IV. Further Topics: Convex Relaxation

Convex relaxation

The art and science of *convex relaxation* revolves around taking a non-convex problem that you want to solve, and replacing it with a convex problem which you can actually solve – the solution to the convex program gives information about (usually a lower bound) the solution to the original program, although sometimes we can get lucky and still solve the original problem. Usually this is done by either "convexifying" the constraints or convexifying the functional – we will see examples of both below.

The Assignment Problem

Suppose that we have N tasks that we want to assign to M different agents. The payout for agent m performing task n is $C_{m,n}$. We want to assign the agents to maximize the total payout, given by the sum of the payouts associated with each assignment. We have the constraint that no agent can be assigned to more than one task and no task can be performed by more than one agent. This is a classic optimization problem in operations research, and we will see below that despite its combinatorial nature, it can be solved exactly using linear programming.

Assume to start that $M \geq N$ and that $C_{m,n} \geq 0$, so we will want to assign every task to an agent, but some agents may not get a task. (We can drop these assumptions, we make them here merely for convenience.) If we arrange the payouts $C_{m,n}$ as entries in an $M \times N$ matrix $\boldsymbol{C}$, then the goal is to choose N entries in this matrix that maximize the sum under the constraint that there is exactly one entry chosen per column and at most one entry chosen per row.

Example. On the next page is a chart whose rows are indexed by NFL teams and columns indexed by week of the season. The number in entry (m, n) is the probability that team m wins in week n multiplied by a weight (indicated in parentheses next to the team — the worse the team was last year, the higher the weight). You choose one team to win every week and you can only choose each team once; if they win you increase your score by the number in the box, otherwise you get nothing. If you have to pick all 18 weeks at once, what is the optimal selection sequence?

The answer is given below with the optimal choices in gray boxes ...

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	ATL (25)	8.93	4.19	8.56	12.75	3.98	7.89	5.08	12.75	7.4	9.26	12.77	7.09	10.4	0	5.93	5.34	9.82	6.32
2	BAL (19)	11.9	11.57	8.91	8.67	10.56	11.07	14.06	6.46	8.69	0	14.6	11.44	11.4	9.95	8.7	14.94	12.54	7.82
3	CAR (27)	15.39	10.57	11.24	12.24	10.15	5.51	8.16	13.23	6.64	17	6.26	11.07	0	11.71	12.84	15.27	5.26	7.65
4	CHI (26)	9.75	5.18	16.62	10.19	7.14	12.42	7.54	6.3	10.99	14.7	12.72	12.12	7.94	0	11.27	7.36	11.01	10.47
5	CLE (20)	8.6	11.9	9.52	9.8	7.71	8.51	5.2	7.37	0	6.3	3.64	8.4	12.94	7.69	10.84	11.56	9.85	9.89
6	DEN (24)	13.75	18.61	12.82	11.33	13.39	9.48	18.21	16.38	0	11.23	14.81	14.16	9.6	11.24	14.71	8.2	7.89	13.03
7	DET (31)	12.49	14.95	8.63	17.62	9.18	0	8.47	13.28	9.63	13.48	12.36	8.88	18.69	12.68	14.66	13.47	17.87	6.28
8	HOU (30)	9.89	6.74	10.82	9.6	11.51	0	7.69	10.55	10.18	8.86	12.44	6.25	10.59	5.95	7.69	6.93	15.93	6.38
9	IND (17)	11.39	10.73	8.25	10.13	7.52	12.85	7.12	11.16	7.84	8.23	9.87	11.15	6.82	0	8.11	9.62	9.77	13.39
10	JAX (32)	10.1	11.79	7.29	7.99	19.72	7.81	16.49	10.17	13.96	5.9	0	12.72	12.71	8.45	11.08	14.13	15	12.53
11	LAC (16)	10.31	5.65	12.36	10.88	9.83	9.68	11.78	0	11.26	6.09	8.07	8.02	8.09	10	9.88	6.94	8.2	7.31
12	MIA (18)	10.48	7.04	7.68	6.75	11.09	10.12	11.38	10.29	10.39	12.33	0	14.25	6.49	6.75	5.17	8.21	7.72	13.45
13	MIN (21)	9.78	8.8	15.16	10.79	15.23	9.19	0	13.73	10.49	6.36	11.25	12.54	15.94	12.41	10.98	14.69	6.93	12.55
14	NO (15)	9.64	6.74	8.76	7.29	10.61	7.86	6.81	9.18	8.14	6.76	7.2	5.49	4.7	0	11.44	6.33	6.07	10.75
15	NYG (28)	8.73	17.04	11.33	17.03	7.73	11.69	13.57	12.16	0	19.73	16.84	7.47	14.31	12.04	9.48	8.42	11.9	8.4
16	NYJ (29)	10.84	11.74	9.76	8.81	11.13	5.02	6.99	11.14	7.28	0	7.57	15.49	6.99	4.61	15.28	16.19	10.49	7.33
17	SEA (23)	9.83	5.93	15.13	9.92	6.72	10.78	6.07	13.01	7.45	5.96	0	11.67	5.1	13.02	8.77	4.8	14.68	7.88
18	WSH (22)	15.06	11.39	10.77	6.98	11.21	11.49	8.69	7.56	11.01	7.76	12.88	15.76	10.76	0	14.55	6.8	11.16	10.18

Each possible selection is associated with a matrix $\boldsymbol{P}$ that has exactly one 1 per column. The total payout associated with a matrix $\boldsymbol{P}$ is

$$\langle \boldsymbol{C}, \boldsymbol{P} \rangle = \sum_{m=1}^{M} \sum_{n=1}^{N} P_{m,n} C_{m,n}.$$

The problem of assigning each of the N tasks to one of the M agents can now be recast as searching for the matrix that maxmizes the linear form above:

$$\underset{\boldsymbol{P}}{\text{maximize}} \ \langle \boldsymbol{P}, \boldsymbol{C} \rangle \quad \text{subject to} \quad \boldsymbol{P} \in \Pi, \tag{1}$$

where Π is the set of all $M \times N$ matrices with exactly one 1 per column. This is a finite set, but it is very large; solving the program above by direct search requires testing $M!/(M-N)!$ different permutations. Note that if $M = 32$ and $N = 18$ this is about 3 *septillion* ($\approx 3 \cdot 10^{24}$).

Luckily, direct search is unnecessary. There is a natural convex relaxation for (1) and it turns out this relaxation is exact. To see how this works, suppose for the moment that $M = N$, in which case Π becomes the set of matrices with exactly one 1 in every column *and* every row, i.e., the set of all *permutation matrices*. In this case, the proposal is to replace a search over all permutation matrices with a search over the larger set of *doubly stochastic* matrices whose entries can be arbitrary numbers in $[0, 1]$ but with the constraint that all the row and column sums are equal to 1[1]:

$$\mathcal{B} = \left\{ \boldsymbol{A} : A_{m,n} \geq 0, \ \boldsymbol{A}\mathbf{1} = \mathbf{1}, \ \mathbf{1}^{\mathrm{T}} \boldsymbol{A} = \mathbf{1}^{\mathrm{T}} \right\}. \tag{2}$$

[1] $\boldsymbol{A}\mathbf{1} = \mathbf{1}$ means all the rows sum to one and $\mathbf{1}^{\mathrm{T}}\boldsymbol{A} = \mathbf{1}^{\mathrm{T}}$ means all the columns sum to one.

The relaxation of (1) is

$$\underset{\boldsymbol{P}}{\text{maximize}} \ \langle \boldsymbol{P}, \boldsymbol{C} \rangle \quad \text{subject to} \quad \boldsymbol{P} \in \mathcal{B}. \tag{3}$$

Since the set $\mathcal{B}$ is described by a set of linear inequalities, this is a linear program in N^2 variables. It can be solved in a fraction of a second for $N = 18$ or $N = 32$ with any one of a number of freely available off-the-shelf solvers.

It remains to show that solving (3) is the same as solving (1). As $\Pi \subset \mathcal{B}$, it might be that solutions to (3) are matrices that do not correspond to permutations (i.e. contain entries that are not 0 or 1). But it turns out that no matter what $\boldsymbol{C}$ is, the solution to (3) is always in Π.

The set $\mathcal{B}$ is what is known as the **Birkhoff polytope**. It is a classical result from convex analysis (usually attributed to Birkhoff [Bir46] and von Neumann [vN53]) that every doubly stochastic matrix (every matrix in the set $\mathcal{B}$) can be written as a convex combination of permutation matrices. That is,

$$\mathcal{B} = \text{conv}(\Pi) = \left\{ \boldsymbol{M} : \boldsymbol{M} = \sum_{k=1}^{K} \lambda_k \boldsymbol{P}_k, \ \boldsymbol{P}_k \in \Pi, \ \lambda_k \geq 0, \ \sum_{k=1}^{K} \lambda_k = 1 \right\}.$$

While it is obvious that every convex combination of permutation matrices is doubly stochastic ($\text{conv}(\Pi) \subset \mathcal{B}$), the converse, that every permutation matrix can be written as such a convex combination ($\mathcal{B} \subset \text{conv}(\Pi)$) is non-trivial; we will not prove it here, but refer the reader to the references above. It is also a fact from linear programming, which we review in the next section below, that a linear program over a polytope generated from a finite number of vertices must have a solution set that includes one of these vertices. Thus we know that (3) is solved by a matrix $\boldsymbol{P} \in \Pi$.

The solution to (3) can be non-unique; this will happen only when the solution to (1) is non-unique. In this case, the solution set might contain matrices not in Π, but there will always be at least one member of Π in the set. In practice, it is easy to move from any member of the solution set to a $\boldsymbol{P} \in \Pi$ that is also in the set.

Finally, the discussion above relied on the condition $M = N$. For $M > N$, we can simply add $M - N$ columns of the return matrix $\boldsymbol{C}$ to make it square. If we fill these columns with entries that are less than the minimum payout for any agent in any task (0 if the $C_{m,n}$ are non-negative, for example), then we can solve (3) with the augmented return matrix and take the first N columns. Similar tricks can be applied to $M < N$ by adding dummy agents.

Linear programming on polytopes

In this section, we establish the fact that a linear program over a polytope always has one of the vertices as a solution. We start by considering the special case of linear programming over the unit simplex:

$$\underset{\boldsymbol{x} \in \mathbb{R}^N}{\text{maximize}} \ \boldsymbol{c}^{\mathrm{T}} \boldsymbol{x} \quad \text{subject to} \quad \sum_{n=1}^{N} x_n = 1, \quad \boldsymbol{x} \geq \boldsymbol{0}. \tag{4}$$

It is self-evident that a solution to this program is

$$x_n^\star = \begin{cases} 1, & n = n_{\max} \\ 0, & \text{otherwise}, \end{cases}$$

where

$$n_{\max} = \arg \max_n c_n.$$

The solution to (4) need not be unique; if $\boldsymbol{c}$ is maximized in more than one place then $\boldsymbol{x}^\star$ can spread its mass in an arbitrary way between these places and still solve (4).

Now consider maximizing a linear form over a polytope $\mathcal{P}$

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{maximize}} \ \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} \quad \text{subject to} \quad \boldsymbol{x} \in \mathcal{P}, \tag{5}$$

where $\mathcal{P}$ is the convex hull of a finite number of vertices $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_K$,

$$\mathcal{P} = \left\{ \boldsymbol{x} : \boldsymbol{x} = \lambda_1 \boldsymbol{v}_1 + \cdots + \lambda_K \boldsymbol{v}_K, \ \lambda_k \geq 0, \ \sum_{k=1}^K \lambda_k = 1 \right\}.$$

In this case, one of these vertices will *always* be a maximizer of (5). To see this, we note that solving (5) is the same as solving

$$\underset{\lambda_1,\ldots,\lambda_K}{\text{maximize}} \ \boldsymbol{c}^{\mathrm{T}} \left(\sum_{k=1}^K \lambda_k \boldsymbol{v}_k \right) \quad \text{subject to} \ \ \lambda_k \geq 0, \ \sum_{k=1}^K \lambda_k = 1,$$

and then taking $\boldsymbol{x}^\star = \sum_{k=1}^K \lambda_k^\star \boldsymbol{v}_k$. We can write this program more compactly as

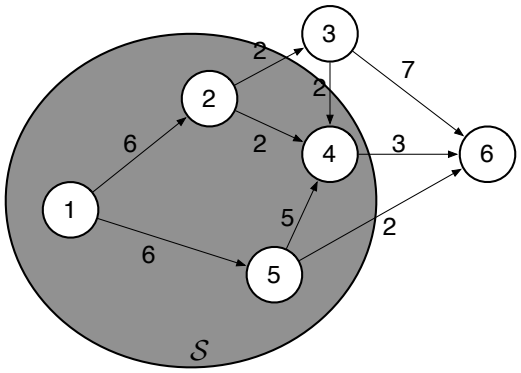
$$\underset{\boldsymbol{\lambda}\in\mathbb{R}^K}{\text{maximize}} \ \boldsymbol{d}^{\mathrm{T}}\boldsymbol{\lambda} \quad \text{subject to} \quad \sum_{k=1}^K \lambda_k = 1, \quad \boldsymbol{\lambda} \geq 0,$$

where the kth entry of $\boldsymbol{d}$ is given by $d_k = \boldsymbol{c}^{\mathrm{T}}\boldsymbol{v}_k$. This has the same form as (4), and so the set of optimal solutions always includes $\boldsymbol{\lambda}^\star$ that is one in one location and zero in all others — the corresponding $\boldsymbol{x}^\star$ will then simply be the corresponding $\boldsymbol{v}_k$.

Minimum-Cut

Suppose that we have a directed graph with vertices indexed by $1, \ldots, N$. By convention we will denote vertex 1 as the "source" and vertex N as the "sink". Between each pair of vertices (i, j) there is a value $C_{i,j} \geq 0$ — if there is no edge from i to j, we take $C_{i,j} = 0$. A **cut** partitions the vertices into two sets: a $\mathcal{S}$ which contains the source, and a set $\mathcal{S}^c$ which contains the sink. The value $\text{cut}(\mathcal{S})$ of the cut is the sum of the $C_{i,j}$ on all the edges that originate in $\mathcal{S}$ and terminate in $\mathcal{S}^C$.

In example below, we have $N = 6$ and $\mathcal{S} = \{1, 2, 4, 5\}$:



We have $\text{cut}(\mathcal{S}) = 2 + 3 + 2 = 7$.

In general, the value associated with a cut $\mathcal{S}$ is

$$\text{cut}(\mathcal{S}) = \sum_{i \in \mathcal{S}, j \notin \mathcal{S}} C_{i,j}.$$

If we take the vector $\boldsymbol{\nu} \in \mathbb{R}^N$ as

$$\nu_i = \begin{cases} 1, & i \in \mathcal{S}, \\ 0, & i \notin \mathcal{S} \end{cases}$$

then we can write the problem of finding the *minimum* cut as

$$\begin{aligned} \underset{\boldsymbol{\nu}}{\text{minimize}} \quad & \sum_{i=1}^{N}\sum_{j=1}^{N} C_{i,j}\max(\nu_i-\nu_j,0) \\ \text{subject to} \quad & \nu_i \in \{0,1\},\ i=1,\dots,N, \\ & \nu_1 = 1, \quad \nu_N = 0. \end{aligned}$$

To make the objective function simpler, we introduce $\lambda_{i,j}$, and the minimum cut program can be rewritten as

$$\begin{aligned} \text{(MINCUT)} \quad \underset{\boldsymbol{\Lambda},\boldsymbol{\nu}}{\text{minimize}} \quad & \sum_{i=1}^{N}\sum_{j=1}^{N} \lambda_{i,j}C_{i,j} \\ \text{subject to} \quad & \lambda_{i,j} = \max(\nu_i-\nu_j,0),\ i,j=1,\dots,N, \\ & \nu_i \in \{0,1\},\ i=1,\dots,N, \\ & \nu_1 = 1, \quad \nu_N = 0. \end{aligned}$$

We will not do so here, but one reason why the minimum-cut problem is of interest is that its dual is the *maximum-flow* problem (i.e., given a directed graph as above what is the largest flow possible from the source to the sink if we associate the $C_{i,j}$ as maximum capacities).

As it is stated, there are two things making this program nonconvex – we have non-affine equality constraints relating $\lambda_{i,j}$ to ν_i and ν_j, and we have binary constraints on ν_i. If we simply drop the integer constraint, and relax

$$\lambda_{i,j} = \max(\nu_i-\nu_i,0) \quad \text{to} \quad \lambda_{i,j} \geq \nu_i - \nu_j \ \text{ and } \ \lambda_{i,j} \geq 0,$$

we are left with the linear program

$$\begin{aligned} \text{(LP-R)} \qquad \underset{\boldsymbol{\Lambda},\boldsymbol{\nu}}{\text{minimize}} \quad & \langle \boldsymbol{\Lambda}, \boldsymbol{C} \rangle \\ \text{subject to} \quad & \lambda_{i,j} \geq \nu_i - \nu_j, \quad \lambda_{i,j} \geq 0,\ i,j=1,\dots,N, \\ & \nu_1 = 1,\ \nu_N = 0. \end{aligned}$$

Note that the domain we are optimizing over in the LP relaxation is larger than the domain in the original formulation – this means that every valid cut (feasible $\boldsymbol{\Lambda}, \boldsymbol{\nu}$ for the original program) is feasible in the LP relaxation. So at the very least we know that

$$\text{LP-R}^\star \;\leq\; \text{MINCUT}^\star.$$

But the semi-amazing thing is that the solutions to the two programs turn out to agree.

We show this by establishing that for every solution of the relaxation, there is at least one cut with value less than or equal to LP-R*. We do this by generating a *random cut* (with the associated probabilities carefully chosen) and show that in expectation, it is less than LP-R*.

Let Z be a uniform random variable on $[0, 1]$. Let $\boldsymbol{\Lambda}^\star, \boldsymbol{\nu}^\star$ be solutions to (LP-R). Create a cut $\mathcal{S}$ with the rule:

$$\text{if } \nu_n^\star > Z, \text{ then take } n \in \mathcal{S}.$$

The probability that a particular edge $i \to j$ is in this cut is

$$\begin{aligned}
\mathrm{P}\left(i \in \mathcal{S}, j \notin \mathcal{S}\right) &= \mathrm{P}\left(\nu_j^\star \leq Z \leq \nu_i^\star\right) \\
&\leq \max(\nu_i^\star - \nu_j^\star, 0) \\
&\leq \lambda_{i,j}^\star,
\end{aligned}$$

where the last inequality follows simply from the constraints in (LP-R). This cut is random, so its value is a random variable, and its expectation is

$$\begin{aligned}
\mathrm{E}[\mathrm{cut}(\mathcal{S})] &= \sum_{i,j} C_{i,j}\, \mathrm{P}\left(i \in \mathcal{S}, j \notin \mathcal{S}\right) \\
&\leq \sum_{i,j} C_{i,j} \lambda_{i,j}^\star \\
&= \text{LP-R}^\star.
\end{aligned}$$

Thus there must be a cut whose valueis at most LP-R*. This establishes that

$$\text{MINCUT}^\star \leq \text{LP-R}^\star.$$

Of course, combining this with the result above means than

$$\text{MINCUT}^\star = \text{LP-R}^\star.$$

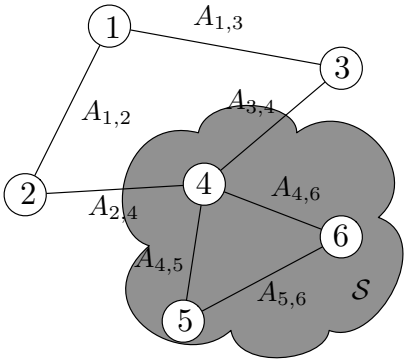
This is an example of a wonderful situation where convex relaxation costs us nothing, but makes solving the program computationally tractable.

Maximum-cut

A good resource for this section and the next is Ben-Tal and Nemirovski [BTN01].

This problem has a very similar setup as the minimum-cut problem, but it is different in subtle ways. We are given a graph; this time the edges are undirected, and have positive weights $A_{i,j}$ associated with them. Since the graph is undirected, $A_{i,j} = A_{j,i}$ and so $\boldsymbol{A}$ is symmetric. We will also assume that $A_{i,i} = 0$ for all i.

As before, a cut partitions the vertices into two sets, $\mathcal{S}$ and $\mathcal{S}^c$ – these sets can be arbitrary; there is no notion of source and sink here. For example, the cut in this example:



has value $\text{cut}(\mathcal{S}) = A_{2,4} + A_{3,4}$. The problem is to find the cut that **maximizes** the weights of the edges going between the two partitions.

We can specify a cut of the graph with a binary valued vector $\boldsymbol{x}$ of length N, where each $x_n \in \{-1, 1\}$. We set $x_n = 1$ if vertex n is in

$\mathcal{S}$ and $x_n = -1$ if vertex n is in $\mathcal{S}^c$. The value of the cut is

$$\text{cut}(\mathcal{S}) = \frac{1}{4}\sum_{i=1}^{N}\sum_{j=1}^{N} A_{i,j}(1 - x_i x_j).$$

Note that if $x_i \neq x_j$, then $(1 - x_i x_j) = 2$, while if $x_i = x_j$, then $(1 - x_i x_j) = 0$. The factor of $1/4$ in front comes from the fact that $(1 - x_i x_j) = 2$ for edges in the cut, and that we are counting every edge twice (from i to j and again from j to i). Notice that we can write this value as a quadratic function of $\boldsymbol{x}$:

$$\text{cut}(\mathcal{S}) = \frac{1}{4}\left(\mathbf{1}^{\mathrm{T}}\boldsymbol{A}\mathbf{1} - \boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x}\right)$$

The maximum-cut problem is find the cut with the largest value:

$$\begin{aligned} \text{(MAXCUT)} \quad & \underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{maximize}} \quad \frac{1}{4}\mathbf{1}^{\mathrm{T}}\boldsymbol{A}\mathbf{1} - \frac{1}{4}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x} \\ & \text{subject to} \quad x_i \in \{-1, 1\}. \end{aligned}$$

Right now, this looks pretty gnarly, as $\boldsymbol{A}$ has no guarantee of being PSD, and we have integer constraints on $\boldsymbol{x}$. We can address the first concern by re-writing this as a search for a matrix $\boldsymbol{X} = \boldsymbol{x}\boldsymbol{x}^{\mathrm{T}}$. As now[2] $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x} = \langle \boldsymbol{X}, \boldsymbol{A}\rangle$, we have

$$\begin{aligned} \underset{\boldsymbol{X}\in\mathbb{R}^{N\times N}}{\text{maximize}} \quad & \frac{1}{4}\mathbf{1}^{\mathrm{T}}\boldsymbol{A}\mathbf{1} - \frac{1}{4}\langle \boldsymbol{X}, \boldsymbol{A}\rangle \\ \text{subject to} \quad & \boldsymbol{X} \succeq \mathbf{0} \\ & X_{i,i} = 1, \quad i = 1, \ldots, N \\ & \text{rank}(\boldsymbol{X}) = 1. \end{aligned}$$

[2]Recall that the standard inner product between two $M \times N$ matrices is $\langle \boldsymbol{X}, \boldsymbol{A}\rangle = \sum_{m,n} A_{m,n}X_{m,n} = \text{trace}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{X})$. This is exactly the same as "flattening out" the matrices as vectors and using the standard Euclidean inner product.

You should be able to convince yourself that $\boldsymbol{X}$ is feasible above if and only if it can be written as $\boldsymbol{X} = \boldsymbol{x}\boldsymbol{x}^{\mathrm{T}}$, where the entries of $\boldsymbol{x}$ are ± 1.

The recast program looks like a semidefinite program (SDP), except for the rank constraint. The relaxation, then, is to simply drop it and solve

$$\begin{aligned} \text{(MAXCUT-R)} \quad & \underset{\boldsymbol{X} \in \mathbb{R}^{N \times N}}{\text{maximize}} \ \frac{1}{4}\mathbf{1}^{\mathrm{T}}\boldsymbol{A}\mathbf{1} - \frac{1}{4}\langle \boldsymbol{X}, \boldsymbol{A} \rangle \\ & \text{subject to} \quad \boldsymbol{X} \succeq \mathbf{0} \\ & \qquad\qquad\ \ X_{i,i} = 1, \ i = 1, \ldots, N. \end{aligned}$$

As we are optimizing over a larger set, the optimal value of MAXCUT-R will in general be larger than MAXCUT:

$$\text{MAXCUT-R}^{\star} \ \geq \ \text{MAXCUT}^{\star}.$$

But there is a classic result [GW95] that shows it will not be too much larger:

$$\text{MAXCUT}^{\star} \ \geq \ (0.87856) \cdot \text{MAXCUT-R}^{\star}.$$

The argument again relies on looking at the expected value of a random cut. Let $\boldsymbol{X}^{\star}$ be a solution to MAXCUT-R. Since $\boldsymbol{X}^{*}$ is PSD, it can be factored as

$$\boldsymbol{X}^{\star} = \boldsymbol{V}^{\mathrm{T}}\boldsymbol{V}.$$

With $\boldsymbol{v}_j$ as the j^{th} column of $\boldsymbol{V}$, this means $X^{\star}_{i,j} = \langle \boldsymbol{v}_i, \boldsymbol{v}_j \rangle$. Since along the diagonal we have $X^{\star}_{i,i} = 1$, this means that $\|\boldsymbol{v}_i\|_2 = 1$ as well. We can associate one column $\boldsymbol{v}_i$ with each vertex in the original

problem. To create the cut, we draw a vector $\boldsymbol{z}$ from the unit-sphere (so $\|\boldsymbol{z}\|_2 = 1$) uniformly at random,[3] and set

$$\mathcal{S} = \{i \ : \ \langle \boldsymbol{v}_i, \boldsymbol{z} \rangle \geq 0\}.$$

It should be clear that the probability that any fixed vertex is in $\mathcal{S}$ is $1/2$. But what is the probability that vertex i and vertex j are on different sides? The probability of this is simply the ratio of the angle between $\boldsymbol{v}_i$ and $\boldsymbol{v}_j$ to π:

$$\mathrm{P}\left(i \in \mathcal{S}, j \notin \mathcal{S}\right) + \mathrm{P}\left(i \notin \mathcal{S}, j \in \mathcal{S}\right) = \frac{\arccos\langle \boldsymbol{v}_i, \boldsymbol{v}_j \rangle}{\pi} = \frac{\arccos X^\star_{i,j}}{\pi}.$$

Thus the expectation of the cut value is

$$\begin{aligned} \mathrm{E}[\mathrm{cut}(\mathcal{S})] &= \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} A_{i,j}\left(\mathrm{P}\left(i \in \mathcal{S}, j \notin \mathcal{S}\right) + \mathrm{P}\left(i \notin \mathcal{S}, j \in \mathcal{S}\right)\right) \\ &= \sum_{i=1}^{N}\sum_{j=1}^{N} A_{i,j}\frac{\arccos X^\star_{i,j}}{2\pi}. \end{aligned}$$

There must be at least one cut that has a value greater than or equal to the mean, so we know that

$$\mathrm{MAXCUT} \ \geq \ \mathrm{E}[\mathrm{cut}(\mathcal{S})].$$

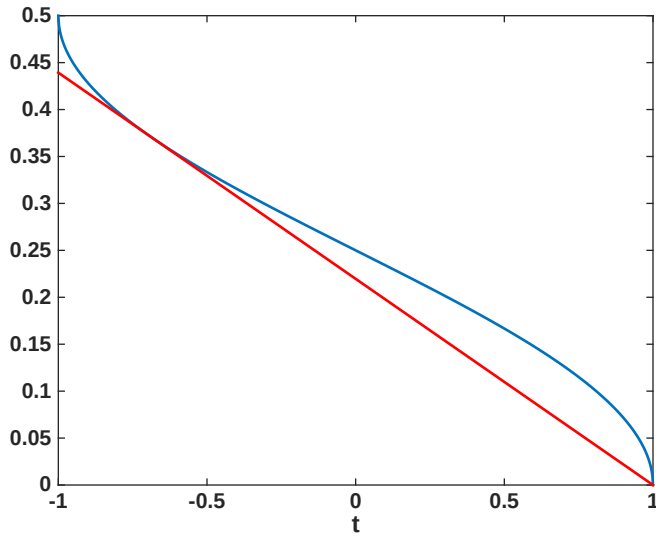
Let's compare the terms in this sum to those in the objection function for MAXCUT-R. We know that the entries in $\boldsymbol{X}^\star$ have at most unit magnitude[4] $-1 \leq X^\star_{i,j} \leq 1$, and it is a fact that:

$$\frac{\arccos t}{2\pi} \ \geq \ (0.87856)\frac{1}{4}(1-t), \quad \text{for} \ \ t \in [-1, 1].$$

[3] In practice, you could do this by drawing each entry Normal$(0, 1)$ independently, then normalizing.

[4] This follows from $X^\star_{i,j} = \langle \boldsymbol{v}_i, \boldsymbol{v}_k \rangle$, $\|\boldsymbol{v}_i\|_2 = 1$, and Cauchy-Swartz.

Here is a little "proof by plot" of this fact:



$$\text{blue} = \frac{\arccos t}{2\pi}, \text{ red} = (0.878856)\tfrac{1}{4}(1-t).$$

Thus

$$\begin{aligned}
\text{MAXCUT}^\star &\geq \text{E}[\text{cut}(\mathcal{S})] \\
&= \sum_{i=1}^{N}\sum_{j=1}^{N} A_{i,j}\frac{\arccos X^\star_{i,j}}{2\pi} \\
&\geq (0.87856)\sum_{i=1}^{N}\sum_{j=1}^{N}\frac{1}{4}A_{i,j}(1-X^\star_{i,j}) \\
&= (0.87856)\cdot \text{MAXCUT-R}^\star
\end{aligned}$$

Quadratic equality constraints

The integer constraint $x_i \in \{-1, 1\}$ in the example above might also be interpreted as a quadratic *equality* constraint:

$$x_i \in \{-1, 1\} \quad \Leftrightarrow \quad x_i^2 = 1.$$

As we are well aware, quadratic (or any other nonlinear) equality constraints make the feasibility region nonconvex.

We consider general nonconvex quadratic programs of the form

$$\begin{aligned} \underset{\boldsymbol{x} \in \mathbb{R}^N}{\text{minimize}} \;\; & \boldsymbol{x}^{\mathrm{T}} \boldsymbol{A}_0 \boldsymbol{x} + 2\langle \boldsymbol{x}, \boldsymbol{b}_0 \rangle + c_0 \\ \text{subject to} \;\; & \boldsymbol{x}^{\mathrm{T}} \boldsymbol{A}_m \boldsymbol{x} + 2\langle \boldsymbol{x}, \boldsymbol{b}_m \rangle + c_m = 0, \quad m = 1, \ldots, M, \end{aligned}$$

where the $\boldsymbol{A}_m$ are symmetric, but not necessarily $\succeq \boldsymbol{0}$. We will show how to recast these problems as optimization over the SDP cone with an additional (nonconvex) rank constraint. Then we will have a natural convex relaxation by dropping the rank constraint. This general methodology works for equality or (possibly nonconvex) inequality constraints, but for the sake of simplicity, we will just look at equality constraints.

We can turn a quadratic form into a trace inner product with a rank 1 matrix as follows. It is clear that

$$\begin{aligned} \boldsymbol{x}^{\mathrm{T}} \boldsymbol{A} \boldsymbol{x} + 2\boldsymbol{b}^{\mathrm{T}} \boldsymbol{x} + c &= \begin{bmatrix} \boldsymbol{x}^{\mathrm{T}} & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{A} & \boldsymbol{b} \\ \boldsymbol{b}^{\mathrm{T}} & c \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ 1 \end{bmatrix} \\ &= \operatorname{trace}\left(\begin{bmatrix} \boldsymbol{A} & \boldsymbol{b} \\ \boldsymbol{b}^{\mathrm{T}} & c \end{bmatrix} \boldsymbol{X}_x \right), \quad \boldsymbol{X}_x = \begin{bmatrix} \boldsymbol{x} \\ 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{x}^{\mathrm{T}} & 1 \end{bmatrix}. \end{aligned}$$

This means we can write the nonconvex quadratic program as

$$\begin{aligned} \underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}} \quad & \operatorname{trace}\left(\begin{bmatrix} \boldsymbol{A}_0 & \boldsymbol{b}_0 \\ \boldsymbol{b}_0^{\mathrm{T}} & c_0 \end{bmatrix} \boldsymbol{X}_x\right) \\ \text{subject to} \quad & \operatorname{trace}\left(\begin{bmatrix} \boldsymbol{A}_m & \boldsymbol{b}_m \\ \boldsymbol{b}_m^{\mathrm{T}} & c_m \end{bmatrix} \boldsymbol{X}_x\right) = 0, \quad m = 1, \ldots, M. \end{aligned}$$

With

$$\boldsymbol{F}_m = \begin{bmatrix} \boldsymbol{A}_m & \boldsymbol{b}_m \\ \boldsymbol{b}_m^{\mathrm{T}} & c_m \end{bmatrix},$$

we see that this program is equivalent to

$$\begin{aligned} \underset{\boldsymbol{X}\in\mathbb{R}^{N\times N}}{\text{minimize}} \quad & \langle \boldsymbol{X}, \boldsymbol{F}_0 \rangle \\ \text{subject to} \quad & \langle \boldsymbol{X}, \boldsymbol{F}_m \rangle = 0, \quad m = 1, \ldots, M \\ & \boldsymbol{X} \succeq \boldsymbol{0} \\ & \operatorname{rank}(\boldsymbol{X}) = 1. \end{aligned}$$

Again, we can get a convex relaxation simply by dropping the rank constraint. How well this works depends on the particulars of the problem. There are certain situations where it is exact; one of these is when there is a single non-convex inequality constraint.There are other situations where it is not exact but is provably good – one example is maximum-cut. There are other situations where it is arbitrarily bad.

Example: Phase retrieval

In coherent imaging applications, a not uncommon problem is to reconstruct an unknown vector $\boldsymbol{x}$ from measurements of the *magnitude* of a series of linear functionals. We observe

$$y_m = |\langle \boldsymbol{x}, \boldsymbol{a}_m \rangle|^2 \ \ (+ \text{ noise}), \quad m = 1, \ldots, M.$$

For instance, if $\boldsymbol{a}_m$ are Fourier vectors, we are observing samples of the magnitude of the Fourier transform of $\boldsymbol{x}$. If we also measured the phase, then recovering $\boldsymbol{x}$ is a standard linear inverse problem (and if we have a complete set of samples in the Fourier domain, you can just take an inverse Fourier transform). But since we do not get to see the phase, we have to estimate it along with the underlying $\boldsymbol{x}$ – this problem is often referred to as **phase retrieval**.

We can rewrite the measurements as

$$\begin{aligned} y_m = \langle \boldsymbol{a}_m, \boldsymbol{x} \rangle \langle \boldsymbol{x}, \boldsymbol{a}_m \rangle = \boldsymbol{x}^{\mathrm{H}} \boldsymbol{a}_m \boldsymbol{a}_m^{\mathrm{H}} \boldsymbol{x} &= \operatorname{trace}(\boldsymbol{a}_m \boldsymbol{a}_m^{\mathrm{H}} \boldsymbol{x} \boldsymbol{x}^{\mathrm{H}}) \\ &= \langle \boldsymbol{X}, \boldsymbol{A}_m \rangle_F, \end{aligned}$$

where $\boldsymbol{A}_m = \boldsymbol{a}_m \boldsymbol{a}_m^{\mathrm{H}}$ and $\boldsymbol{X} = \boldsymbol{x}\boldsymbol{x}^{\mathrm{H}}$. So solving the phase retrieval problem is the same as finding an $N \times N$ matrix with the following properties:

$$\langle \boldsymbol{X}, \boldsymbol{A}_m \rangle_F = y_m, \quad m = 1, \ldots, M, \quad \boldsymbol{X} \succeq \boldsymbol{0}, \quad \operatorname{rank}(\boldsymbol{X}) = 1.$$

The first condition is just that $\boldsymbol{X}$ obeys a certain set of linear equality constraints; the second is that $\boldsymbol{X}$ is in the SDP cone; the third is a nonconvex constraint.

One convex relaxation for this problem simply drops the rank constraint and finds a feasible point that obeys the first two conditions. Under certain conditions on the $\boldsymbol{a}_m$, there will only be one point in this intersection once M is mildly larger than N.

Sparse solutions to linear systems of equations

Many, many problems in scientific computing and data science amount to solving a linear system of equations. That is, given an $M \times N$ matrix $\boldsymbol{A}$ and a response vector $\boldsymbol{y} \in \mathbb{R}^M$, we want to find $\boldsymbol{x} \in \mathbb{R}^N$ such that

$$\boldsymbol{A}\boldsymbol{x} \approx \boldsymbol{y}.$$

The applications here are too numerous to list, but we will settle for kernel regression in machine learning and statistics, medical imaging, seismic exploration, radar, and channel estimation in wireless communications.

As we have seen continuously throughout this course, the starting point for this problem is the least-squares optimization program

$$\underset{\boldsymbol{x} \in \mathbb{R}^N}{\text{minimize}} \ \|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|_2^2.$$

In some sense, this will return the $\hat{\boldsymbol{x}}$ that come closest to explaining $\boldsymbol{y}$.

There can, however, be many solutions to the least-squares program above. This is certainly true when $\boldsymbol{A}$ is under determined, that is it has fewer rows than it has columns. We now need to choose between these solutions.

One criteria we can use is to choose the $\boldsymbol{x}$ that has the smallest number of non-zero terms, i.e. is the "sparsest". This has appeal across many of the applications listed above. For example, in regression we might want to choose the smallest number of features that explain the response. In imaging, we can often express a target as a sparse superposition of pre-determined basis functions (indeed, this is the main idea behind every image or video compression algorithm).

We might modify the least-square problem by introducing a sparsity constraint

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}} \ \|\boldsymbol{y}-\boldsymbol{A}\boldsymbol{x}\|_2^2 \quad \text{subject to} \quad \text{nnz}(\boldsymbol{x}) \leq S,$$

where $\text{nnz}(\cdot)$ returns the number of non-zero terms, or by introducing a penalty (or regularizer) which is basically putting the above in Lagrange form,

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}} \ \|\boldsymbol{y}-\boldsymbol{A}\boldsymbol{x}\|_2^2 + \lambda \cdot \text{nnz}(\boldsymbol{x}),$$

It should be clear that $\text{nnz}(\cdot)$ is not a convex function, and indeed both of the programs above have been shown to be NP-hard [Nat95].

There is, however, a natural convex relaxation to the above. In place of $\text{nnz}(\cdot)$, we use the ℓ_1 norm. We replace the above with

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}} \ \|\boldsymbol{y}-\boldsymbol{A}\boldsymbol{x}\|_2^2 + \lambda\|\boldsymbol{x}\|_1.$$

The above is called the 'LASSO' [Tib96] in statistics and 'basis pursuit' [CDS99] in signal processing. Roughly speaking, this relaxation works since signals that are sparse (have a small number of nonzero terms) have small ℓ_1 norm relative to their Euclidean norm.

There is a rich theory for using ℓ_1 norm minimization for solving systems of equations. One typical result is that for "generic" $M \times N$ matrices $\boldsymbol{A}$ and an observation $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}^\star$, then the solution to

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}} \ \|\boldsymbol{x}\|_1 \quad \text{subject to} \quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{y},$$

will be exactly $\boldsymbol{x}^\star$ when M is on the order of non-zero terms, even when $M \ll N$ [CRT06a, CRT06b, Don06]. Hence we can invert under determined systems even when the solutions are indeed sparse.

Throughout this course, we have seen many algorithms for solving these types of problems.

Low rank matrix recovery

Consider the following fundamental, easily stated problem. Suppose there is a $M \times N$ matrix

$$\boldsymbol{X} = \begin{bmatrix} X_{1,1} & X_{1,2} & X_{1,3} & X_{1,4} & X_{1,5} \\ X_{2,1} & X_{2,2} & X_{2,3} & X_{2,4} & X_{2,5} \\ X_{3,1} & X_{3,2} & X_{3,3} & X_{3,4} & X_{3,5} \\ X_{4,1} & X_{4,2} & X_{4,3} & X_{4,4} & X_{4,5} \\ X_{5,1} & X_{5,2} & X_{5,3} & X_{5,4} & X_{5,5} \end{bmatrix}$$

whose entries we only partially observe,

$$\boldsymbol{X} = \begin{bmatrix} X_{1,1} & - & X_{1,3} & - & X_{1,5} \\ - & X_{2,2} & - & X_{2,4} & - \\ - & X_{3,2} & X_{3,3} & - & - \\ X_{4,1} & - & - & X_{4,4} & X_{4,5} \\ - & - & - & X_{5,4} & X_{5,5} \end{bmatrix}.$$

Is it possible to "fill in the blanks"?

Of course, in general the answer is "no". But what if the matrix is structured in that it has rank $R \ll \min(M, N)$? Then the answer (under some conditions on the underlying matrix) is "yes". Revealing just a few entries per row/column makes the problem identifiable: there is only one low rank matrix that can have exactly those entries. This matrix can be recovered (the entries "filled in") by solving

$$\underset{\boldsymbol{X}}{\text{minimize}} \ \text{rank}(\boldsymbol{X}) \quad \text{subject to} \quad X_{m,n} = Y_{m,n}, \quad (m,n) \in \mathcal{I},$$

where $\mathcal{I}$ is the set of observed indices, and $Y_{m,n}$ are their observed values.

This is again an NP-hard problem. But also again there is a natural convex relaxation (first made popular in [Faz02]), we replace rank with the *nuclear norm*:

$$\underset{\boldsymbol{X}}{\text{minimize}} \ \|\boldsymbol{X}\|_{\text{nn}} \quad \text{subject to} \quad X_{m,n} = Y_{m,n}, \quad (m,n) \in \mathcal{I}.$$

The nuclear norm $\|\boldsymbol{X}\|_{\text{nn}}$ is the sum of the singular values of $\boldsymbol{X}$. This is actually the dual norm of the standard matrix operator norm (maximum singular value). Note the parallels to the vector case: we relax the number of non-zero singular values (the rank) to the sum, just as we relaxed the number of non-zero entries in a vector to the sum of the absolute values. Also, just as the ℓ_1 norm is dual to the ℓ_∞ norm, the nuclear norm (sum of singular values) is dual to the operator norm (maximum of singular values).

There is also a rich theory for recovering low-rank matrices from partial observations; seminal works include [RFP10, CR09], and a review can be found in [DR16].

References

[Bir46] G. Birkhoff. Three observations on linear algebra. *Univ. Nac. Tacumán Rev. Ser. A*, 5:147–151, 1946.

[BTN01] A. Ben-Tal and A. Nemirovski. *Lectures on Modern Covex Optimization*. SIAM, 2001.

[CDS99] S. S. Chen, D. L. Donoho, and M. A. Saunders. Atomic decomposition by basis pursuit. *SIAM J. Sci. Comput.*, 20(1):33–61, 1999.

[CR09] E. Candès and B. Recht. Exact matrix completion via con-

vex optimization. *Found. of Comput. Math.*, 9(6):717–772, 2009.

[CRT06a] E. Candès, J. Romberg, and T. Tao. Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information. *IEEE Trans. Inform. Theory*, 52(2):489–509, February 2006.

[CRT06b] E. Candès, J. Romberg, and T. Tao. Stable signal recovery from incomplete and inaccurate measurements. *Comm. on Pure and Applied Math.*, 59(8):1207–1223, 2006.

[Don06] D. L. Donoho. Compressed sensing. *IEEE Trans. Inform. Theory*, 52(4):1289–1306, April 2006.

[DR16] M. A. Davenport and J. Romberg. An overview of low-rank matrix recovery from incomplete observations. *IEEE J. Selected Topics in Sig. Proc.*, 10(4):608–622, 2016.

[Faz02] M. Fazel. *Matrix rank minimization with applications*. PhD thesis, Stanford University, March 2002.

[GW95] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. Assoc. Comp. Mach.*, 42(6):1115–1145, November 1995.

[Mir75] L. Mirsky. A trace inequality of John von Neumann. *Monatshefte für Mathematik*, 79:303–306, 1975.

[Nat95] B. K. Natarajan. Sparse approximate solutions to linear systems. *SIAM J. Comput.*, 24(2):227–34, 1995.

[RFP10] B. Recht, M. Fazel, and P. A. Parrilo. Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization. *SIAM Review*, 52(3):471–501, 2010.

[Tib96] R. Tibshirani. Regression shrinkage and selection via the Lasso. *J. Royal Stat. Soc. Ser. B*, 58(1):267–288, 1996.

[vN53] J. von Neumann. A certain zero-sum two-person game equivalent to an optimal assignment problem. *Ann. Math. Studies*, 28:5–12, 1953.