Independent Study Report : Optimization

Yunqiu Guo

Spring 2018,May

1 Linear Programming

1.1 Introduction of linear programming

A *linear program* is a class of optimization problems where we maximize or minimize a linear function of $x \in \mathbb{R}^n$ subject to linear constraints.

Example 1.1.

$$\begin{array}{rrcrcl} \max & x_1 & + & x_2 & & \\ s.t. & x_1 & & & \leq & 2 \\ & x_1 & + & 2x_2 & \leq & 4 \\ & x_1 & , & x_2 & \geq & 0 \end{array}$$

This linear program consists of both inequality and equality constraints, we can transform this LP into standard form linear program by introducing *slack variables*. See the continued Example 1.1 below.

Example 1.2.

$$\begin{array}{rcrcrcrcl} x_1 & & & + & x_3 & & & = & 2 \\ x_1 & + & 2x_2 & & & + & x_4 & = & 4 \\ x_1 & , & x_2 & , & x_3 & , & x_4 & \geq & 0 \end{array}$$

The system we arrive in Example 1.2 is a standard form for linear program,by adding x_3 to the first constraint and then change the inequality to equality constraint. Since the original constraint is less than or equal, to make these two constraints mathematically equal,we need to add one more constraint $x_3 \geq 0$. Similar procedure with the second constraint.

Formly, while the linear constraints could be equality or inequality constraints, all linear programs can be re-written such that only *nonnegativity* constraints and linear *equality* constraints are involved. Linear programs of this form is said to be in a *standard form*:

$$\begin{array}{rrcrcrcrcl} \max / \min & c_1x_1 & + & c_2x_2 & + & \dots & + & c_nx_n & & \\ s.t. & a_{11}x_1 & + & a_{12}x_2 & + & \dots & + & a_{1n}x_n & = & b_1 \\ & a_{21}x_1 & + & a_{22}x_2 & + & \dots & + & a_{2n}x_n & = & b_2 \\ & & & & & \vdots & & & & \\ & a_{m1}x_1 & + & a_{m2}x_2 & + & \dots & + & a_{mn}x_n & = & b_m \\ & x_1, & & x_2, & & \dots, & & x_n & \geq & 0 \end{array} \tag{1}$$

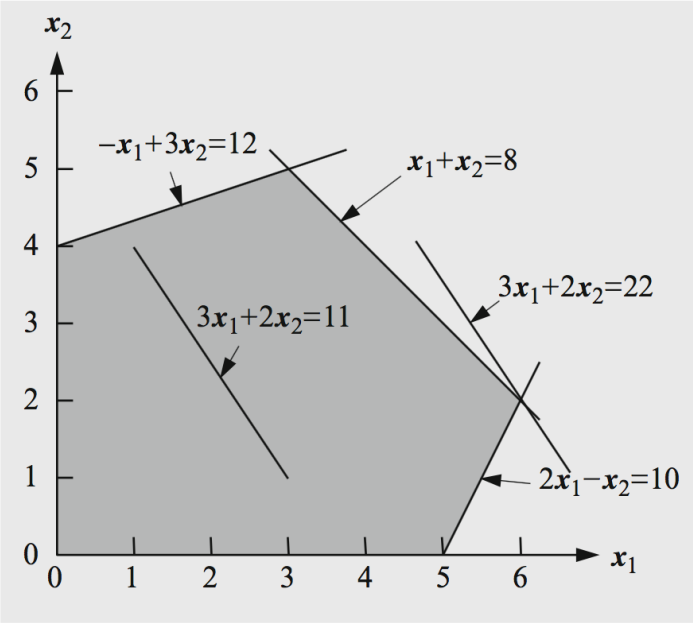


Figure 1: This is the feasible region of Example 1.1,figure from [Van14]

Equivalently, in matrix notation:

$$\begin{array}{rl} \max / \min & c^T x \\ s.t. & Ax = b \\ & x \geq 0. \end{array} \tag{2}$$

1.2 The Geometry of Linear Programs

When the number of variables in the linear program is less than 3, we can graph the set of feasible solutions together with the level sets of the objective function.

To illustrate the geometry representation of a linear program, look at the following classic example. See fig. 1.

Example 1.3.

$$\begin{array}{lccccc} \max & 3x_1 & + & 2x_2 & & \\ s.t. & -x_1 & + & 3x_2 & \leq & 12 \\ & x_1 & + & x_2 & \leq & 8 \\ & 2x_1 & - & x_2 & \leq & 10 \\ & x_1 & , & x_2 & \geq & 0 \end{array}$$

Here, each constraint is a half-plane, we can first treat the constraints as equality and draw the corresponding straight line, and then choose the half plane according to the inequality.

1.3 Simplex Method

1.3.1 Simplex Method Algorithm

Consider the general linear program in a standard equality form:

$$\begin{array}{lrcl} \min & c^T x & & \\ s.t. & Ax & = & b \\ & x & \geq & 0 \end{array}$$

$z = c^T x$ is defined as the *objective value* of the LP.

$$\begin{aligned} z - c^T x &= 0 \\ Ax &= b \\ x &\geq 0 \end{aligned} \qquad (*)$$

(*) is the linear program we are studying for the simplex method.

The sequence of steps to solve (*) is known as the following simplex method.

Step 1: Initialization

(1) Find an initial feasible basis B (later we introduced the phase 1 method)

Feasible solutions: feasible solutions are the solutions that satisfy all the constraints; *Basic feasible solutions*, meaning feasible solution that is also basic.

Remark. For a solution to be basic, the basis of the solution should be linearly independent.

(2) Form a complementary non-basis N.

(3) Corresponding tableau:

$$\begin{array}{rcrcll} z & - & \sum_{j\in N} \bar{c}_j x_j & = & \bar{v} & \\ x_i & + & \sum_{j\in N, i\in B} \bar{a}_{ij} x_j & = & \bar{b}_i & i \in B \end{array}$$

Step 2: Check optimality

(1) If each reduced cost satisfies $\bar{c_j} > 0$, for all $j \in N$, then iteration stops, that is the current solution is optimal.

(2) j_{enter}

If we don't stop in step1, then we will get a list of indices subh that the reduced cost c_j is negative, that is $J = \{j : c_j \bar{<} 0\}$.

We choose the first index in list J, j_{enter} to enter the basis, as c_j and start the iteration to minimize z.

Step 3: Check Unboundedness

(1) If all $\bar{a}_{ij} < 0, \forall i \in B$, we can make the value t as small as we can, further,$z = \bar{v} + \bar{c}_j t$ will become as small as we like. So in this case, the LP is unbounded.

$$x_i = \bar{b}_i - \bar{a}_{ij} t, i \in B$$

(2) r_{leave}

If we don't stop in the previous step, we may get at least one $i \in B$, $\bar{a}_{ij} < 0$ s.t.

$$x_i = \bar{b}_i - \bar{a}_{ij} t \geq 0, i \in B$$

Make sure that

$$t \leq \frac{\bar{b}_i}{\bar{a}_{ij}}, \forall i \in \bar{B}, \bar{B} = \{i \in B : \bar{a}_{ij} > 0\}$$

$$\bar{t} = \min \left\{ \frac{\bar{b}_i}{\bar{a}_{ij}} : i \in B \right\}$$

Let L denote the set of indices i attaining the minimum in the definition of $\bar{t}$. Choose the first index in the set L to be r_{leave}.

Step 4: Update

Update the new basis B and new B^{-1} and go back to the first step.

1.3.2 Phase 1

To begin the simplex method for solving a linear program, we need to find an intial feasible tableau. This is called the *phase 1* problem of simplex.

The general idea of "Phase 1" of the simplex method is, corresponding to each constraint, to introduce an "artificial variable", u_i,which we can think of it as the error in the constraint. Then we can transform the original constraint LP system:

$$\sum_{j=1}^{n} a_{ij} x_j = b_i, (i = 1, 2, ...m)$$

To the following constraints:

$$\sum_{j} a_{ij} x_j + u_i = b_i (i = 1, 2, ..., m)$$

We then may want to try force $u_i \to 0$ by using the simplex method we've discussed before to minimize the sum:

$$\min \left\{ \sum_{i=1}^{m} u_i : \sum_{j} a_{ij} x_j + u_i = b_i (i = 1, 2, ..., m) holds, x \geq 0, u \geq 0 \right\}$$

Example 1.4. Consider the following LP

$$\begin{array}{lccccccc} \max & 2x_1 & + & x_2 & & & & \\ s.t. & x_1 & + & x_2 & & & = & 3 \\ & -x_1 & + & x_2 & - & x_3 & = & 1 \\ & x_1, & & x_2, & & x_3 & \geq & 0 \end{array}$$

we introduce the *artificial variable* here,

$$x_4 = 3 - (x_1 + x_2)$$
$$x_5 = 1 - (-x_1 + x_2 - x_3)$$

Then we force these errors to be zero, our objective will become,

$$\min\ x_4 + x_5,\ x_1, x_2, x_3, x_4, x_5 \geq 0 (*)$$

Important! Now, the original LP is feasible $\Leftrightarrow$ $(*)$ has an optimal solution with value zero.

Then we are able to reduce the problem of feasibility of LP to the problem of solving an auxiliary LP.

$$\begin{array}{lrcrcrcrcrcl} \max & w & = & -x_4 & - & x_5 & & & & & \\ s.t. & x_1 & + & x_2 & & & + & x_4 & & & = & 3 \\ & -x_1 & + & x_2 & - & x_3 & & & + & x_5 & = & 1 \\ & x_1, & & x_2, & & x_3, & & x_4, & & x_5 & \geq & 0 \end{array}$$

Use the simplex method again to solve this new LP, and aiming to get the objective value 0. And by contrast with original simplex method, the advantage of the auxiliary problem is availability of an obvious initial feasible tableau, which is just the *artificial variable* x_4 and x_5.

1.4 Linear Programming Duality

1.4.1 Finding the Dual of a Linear Program

How might we judge how good a solution of LP is ? If we have a feasible solution, how do we certify that it is in fact optimal?

Given a linear program in standard form,

$$\begin{array}{lrcl} \max & c^T x & & \\ s.t. & Ax & = & b \\ & x & \geq & 0 \end{array}$$

we may define the *dual problem* as the linear program:

$$\begin{array}{lrcl} \min & b^T y & & \\ s.t. & A^T y & \geq & c \end{array}$$

Example 1.5. For instance, for the primal problem,

$$\begin{array}{lrcrcrcl} \max & x_1 & - & x_2 & + & 7x_3 & & \\ s.t. & 2x_1 & - & x_2 & + & x_3 & = & 1 \\ & x_1 & + & x_2 & + & 2x_3 & = & 5 \\ & x_1, & & x_2, & & x_3 & \geq & 0 \end{array}$$

the dual is,

$$\begin{array}{lrcrcl} \min & y_1 & + & 5y_2 & & \\ s.t. & 2y_1 & + & 7y_2 & \geq & 1 \\ & -y_1 & + & y_2 & \geq & -1 \\ & y_1 & + & 2y_2 & \geq & 7 \end{array}$$

1.4.2 Weak Duality

Before the weak duality, first consider the following concepts: *duality and complementary slackness*

$$(P) : \min c^T x$$

$$s.t. Ax = b, x \geq 0,$$

$A \in R^{m \times n}$, $b \in R^m$, $c \in R^n$ $m \leq n$. Solution vector for (P) : $x \in \mathbb{R}^n$

$$(D) : \max b^T y$$

$$s.t. A^T y \leq c$$

or

$$\max b^T y$$

$$s.t. A^T y + s = c$$

$$s \geq 0$$

$s \in \mathbb{R}^n$ $y \in \mathbb{R}^m$. Solution vector for (D): $(s, y) \in \mathbb{R}^{n+m}$. A pair of feasible solutions x,(s, y) satisfies **complementary slackness** provided that

$$x_1 s_1 = 0$$

$$...$$

$$x_n s_n = 0$$

that is for each i, $x_i = 0$ or $s_i = 0$. so: if $x_i > 0$, then $s_i = 0$;
if $s_i > 0$, then $x_i = 0$.

Remark. It is also possible that $x_i = 0$ and $s_i = 0$.

Since $x_i s_i \geq 0$, $\forall i$,
Then,

$$x_1 s_1 = 0$$

$$...$$

$$x_n s_n = 0$$

is equivalent to $\sum_{i=1}^{n} x_i s_i = 0$ or $x^T s = 0$.
Note: If x is feasible, s is also feasible, then $x^T s \geq 0$.
If x and s are complementary, $x^T s = 0$.

Theorem (The Weak Duality Theorem)**.** If x is feasible for (P) and (s, y) is feasible for (D), then $c^T x \geq b^T y$

Proof. $b^T y = (Ax)^T y = x^T (A^T y) = x^T (c - s) = x^T c - x^T s \leq x^T c = c^T x.$ □

Theorem (Fundamental Theorem of LP)**.** (1) If LP has feasible solution. it ha a basic feasible solution.
(2) If LP has feasible solution and it's bounded, it has an optimal basic solution.
(3) If LP doesn't have optimal solution, it is infeasible or unbounded.

1.4.3 Strong Duality and Farkas's Lemma

Theorem (The Strong Duality Theorem)**.** If (P) or (D) has finite optimal value, then so does the other. Furthermore, their optimal values are equal and the optimal solution to both (P) and (D) exists.

Proof. Suppose, w.l.o.g. (P) has an optimal solution and suppose that x^* is a basic optimal solution, with basis B.

$\bar{c} = c^T - c_B^T A_B^{-1} A$	$-c_B^T A_B^{-1} b$
$A_B^{-1} A$	$A_B^{-1} b$

□

According to the simplex tableau, when

$$\bar{c} \geq 0$$

we stop the loop.

$$\Rightarrow c^T - c_B^T A_B^{-1} A \geq 0$$
$$\Rightarrow c \geq A^T [(A_B)^{-T} c_B]$$

Then let $y^* = ((A_B)^{-1})^T c_B$, $s^* = c - A^T y^*$

Theorem (Farkas' Lemma)**.** Let $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, the lemma states that *exactly* one of the following is true:

1. $\exists x \in \mathbb{R}^n : Ax = b, x \geq 0$
2. $\exists y \in \mathbb{R}^m$: $A^T y \geq 0, b^T y < 0$

1.4.4 Duality Gap

In optimization,the *duality gap* means the difference primal and dual solutions. If we denote the primal optimal solution as $p*$ and we denote the dual optimal solution as $d*$, the gap is $p* - d* \geq 0$(for minimization). $p* - d*$ is zero if and only if the strong duality theorem holds. Otherwise, the weak duality holds.

1.5 Interior point Method

1.5.1 KKT condition

Consider the nonlinear optimization problem:

$$\begin{array}{rlcl} \min / \max & f(x) & & \\ s.t. & g_i(x) & \leq & 0 \\ & h_j(x) & = & 0 \end{array}$$

where $g_i(i = 1, 2, ...m) \in \mathbb{R}^m$ is the *inequality* constraints and $h_j(j = 1, 2, ...n) \in \mathbb{R}^n$ is the *equality* constraints. The number of equalities and inequalities are denoted as m, n respectively. If x^* is the local minimum and the minimization

problem satisfies the following conditions,then there exists constants $\mathbf{u}$, λ (the Lagrangian multipliers) such that

$$\nabla f(x^*) + \sum_{i=1}^{m} u_i \nabla g_i(x^*) + \sum_{j=1}^{n} \lambda_j \nabla h_j(x^*) = 0 \qquad (K_1)$$

Primal feasibility:

$$g_i(x^*) \leq 0, i = (1, 2, ...m) \qquad (K_2)$$
$$h_j(x^*) = 0, j = (1, 2, ...n)$$

Dual feasibility:

$$u_i \geq 0, i = (1, 2, ...m) \qquad (K_3)$$

Complementary Slackness:

$$u_i g_i(x^*) = 0, i = (1, 2, ...m) \qquad (K_4)$$

Remark:

Recall that given an objective function $f(x)$, $f : \mathbb{R}^n \to \mathbb{R}$ and a set of constraint functions $c_i(x)$, $c_i : \mathbb{R}^n \to \mathbb{R}$ one can define the Lagrangian as

$$L(x, \lambda) = f(x) - \sum_{i=1}^{m} \lambda_i c_i(x)$$

where $\lambda = (\lambda_1, \lambda_2, ..., \lambda_m)$ is a collection of Lagrangian multipliers.

1.5.2 IPM

Consider an instance of linear programming problem in a standard form:

$$\min c^T x$$

$$s.t. Ax = b$$

$$x \geq 0$$

where $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$. We associate a barrier function $B(x, \mu)$

$$B(x, \mu) = c^T x - \mu \sum_{j=1}^{n} \ln(x_j), \mu > 0$$

Then instead of considering the original LP, we consider the problem of minimizing the above barrier function s.t. $Ax = b$.

$$\min_{x \in R^n} B(x, \mu) s.t. Ax = b$$

Associated with this new barrier problem is a Lagrangian defined by

$$L(x, \lambda, s) = c^T x - \mu \sum_{i=1}^{n} \ln x_j - \lambda^T (Ax - b)$$

Let $g(x) = \nabla_x L(x, \mu) = c - A^T\lambda - \mu X^{-1}e$ where $X = diag(x_1, x_2, ...x_n)$, $e = (1, 1, 1..., 1)^T$ Then the KKT conditions can be written as

$$\begin{aligned} A^T\lambda + \mu X^{-1}e &= c \\ Ax &= b \\ XSe &= \mu e \end{aligned}$$

If we drive $\mu \to 0$, then the minimum to the barrier function will be a minimum to the original LP.

Algorithm for Primal-Newton Barrier Method IPM
(adapted from lecture notes,[Rob12])
Choose $\rho \in (0, 1)$ and $\mu_0 > 0$
Choose a point x_0 as the initial starting point s.t. $Ax_0 = b$ and $x \geq 0$.
for $k = 0, 1, 2, ...$ **do**
$u_k = \rho u_{k-1}$
Compute the constrained Newton direction p_B using the formula.
Solve $argmin_\alpha B(x_k, \mu_k)$ where
$x_k = x_{k-1} + \alpha p_B$ s.t. $Ax_k = b$
update x_k
update $k \leftarrow k + 1$

2 Computational Results

2.1 Simplex Method

2.1.1 Stage 1

In Stage one, we basically tried to figure out the implementation of simplex method by assuming that *the initial basis are somehow given.*

We used four inputs for $simplex(A, b, c, B_0)$ (including the initial basis). After doing the initialization, the first problem to concern is *the number of iterations.* In the implementation attached in the *appendix A*, for the iteration conditions,for instance, n_{max},we choose the number of iterations to be small, that is, $n_{max} = 10$, and also the duality gap is small,etc.In the future, to do tests with more dimensions or more constraints, we can modify these iteration conditions and get improvement.

Then the second thing to notice is that *the sign of termination of the loop* is that when we can not find any negative indices, then we will end up the loop, meaning that we've already found the optimal solution.

One computational difficulty that I would like to talk here is when we are doing the *update* part, we need to update the matrix B with the new basis after leave/enter indices, and also compute the new updated $inv(B)$. One way to realize this, is just to simply use the MATLAB built-in function,$inv(X)$, which automatically compute the inverse matrix of the input matrix use the LU decomposition; The other alternative to compute the updated inverse matrix B is to do row reduce computation by hand and then implement a more concrete computing algorithm. The general idea is to update the leaving index column to be new entered index

corresponding data. To be more specific, let B denote the matrix we choose from A according to the basis index B_0, and then update each row of B^{-1} as

$$\begin{cases} B_i^{-1} * A[j_{enter}]_i, \text{if not the leaving index row} \\ B_i^{-1} - B_{r_{leave}}^{-1} * (B^{-1} * A[j_{enter}])_i / (B^{-1} * A[j_{enter}])_{r_{leave}}, \text{if is the leaving index row} \end{cases}$$

In the implementation attached in the appendix, we figured out the second implementation, which indeed saved time and space.

The main test case we've used is mentioned before as Example 1.2, and the test files can be changed into various dimensions or constraints in the future to do more interesting experiments.

2.1.2 Stage 2

As mentioned in Stage 1, the initial basis/tableau are given, now we are trying to compute the initial feasible basis by MATLAB using the phase 1 algorithm (mentioned in previous section). We re-call the simplex1 function in this part to find the possible initial feasible basis.

Debug difficulty mainly depends on your implementation for simplex1,(because our implementation needs to call simplex1 in phase1.)

Future directions: We can randomly generate matrix combinations and objective functions and then put them in the the simplex function and use tic toc to trace the running time. And then observe the relationship between matrix dimensions, that is the number of variables and running time. Maybe with n being increased, the simplex method performs slower than for smaller dimension matrix.

2.2 IPM

2.2.1 Stage 1

Under the Stage 1 implementation, it's very hard for the solution to converge to the actual test cases, because x_k can easily go out of the domain. Since stage 1 is the interior point method using unit step size without introducing the variable *alpha*, that is we update the x vector by simply adding a Newton direction vector and not considering the adjustment, i.e. sizing variable. Later in Stage 2, we may scale this Newton direction by multiply the step size.

2.2.2 Stage 2

We then improve the algorithm, by finding the min of α s.t.the barrier function

$$\min_{\alpha} B(x_k, \mu_k), x_k = x_{k-1} + \alpha p_B$$

reaches its minimum. Since $|\alpha| \in (0, 1)$, we can prevent the updated x from getting to close to the boundary or getting out of the domain,etc.

The method we use here to do the minimization for argument α is Newton's method.(See Appendix)

The main difficulty is we should gather together all the KKT conditions and constraints to derive a matrix representation:

$$\begin{pmatrix} H & A^T \\ A & 0 \end{pmatrix} \begin{pmatrix} -p_B \\ \lambda \end{pmatrix} = \begin{pmatrix} c - \mu X^{-1} e \\ 0 \end{pmatrix}$$

(The KKT conditions are derived in section1.5) and based on this,we should correctly translate the newton direction vector p_B expression into numerical representation with taking care of all the matrix dimension when doing multiplication.

One future possible direction that we can play on is to compare the running time and performance of Interior Point Method and Simplex. In different occasions, those two functions' behavior may vary from each other. And also test on longer problems with huge dimensions and even randomly generated inputs.

The test case we used for IPM is the same as simplex, Example 1.2 and the IPM gave us an approximately right solution after 5 iterations.

3 Clustering

3.1 Intro to Clustering and Motivation for SURE

Cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense) to each other than to those in other groups (clusters). It is a main task of exploratory data mining, and a common technique for statistical data analysis, used in many fields, including machine learning, pattern recognition, image analysis, information retrieval, bioinformatics, data compression, and computer graphics.

There are several types of clustering algorithm,including Connectivity-based clustering, centroid-based clustering(such as k-means clustering), also distributed-based/density-based,etc..) These clustering problems are NP hard: it is computationally expensive to find exact optimal clusters. The "linear programming (LP) relaxations" are closely related to the original problems but computationally easier to solve. In general, the optimal solution to the LP relaxations of computationally-hard problems are usually approximations to the true optimal solutions.

Inspired by the previous work done by Rachel Ward et.al [ABC$^+$15], our SURE project, in the intersection of mathematics and data science, aims to study under what conditions will an approximation to a clustering problem yield exact solution. To be more specific, for the the k-median problem, the authors showed that in fact the optimal solution to the LP relaxations of are "tight" or exact optimal solutions, provided that the points to be clustered do belong to k clusters that are far enough apart from one another. However, the authors observed that in computational experiments (in randomly- generated problems), the optimal solutions to the LP-relaxations of the k-median clustering problems are tight even in a weaker setting, such as when the points are drawn from a single distribution but multiple clusters. Therefore, we've decided to go further to see whether the integrality of LP relaxation for k-medians still holds for data points who are not in clustering setting. (Ideas inspired by [Ban15])

3.2 Background and Summary of Paper

3.2.1 Integrality and k-median relaxation

The k-median problem, expressed in the form of an integer programming problem, has a natural linear programming relaxation, given by relaxing the integral

constraints to interval constraints.

$$\min_{z\in\mathbb{R}^{n\times n}} \sum_{p,q} d(p,q)z_{pq}$$

$$s.t. \sum_{p\in P} z_{pq} = 1, \forall q \in P$$

$$z_{pq} \leq y_p, \forall p, q \in P$$

$$\sum_{p\in P} y_p = k$$

$$z_{pq}, y_p \in [0,1], \forall p, q \in P$$

This is the linear program studied. Then the dual LP for it is,

$$\max_{\alpha\in\mathbb{R}^n} \sum_{q\in P} \alpha_q - k\xi$$

$$s.t. \alpha_q \leq \beta_{pq} + d(p,q), \forall p, q \in P$$

$$\sum_{q} \beta_{pq} \leq \xi, \forall p \in P$$

$$\beta_{pq} \geq 0, \forall p, q \in P$$

$y_p \in \{0,1\}$ indicates whether the point $p \in P$ is a center or not. The variable $z_{pq} \in \{0,1\}$ $\forall p, q \in P$ indicates whether or not the point p is the center for the point q.

From [ABC+15, Lemma 5] in the paper, we've learned that if we choose feasible dual variables $\alpha_1, ..., \alpha_k$ to satisfy

1. Each center sees exactly its own cluster, i.e. $(\alpha_j - d(c_j, q))_+ > 0$ if.f. $q \in A_j$
2. $\sum_{q\in A_1}(\alpha_1 - d(s,q))_+ + ... + \sum_{q\in A_k}(\alpha_k - d(s,q))_+$ attains its maximum in the centers $c_1, .., c_k$.
3. Each of the terms $n_i\alpha_i - \min_{p\in A_j} d(p,q)$ in the average are the same.

then the above $k-median$ LP is integral and the partition $A_1, A_2, ..A_k$ is optimal.

To provide a set of conditions in the data points to guarantee such feasible dual variables, assume the sets $A_1, ..A_k$ are contained in disjointed balls $B_{r_1}(c_1)...B_{r_k}(c_k)$ respectively. The conditions basically ask that the clusters satisfy the *separation* and *center dominance* conditions.

3.2.2 Definition of Separation & Center dominance

1. Separation: Let the sets $A_1, ..., A_k$ in X, $|A_1| = ... = |A_k| = n$ such that

   $$OPT_1 \leq ... \leq OPT_k$$

   if they are included in k disjoint balls

   $$A_1 \subset B_1(c_1), ..., A_k \subset B_1(c_k), d(c_i, c_j) = 2 + \delta_{ij}, i \neq j, \delta_{ij} > 0$$

and the distance between two balls $B_1(c_i)$ and $B_1(c_j)$ satisfies:

$$\min_{1\le i,j\le k} \delta_{ij} > \frac{OPT_k - OPT_1}{n}$$

$\frac{OPT_k - OPT_1}{n}$ measures the difference between two clusters.

2. Center dominance: $A_1, ..., A_k$ satisfy center dominance int he interval $(a, b) \subset (1, 1 + \min_{1\le i,j\le k} \delta_{ij})$ if

$$b - a > \frac{OPT_k - OPT_1}{n}$$

and $\forall \alpha_1, ..., \alpha_k \in (a, b)$ there exists $\tau_1, ..., \tau_k > 0$ such that $\forall x \in B_{\tau_j}(c_j), j = 1, ..., k$, we have

$$B_{\alpha_i}(x) \cap B_{r_i}(c_i) = B_{r_j}(c_j), i = j$$

$$B_{\alpha_i}(x) \cap B_{r_i}(c_i) = \emptyset, i \neq j$$

$$\max_{y\in A_j \setminus B_{\tau_j}(c_j)} P^{(\alpha_1,...,\alpha_k)}(y) < \max_{y\in B_{\tau_j}(c_j)} P^{(\alpha_1,...,\alpha_k)}(y)$$

The center dominance basically states that the contribution function attains its max in a small neighborhood of the center of each ball, as long as the parameters α are chosen from some small interval.

3.2.3 Failure of LLOYD's method and k-means

It is being said in the paper that when the number of clusters and the dimension of the space is large enough, both LLOYD and k-means clustering method fails with high probability to exactly recover the clusters.

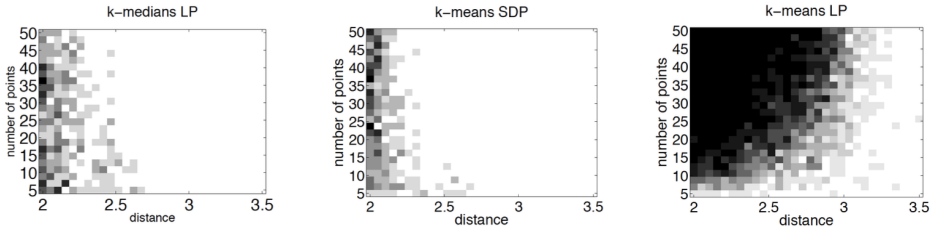
3.3 Open questions and insights for SURE

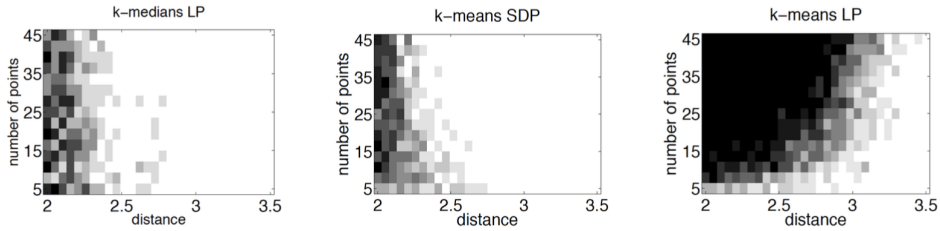
3.3.1 Simulation for the data points

In [ABC+15], the author's input consists of k disjoint unit-radius balls in $\mathbb{R}^m$ such that the centers of distinct balls are separated by distance $\Delta \geq 2$. Then randomly draw $N = kn$ points. and the n points are i.i.d. uniformly within each ball. Then implement and solve the convex optimization using MATLAB.

For each value of Δ and n, they repeat the experiment 10 times and plot.

The experiment result shows that k-median LP is much better than the k-means SDP and LP. To be more specific, we can see from the following simulation results in [ABC+15],





that regardless of the number of clusters k we've choose, k-means LP obtains darker color, which means with higher probability of success.

3.3.2 Open questions we can study and Insights

The analysis in [ABC$^+$15] for k-median LP shows that for any data points with separation $2+\epsilon$ and any number of clusters k, the solution of the k-median LP is the planted clusters with high probability if n is large enough. n is the number of points we choose within each ball is large enough. Here, we raise the first point to be studied further.

1. **How large should n be?** As said before, when the number of points we draw from each cluster increases, the probability of recovering the clusters will also increase. But it still remains to be quantified that how precise should this n be. We can possibly derive a formula or constraint on the number n with regard to the other factor variable k and and Δ.

2. Also for the k-means LP, is it possible to do **rounding** from the failure results in order to recover the expected clustering results?

   In experiments, getting the integral solutions is ideal. We will also encounter a lot of fractional solutions. The definition of clustering with fractional solutions is yet vague. We may define a rule to give meaning to the fractional solutions representation of clustering. And thus by doing rounding, we can probably give meaning to the failure results and it will be recovered to the expected clustering results.

3. **More general possible conclusions**

   Right now, the author uses the equal radius for each unit ball and n equal points are drawn from each cluster. If we relax these conditions, will the exact recovery still exist?

4. The balls overlap and/or when the points are drawn according to **a mixture of Gaussians**.

   Right now, in [ABC$^+$15], n points are drawn uniformly within each ball, if our balls has overlapping and for instance, if we draw data points from a mixture of Gaussian distribution, with μ shifted, what will happen?

5. **Relax the notion of integrality**. The last point is similar to the rounding rules. If we relax our notion of integrality, that is , we instead ask whether the convex optimization provide us a near-optimal solution.(This is mainly for k-means algorithm)

References

[ABC+15] Pranjal Awasthi, Afonso S. Bandeira, Moses Charikar, Ravishankar Krishnaswamy, Soledad Villar, and Rachel Ward. Relax, no need to round: integrality of clustering formulations. In *ITCS'15—Proceedings of the 6th Innovations in Theoretical Computer Science*, pages 191–200. ACM, New York, 2015.

[Ban15] Afonso Bandeira. Euclidean clustering. 2015.

[Rob12] Robert Robere. Interior point methods and linear programming. 2012.

[Van14] Robert J. Vanderbei. *Linear programming*, volume 196 of *International Series in Operations Research & Management Science*. Springer, New York, fourth edition, 2014. Foundations and extensions.

A Implementation of Simplex method in MATLAB

```
% Implementation of the simplex method.
% min z= c'*x
% subject to  Ax = b
%             x >= 0
% Function input parameters :
% A: The constraints    b: given R.H.S. vector  c: cost vector
% B_0: indices vector corresponding to the initial feasible basis B
% Returns: x_opt: optimal solution vector
%          z_opt : optimal objective value
% Find the initial tableau??
%
function [x_opt, z_opt] = simplex_1(A,b,c,B_0)
x = zeros (length(c),1);
B = A (:,B_0);
inv_B = inv(B);
x(B_0) = inv_B * b;

% set the maximum iteration time  to be 10
n_max = 10;
for n = 1: n_max
    % the reduced cost : (for j in N, nonbasic vector)
    c_B = c(B_0);
    p = (c_B' * inv_B )';
    c_j = c' - p' * A; % vector of reduced costs

    J = find (c_j < -exp(-16)); % find negative indices

    if (isempty(J)) % all of the indices are positive
        z_opt = -c' * x ;
```

```
        x_opt = x;
        return;
    end

    j_enter = J(1); %choose the first index to enter


    u = inv_B * A(:,j_enter); %used later also for inverse computation
    U = find (u>0);
    if (isempty(U))
        z_opt = -inf; % unbounded, z will become as small as we like
        x_opt = []; % we can't find such basic solution
        return
    end

    t_bar = min(x(B_0(U))./u(U));
    L = find (x(B_0)./u == t_bar);
    % then the leaving index can be any index from set L

    r_leave = L(1);

    %update
    x(B_0)= x(B_0)- t_bar * u;
    x(j_enter) = t_bar;
    B_0(r_leave) = j_enter;


    B=A(:,B_0);
    [m,~]= size(inv_B);
    for i = 1:m
        if i~=r_leave
           newinv_B(i,:)=inv_B(i,:)-inv_B(r_leave,:)*u(i)/u(r_leave);
        else
            newinv_B(i,:)=inv_B(i,:)/u(i);
        end
    end
    inv_B= newinv_B;
end
end

% Test case 1
A = [1 2 2 1 0 0;
     2 1 2 0 1 0;
     2 2 1 0 0 1
     ];
 b = [20 20 20]';
 c = [-10 -12 -12 0 0 0]';
 B_0 = [4 5 6];
```

```
 [x z] = simplex_1(A,b,c,B_0)

x =

     4
     4
     4
     0
     0
     0


z =

   136

% Test case 2
A= [-1 3 1 0 0;
     1 1 0 1 0;
     2 -1 0 0 1];
b = [12 8 10]';
c = [-3 -2 0 0 0]';
B_0 = [3 4 5];
[x z]=simplex_1(A,b,c,B_0)

test2

x =

     6
     2
    12
     0
     0


z =

    22

%Test case 3
A= [ 2 1 1 0;
     1 2 0 1];
b = [4 3]';
c = [-1 -1 0 0]';
B_0 = [3 4];
[x z]=simplex_1(A,b,c,B_0)
```

```
test3

x =

    1.6667
    0.6667
         0
         0


z =

    2.3333
```

B Implementation of Interior Point Method in MATLAB

```
function [x_opt] = IPM(A,b,c,u,x) % x_0 has to satisfy that Ax_0 = b
    p = 0.5; % choose p in (0,1)b
    [m n] = size(A);
    n_max = 10;
    for k = 1: n_max
        u = p * u;% u_(k+1) = p * u_(k)
        X = diag(x);
        X
        X2 = diag(x.^2);
        X2
        e = ones(n,1);
        M = (A * X2) * A';% M = AX^2A^{T}
        M
        z_1 = A * (c.*(x'.^2)) - u*A* x';
        z_1
        lamda = M\z_1;
        d = 1/u * X2 *(A'* lamda - c + u * (e./x'));
        % inplement for the alpha with min_alpha
%         alpha_0 = 0.5; % how to choose the initial start point near the
%         alpha = root_finding(f,alpha_0);
        %v = d./(x + a * d);
        %f_alpha = c'* d + u*sum(v);
        %df_alpha = u* sum(v*v);
        tol = 1e-8;
        nmax = 1e1;
        alpha_0 = 0.5;
        %alpha_opt = fminsearch(f_alpha,alpha_0);
        alpha_opt = newton(c,d,x,u,tol,nmax,alpha_0);
        alpha_opt
        d1= alpha_opt*d';
        d1
```

```
        x = x+d1;% x_(k+1) = x_(k)+alpha_opt*d'
        x
        pause;
    end
    x_opt = x;
    return;
end

function y = f_alpha(alpha,c,d,x,u)
     y =c'* d - u*sum(d./(x + alpha * d));
end

function y = df_alpha(alpha,d,x,u)
     y = u*sum((d./(x + alpha * d)).*((d./(x + alpha * d))));
end



function [alpha_opt] = newton (c,d,x,u,tol,nmax,alpha_0)
    y = @f_alpha;
    dy = @df_alpha;
    alpha(1)= alpha_0 - y(alpha_0,c,d,x,u)/dy(alpha_0,d,u,x);
    ex(1) = abs(alpha(1)-alpha_0);
    k = 2;
    while (ex(1) >= tol) && (k<= nmax)
        alpha(k) = alpha(k-1)- y(alpha(k-1),c,d,x,u)/dy(alpha(k-1),d,u,x)
        ex(k) = abs(alpha(k)-alpha(k-1));
        k= k+1;
    end
    alpha_opt = alpha(k-1);
end
```

Result:

```
alpha_opt =

    0.5012


d1 =

    0.0126   -0.0070    0.0337   -0.0056   -0.0323


x =

    5.9894    2.0070   11.9685    0.0036    0.0282
```