Advanced Algorithms and Data structures
Assignment two

Magnus Goltermann (xzb187), Thomas Busk-Jepsen (tnr653)
Mia Rahlff Pedersen(bvx284)

December 17, 2024

1 A

1.1 29.1-9

To give an example of a linear program where the feasible region is not bounded but the optimal objective valie is finite. We condider optimizing $x_1 + x_2$ under the constraints $x_1 \geq x_2$, $-x_1 + 2 \geq x_2$ and $x_1 \leq 1$. Here we see that the optimal point is the vertex (1.1), the graph can be seen below.

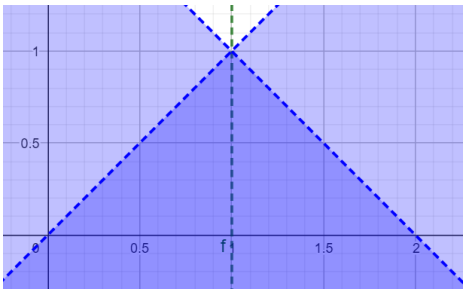

Figure 1: Example of non bounded linear program

1.2 29.2-3

The linear program for finding the shortest path between just two vertices are given as

$$\begin{aligned} &\text{maximize} \quad d_t \\ &\text{subject to} \\ &\quad d_v \leq d_u + w(u,v) \quad \text{for each edge } (u,v) \in E, \\ &\quad d_s = 0. \end{aligned} \tag{1}$$

Then doing so for vertices must mean just taking the sum of all d_t as

$$\begin{aligned} &\text{maximize} \sum_{v \in V} \quad d_t \\ &\text{subject to} \\ &\quad d_v \leq d_u + w(u,v) \quad \text{for each edge } (u,v) \in E, \\ &\quad d_s = 0. \end{aligned} \tag{2}$$

1.3 29.2-6

If we look at the maximum bipartite problem as a flow problem, where we assign a source that goes into all vertices in L and a sink that goes into all vertices in R where all weights/capacities are 1, as this will constrain the amount of edges going from each vertex in L to 1, and going into each vertex in R as 1, thus keeping the constraints of the maximum bipartite problem. Then the solution of the maximum bipartite problem can just be seen as the maximum flow. Thus defining the linear program as

$$\begin{array}{ll} \max & \sum_{v \in L} f_{sv} \\ \text{s.t} & f_{uv} \leq 1 \text{ for each } u, v \in \{s\} \cup G \cup \{t\} = V \\ & \sum_{v \in V} f_{vu} = \sum_{v \in V} f_{uv} \text{ for each } u \in G \\ & f_{uv} \geq 0 \text{ for each } u, v \in V \end{array}$$

1.4 29.4-3

The primal maximum-flow linear program is defined as

$$\begin{aligned} &\text{maximize} \sum_{v \in V} f_{sv} - \sum_{v \in V} f_{vs}, \\ &\text{subject to:} \\ &f_{uv} \leq c(u,v), \quad \text{for each } u, v \in V, \\ &\sum_{v \in V} f_{vu} = \sum_{v \in V} f_{uv}, \quad \text{for each } u \in V - \{s,t\}, \\ &f_{uv} \geq 0, \quad \text{for each } u, v \in V \end{aligned}$$

We need to start of by writing this in standard form, as there is an equality constraint in this, that violates the standard form. This gives us

$$\begin{aligned}
&\text{maximize} \sum_{v \in V} f_{sv} - \sum_{v \in V} f_{vs}, \\
&\text{subject to:} \\
&f_{uv} \leq c(u,v), \quad \text{for each } u, v \in V, \\
&\sum_{v \in V} f_{vu} \leq \sum_{v \in V} f_{uv}, \quad \text{for each } u \in V - \{s,t\}, \\
&\sum_{v \in V} f_{vu} \geq \sum_{v \in V} f_{uv}, \quad \text{for each } u \in V - \{s,t\}, \\
&f_{uv} \geq 0, \quad \text{for each } u, v \in V
\end{aligned}$$

However, now we need to negate the greater-than-or-equal, to obtain less-than-or-equal. Giving us

$$\begin{aligned}
&\text{maximize} \sum_{v \in V} f_{sv} - \sum_{v \in V} f_{vs}, \\
&\text{subject to:} \\
&f_{uv} \leq c(u,v), \quad \text{for each } u, v \in V, \\
&\sum_{v \in V} f_{vu} \leq \sum_{v \in V} f_{uv}, \quad \text{for each } u \in V - \{s,t\}, \\
&-\sum_{v \in V} f_{vu} \leq -\sum_{v \in V} f_{uv}, \quad \text{for each } u \in V - \{s,t\}, \\
&f_{uv} \geq 0, \quad \text{for each } u, v \in V
\end{aligned}$$

We then bring variables, from the right-hand side to the left hand, to avoid having variables on the right.

$$\begin{aligned}
&\text{maximize} \sum_{v \in V} f_{sv} - \sum_{v \in V} f_{vs}, \\
&\text{subject to:} \\
&f_{uv} \leq c(u,v), \quad \text{for each } u, v \in V, \\
&\sum_{v \in V} f_{vu} - \sum_{v \in V} f_{uv} \leq 0, \\
&-\sum_{v \in V} f_{vu} + \sum_{v \in V} f_{uv} \leq 0, \\
&f_{uv} \geq 0, \quad \text{for each } u, v \in V
\end{aligned}$$

Now to turn this into the dual, we need to make this a minimization problem and exchange the roles of coefficients on the right-hand sides and the objective

function, and replace each less-than-or-equal-to by a greater-than-or-equal-to.

$$\begin{aligned}
&\text{minimize} \sum_{v \in V} y_{uv} \cdot c(u,v), \\
&\text{subject to:} \\
&y_{uv} - z_u + z_v \geq 0, \\
&z_s - z_t \geq 1, \\
&f_{uv} \geq 0, \quad \text{for each } u, v \in V
\end{aligned}$$

As y_{uv} is the dual variable associated with the capacity constraint, and $c(u,v)$ is the coefficient on the right hand side, put in the objective function. z corresponds to the flow conservation.

As for the interpretation of this formulation as a minimum-cut problem, one can see that z is not in the objective function, rather it is in the constraints as the difference between z values.

2 B

To get a 5-sided polygon to have a full edge as a solution as 1, we can define the linear program as

$$\text{Max } x_1 + x_2$$

st.

$$\begin{aligned}
x_1 + x_2 &\leq 1 \\
x_1, x_2 &\leq 1 \\
x_1, x_2 &\geq -1
\end{aligned}$$

Here we have that the line $x_1 + x_2 = 1$ are all solutions to this.

3 C

We start of by letting U_1 and U_2 represent $|f(1)|$ and $|f(2)|$ respectively. We also let x_1 and x_2 represent $f(1)$ and f(2) respectively. We can then form our linear program as follows

$$\begin{aligned}
&\text{Max.} \, U_1 + U_2, \\
&\text{s.t} \\
&x_1 - U_1 \leq 0, \\
&-x_1 - U_1 \leq 0, \\
&x_2 - U_2 \leq 0, \\
&-x_2 - U_2 \leq 0, \\
&U_1, U_2 \geq 0
\end{aligned}$$

This satisfy the requirement that all the entries in the matrix A and in the vectors b,c must be affine functions in f(1) and f(2). As it is a sum of linear terms, with no non-linear operations.

This primal linear program can be formed into the dual, by exchanging the roles of coefficients on the right-hand sides and the objective function, and replace each less-than-or-equal-to by a greater-than-or-equal-to, and also turn it into a minimization problem.

$$\begin{aligned}
&\text{Min. } y_1 - y_2 + y_3 - y_4, \\
&\text{s.t} \\
&-y_1 + y_2 \geq 1, \\
&-y_3 + y_4 \geq 1, \\
&y_1, y_2, y_3, y_4 \geq 0
\end{aligned}$$

4 Randomized algorithms

4.1 *

To find a function $f(n)$ such that the expected depth is bound by that function by just a magnitude, we set up the equation as

$$c_1 f(n) \leq \mathbb{E}[d(x)] \leq c_2 f(n) \tag{3}$$

To figure out at what depth we expect to find x, it will be the depth at which we on expectation expect to have x as the pivot. At every pivot, there is a $\frac{1}{s}$ chance that x is the pivot, and since the array is on average halved at reach recursive call we get the depth to be logarithmic of n, and thus the equation can be rewritten into

$$c_1 log(n) \leq \mathbb{E}[d(x)] \leq c_2 log(n) \tag{4}$$

and thus $\mathbb{E}[d(x)] = \Theta(log(n))$

In randomized quicksort, each pivot is chosen uniformly at random from the elements currently under consideration. Over the course of the algorithm, each element x gets "partitioned out" of the input when a pivot that splits its position off from the rest is chosen. The expected number of pivots chosen before x is isolated (and thus the depth at which x ends up in). First we set up an indicator variable Y_{ij} to be

$$Y_{ij} = \begin{cases} 1 & \text{if } x(j) \text{ is the pivot while } x(i) \text{ is in the subproblem} \\ 0 & \text{otherwise} \end{cases}$$

Where $x(i)$ is the x we are "looking" for, and thus Y_{ij} is 1 whenever $x(i)$ is still in the subproblem, and thus the depth can be seen as the sum of Y_{ij} over all

the pivots:

$$d(x(i)) = \sum_{\substack{j=1 \\ j \neq i}}^{n} Y_{ij}$$

And thus we just take the expectation of that to get the expected depth to be

$$\mathbb{E}[d(x(i))] = \sum_{\substack{j=1 \\ j \neq i}}^{n} \mathbb{E}[Y_{ij}]$$

And since its the expectation of an indicator variable, it is just the probability of it being 1, and thus:

$$\mathbb{E}[Y_{ij}] = P(Y_{ij} = 1) = \frac{1}{|S_{ij}|} = \frac{1}{|j - i| + 1}$$

which is the probability of the first pivot being $x(j)$ while $x(i)$ is kept in the subproblem, and since its uniformly at random we get it to be one over the size of the set. Putting that back into the sum:

$$\mathbb{E}[d(x(i))] = \sum_{\substack{j=1 \\ j \neq i}}^{n} \mathbb{E}[Y_{ij}] = \sum_{\substack{j=1 \\ j \neq i}}^{n} \frac{1}{|j - i| + 1}$$

Which is a harmonic series that grows like $ln(n)$ we get a bound of the depth to be

$$\mathbb{E}[d(x(i))] = \Theta(\log n)$$

4.2 **

To determine how many runs of randomized contractions we need to be 99% sure that a minimum cut is found we first look at the probability that a single run will find the min cut. As given in the lecture that is:

$$\frac{2}{n(n-1)}$$

This means that the probability of not getting the correct result is 1-$\frac{2}{n(n-1)}$. That means that after k independent runs the chance of failure is:

$$Pr_k = (1 - \frac{2}{n(n-1)})^k$$

We then want the probability of failure to be 0.01% to get the 99% interval, giving us:

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

Simply rearranging this and solving for k we get:

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

4.3 1.2

To show that there are inputs for which this algorithm finds a min cut is exponentially small. We will consider a graph where every vertex is connected to multiple other vertexes, except for one vertex that is only connected to one other vertex. This edge that connects the sole vertex we will call $\{v_s, v_t\}$. To begin with there are n vertexes, as the algorithm choses two vertexes at random we can describe the probability of not merging v_s as following:

$$\frac{\binom{n-1}{2}}{\binom{n}{2}} = \frac{(n-1)(n-2)}{n(n-1)} = \frac{n-2}{n}$$

As we keep doing this until we only have two vertexes left we can describe the chances of having the correct edge as following:

$$\prod_{k=2}^{n-1} \frac{k-1}{k}$$

Simplified into

$$\frac{1}{n-1}$$

Meaning that for large n we have an incredible low chance of succeeding with this method.

To show that the probability of finding a minimum cut is exponentially small with the modified algorithm that takes two vertices at random to coalesce at eatch step, we will consider the following graph.

The graph consists of two complete graphs K_n and K_n only connected by a single edge $\{u, v\}$. Here we see that the left cluster K_n has the vertices $\{u, a_1, a_2, ..., a_{n-1}\}$ and the right cluster has the vertexes $\{v, b_1, b_2, ..., b_{n-1}\}$, and then there are a single edge $\{u, v\}$ connecting the two clusters.

It is clear to see that the minimum edge is the edge $\{u, v\}$, so any algorithm that is successful must end the contraction process with u and v as the final two vertices.

We see that with our graph we have $2n$ vertices (n vertices in each cluster). Therefore we know that the total number of possible vertex pairs are $\binom{2n}{2} = n(2n-1)$.

The safe pairs are the ones that belong in the same cluster, either the vertexes a or b, there fore the number of safe pairs becomes:

$$\binom{n-1}{2} + \binom{n-1}{2} = 2 \cdot \frac{(n-1)(n-2)}{2} = (n-1)(n-2).$$

As for large n we get that the possibilities are approximately $2n^2$ and the safe pairs are approximately n^2, we get that the probability that we pick a safe pair is $\frac{1}{2}$.

Since we then need to perform $2(n-1)$ contractions we then get that the probability of picking a safe pair is

$$\left(\frac{1}{2}\right)^{2(n-1)}$$

Meaning that the probability of finding a minimum cut decays exponentially with the amount of vertexes.

4.4 1.3

The Las Vegas Algorithm runs the Monte Carlo Algorithm until a correct output is found. We define p to be a random variable representing the number of iterations required to produce a correct output. Since each run of the Monte Carlo Algorithm is independent and succeed with probability $\gamma(n)$, p follows a geometric distribution with success probability $\gamma(n)$. The expected value for a geometric random variable is

$$\mathbb{E}[p] = \frac{1}{\gamma(n)}$$

We know that the Monte Carlo Algorithm runs in $T(n) + t(n)$ time. The Las Vegas Algorithm will thereby run in $(T(n) + t(n)) \cdot \mathbb{E}[p]$ time, as it will run Monte Carlo until it is correct, which it will be at the expected value. As we know $\mathbb{E}[p] = \frac{1}{\gamma(n)}$, we can write it as

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

. Thus we have obtained a Las Vegas Algorithm that always give a correct answer to Π and runs in the expected time at most

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