1, \ldots, m$. The larger the q_j, the easier the is jth testing task; also, the larger the p_i, the stronger is the ith testee.

If these assumptions hold, the probability that the ith testee successfully finishes the jth testing task should be $(p_i q_j)$. The probability mass function f of the distribution over possible outcomes $x_{i,j}$ should be a Bernoulli distribution

$$f(x_{i,j}|p_i q_j) = (p_i q_j)^{x_{i,j}} (1 - p_i q_j)^{1-x_{i,j}}. \tag{1}$$

The likelihood can then be given as

$$L = \prod_{i=1}^{m} \prod_{j=1}^{n} f(x_{i,j}|p_i q_j) = \prod_{i=1}^{m} \prod_{j=1}^{n} (p_i q_j)^{x_{i,j}} (1 - p_i q_j)^{1-x_{i,j}}. \tag{2}$$

9.4. Joint Estimation

Note that, the maximization of L is equivalent to the maximization of its logarithm

$$\ln L = \sum_{i=1}^{m} \sum_{j=1}^{n} [x_{i,j} (\ln p_i + \ln q_j) + (1 - x_{i,j}) \ln(1 - p_i q_j)]. \tag{3}$$

We design a coordinate descent algorithm to find a local optimal parameter set of p_i and q_j.

Suppose we fix q_j and consider the influence of p_i on $\ln L$ only. Let $\partial \ln L / \partial p_i = 0$, we can get

$$\sum_{j=1}^{n} \left[\frac{x_{i,j}}{p_i} - \frac{(1 - x_{i,j}) q_j}{(1 - p_i q_j)} \right] = 0 \tag{4}$$

or equivalently

$$\sum_{j=1}^{n} \frac{x_{i,j} - p_i q_j}{p_i (1 - p_i q_j)} = 0. \tag{5}$$

However, we cannot find an elegant closed-form solution for p_i from (5), especially when n is large.

9.4. Joint Estimation

Moreover, we have

$$\frac{\partial^2 \ln L}{\partial p_i^2} = -\sum_{j=1}^{n} \left[\frac{x_{i,j}}{p_i^2} + \frac{(1 - x_{i,j}) q_j^2}{(1 - p_i q_j)^2} \right] < 0. \tag{6}$$

This indicates that $\ln L$ is convex in terms of p_i when q_j is fixed. Therefore, we can use either the gradient descent algorithm or Newton's method to numerically find the optimal solution p_i^* for $\ln L$ when no constraints are considered.

Since we require $p_i \in [0, 1]$, we still need to check whether the optimal value of $\ln L$ can be reached at $p_i = \sum_{j=1}^{n} x_{i,j} / \sum_{j=1}^{n} q_j$. When $p_i \to 0^+$, $\ln L \to -\infty$. Therefore, when we fix q_j, the optimal value of $\ln L$ should be reached at

$$p_i = \min \left\{ p_i^*, 1 \right\}. \tag{7}$$

Noting the commutation relation between p_i and q_j, when we fix p_i, the optimal value of $\ln L$ in terms of q_j can be reached via Newton's algorithm, too.

Summarizing the above analysis, we propose the following coordinate descent algorithm.

9.4. Joint Estimation

In this model, we assume that the relative difficulty level of the jth testing task can be denoted by a weighting coefficient $\beta_j \in (0, +\infty)$, $j = 1, \ldots, n$; also, the relative capability of the ith testee can be denoted by a weighting coefficient $\alpha_i \in (0, +\infty)$, $i = 1, \ldots, m$. We denote the probability that the ith testee passes the jth testing task as $\theta_{i,j}$.

In Section II, we do not make any *a priori* assumption of $\theta_{i,j}$. This may bring overestimation. For example, if a special player gets a base hit, it does not indicate that this player can always get bases hit. Therefore, it is more safe to assume that $\theta_{i,j}$ follows a beta distribution characterized by α_i and β_j [13]. The probability density function of $\theta_{i,j}$ given α_i and β_j is

$$\pi(\theta_{i,j}|\alpha_i, \beta_j) = \frac{\Gamma(\alpha_i + \beta_j)}{\Gamma(\alpha_i)\Gamma(\beta_j)} \theta_{i,j}^{\alpha_i - 1} (1 - \theta_{i,j})^{\beta_j - 1} \qquad (8)$$

where $\Gamma()$ denotes the Gamma function. The larger the β_j, the more difficult is the jth testing task; also, the larger the α_i, the stronger is the ith testee.

9.4. Joint Estimation

Given $\theta_{i,j}$, the probability mass function f of the distribution over possible outcomes $x_{i,j}$ should be a Bernoulli distribution

$$f(x_{i,j}|\theta_{i,j}) = \theta_{i,j}^{x_{i,j}}(1-\theta_{i,j})^{1-x_{i,j}}. \tag{9}$$

Combining (8) and (9), we can get the probability mass function f of the distribution over $x_{i,j}$ as

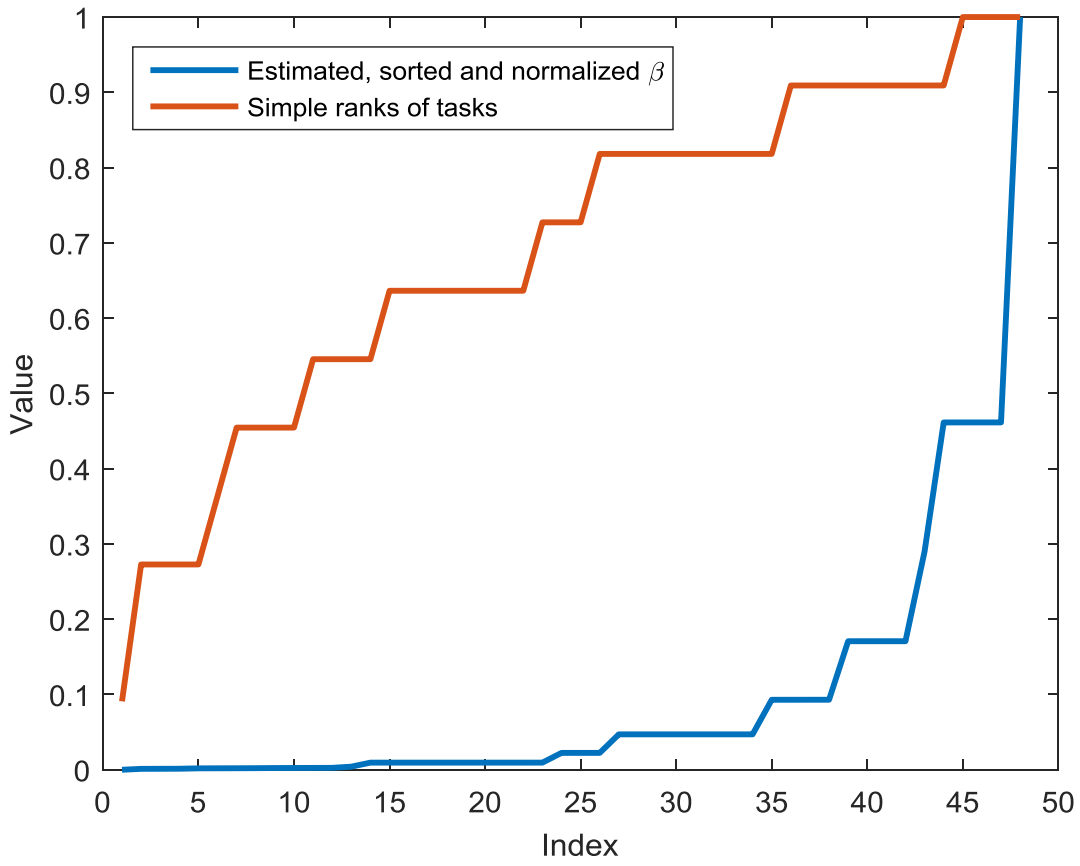
$$\begin{aligned} f(x_{i,j}|\alpha_i,\beta_j) &= \int_0^1 f(x_{i,j}|\theta_{i,j})\pi(\theta_{i,j}|\alpha_i,\beta_j)d\theta_{i,j} \\ &= \frac{\alpha_i^{x_{i,j}}\beta_j^{1-x_{i,j}}}{\alpha_i+\beta_j}. \end{aligned} \tag{10}$$

Thus, the likelihood can then be given as

$$L = \prod_{i=1}^{i=m}\prod_{j=1}^{j=n} f(x_{i,j}|\alpha_i,\beta_j) = \prod_{i=1}^{i=m}\prod_{j=1}^{j=n}\frac{\alpha_i^{x_{i,j}}\beta_j^{1-x_{i,j}}}{\alpha_i+\beta_j}. \tag{11}$$

9.4. Joint Estimation

在测试数据较少时，使用 Beta 分布型的先验假设，有助于同时估计测试任务的难度和测试系统的智能程度



9.5. Robust PCA

In many situations, we need to decompose a matrix $A \in \mathbb{R}^{m,n}$ into the sum of two matrices. One matrix B has small rank; while the other matrix C is sparse [1]-[2]. So, we can reach the following optimization problem

$$\min_{B,C \in \mathbb{R}^{m,n}} \quad \text{rank}(B) + \|C\|_0 \tag{2}$$

$$\text{s.t.} \quad A = B + C \tag{3}$$

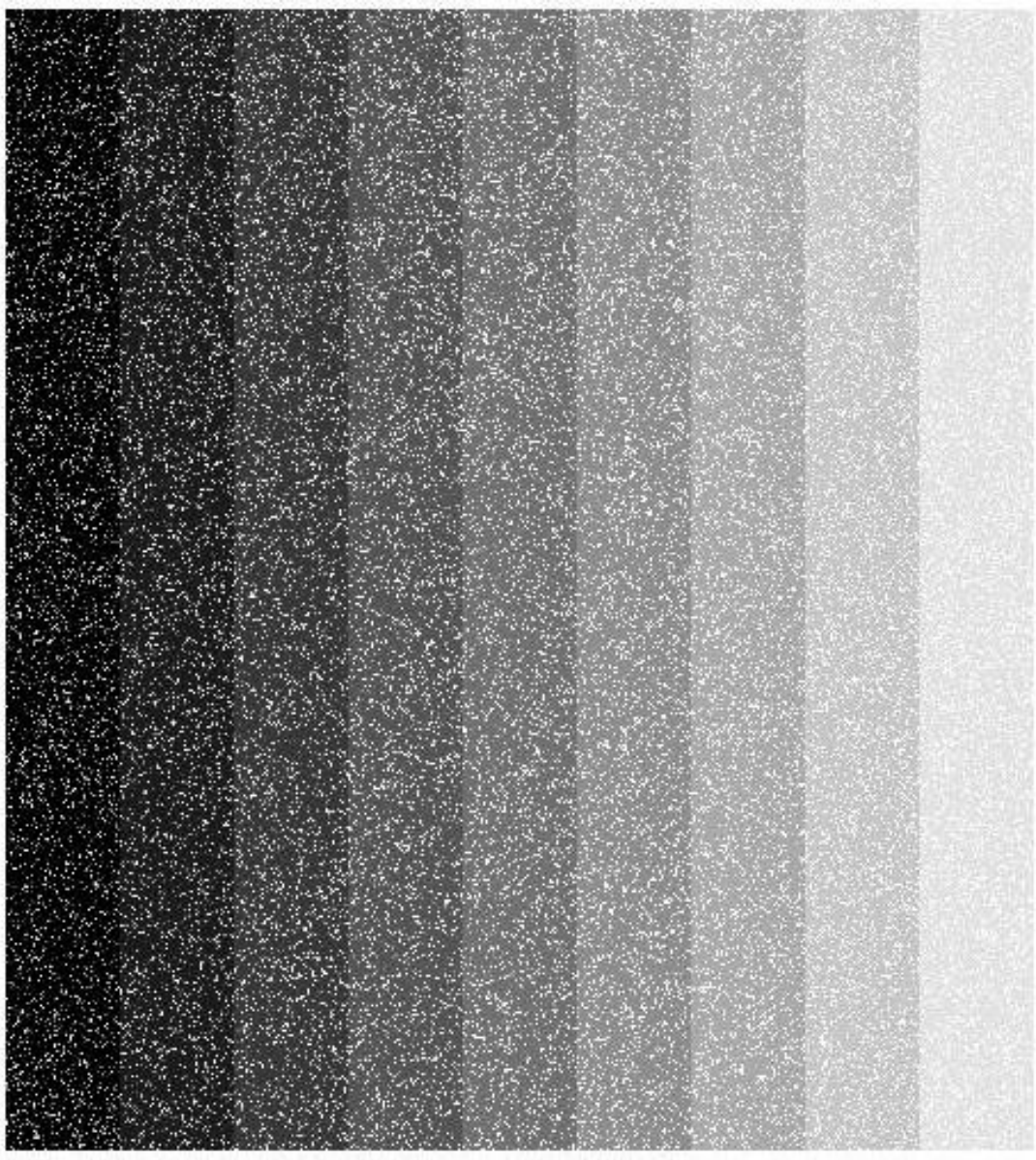
However, this is not a convex optimization problem. So, we often relax the problem and turn to solve the following optimization problem

$$\min_{B,C \in \mathbb{R}^{m,n}} \quad \|B\|_* + \|C\|_1 \tag{4}$$

$$\text{s.t.} \quad A = B + C \tag{5}$$

where $\|B\|_* = \text{trace}(\sqrt{B^T B})$ denotes the nuclear norm of matrix B.

9.5. Robust PCA



9.6. References

[1] http://en.wikipedia.org/wiki/Cut_(graph_theory)

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, *Introduction to Algorithms*, 3rd ed., MIT Press, 2009.

[3] R. M. Karp, "Reducibility among combinatorial problems," *Complexity of Computer Computation*, R. E. Miller, J. W. Thacher, eds., Plenum Press, New York, pp. 85-103, 1972.

[4] M. X. Goemans, D. P. Williamson, "Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming," *Journal of the ACM*, vol. 42, no. 6, pp. 1115-1145, 1995.

[5] C. H. Papadimitriou, K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Dover Publications, Mineola, New York, 1998.

[6] A. Megretski, "Relaxations of quadratic programs in operator theory and system analysis," In *Systems, Approximation, Singular Integral Operators, and Related Topics*, A. A. Borichev, N. K. Nikolski, eds., Bordeaux, 2000, *Operator Theory Advances and Applications*, vol. 129, Birkhäuser Verlag, Boston, MA, 2001, pp. 365-392. http://web.mit.edu/ameg/www/images/iwota_final.pdf

[7] E. de Klerk, *Aspects of Semidefinite Programming: Interior Point Algorithms and Selected Applications*, Kluwer Academic Publishers, 2004.

[8] Z. Luo, W. Ma, A. M.-C. So, Y. Ye, S. Zhang, "Semidefinite relaxation of quadratic optimization problems," *IEEE Signal Processing Magazine*, vol. 27, no. 3, pp. 20-34, 2010.

[9] J. Chen, *Introduction to Tractability and Approximability of Optimization Problems*, http://faculty.cs.tamu.edu/chen/notes/opt.pdf

[10] R. Yu, C. Russell, L. Agapito, "Solving Jigsaw puzzles with linear programming" https://arxiv.org/abs/1511.04472

[11] F. A. Andalo, G. Taubin, S. Goldenstein, "PSQP: Puzzle solving by quadratic programming," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 2, pp. 385-396, 2017.

[12] K. Son, J. Hays, D. B. Cooper, "Solving square jigsaw puzzle by hierarchical loop constraints," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 41, no. 9, pp. 2222-2235, 2019.

[13] Computational Jigsaw Puzzle Solving. http://icvl.cs.bgu.ac.il/automatic-jigsaw-puzzle-solving/

[14] https://github.com/guansanghai/CVX-Jigsaw

[15] C. Zhang, Y. Liu, L. Li, N.-N. Zheng, F.-Y. Wang, "Joint task difficulties estimation and testees ranking for intelligence evaluation," *IEEE Transactions on Computational Social Systems*, vol. 6, no. 2, pp. 221-226, 2019.