AADS - Assignment 2

Aditya Fadhillah (hjg708), Mads Nørregaard (gnz359), Nikolaj Schaltz (bxz911)

January 8, 2025

Contents

1 Exercises from CLRS

1.1 CLRS 29.1-9

Give an example of a linear program for which the feasible region is not bounded, but the optimal objective value is finite.

Answer:

An example of said linear program can be that we want to maximize $z = -x$ subject to:
$x + 2y \geq 3$
$x - 2y \geq -1$
$x, y \geq 0$

The feasible region is above $y = x + 1$, below $y = 3$ and y is strictly positive. The feasible region is not bounded for y, so it can increase indefinitely, but there is a bound on x so it cannot decrease indefinitely. Therefore, the optimal objective value must be finite, given the bound on x and we want to maximize $-x$.

1.2 CLRS 29.2-3

In the single-source shortest-paths problem, we want to find the shortest-path weights from a source vertex s to all vertices $v \in V$. Given a graph G, write a linear program for which the solution has the property that d_v is the shortest-path weight from s to v for each vertex $v \in V$.

Answer:

$$\text{Maximize} \sum_{v \in V} d_v$$

$$\text{Subject to:}$$

$$\begin{aligned} & d_s = 0, \\ & d_v \leq d_u + w(u, v), \quad \forall (u, v) \in E \end{aligned}$$

The objective function $\sum_{v \in V} d_v$ maximizes the shortest total distances. The constraint $d_s = 0$ sets the distance from the source to itself as 0. The triangle inequality constraints $d_v \leq d_u + w(u, v)$ ensure that d_v satisfies the shortest-path condition.

The solution d_v represents the shortest-path weight from s to v for all $v \in V$, satisfying the shortest-path properties.

1.3 CLRS 29.2-6

Write a linear program that, given a bipartite graph $G = (V, E)$, solves the maximum-bipartite-matching problem.

Answer:

It is assumed we are not supposed to solve the problem step-by-step, only detail how said program will work and its variables/properties.

We partition V into two disjoint subsets U and W so that $V = U \cup W$. For the linear program, we want to find the max. amount of edges matching, i.e., the largest subset of edges where no two edges share ending vertices.

For each edge in G, (u, v), we assign a variable x_{uv}, which can have the following value:

$$x_{uv} = \begin{cases} 1 & \text{if edge } (u, v) \text{ is matching} \\ 0 & \text{otherwise} \end{cases}$$

This will act as the decision variable. It is constrained by it only being able to have a binary value:

$$0 \leq x_{u,v} \leq 1 \forall (u,v) \in E$$

We have to add another constraint to the edges; each vertex can only have at most one matching edge:

$$\sum_{(u,v)\in E} x_{uv} \leq 1, \forall u \in U$$
$$\sum_{(u,v)\in E} x_{uv} \leq 1, \forall v \in W$$

We now have the following linear problem; maximize the amount of edges $max \sum_{e \in E} x_{uv}$, subject to:

$$\sum_{(u,v)\in E} x_{uv} \leq 1, \forall u \in U$$
$$\sum_{(u,v)\in E} x_{uv} \leq 1, \forall v \in W$$

and where:

$$0 \leq x_{u,v} \leq 1 \forall (u,v) \in E$$

While the optimal solution is always a binary integer, there is a chance that solutions with fractions instead of binary integers exist. This can be solved by relaxing the program to only yield integer solutions, through means such as rounding the solution, thus making the linear program optimal for solving the problem.

1.4 CLRS 29.4-3

Write down the dual of the maximum-flow linear program, as given in lines (29.47)-(29.50) on page 860. Explain how to interpret this formulation as a minimum-cut problem.

First we look at the capacity constraint (line 29.48). We have that for every capacity constraint $c(u,v)$ on edge (u,v), we will have a new dual variable:

$$c'_{uv}$$

For the flow constraint (line 29.49), we have that for every flow constraint on vertex $u \notin \{s,t\}$, we have a dual variable:

$$f'_u$$

We thus have the constraints for the dual of the program:

$$f'_u - f'_v \leq c'_{uv} \quad \forall (u,v) \in E \qquad (29.48),$$
$$f'_s = 1, \quad f'_t = 0 \qquad (29.49),$$
$$c'_{uv} \geq 0 \quad \forall (u,v) \in E \qquad (29.50).$$

Finally, using the constraints and dual variables, as well as the fact that we want to interpret this problem as a min-cut problem, we have the following problem definition:

$$\min \sum_{(u,v)\in E} c_{uv} c'_{uv}$$

subject to:

$$f'_u - f'_v \leq c'_{uv} \quad \forall (u,v) \in E \qquad (29.48),$$
$$f'_s = 1, \quad f'_t = 0 \qquad (29.49),$$

$$c'_{uv} \geq 0 \quad \forall (u,v) \in E \qquad (29.50).$$

The capacity constraint has been changed to include the dual variable. c'_{uv} indicates if an edge contributes to the cut. The sum in the objective definition is the sum of the capacities of the edges included in the cut. The flow constraint f'_u represents if a given vertex u is closer to the source s or sink t, where a value of 1 represents it being closer to s and a value of 0 represents being closer to t. We call this being closer to the source side (if 1) or closer to the sink side (if 0). $f'_u - f'_v \leq c'_{uv}$ constraint ensures that edges crossing the cut will be correctly accounted for.

Overall, the goal of the dual of the linear problem is to minimize the capacity of the edges crossing from the source side to the sink side. This problem corresponds to solving the min-cut problem and also aligns with the max-flow min-cut theorem, which states the value of the max. flow equals the capacity of a min. cut.

2 Standard Form LP

Write an LP in standard form in the plane so that: (1) the feasible region is a convex polygon with 5 edges, (2) the maximum of the LP is 1 and (3) the maximum is achieved on a full edge of the feasible region (and not just a single vertex).

Answer:

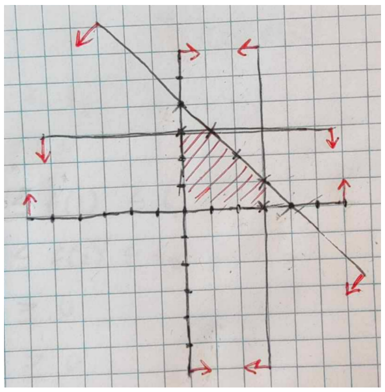

Figure 1: The graph shown on paper

Maximize:

$$z = x_1 + x_2$$

Subject to:

$$\begin{aligned} x_1 + x_2 + x_3 &= 1, & (1)\\ x_1 + x_4 &= 0.75, & (2)\\ x_2 + x_5 &= 0.75, & (3)\\ x_1 - x_6 &= 0, & (4)\\ x_2 - x_7 &= 0. & (5) \end{aligned}$$

All variables are non-negative:

$$x_1, x_2, x_3, x_4, x_5, x_6, x_7 \geq 0.$$

The feasible region described by these constraints is a convex polygon with five edges. The objective function $z = x_1 + x_2$ achieves its maximum value of 1 on the edge defined by $x_1 + x_2 = 1$, between the points $(0.25, 0.75)$ and $(0.75, 0.25)$. This ensures the maximum is achieved on a full edge of the feasible region, as required. The slack variables $x_3, x_4, \ldots, x_7$ allow us to convert the inequalities into equalities, ensuring the problem is in standard form.

3 Non-linearities in LP

This exercise demonstrates that linear programming (LP) can represent non-linearities such as absolute values. Let $f : \{1, 2\} \to \mathbb{R}$. Write an LP whose value is $|f(1)| + |f(2)|$, and write the dual of this LP.

In your solutions, all entries in the matrix A and the vectors b, c must be affine functions in $f(1)$ and $f(2)$; that is, each entry must have the form $df(1) + ef(2) + g$, where $d, e, g \in \mathbb{R}$.

For example, in dimension $n = 3$, the vector b can be

$$b = \begin{pmatrix} 3f(1) \\ 0 \\ -2f(2) + 1 \end{pmatrix} \tag{6}$$

but not

$$b = \begin{pmatrix} |f(1)| \\ 0 \\ 0 \end{pmatrix} \tag{7}$$

Answer:

To represent $|f(1)| + |f(2)|$ as a linear program, we introduce variables t_1 and t_2 to represent $|f(1)|$ and $|f(2)|$, respectively. Using the fact that the absolute value of a real number x, we apply this idea to $f(1)$ and $f(2)$.

Objective function:

We introduce variables t_1 and t_2 to represent $|f(1)|$ and $|f(2)|$. The objective function becomes:

$$\text{Minimize } z = t_1 + t_2 \tag{8}$$

To ensure $t_1 = |f(1)|$ and $t_2 = |f(2)|$, we add constraints.

Primal LP:

Therefore, the primal LP becomes:

$$\text{Minimize } z = t_1 + t_2 \tag{9}$$

Subject to:

$$t_1 \geq -f(1), \tag{10}$$
$$t_1 \geq f(1). \tag{11}$$
$$t_2 \geq -f(2), \tag{12}$$
$$t_2 \geq f(2). \tag{13}$$

Matrix form:

We rewrite the LP in standard matrix form Minimize $c^T x$ subject to $Ax \geq b$, where x includes both the variables t_1, t_2 and slack variables representing $f(1)$ and $f(2)$.

$$x = \begin{pmatrix} t_1 \\ t_2 \end{pmatrix}, \quad c = \begin{pmatrix} 1 \\ 1 \end{pmatrix}. \tag{14}$$

The constraints are:

$$Ax \geq b, \tag{15}$$

where:

$$A = \begin{pmatrix} -1 & 0 \\ 1 & 0 \\ 0 & -1 \\ 0 & 1 \end{pmatrix}, \quad b = \begin{pmatrix} -f(1) \\ f(1) \\ -f(2) \\ f(2) \end{pmatrix}. \tag{16}$$

Find dual LP:

To write the dual of the primal LP, we use the standard procedure for constructing the dual.

The primal problem is:

$$\text{Minimize } c^T x \qquad \text{subject to } Ax \geq b. \tag{17}$$

The dial LP becomes:

$$\text{Maximize } c^T x \qquad \text{subject to } A^T y = c, \quad y \geq 0. \tag{18}$$

The dual variables $y_1, y_2, y_3, y_4, y_5, y_6$ correspond to the primal constraints.

The matrix A^T is:

$$A^T = \begin{pmatrix} -1 & 1 & 0 & 0 \\ 0 & 0 & -1 & 1 \end{pmatrix}.$$

From $A^T y = c$, we get:

$$\begin{pmatrix} -1 & 1 & 0 & 0 \\ 0 & 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

This gives the constraints:

$$-y_1 + y_2 = 1, \tag{19}$$

$$-y_3 + y_4 = 1. \tag{20}$$

Additionally, $y \geq 0$ gives:

$$y_1, y_2, y_3, y_4 \geq 0.$$

Dual LP:

The dual LP can now be written as:

$$\text{Maximize } -y_1 f(1) + y_2 f(1) - y_3 f(2) + y_4 f(2),$$

subject to:

$$-y_1 + y_2 = 1, \tag{21}$$

$$-y_3 + y_4 = 1, \tag{22}$$

$$y_1, y_2, y_3, y_4 \geq 0. \tag{23}$$

In both the primal and dual LPs, all entries in A, b, and c are affine functions of $f(1)$ and $f(2)$.

The primal LP minimizes $t_1 + t_2$ with constraints ensuring that $t_1 = |f(1)|$ and $t_2 = |f(2)|$. The dual LP maximizes a linear combination of $f(1)$ and $f(2)$, subject to constraints derived from the primal. Both programs use affine functions of $f(1)$ and $f(2)$ in their matrices and vectors, meeting the requirements of the exercise.

4 Exercises for randomized algorithms

4.1

For any given key $x \in S$, let $d(x)$ be the depth of x in the search tree generated by `RandQS`. Give a lower and an upper bound, as a function of the number of keys n, on the expected value of $d(x)$ such that these bounds are within a constant factor of each other. That is, find a function $f(n)$ and prove that

$$\mathbb{E}[d(x)] = \Theta(f(n)).$$

Answer:

The depth $d(x)$ is the number of comparisons involving x until x becomes a pivot point or is in its final position. To calculate the depth, the depth of x is increased by 1 every time a partition is made and x was not the pivot point. The depth is calculated recursively as quicksort is called on each side of the pivot in a partition. Therefore we can write the depth as:

$$d(x) = 1 + d_{partition}(x)$$

Where $d_{partition}(x)$ is the depth of x in either the left or right partition.

Now the probability that x is not chosen as the pivot needs to be calculated. The probability is based on the rank of x which is where in the sorted list x is located, eg if x = 2 in the list from 1-20 and the pivot is 4 then in the next partition the chance of choosing x as the pivot is way higher compared to if x is higher than the pivot. The probabilities of x being on the left or on the right is given by the following where n is the amount of numbers in the array and i is the rank of x.

$$\text{Chance of left side:}\frac{i-1}{n-1} \qquad \text{Chance of right side:}\frac{n-i}{n-1}$$

This gives the expected depth as the function:

$$\mathbb{E}[d(x)] = 1 + \frac{i-1}{n-1}\mathbb{E}[d(x)] + \frac{n-i}{n-1}\mathbb{E}[d(x)]$$

We can simplify this expression because the symmetry caused by the randomization makes the expected depth equal for all ranks which makes the expected depth dependant on on the size n rather than x's position. Using this we can introduce k which is the size of partition where x ends up after a pivot, and let $D(n) = E[d(x)]$

$$D(n) = 1 + \frac{1}{n}\sum_{k=1}^{n-1}\left(\frac{k}{n-1}D(k) + \frac{n-k}{n-1}D(n-k)\right).$$

We rewrite this to use a recursive structure:

$$D(n) = 1 + \frac{2}{n}\sum_{k=1}^{n-1}\frac{k}{n}D(k).$$

This can be simplified further using continous approximation of large n:

$$D(n) \sim 2\int_1^n \frac{D(k)}{k}dk.$$

The solution to this integral recurrence is $D(n) \sim 2H_{n-1}$, where H_{n-1} is the $(n-1$)-th harmonic number:

$$H_{n-1} = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n-1}.$$

For large $n_1 H_{n-1} \sim \ln n$, so we can rewrite it as:

$$D(n) \sim 2\ln n.$$

This gives the expected depth:

$$\mathbb{E}[d(x)] = \Theta(ln(n)).$$

Where the upper and lower bound are within a constant factor.

4.2

How many runs of randomized contraction do we need to be 99% sure that a minimum cut is found?

Answer:

From slides we have that the probability of success in a single run is $P_{success} = \frac{2}{n(n-1)}$ with n being the amount of vertices in the graph. We can find the probability of failure by using the function:

$$P_{failure} = 1 - P_{success} = 1 - \frac{2}{n(n-1)}$$

This is needed as we want to handle multiple runs of the algorithm. Since we want to find how many runs are needed we need to find the probability after k runs:

$$p_{\text{failure after k runs}} = (1 - \frac{2}{n(n-1)})^k$$

We find the probability of success after k runs by substracting the failure after k runs with 1:

$$p_{\text{success after k runs}} = 1 - (1 - \frac{2}{n(n-1)})^k$$

Now we can find the amount of runs required for 99% probability, this is done by solving:

$$1 - (1 - \frac{2}{n(n-1)})^k \geq 0.99$$

We simplify:

$$(1 - \frac{2}{n(n-1)})^k \leq 0.01$$

We take the natural logarithm (ln) which allows us to remove the power:

$$k \cdot ln(1 - \frac{2}{n(n-1)}) \leq ln(0.01)$$

We can approximate as $ln(1-x) \approx -x$ for small x:

$$k \cdot (-\frac{2}{n(n-1)}) \leq ln(0.01)$$

We now solve for k:

$$k \geq \frac{-ln(0.01}{\frac{2}{n(n-1)}}$$

Which when -ln(0.01) is calculated gives:

$$k \geq \frac{4.605 \cdot n(n-1)}{2}$$

Since we do not have a given n for the assignment we are done.

4.3 PDF 1.2

Suppose that at each step of our min-cut algorithm, instead of choosing a random edge for contraction we choose two vertices at random and coalesce them into a single vertex. Show that there are inputs on which the probability that this modified algorithm finds a min-cut is exponentially small.

Answer:

We have a graph G_n with n vertices $(v_1, \ldots, v_n)$ with two properties; there is an edge between v_1 and v_n, and the first $\frac{n}{2}$ vertices constitute a clique. This means there is an edge between any vertex in $(v_1, \ldots, v_{\frac{n}{2}})$ and the renaming $\frac{n}{2}$ edges constitute a clique. In this case, (v_1, v_n) is the min-cut in G, which is also a unique min-cut given the clique properties.

The new min-cut algorithm will instead ONLY return the min-cut if no merges between vertices from the first clique with vertices from the second clique are done while the algorithm is running. If we let E_i be an output returned if vertices between the two cliques are never merged in step i of the algorithm, we write the probability of this event as:

$$\mathrm{P}\left(E_i \mid E_{i-1}, \cdots, E_1\right) = \frac{\binom{r_i}{2} + \binom{s_i}{2}}{\binom{r_1+s_1}{2}} = 1 - \frac{2r_i s_i}{(r_i+s_i)^2 - (r_i+s_i)} \leq 1 - \frac{2r_i s_i}{n^2}$$

where r_i is the amount of vertices remaining in the first clique and s_i is the amount of vertices remaining in the second clique. The sum of r_i and s_i can be written as:

$$r_i + s_i = n - i + 1$$

For $1 \leq i \leq \frac{n}{5}$, we have

$$r_i, s_i \geq \frac{n}{5}$$

The probability is thus:

$$\mathrm{P}\left(E_i \mid E_{i-1}, \cdots, E_1\right) \leq 1 - \frac{2 \cdot (n/5) \cdot (n/5)}{n^2} = \frac{23}{25}$$

This shows a very high probability, however this also means that:

$$\mathrm{P}\left(E_1, E_2, \cdots, E_{n-1}\right) \leq \mathrm{Pr}\left(E_1, E_2, \cdots, E_{\lceil n/5 \rceil}\right) \leq \left(\frac{23}{25}\right)^{\lceil n/5 \rceil}$$

This yields an exceptionally small probability of finding the min-cut, which also makes sense given that the probability decreases for every iteration of the algorithm, as more and more elements are merged.

4.4 PDF 1.3

Consider a Monte Carlo algorithm A for a problem Π whose expected running time is at most $T(n)$ on any instance of size n and that produces a correct solution with probability $\gamma(n)$. Suppose further that given a solution to Π, we can verify its correctness in time $t(n)$. Show how to obtain a Las Vegas algorithm that always gives a correct answer to Π and runs in expected time at most $\frac{(T(n)+(tn))}{\gamma(n)}$.

Answer:

We assume $T(n)$ is the worst case runtime of A. We also assume we are only going to explain how to find said solution and not write any detailed pseudocode or calculations.

Given the properties of Monte Carlo and Las Vegas algorithms, MC algorithms always solves problems in an expected time $T(n)$ with probability $\gamma(n)$ of returning a correct solution. LV algorithms always return the correct solution in average time $\frac{T(n)}{\gamma(n)} + t(n)$, where $t(n)$ is the time it takes to verify the solution.

First to guarantee a correct solution, we will let the algorithm run forever, as doing this will eventually lead to a correct solution, which eliminates the chance of ever returning a wrong solution. If the algorithm returns a solution, it will be verified. If the solution is wrong, it will discard the solution and retry again, continuing the loop forever until a correct solution is found.

The probability of a correct solution being found for a single iteration of the algorithm is always $\gamma(n)$ and the expected amount of iterations to obtain a correct solution is $\frac{1}{\gamma(n)}$, as for each iteration we run the MC algorithm A in time $T(n)$ and verify the solution in time $t(n)$. The expected time for each iteration is thus $T(n) + t(n)$.

The total expected time is thus:

$$ET = \frac{(T(n) + (t(n))}{\gamma(n)}$$

or in words, the product of the expected amount of iteration and the expected time for each iteration.