

# 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 visit 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 buildings 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  $\vec{c} = [c_1, c_2, \dots, c_n]^T$  and in the end we want to pick  $\vec{x}$  such that the linear combination 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:

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

We can gather our  $\vec{a}_i$  vectors into a single matrix  $\mathbf{A} = \begin{bmatrix} \dots \vec{a}_1 \dots \\ \dots \vec{a}_2 \dots \\ \vdots \\ \dots \vec{a}_n \dots \end{bmatrix}$  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  $\vec{x}$ :

$$\begin{aligned} \min \vec{c} \cdot \vec{x} \\ \mathbf{A}\vec{x} &= \vec{b} \\ \vec{x} &\geq 0 \end{aligned}$$

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 the simplex algorithm.

Linear programming has a large number of applications as it is a general approach of taking a constrained linear optimization problem into a form where a solution can easily be found.

## 2.2 Formalization of Semidefinite Programming

Instead of having just  $n$  variables that we have a matrix  $\mathbf{X}$  of  $n^2$  elements. Thus instead of finding an optimal  $\vec{x}$  we are trying to find an optimal  $\mathbf{X} \in \mathbb{R}^{n \times n}$  that is symmetric positive semidefinite (SPSD) such that

$$\begin{aligned} \min &= \sum_{i=1}^n \sum_{j=1}^n x_{i,j} c_{i,j} \\ \mathbf{X} &\succeq 0 \end{aligned}$$

Where  $c_{i,j}$  is the coefficient corresponding to the  $x_{i,j}$  variable and  $\mathbf{X} \succeq 0$  ensures that  $\mathbf{X}$  is SPSPD. We can gather all  $n^2$  coefficients into a matrix  $\mathbf{C}$  and then rewrite our minimization term as:

$$\min \mathbf{C} \bullet \mathbf{X}$$

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

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

Of important note is the fact that  $\mathbf{C} \bullet \mathbf{X} = \text{trace}(\mathbf{C}^\top \mathbf{X})$  where trace is the sum of the diagonal terms for a matrix. Calculating the trace of the matrix multiplication is not as efficient as the  $\bullet$  operator but the notation comes up quite frequently within the literature so it is important to know.

Returning back to our formalization of SDP, we note that with more variables comes with more constraints. Suppose that we have  $n$  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:

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

This formally defines SDP. In total we can write out a rigorous form of semidefinite programming.

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

The parallels between semidefinite programming and linear programming should be quite obvious. In many ways, semidefinite programming is linear programming with more variables of control and more constraints.

## 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 actual solution (brute force), or a solution (integer programming) and check how good our bound is

<https://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 to 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