1 Introduction

- how SDP encapsulates Linear programming - use cases for SDP

Our goal is to ...., using Semidefinite programming. ... we find that results ....

-

1.1 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 vist each building exactly one? (can't not teleport). If so, what is the shortest possible distance you will need to travel. We also have the added constraint that you start at the parking lot and will need to come back to the parking lot.

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.

1.2 Max Cut

2 Semidefinite Programming

2.1 Recap of Linear Programming

To give background on semi-definite programming we first start 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 maximized / minimized. To make this more complicated suppose that these variables each have a constraint that must be met.

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:

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

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

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

Without constraints the problem is trivial as we just set $\vec{x}$ to an arbitrarily large / small value depending on the problem. Suppose that we must also satisfy a system of equations. We let $\vec{a}_i$ and $\vec{b}$ be a n length vectors. Then we want our pick of $\vec{x}$ to satisfy:

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

We can gather our $\vec{a}_i$ vectors into a single matrix so that we only need to solve:

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

In total we can define a linear programming problem as the following where we want to find an optimal $\vec{x}$:

$$\min \vec{c} \cdot \vec{x}$$
$$\mathbf{A}\vec{x} = \vec{b}$$

What is nice about linear programming is that it is a convex optimization problem. Finding a solution to an instance of linear programming can be done with a variety of algorithms including *simplex algorithm* **TODO: Double check this**.

This is what linear programming solves and is usable in a large variety of problems such as **TODO: Need to find more examples**.

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)

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