CSCI-4968: ML and Optimization Spring 2023

Semidefinite Programming

For approximating Max Cut and Travelling Salesman

1 Motivation

1.1 Max Cut

We would like to partition our graph into two set of vertices such that the sum of the edge weights between the two partitiions is maximized.

Applications include examples: Clustering - Intuition: The distances between the two partitiions are the greatest so they have a high likelihood of belonging to two different categories.

1.2 Travelling Salesman Problem

Imagine you are prospective student touring RPI. You have a list of buildings you want to visit. You have on hand the distance between each pair of buildings. Is it possible to find a path such that each buidling is visited exactly once? If so, what is the shortest possible distance you will need to travel. Since you toruing the college, it is likely you have some method of transportation to came from and need to come back to . We have the added constraint that you start at the parking lot and will need to come back to the parking lot. This is know as the travelling salesman. As our goal is to find an *efficient* route, we will be satisfied with an approximation. Or a series of buidlings to visit that is close the optimal.

2 Semidefinite Programming

2.1 Recap of Linear Programming

To give background on semi-definite programming, we begin with a brief recap of linear programming. Suppose that you have control over a set of variables and you are attempting to find a selection for each of these variables such that some linear combination is either maximized or minimized.

Formally we can write this out as the following. Suppose that we have n variables that we have control over which we can represent as:

$$\vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

This problem is made trivial as we can set $\vec{x}$ to be arbitrarily large or small value depending on the optimization direction. The solution becomes more meaningful if each variable has a constraint that must be met.

Say for each of these n variables we have an associated coefficient $\vec{c} = \begin{bmatrix} c_1, c_2, ..., c_n \end{bmatrix}^\top$. We would like to find $\vec{x}$ such that the linear combinatrion of the two $(\vec{c} \cdot \vec{x})$ is minimized

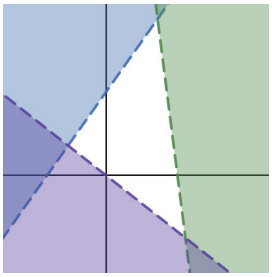
$$\min \quad \vec{c} \cdot \vec{x}$$

We can extend this idea to a system of equations. We define $\vec{a}_i$ and $\vec{b}$ to vectors in $\mathbb{R}^n$. We would like to

find $\vec{x}$ such that

$$\begin{aligned}\vec{a}_1 \cdot \vec{x} &= b_1\\ \vec{a}_2 \cdot \vec{x} &= b_2\\ &...\\ \vec{a}_n \cdot \vec{x} &= b_n\end{aligned}$$

Graphically, each constraint creates a *half-space*. The intersection of the halfspaces (the white space) is our solution set. Our goal is the find the minimize value of the intersections.



We can gather our $\vec{a}_i$ vectors into a single matrix. Let $\mathbf{A} = [\vec{a_1}^\top \cdots \vec{a_n}^\top]$. Solving the above systems of equations is equivalent to solving:

$$\mathbf{A}\vec{x} = \vec{b}$$

In summary, a linear programming asks for an optimal solution, $\vec{x}$, given a set of linear constraints

$$\begin{aligned}\text{minimize} \quad & \vec{c} \cdot \vec{x}\\ \text{subject to} \quad & \mathbf{A}\vec{x} = \vec{b}\end{aligned}$$

We note that our constraints form a convex set. Thus, linear programming is a subset of a convex optimization problem Finding a solution to an instance of linear programming can be done with a variety of algorithms including *simplex algorithm*.

Many real world problems can be modelled as a set of linear contraints. One use case arises in manufacturing. Given a set of products, how much of each kind should be produced to either minimize cost or maximize profits?

2.2 Formalization of Semidefinite Programming

Instead of having just n variables to pick suppose that we have n^2 variables. Thus instead of finding a $\vec{x}$ we are trying to find $\mathbf{X}$. With more variables comes more constraint so suppose that we n equations to satisfy. For each equation we have a matrix $\mathbf{A}_i$ and a scalar value b_i defining our constraint. Then for a given constraint we have:

$$\mathbf{A}_i \bullet \mathbf{X} = b_i$$

Where the $\bullet$ operator is simply taking the sum of all elements after doing element wise multiplication on matricies two matricies $\mathbf{A}$ and $\mathbf{X}$ with the same shape. Formally we can write it as:

$$\mathbf{A} \bullet \mathbf{X} = \sum_{i=1}^{n}\sum_{j=1}^{n} \mathbf{A}_{i,j}\mathbf{X}_{i,j}$$

An easy way of

In total we can write out a rigorous form of Semdefinite programming.

$$\begin{aligned}\min \mathbf{C} \bullet \mathbf{X}&\\ \mathbf{A}_i \bullet \mathbf{X} &= b_i \text{ for } i = 1, 2, ..., n\\ \mathbf{X} &\succeq 0\end{aligned}$$

3 Relaxation

3.1 Proofs

- performance guarantees - proof of relaxations (how accurate + how it works)
Integrality Gap: ratio btwen optimal / relaxation

4 Experimental Results

4.1 Visualization

- table of result - for small examples, we find the acutal solution (brute force), or a solution (integer programming) and check how good our bound is

vhttps://www.cs.cmu.edu/ anupamg/adv-approx/lecture14.pdf used "randomized rounding" to find a max cut. i.e. if p_{ij} is close to -1, then we should cut it. We attempt do do something similar

5 Discussion

- can we prove the runtime - can we prove how good of a lower bound we have? - what are some easy things to prove?

5.1 Scope and limitations

6 References