Convex Semi-definite Programs for the Fresco Problem

Arka Adhikari
Princeton University
Advisor:Amit Singer

April 18, 2016

Abstract

Computer Vision is the problem of having computers being able to recognize and categorize objects in an appropriate manner. An important problem in this field is the problem of 2-D puzzle reconstruction; we would need to have the computer be able to tell whether two distinct pieces could fit together and then reconstruct the puzzle given this information. This paper will detail various algorithms that could be used to solve this problem. We outline four possible approaches to this problem. Three of these approaches involves embedding the puzzle as points in R2 and using natural metrics of the plane to quantify errors from measured data to use in our cost function. Our last approach treats the problem as a completely abstract 'game', in the sense of the Unique Games conjecture in Computer Science, and uses an approximation algorithm formulated for the Unique Games Problem to derive a solution. We discuss limitations of these algorithms and how they could be improved to solve real life problems.

Contents

1 Introduction

The field of computer vision has been created due to fundamental differences between the nature of machine and man; namely, machines do not have the natural ability of man to recognize and classify objects. As a consequence, complicated algorithms are needed for computers to be able to solve the problem of recognition and reconstruction.

An important starting point for problems of this type is the issue of puzzle solving, or two-dimensional reconstruction. The computer has to be able to determine the boundary of each of the pieces and, afterwards, be able to determine which of the pieces would best fit next to one another. Afterwards, the computer must be able to use this information gleaned by boundary information to determine how the pieces fit together in a global structure. This specific issue is slightly easier than general computer vision because of the dimension reduction; since the only image a computer can see in one instance is a two -dimensional projection, multiple projections are in general necessary to reconstruct a three dimensional object. The two dimensional analogue is given with a canonical projection that gives understanding of the entire nature of the problem. Nevertheless, few algorithms are known to efficiently solve the problem.

We will deal with the latter half of the problem of two dimensional reconstruction. We assume as given orientation information given between any pair of two pieces believed to be connected to each other and ,from this, we try to determine the global connection between pieces and their relative orientations. The issues with this problem come from the fact that the data given contains many 'bad' edges, e.g. pairs of pieces that are believed to be connected to each

other, but really do not have such a connection. These types of errors are not readily recognized by convex cost models and simple models do not readily solve this problem.

Older approaches to this problem, an example of which is illustrated in the paper of Singer and Lo, try to solve this problem by trying to explicitly determine which of the edges given are outliers by geometric considerations. The idea pursued in the latter paper involved creating a probability distribution on the set of edges, the probability distribution serving as a measure of whether the given edge was indeed valid. Updates are made to the probability model based on whether small cycles were found involving the edge. The motivation behind this idea based on the fact that if two pieces are adjacent, then they would also have multiple small-hop neighbors in common. These common neighbors would lead to multiple small cycles involving a correct edge. However, we would not expect that two pieces that by mere chance could fit together locally could fit together in a global structure by having multiple common neighbors.

The only issue with the latter method involves the fact that it becomes difficult to remove all outliers in the data. Thus, we would still need a solver that could still function in the presence of wrong edges. To this end, we try to use the geometric consideration given as motivation in the earlier paper; that two pieces that are only locally adjacent by chance could not in general fit together due to the interference of correct edges, which establish geometric constraints that would prohibit the placement and use of bad edges. We try to create cost functions and convex programs that would incorporate such information into their cost function.

Our first methods involve embedding the pieces in R^2 by considering the pieces as a set of points in R^2. This involves a significant data reduction that could still be used to reconstruct the solution. As a measure of error, we use natural metrics inherited from R^2 to establish measures on how much our data deviates from the measured data. These natural metrics can readily be established into convex functions and can be efficiently solved by classical convex solvers; as such, these algorithms are very fast.

Our final method involves changing the paradigm entirely and treat the problem of puzzle solving in much the same way as a human would solve it. We reduce the problem to the instance of the Unique Games problem. Once we have described the problem in terms of the unique games approach, then we use an approximation algorithm inherited from the unique games approach in order to solve the problem.

2 The method of SNL-SDP

2.1 Background on SNL-SDP

The problem that SNL-SDP attempts to solve is as follows. We are given a set of n radio towers whose locations we want to find out. The only information that we have to aid our task are noisy inter-pair distances between some of the

radio towers. Mathematically, the problem statement can be described using the following language.

In the plane R^2 , we know there exists n points $[P_i|i \in 1..n]$ and we wish to determine the actual values of the coordinates x_i and y_i of these n points: $[P_i = (x_i, y_i)|i = 1..n]$. The information given to us to solve this question consists of noisy inter-distance information; namely, we are given the set of values

$$[d^*_{(i,j)} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}[1 + \epsilon_{(i,j)}]|(i,j)\ inE] \tag{1}$$

where E is some subset of $[(a,b)|a \in 1..n, b \in 1..n, a < b]$ and the $\epsilon_{(i,j)}$ are independent identically distributed gaussian variables of mean 0 and variance 1(or as appropriate for the measuring apparatus). As a shorthand, we use the symbol d_{ij} to denote the actual distance between points namely $d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$.

The SNL-SDP approach represents this question as the minimization of an appropriate cost function, as is the paradigm of optimization. The simplest type of cost function is the minimization of the sum of squares; indeed, this is the workhorse for optimization, rather akin to the use of gaussian for statistical inference. The square function has the nice properties that it is smooth(and convex) and always positive. These properties make the square function elegant and make it easier to solve for minima using calculus. In some simple instances, it is actually possible to get a closed form solution using least squares. The importance of emphasizing this fact will come into play later in which I will contrast least squares with other types of error models.

The cost function we first 'try' to implement is as follows:

$$\min_{(x'_i, y'_i)} \Sigma_{(i,j)\in E}|(d^*_{ij})^2 - (x'_i - x'_j)^2 + (y'_i - y'_j)^2|^2 \tag{2}$$

One can also take instead of the second power of the difference from d^*_{ij} , the first power of the difference with d^*_{ij}. Unfortunately, neither power results in an expression that is convex in the variables x_i. We need a slight relaxation in order to make this a convex expression.

The first trick to notice is that expressions of the form $(x'_i - x'_j)^2 + (y'_i - y'_j)^2$ can be expressed in terms of the matrix components of an appropriate semi-definite matrix. First define X to be the n by 2 matrix with rows $[x_i, y_i]$; the matrix $Y = XX^T$ will have coordinates $[Y]_{ij} = x_i x_j + y_i y_j$. Thus, we can express $(x'_i - x'_j)^2 + (y'_i - y'_j)^2$ in terms of the matrix Y as $Y_{ii} + Y_{jj} - 2Y_{ij}$.

Thus, an equivalent representation of our earlier cost function is

$$\begin{aligned} \min_{X \in R^{nx2}} \quad & \Sigma_{(i,j)\in E}|(d^*_{ij})^2 - Y_{ii} - Y_{jj} + 2Y_{ij}|^2 \\ \text{subject to } & qquad Y = XX^T \end{aligned}$$

The restriction that Y is of rank 2, which comes as a result of our prescription of Y as coming from the product of a rank 2 matrix with itself, is not a convex

constraint. It is now natural to see that the convex relaxation of the above problem is to let Y vary over the class of all positive semi-definite matrices.

We now arrive at the manifestly convex problem

$$\begin{aligned} \min_{Y \in R^{nxn}} \quad & \Sigma_{(i,j)\in E} |(d_{ij}^*)^2 - Y_{ii} - Y_{jj} + 2Y_{ij}|^2 \\ \text{subject to} \quad & Y \succeq 0 \end{aligned}$$

In order to 'retrieve' the n by 2 matrix we have to undergo a procedure of rounding. By standard linear algebra, one can write the expression Y as $(UD)(UD)^T$, where U is an n by n orthogonal matrix and D is a diagonal matrix(one can assume that the eigenvalues of D are ordered$\lambda_n > ... > \lambda_1 \geq 0$). The two-dimensional projection we use will then be the first two columns of the matrix UD.

It can be shown that over all matrices X in R^{nx2} that UD is 'closest' to Y in the following sense: $(UD)(UD)^T$ is closest to Y in the following norm $\|A\| = \max_{\|v\|=1, v\in R^{1xn}} v^t A v$

This concludes our discussion on the theoretical basis of the SNL-SDP algorithm

2.2 Formulation of the Fresco Problem as an Instance of SNL-SDP

As many of our later approaches are minor variations of the cost function used in SNL-SDP, many of the discussions that we undergo here will be useful in our further considerations.

The pieces of our Fresco have a certain correspondence to the radio towers that we try to localize in the SNL-SDP method. Geometrically, one knows that we can localize the entire piece given the location of three points on the piece that are not linear. Thus, for our problem of localization of pieces, we consider each fresco piece as the union of three points whose coordinates we wish to determine using the SNL-SDP algorithm.

The information that is given to us includes information on which of the pieces are connected to each other and the relative orientation of pieces that are adjacent to each other. As a result, one is able to find noisy distance information about relative distances between the points on the pieces that are considered to be adjacent.

However, the error model slightly differs from the error model given in the previous section. In the case of the Fresco, we are not completely aware of which pieces are supposed to fit together. Thus, in some instances, we get information that is completely wrong; e.g. we say two pieces are supposed to fit together even when they are not. In this case, the error model for the estimated distances d_{ij}^* will be completely random data rather than the Gaussian Error we proposed earlier.

The error model above results in great amounts of error; namely, it is possible to get data that proposes that two pieces are close together even when they are

far apart. In two dimensions, this data will be inconsistent. However, in higher dimensions, it is possible for a solution to exist in higher dimensions; afterward, the rounding procedure amounts in an incorrect solution.

Pure SNL-SDP is not robust to this type of errors, so our goal is to add appropriate constraints to SNL-SDP that are robust to the error models that we are proposing. One obvious geometric constraint is that the pieces should not like on top of each other. This results in a spreading constraint on the problem; namely, for any two points on distinct pieces namely t_1 on piece p_1 and point t_2 on piece p_2, there should exist a lower bound on the distances between the points p_1.

With these modifications above, we are now going to construct the mathematical formulation of our problem.

We are given N pieces $P_1 ... P_N$ along with information about which of these pieces are adjacent to each other along with information about how to orient each of the adjacent pieces so that they fit together. After going through the following transformation, we can turn this into numeric data.

For each of the pieces P_i we choose 3 points $p_{i,1}, p_{i,2}, p_{i,3}$. These 3 pieces are not adjacent to each other. Additionally, from the geometry of the piece, we can determine for each point $p_{i,j}$ the smallest distance from the point to the boundary of the piece containing it. We call this smallest distance from $p_{i,j}$ to the boundary $r_{i,j}$.

From the adjacency and rotation information, we shall be able to determine appropriate distance information. Let E be the set of pairs of all 'adjacent pieces'. For each $(i,j) \in E$ we can get distance information $d_{(i,k),(j,l)}$ for all $k \in 1,2,3$ and $l \in (1,2,3)$. We then impose the standard SNL-SDP method with the additional constraints that the distance between point $p_{i,j}$ and $p_{k,l}$ is greater than $r_{i,j} + r_{k,l}$ if $k \neq l$ and equality constraints for the distances between points on the same piece.

Namely, we get the following optimization problem

$$\begin{aligned} \min_{Y \in R^{nxn}} \quad & \Sigma_{(i,j)\in E} |(d^*_{ij})^2 - Y_{ii} - Y_{jj} + 2Y_{ij}|^2 \\ \text{subject to} \quad & Y \succeq 0 \\ & Y_{(i,k)(i,k)} - Y_{(j,l)(j,l)} - 2Y_{(i,k)(j,l)} \geq (r_{(i,k)} + r_{(j,l)})^2 \end{aligned}$$

2.3 Testing data set formulation

The general model for data set testing will be the same in most of the following sections. We will mention necessary differences as appropriate.

A puzzle of n 'pieces' come from the Voronoi cell decomposition from choosing n random points from the $[0,1]x[0,1]$. Construction of the Voronoi cells and adjacency matrix information comes directly from Matlab functionality. We choose the three points from each piece in the manner outlined.

For each piece, we try to find the circle of maximum radius that can be inscribed in that circle. This comes as the result of solving the following opti-

mization question

$$\begin{aligned} \max_{(x,y)} \quad & r \\ \text{subject to} \quad & \frac{|(a_i^{p_j})^2 x + b_i^{p_j} y + c_i^{p_j}|}{\sqrt{(a_i^{p_j})^2 + (b_i^{p_j})^2}} \geq r, \forall i \\ & a_i^{p_j} x + b_i^{p_j} y + c_i^{p_j} (\geq or \leq) 0 \text{ as appropriate} \end{aligned}$$

where $a_i^{p_j} x + b_i^{p_j} y + c_i^{p_j}$ are the equations of the boundary lines for the piece p_j.

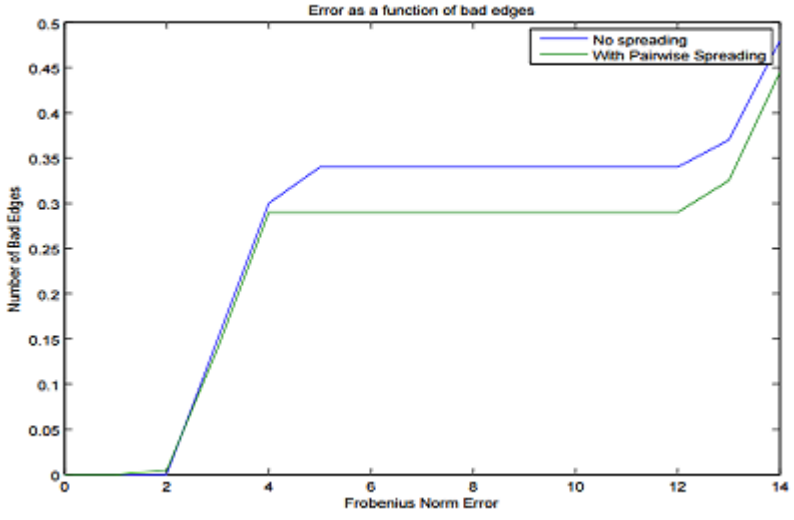
One of the points we use for piece p_j will be the solution to the above optimization question: $P_{(j,1)} = (x_i^S, y_i^S)$. The associated radius $r_{(j,1)}$ will be the solution to the above equation.

The other two points we choose are random on the piece. We set the corresponding radius information to be 0(We hope that enforcing spreading between the center of each piece would be sufficient).

Our error model is as follows: we first pick a number of m to be the total number of bad edges to introduce. Afterwards, we randomly select m edges from the set of edges that we did not choose to include in our set. On this set of m edges we introduce random distance information between the points lying on this data.

Our testing involves modifying m to determine how robust the SNL-SDP algorithm is to data.

2.4 Discussion of Results



The above graph depicts the difference in the performance of algorithm between the case in which we add no pairwise distance spreading constraints and the case in which we include a pairwise distance spreading constraint when we try to solve a 20 piece puzzle and progressively add bad edges as a measure of the error. What can be noticed is that though the inclusion of the pairwise spreading constraint does result in less error overall, the two curves have the same shape.

In particular, what is very negative is the fact that the error rate jumps up at the inclusion of even 2 bad edges. This amount of error shows that there is very little robustness even with the introduction of the pairwise spreading constraints. We explain this behavior as follows.

When we looked at the spectral gap between the second and the third eigenvalues, we noticed that the gap was very small. Namely, this indicates that we are dealing with a naturally high dimensional solution; we are dealing with a folded puzzle. What this would indicate is that when we apply the projection to lower dimensions, the pieces would still lie on top of each other.

Ultimately, we see that just applying pairwise spreading constraint by itself is not sufficient to guarantee that we obtain a low dimensional solution. However, there are other papers that themselves have attempted to apply a spreading approach to try to obtain low dimensionality and they have been successful. What they did was try to create pairwise spreading constraints that involve more than once piece. We believe that this may be able to ensure that we get a 2-D solution.

However, the difficult geometry of the pieces rather makes it difficult to provide very tight constants. One should note that finding such tight constants are difficult even if one were dealing with circular pieces of varying radius(which we used to create the pairwise spreading constraints)

3 SNL-SDP with L1 loss using the Manhattan Norm

3.1 Background

We believe that the errors of the previous section were due to the fact that the l^2 cost function was too susceptible to the presence of outliers. We believed that the reason for this was due to the fact that the solutions that were given by our SNL-SDP solver were inherently high dimensional. Thus, the projection to 2 dimensions resulted in a solution that gave overlaps of pieces.

This time, we attempt to create a manner of cost function that is more robust to error using an L1 type of error. Namely, instead of trying to measure the deviance from the square of the estimated distance d_{ij}^2, we want to create an error model that measures the deviation from d_{ij}. However, this requires significant modification in our cost function, as we can not represent the first power of the distances as a convex function of the terms of the semi-definite

matrix.(Manifestly, one can se this from the fact that the square root function is not a convex function)

Our first relaxation tries to approximate $\sqrt{x^2+y^2}$ with the manhattan norm $|x|+|y|$. Now we are dealing with the convex absolute value function; however, the manhattan norm is not rotationally invariant, which causes some problems with our formulation.

The best relationship we can derive completely is the fact that $\sqrt{2}[|x|+|y|] > \sqrt{x^2+y^2} > |x|+|y|$. However, for two points $pi = (x_i, y_i)$ and $p_j = (x_j, y_j)$ and distance d_{ij} from each other, it is unclear what value we want $|x_i - x_j| + |y_i - y_j|$ to be. Specify the wrong value and the problem may not be solvable exactly.

Given our earlier inequality, we suggest trying to minimize a cost function of the form $\Sigma_{ij}|\sqrt{2}d_{ij} - |x_i - x_j| - |y_i - y_j||$. Compared to choosing d_{ij} as a maximum, this does not promote putting pieces on a single axis; however, our choice is rather arbitrary.

Our full cost function will be

$$\min \Sigma_{(i,j)\in E,k,l}|\sqrt{2}d_{(i,k)(j,l)} - |x_{(i,k)} - x_{(j,l)}| - |y_{(i,1)} - y_{(j,l)}|| \quad (3)$$

$$\text{subject to} r_{(i,k)} + r_{(j,l)} < |x_{(i,k)} - x_{(j,l)}| + |y_{(i,1)} - y_{(j,l)}| \forall i, j, k, l, i \neq j \quad (4)$$

$$d_{(i,k),(i,l)} < |x_{(i,k)} - x_{(j,l)}| + |y_{(i,k)} - y_{(j,l)}| < \sqrt{2}d_{(i,k),(i,l)} \quad (5)$$

To make this into a convex program, we use the standard relaxation for absolute values(e.g. splitting into positive and negative parts).

Namely, we replace each instance of $|x_{(i,k)} - x_{(j,l)}|$ with $\lambda^+_{ikjl} + \lambda^-_{ikjl}$ and add the further conditions that $\lambda^+_{ikjl} - \lambda^-_{ikjl} = |x_i - y_j|$.(And similar variables for y of course, say ν)

3.2 Results and Discussion

The results of this solver were always trivial. Namely, $x_{(i,k)}$ and $y_{(j,l)}$ were always set to 0 in our problem. By a further mathematical analysis, we were able to deduce that this would be the case in all instances. The mathematical analysis will be as below.

Observe as follows, if we set $\lambda^+_{ijkl} = \lambda^-_{ijkl}$ and similarly for ν , we can then make $x_{(i,k)}$ and $y_{(j,l)}$ equal to 0 for all pairs of points. Even with the above restriction, it is still very easy to satisfy the constraints and minimize the cost function as we can set the sum $\sqrt{2}d_{ij,kl} = \lambda^+_{ijkl} + \lambda^-_{ijkl} + \nu^+_{ijkl} + \nu^-_{ijkl}$. This system of equations can be solved in a multitude of ways as well due to the fact it is a combination of many equations none of whom share a variable in common. The spreading condition is readily satisfied by ensuring the equality conditions we maintained above; if not, then we are obviously given as data a condition that cannot possibly be achieved by any solution. We could then remove the incorrect condition and proceed on the original data. Thus, we can see that all solutions to this formulation are trivial.

The main issue with this formulation is due to the fact that the slack variables we introduce do not couple very strongly to each other. This lack of relation

between the slack variables allows the solution to collapse to such a useless value. If we were to adopt this manhattan norm scheme, we would need to have a way that does not use such weak coupling in slack variables.

4 L1 norm using Schur Complement

4.1 motivation

Our motivation for the $L1$ norm is to have a cost function that is robust to error. As said in the earlier section, an equality of the form $y = \sqrt{x}$ cannot be part of a convex problem. We decide to relax this constraint by setting $y \leq \sqrt{x}$ instead. This can be written in terms of the positive semi-definite condition:

$$M_{(i,k)(j,l)} = \begin{bmatrix} 1 & y_{(i,k),(j,l)} \\ y_{(i,k),(j,l)} & x_{(i,k),(j,l)} \end{bmatrix}$$

with $M \geq 0$. This is the insight of the paper of Simonetto and Leus

The cost function will simply be

$$\max \Sigma_{(i,j)\in E,k,l} |y_{(i,k),(l,j)} - d_{(i,k),(l,j)}| \text{subject to} M_{(i,k)(j,l)} \geq \tag{6}$$

$$x_{(i,k),(j,l)} = X_{ikik} + X_{jljl} - 2X_{ikjl} \tag{7}$$

$$X \geq 0 \tag{8}$$

$$y_{(i,k),(j,l)} = r_{(i,k)} + r_{(k,l)} \tag{9}$$

4.2 Results and Discussion

When applying the solver to our test models, the answer is trivial in this case as well. We found that the cost function was always 0, which it should not have been, while the obtained matrix X had little to no relevance to the problem, even when given completely correct data.

The reason for this is that the inequality on the y variables gives no control on the matrix X. We can readily set $y_{(i,k),(j,l)}$ to the appropriate value $d_{(i,k),(j,l)}$. Then, to satisfy the inequality constraints created on the matrix X, we may merely just increase the values on the diagonal arbitrarily high as to satisfy all inequality constraints while the elements off the diagonal are all 0. This will result in a matrix which trivially satisfies the property of being positive Semi-Definite and also satisfy all of the inequality constraints that we have established. Additionally, the spreading constraints should be trivially satisfied if the equality of the y variables to the d variables hold. The auxiliary variable we have included in this circumstance was not enough to guarantee good controls on our matrix structure.

5 The Unique Games Approach for 'L^0' error

5.1 Background and Motivation

Given the lack of robustness in the $L62$ methods and our inability to sufficiently relax the l^1 formulations in a manner in which we are actually able to obtain non-trivial results.

We hope that trying to use a form of error will capture the robustness that we desire while being able to have a sufficiently good relaxation so that we we are able to obtain solutions that are legitimately non-trivial.

Here we must introduce the concept of what I would like to call 'L^0' (to represent that we wish to use this error to reduce the errors generated by outliers to the maximum extent possible). As we have mentioned before, in non-rigorous terms, an outlier that is 10 times more than normal will be exacerbated by a factor of 100 if we used the square norm error. However, notice that even taking a single power will still create a large error factor that the solver will unfortunately try to fit.

What would be ideal is if we could completely ignore the factor of 10 that appears in the outlier. A nice idea would be to try to take pth powers where the value of p goes to 0. However, this power notion takes all measures of distances to 1 except if they are the same point. Using the SNL-SDP distance formulation and strictly applying the L^0 distance would be completely useless.

However, it would be nice to relax it and incur an error of 0 if two pieces that we believe to be adjacent are actually adjacent in the solution configuration while we occur an error of 1 if the opposite happens.

The notion of 'adjacency' would be hard to easily capture in the context that we are considering, so let us turn to a toy example where a notion of adjacency can rather be easily defined.

Consider the following toy example which makes the notion of adjacency that we want to capture exceptionally clear.

We have n^2 pieces of equal sized squares that are the pieces of a puzzle that fits on an $n \times n$ grid. Each square piece will fit into one and exactly one piece of the grid. We assume here that each of the pieces have an obvious rotational orientation.

This reduces the problem placement to deciding amongst a discrete set of possible positions, rather than a continuous set of positions. We can thus define a finite set of indicators $p_{ij} = \{1, 0\}$ designating whether piece i is within square j of the grid if the value is 1 and 0 otherwise. In the notation given previously, I assume that we have some integer indexing of the set of pieces and of the set of squares on the grid. Notice that for each i, we have that p_{ij} is 1 for exactly 1 value of j.

The notion of adjacency is rather obvious here; in general, we can describe the square grid as being a graph G with the vertices of the graph being the set of squares while the edges of the graph connect the grids of the square that share a common edge in the grid.

Adjacency will be demonstrated as follows we can say that piece i is adjacent

to piece j if for the position k such that $p_{ik} = 1$ we have that $p_{jl} = 1$ for one of the grid squares l such that in the graph G mentioned in the previous paragraph, the grid squares l and k are connected by and edge. IN mathematical language, given the graph $G = \{(V, E)\}$, the pieces i and j are connected if $\exists k, l \text{s.t.} p_{ik} = p_{jl} = 1 \text{and}(k, l) \in E$.

In this toy example, we can construct a cost function that will try to put the desired pieces adjacent to each other. Let G' be the graph of pieces where the vertices V' is the set of the pieces and E', the edge set, contains pairs of pieces that we believe should be adjacent to each other. The cost function that we try to minimize is as follows.

$$\min \Sigma_{(i,j)\in E',(k,l)\in E} |p_{ik} - p_{jl}| \tag{10}$$

$$\text{subject to} p_{ik} = 0, 1 \tag{11}$$

$$\forall i, \exists \text{ exactly one } k \text{s.t.} p_{ik} = 1 \tag{12}$$

$$\forall k, \exists \text{ at most one } i \text{s.t.} p_{ik} = 1 \tag{13}$$

The above is the exemplar of an integer program. The constraints ensure that each piece is assigned to exactly one of the grid squares and that each grid square can contain no more than exactly 1 point. The cost function will be minimized when we the pieces we say are adjacent to each other are actually placed adjacent to each other.

Let us write this in a manner that can be written in a relaxed form. First, one must see that minimizing $\Sigma|p_{ik} - p_{jl}|$ will be equivalent to maximization of $p_{ik}p_{jl}$ when we maintain the integer programming constraints.

Notice that $p_{ik}p_{jl}$ will be the terms of an appropriate gram matrix. Thus, it would now be fruitful to rewrite the constraints in terms of these gram matrix terms.

Namely, we would like to set $p_{ik}p_{il} = 0$ for all i,k, and l. Additionally, we would like $\Sigma p_{ik}^2 = 1$. Combining this constraint with the first constraint would imply that exactly one of the p_{ik} values can be nonzero. The condition on the squares would finally imply that this value is either 1 or -1. Finally $p_{il}p_{jl}$ should be 0 for all i,j, and l to ensure that no square can be covered by two pieces.

Thus, our final problem will be

$$\min \Sigma_{(i,j),(k,l)} p_{ik}p_{jl}C_{(ik,jl)} \tag{14}$$

$$\text{subject to } p_{ik}p_{il} \forall i, k, l \tag{15}$$

$$\Sigma_k p_{ik}^2 \forall i \tag{16}$$

$$p_{il}p_{jl} = 0 \forall i, j, l \tag{17}$$

where $C_{ik,jl}$ is a matrix that is 1 if we have that i and j are pieces that we believe to be adjacent to each other and k and l are positions on the grid that are adjacent to each other.

The relaxation to the set of positive semi-definite matrices will come by replacing $p_{ik}p_{jl}$ as the terms of a matrix elementP_{ikjl} which will be positive

semidefinte and rank 1. The other conditions can easily be seen to correspond to linear sums of some of the terms of the matrix P. Relaxing this equation is only a matter of removing the rank one constraint on the matrix P.

It should be clear that given the matrix P, one can do a standard SVD decomposition to find the vector form $[p_{ik}]$ for each i, one then finds the value of k where the value p_{ik} is the highest and then chooses this value to be the position in which we place the piece i.

5.2 Unique Games Approach

Readers who are very well-informed with the literature on NP-Hardness can be aware that the puzzle solving instance that we have mentioned above is indeed a somewhat veiled modified instance of the Unique Games approach.

In the Unique Games problem, we have a graph G in which we try to color each of the vertices of the graph with one of k colors. The restrictions on the coloring of the graph is as follows, for each of the edges $e = (u, v)$ on the graph we have an associated permutation map $\pi_{(u,v)} : [1, ...k] \rightarrow [1..k]$. The coloring condition is that given u has color c, the color on the vertex v should be $\pi_{(u,v)}c$.

What one can see is that if there is indeed a solution satisfying all of the constraints, then one can easily find a solution by picking color for a random point and then building up the solution from there. However, it is to be noticed that if one only knows that if in the optimal solution only most of the solutions can be satisfied, then it is rather improbable that we can get a configuration that satisfies even close to many.

This difficulty in solving this problem is summarized in the following Conjecture by Subhash Khot called the Unique Games Conjecture

Unique Games Conjecture For any ϵ and δ there exists a large enough k such that any instance of the Unique Games problem (Graph G, permutations π_{uv} number of colors K) with number of colors K greater than this k it is now NP hard to distinguish between the following two possibilities

1. The optimal solution satisfies a fraction $1 - \epsilon$ of the constraints.
2. There is no solution satisfying more than a δ fraction of the constraints.

•

What we have seemingly managed to do in the course of our L^0 relaxation is reduce to attempt to reduce our problem to one that is already known for being difficult to solve. Though this seems like a step backwards, we do have one saving grace in the form of the following theorem

Probabilistically Optimal Algorithms for the Unique Games Problem Given that there is an optimal solution that satisfies a $O(1 - \epsilon)$ fraction of the constraints of a Unique Games Problem, it will be possible to find a solution that satisfies $1 - O(\sqrt{\epsilon log(k)})$ of the constraints of the Unique Games Problem [Makarychev et al.] •

The algorithm given solves our SDP with a much more complicated rounding procedure than we have suggested; indeed, it would probably be very computationally intensive to undergo the complicated selection algorithm suggested by the paper of Makarychev. One issue with our simplified rounding procedure that

it does not take into account trying to satisfy constraints together; the selection procedure of Makarychev ensures that solutions will be applied together.

5.3 Algorithm Simplification by Fourier Transform

Since we have n^2 pieces and n^2 grid squares, our matrix will already have n^4 dimension on each side. This, by itself, sounds like it would make the problem computationally intractable. Indeed, this was the case with some of our initial approaches to the problem. If one were dealing with a general graph rather than a square grid, this is generally a loss that one must accept.

However, the square grid has a regular structure that renders it amenable to dimensionality reduction. Namely, when one applies the discrete fourier transform to the matrix C, then the result is a Block Diagonal Matrix. Since we can implicitly ignore the parts of the matrix that have 0 term, we are only only dealing with a matrix that is of dimension n^3. Here, we have dimension n^2 that comes from each block, while we will have n blocks on the diagonal.

Now, we will describe some of the theory of the discrete Fourier transform, but only to the simpler case of circulant rather than block circulant matrices. The result for circulant matrices can be extended to Block Circulant matrices via the usage of tensor products.

A circulant matrix is of this form

$$M = \begin{bmatrix} C_0 & C_1 & C_2 & ... & C_{n-1} & C_n & \\ C_n & C_0 & & C_1 & ... & C_{n-2} & C_{n-1} \\ ... & ... & ... & ... & ... & ... & \\ C_1 & C_2 & C_3 & ... & ... & C_) & \end{bmatrix}$$

Notice that it will have eigenvalues $\Sigma C_j e^{-2\pi kji}$ and will have eigenvalues $\Sigma e^{-2\pi kji} v_i$ where the v_i are the standard coordinate vectors. Merely by calculating inner product one can easily see that two of these eigenvectors are actually orthogonal to each other. Thus, one can diagonalize the matrix C with some matrix P to $PCP* = D$.

A Small Note in Actually Preparing the Puzzle for the Unique Games Approach

One issue in the unique games approach is that the boundaries of the grid are not exactly the same as the center of the grid. Thus, the unique games model and the dimensionality reduction approach that we have mentioned earlier do not hold exactly.

However, there is a simple mathematical trick for for creating this symmetry, though the geometric implications of the application of the trick would be less than desirable. Instead of thinking of the grid of having an edge, we think of the edges as wrapping around to the other side. What this implies instead is that we are trying to put the pieces of the puzzle onto the torus and solve it there.

Though jigsaw puzzles on the torus have the undesirable property of being rather absurd, they satisfy the nice properties of allowing us to engage in our

Unique Games Approach and associated Dimensionality reduction via regularity of the puzzle. Once a solution on the torus is found by the solver and is sufficiently correct, one can expect a human can determine proper edges of the puzzle for himself.

5.4 The Modification for General Pieces

We do not have the fortune that all of our pieces squares of the same size and are square, so we have to make do in another way. However, the idea of discretization of the puzzle space $[0,1] \times [0,1]$ into a square grid is the proper way to go to establish a natural notion of adjacency as we have done earlier in our toy example.

What we can do for our irregularly shaped pieces is assign each piece to an appropriate grid square based on whether we put some chosen point of the piece into that square. Namely, for each point p_i , we designate some point c_i which we will call the center. The grid location of the center is thence the place we assign the point c_I to.

Another important consideration is to note is that rotational angle of the piece is rather important, as compared to the toy example in which orientation is obvious. To deal with this consideration, what one can do is discretize the space of rotation $[0, 2\pi]$ as finitely as one wants. Then each piece can be designated by the position of its center and its rotation.

Given both of these pieces of variables given two pieces that are supposed to be adjacent to each other say p_1 and p_2 then for any position and associated rotation of p_1 one can find the position and rotations of p_2 such that p_1 and p_2 are placed geometrically next to one another. This creates the permutations π that is desired.

With this type of considerations, one can use any level of discretization that one desires as long as the grid squares of the resulting discretization are not so large so that two pieces are allowed to be found in the same square. Indeed, taking the discretization to infinity, essentially one arrives that the continuous case.

There are two problems with choosing a discretization that is arbitrarily large. Discretizing the grid too much makes the search space extremely large, so the computational time taken is far larger than necessary. The second problem is that we only know distances between pieces up to small noise, so specifying an associated permutation for each edge will become unrealistic if there are too many grid squares.

What one should do is find an appropriate balance of number of grid squares to the size of each piece, which is what we will focus on now.

Our testing model is the same as what was described on previous sections; namely, one constructs the Voronoi diagram to divide the $[0,1]x[0,1]$ square into various pieces. Using the radius finding procedure that has been mentioned in previous sections, the size of the each grid square will then be $\min(r_1...r_n)$. Thus we will deal with a discretization into a $\frac{1}{r_i}$ square.

Notice that by choosing this size, we will be able to ensure that no two pieces can have center in the same square. Additionally, up to a small constant factor(at least less than 3), we cannot make the grid size larger without allowing more than 1 piece having centers in the same square.

5.5 Results and Discussion

One can see that even with the reduction of dimension due to the usage of the block circulant structure of the limit, the time complexity of the algorithm is polynomial of very high power. As such, even in our very small test cases (around 10 pieces), it was taking a prohibitively large time to produce a result for analysis. Running a battery of tests, which would be desirable, was rather impossible in our scenario.

In fact, the only test that we were able to run within a reasonable time frame was the three pieces case. In the three piece case, however, the only solution is that all of the three pieces are adjacent. This is exactly the result of solver, but it is too simple to make a definitive statement as to whether the algorithm is actually successful in being able to derive a solution.

A Convex Program and Positive Semidefinite Matrices

Here we will cover some basic definitions appropriate for convex optimization theory.

A.1 Convex Functions

Convex Functions A convex function f is one which satisfies the inequality $f(\lambda a + (1-\lambda)b) < \lambda f(a) + (1-\lambda)f(b)$. If the function is twice-differentiable, this is equivalent to the second derivative begin non-negative. •

This property allows us the most important theorem for convex functions in optimization: a local minimum is a global minimum.

Proof Assume for contradiction that the point a is a local minimum, but there is a point b is a global minimum strictly less than a. Then we have that $f(\lambda a + (1-\lambda)b) \leq \lambda f(a) + (1-\lambda)f(b) < f(a)$. For λ close to 1, this means that the value of f in some neighborhood around a will have values less than that of $f(a)$ thus, a is not a local minimum of the function. •

In essence, the above property allows for the convergence of methods that use ideas similar to gradient descent(namely the idea of following the direction where the derivative is highest). Since there is only one local minimum, the global minimum, there is no issues in getting stuck in a local minimum that is not optimal.

The second important definition that one must know is the definition of convex set.

Convex Set A convex set satisfies the property that for any two points a, b in the set: the line containing a and b is also within the set.

The above property allows us to also engage in gradient descent without concern. One can imagine that if we could not travel along straight lines, then one may not necessarily be able to travel in the direction of greatest descent. Since we only understand local properties of a function during minimum finding, it would be hard to tell if there exists a global minimum across a gap.

Thus, in the formulation of a convex optimization problem, one needs both a convex function and a convex set. Interior point algorithms use both of these facts to generate a polynomial time solver.

Minimization Property for Convex Functions The local minimum of a convex function is also a global minimum

Pf: Say x_1 is a local minimum and x_2 is a global minimum. By the convex inequality, we have $f(\lambda x_1 + (1-\lambda)x_2) \leq \lambda f(x_1) + (1-\lambda)f(x_2) < f(x_1)$ for λ close to 1. Thus, x_1 cannot be a local minimum. •

A.2 Semi-definite Matrices

Positive Semi-definite Matrices An n by n symmetric matrix ,A, is positive semi-definite if for all vectors v, it is the case that $v^T Av > 0$ for all vectors v.

By standard linear algebra, one knows that symmetric matrices(those that satisfy the property that $A = A^T$) are diagonizable. We will sketch a short proof of this property:

First, notice that if $Av = \lambda v$, then if w is a vector that is orthogonal to v then $0 =< \lambda v, w >=< Av, w >=< v, A^T w >=< v, Aw >$. So A maps the set of vector orthogonal to v to the set of vectors orthogonal to w. Thus, if one is able to find at least 1 eigenvector to A, then one can proceed by induction to find a full orthogonal set of eigenvectors to A. Now, the existence of 1 eigenvector is standard theory(one can find a root of the characteristic polynomial to find a value λ such that $A - \lambda I$ is not invertible, an element in the null space is thus an eigenvector). •

Now, the inequality condition on positive Semi-definite matrices shows that all the eigenvalues are non-negative.

The form of these matrices can comprise both linear and quadratic optimization. The use of SDP in representing sum of squares is clear from the first section of this paper. Also, notice that the diagonizability of positive Semi-Definite Matrices allows us to readily obtain projections via the eigenvalue decomposition. This together makes semi-definite matrices an excellent way to represent distances in any number of dimensions.

The most important part of the set of Positive-Semidefinite matrices, however, is that it is a convex set. Namely $v^T(\frac{A+B}{2})v = \frac{1}{2}[v^T Av + v^T Bv] > 0$. This convexity allows us to travel in the interior of the set for interior point methods.

B Acknowledgments

I would like to thank the help of Graduate Students Yuehaw Khoo and Onur Ozyesil who were very willing to meet with me and help me in much of the implementation issues. I would also like to thank postdoc Justin Solomon for his aid.

C Code Samples at End of Paper

References

[Ye, Biswas et al., 2008] Ye, Biswas et al. (2008) Semidefinite Programming Approaches for Sensor Network Localization with Noisy Distance Measurements *IEEE Transactions on Automation Science and Engineering* (3), 360-371.

[UniqueGamesPaper] oses Charikar, Konstantin Makarychev, Yury Makarychev, SODA 2007, pp. 62-68; Special issue of ACM Transactions on Algorithms, vol. 5, no. 3, article 32, July 2009

[UniqueGamesNotes] onstatnin Marykachev University of Washington Lecute NOtes on Unique Games

[Multireference Alignment] . S. Bandeira, M. Charikar, A. Singer, A. Zhu 'Multireference Alignment using Semidefinite Programming, 5th Innovations in Theoretical Computer Science (2014).

[Maximum Likelihood] istributed Maximum Likelihood Sensor Network Localization Andrea Simonetto, Geert Leu

```
function [ x,Y,RandomPMatrix,AA,C,bb ] = SDPL1cost( X )
%SDPL1cost We solve the fresco problem with an L1 cost function
%   We
subplot(4,1,1)
voronoi(X(:,1),X(:,2));
[RandomPMatrix,Radius ] = RandomPointMatrixRadius( X );

[ DD ] = DistancebtwnPoints( X,RandomPMatrix );
[ ED ] = EqualityConstraints(X,RandomPMatrix );
[m,~]= size(RandomPMatrix);
ColorMatrix=rand(m/3,3);
subplot(4,1,2)

for counter=1:m

plot(RandomPMatrix(counter,1),RandomPMatrix(counter,2),'color',ColorMatri
x(ceil(counter/3),:),'marker','+');
    axis([0 1 0 1])
    hold on
end
hold off
distance= DD+ED;
[blk,At,CC,bb,m,gsize,AA ,C] = SDPLinearData(distance,Radius );
L=[zeros(2*m,2*m+7*gsize);zeros(7*gsize,2*m),eye(7*gsize)];
L2=[eye(2*m),zeros(2*m,7*gsize);zeros(7*gsize,2*m+7*gsize)];
L3=[1,zeros(1,2*m+7*gsize-1)];
L4=zeros(1,2*m+7*gsize);
L4(1,m+1)=1;
M1=[zeros(2*m+7*gsize,2*m+7*gsize);eye(2*m+7*gsize,2*m+7*gsize)];
M2=[zeros(2*m+7*gsize,2*m+7*gsize),eye(2*m+7*gsize,2*m+7*gsize)];
M3=zeros(4*m+14*gsize,4*m+14*gsize);
M3(1:2*m,1:2*m)=eye(2*m,2*m);
B=zeros(2*m+7*gsize,1);
cvx_begin %sdp
variable t
%variable Y(2*m+7*gsize,2*m+7*gsize)
%variable Z(4*m+14*gsize,4*m+14*gsize)
%variable y
variable x(2*m+7*gsize)
minimize(C'*x)
subject to
AA'*x==bb
L*x>=B
%Y*diag(x)>=eye(2*m+7*gsize)
%Z>=zeros(4*m+7*gsize,4*m+7*gsize)
%Z*M1==[Y;eye(2*m+7*gsize,2*m+7*gsize)]
%M2*Z==[Y',eye(2*m+7*gsize,2*m+7*gsize)]

%norm(L2*x,1)-t>=0
L3*x>=t
%L4*x<=-0.0
cvx_end
X=x(1:m,1);
Y=x(m+1:2*m,1);
```

```
XY=[X,Y];

subplot(4,1,3)
for counter = 1:m

plot(X(counter,1),Y(counter,1),'color',ColorMatrix(ceil(counter/3),:),'ma
rker','+');
    axis([-1 1 -1 1])
    hold on
end
hold off

%{
refinemaxit = 1000;
   plotyes  = 0;
   alpha    = [];
   PP       = [];
   printyes = 1;
   if ~exist('OPTIONS'); OPTIONS = []; end

   if isfield(OPTIONS,'refinemaxit'); refinemaxit = OPTIONS.refinemaxit;
end
   if isfield(OPTIONS,'plotyes'); plotyes = OPTIONS.plotyes; end
   if isfield(OPTIONS,'alpha');   alpha   = OPTIONS.alpha; end
   if (plotyes); PP = OPTIONS.PP; BoxScale = OPTIONS.BoxScale; end
   if isfield(OPTIONS,'printyes'); printyes = OPTIONS.printyes; end
   pars.gaptol = 1e-6;
   pars.printlevel = 0;
   if isfield(OPTIONS,'gaptol');     pars.gaptol = OPTIONS.gaptol; end
   if isfield(OPTIONS,'printlevel'); pars.printlevel =
OPTIONS.printlevel; end

   tstart = clock;
   [obj,xx,yy,zz,info] = sqlp(blk,At,CC,bb,pars);
   X=0;Y=0;

  XY=[xx{1}(1:m,1),xx{1}(m+1:2*m,1)];
    display(obj);
    %}
  [XY,~,~] = NewMatchPosition(RandomPMatrix',XY' );
  XY=XY';

  x=XY(1:m,1);
  Y=XY(1:m,2);
  subplot(4,1,4)
for counter=1:m

plot(x(counter,1),Y(counter,1),'color',ColorMatrix(ceil(counter/3),:),'ma
rker','+');
    axis([0 1 0 1])
    hold on
end

end
```

```
function [ output_args ] = SensorNetworkCVX(n )
%SensorNetworkCVX Sensor Network Alg using CVX
%   connected(i,j):Piece i,j connected
%   EC(i,j): Equality constraints between i and j
%   IC(i,j): Inequality constraints between i and j
%   G(i,j,:): Coefficient matrix to multiply Z
%   (G(i,j,:)'*Z*G(i,j,:)==Z[i,i]-2*Z[i,j]+Z[j,j]
%   A(i,j) stores absolute value of difference between our value of
D(i,j)
%   and expected value of D(i,j)
%   D(i,j)= list of expected distances

cvx_begin sdp
variable Z(3*n,3*n)
variable lambdaplus(3*n,3*n)
variable lambdaminus(3*n,3*n)
variable A(3*n,3*n)
minimize sum(sum(A))
subject to


for i=1:n
    for j=1:n
        if( connected(i,j)==1)
            A(i,j)==lambdaplus(i,j)+lambdaminus(i,j);
        end
    end
end
for i=1:n
    for j=1:n
        if(connected(i,j)==1)
            lambdaplus(i,j)-lambdaminus(i,j)== G(i,j,:)'*Z*G(i,j,:)-
D(i,j)*D(i,j);
        end
        if(equality(i,j)==1)
            G(i,j,:)'*Z*G(i,j,:)==EC(i,j);
        end
        if(inequality(i,j)==1)
            G(i,j,:)'*Z*G(i,j,:)>=IC(i,j);
        end
    end
end
Z>=0
cvx_end

end
```

```
function [ l,X ] = UniqueGamesSDP( Z)
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here
[DFTC,n,p] = MakeCirculant(Z);
IMAG=imag(DFTC)

for L1=1:n^2
    CFour(1:2*p,1:2*p,L1)=ExpandComplex(DFTC((L1-1)*p+1:L1*p,(L1-
1)*p+1:L1*p));
end

%Cs=sparse(C);

%n=4;
%p=5;
%C=rand(p*n^2,p*n^2);

%{
D=zeros(p*n^2);
E=zeros(p*n^2);
for i=1:p
    for j=1:n^2

            D((i-1)*n^2+j,(i-1)*n^2+j)=ceil(j/n);
    end
end
for i=1:p
    for j=1:n
        for k=1:n
            E((i-1)*n^2+n*(k-1)+j,(i-1)*n^2+n*(k-1)+j)=j;
        end
    end
end
%}

for k = 0:(n^2-1)
    for j= 0:(n^2-1)
        FTcos(k+1,j+1) = cos(2*pi*k*j/n^2);
        FTsin(k+1,j+1) = sin(2*pi*k*j/n^2);
    end
end
cvx_begin sdp
%variable X(p,p,n^2)
variable Z(2*p,2*p,n^2)
variable DFX(2*p,2*p,n^2)
maximize(trace(sum(Z,3)))
subject to

for k = 1:n^2
       for j = 1:p
            for i = j:p
                DFX(i,j,k) == DFX(i+p,j+p,k);
                DFX(i+p,j,k) == -DFX(i,j+p,k);
            end
```

```
       end
end


for i=1:n^2
    %DFX(:,:,i)==[real(X(:,:,i)),-
imag(X(:,:,i));imag(X(:,:,i)),real(X(:,:,i))];
    DFX(:,:,i)>=0
    Z(1:2*p,1:2*p,i)==CFour(:,:,i)*DFX(:,:,i)
end

DFX(1:p,1:p,1)==ones(p,p);
DFX(1:p,p+1:2*p,1)==zeros(p,p);

for i=2:n^2
    for j=1:p
        DFX(j,j,i)==1/n^2;
        DFX(j,j+p,i)==0
    end
end

for i=1:p
    for j=(i+1):p
        for L=0:(n^2-1)
            if(j==i+1)

                reshape(DFX(i,j,:),[1
n^2])*FTcos(:,L+1)+reshape(DFX(p+i,j,:),[1 n^2])*FTsin(:,L+1)==0;
                reshape(DFX(i,j,:),[1 n^2])*FTsin(:,L+1)-
reshape(DFX(p+i,j,:),[1 n^2])*FTcos(:,L+1)==0;
            else

                reshape(DFX(i,j,:),[1
n^2])*FTcos(:,L+1)+reshape(DFX(p+i,j,:),[1 n^2])*FTsin(:,L+1)>=0;
                reshape(DFX(i,j,:),[1 n^2])*FTsin(:,L+1)-
reshape(DFX(p+i,j,:),[1 n^2])*FTcos(:,L+1)==0;
            end

       end
    end
end
%{
%X==semidefinite(p*n^2);
%{
for i=1:p
    for j=i+1:p

        %if(rand<1/10)
       trace(ones(n^2,n^2)*X((i-1)*n^2+1:i*n^2,(j-1)*n^2+1:j*n^2))==1

        %end
    end
end
```

```
for i= 1:p
    for j=1:n^2
        for b=j+1:n^2
            if(rand<1/10)
            X((i-1)*n^2+j,(i-1)*n^2+b)==0;
            end
        end
        if(rand<1/10)
        X((i-1)*n^2+j,(i-1)*n^2+j)==1/n^2;
        end
    end

end
for i=1:n^2
    v=zeros(1,p*n^2);
    for j=1:p
        v(1,(j-1)*n^2+i)=1;
    end
    if(rand<1/10)
    trace(v'*v*X)<=1;
    end
end


for i=1:p*n^2
    for j=1:p*n^2
        if(rand<1/10)
            X(i,j)>=0;
        end
    end
end

%trace(E*X)==p*n/2;
%trace(D*X)==p*n/2;
%}
%}
cvx_end
X_Orig= zeros(p*n^2,p*n^2)
for k =0:(n^2-1)
    X_Orig((p*k+1):(p*(k+1)),(p*k+1):(p*(k+1)))=DFX(1:p,1:p,k+1)+sqrt(-
1)*DFX((p+1):(2*p),1:p,k+1);
end
V=real((kron(dftmtx(n^2)/sqrt(n^2),eye(p)))'*X_Orig*kron(dftmtx(n^2)/sqrt
(n^2),eye(p)));
[v,d]=eig(V);
l=v(:,1);
end
```