CS 467/567: Linear Programming

Stefan D. Bruda

Winter 2023

- Linear function: $f(x_1, x_2, \ldots x_n) = \sum_{j-1}^{n} a_j x_j$
- Linear constraint: $g(x_1, \ldots, x_n) \bullet b$ for some linear function g and either $\bullet ==$ (linear equality) or $\bullet \in \{\leqslant, \geqslant\}$ (linear inequalities)
- A linear programming problem is the problem of optimizing (minimizing or maximizing) a linear function subject to a finite set of linear constraints; an instance of this problem is a linear program

- Linear function: $f(x_1, x_2, \ldots x_n) = \sum_{j-1}^{n} a_j x_j$
- Linear constraint: $g(x_1, \ldots, x_n) \bullet b$ for some linear function g and either $\bullet ==$ (linear equality) or $\bullet \in \{\leqslant, \geqslant\}$ (linear inequalities)
- A linear programming problem is the problem of optimizing (minimizing or maximizing) a linear function subject to a finite set of linear constraints; an instance of this problem is a linear program
- Solving 2-variable optimization problems:
  - The linear constraints form a convex region in the (x_1, x_2)-Cartesian coordinate system (the simplex)
  - The set of points $f(x_1, x_2) = z$ form a line whose slope is independent of z
  - The goal becomes finding the optimal (maximal/minimal) z with a non-empty intersection between the simplex and the line, which always corresponds to a vertex of the simplex if the simplex is bounded
  - The simplex algorithm starts form an arbitrary vertex of the simplex, keeps moving to a neighbor whose value is no smaller/larger than that of the current vertex
  - The algorithm terminates when it reaches a local maximum/minimum (a vertex with all the neighbors having a smaller/larger objective value)
  - This is also the global maximum/minimum

- Linear function: $f(x_1, x_2, \ldots x_n) = \sum_{j-1}^{n} a_j x_j$
- Linear constraint: $g(x_1, \ldots, x_n) \bullet b$ for some linear function g and either $\bullet ==$ (linear equality) or $\bullet \in \{\leqslant, \geqslant\}$ (linear inequalities)
- A linear programming problem is the problem of optimizing (minimizing or maximizing) a linear function subject to a finite set of linear constraints; an instance of this problem is a linear program
- Solving 2-variable optimization problems:
  - The linear constraints form a convex region in the (x_1, x_2)-Cartesian coordinate system (the simplex)
  - The set of points $f(x_1, x_2) = z$ form a line whose slope is independent of z
  - The goal becomes finding the optimal (maximal/minimal) z with a non-empty intersection between the simplex and the line, which always corresponds to a vertex of the simplex if the simplex is bounded
  - The simplex algorithm starts form an arbitrary vertex of the simplex, keeps moving to a neighbor whose value is no smaller/larger than that of the current vertex
  - The algorithm terminates when it reaches a local maximum/minimum (a vertex with all the neighbors having a smaller/larger objective value)
  - This is also the global maximum/minimum because the simplex is convex
- Idea can be trivially generalized to an n-dimensional spaces

THE STANDARD FORMS OF LINEAR PROGRAMS

- Standard form: Given n real numbers c_i, $1 \leqslant i \leqslant n$ and mn real numbers a_{ij}, $1 \leqslant i \leqslant m$, $1 \leqslant j \leqslant n$, find n real numbers x_i, $1 \leqslant i \leqslant n$ that:
  - maximize the objective function $\sum_{j=1}^{n} c_j x_j$
  - subject to the constraints $\sum_{j=1}^{n} a_{ij} x_j \leqslant b_i$, $1 \leqslant i \leqslant m$ and $x_j \geqslant 0$, $1 \leqslant i \leqslant n$
- Slack form: all constraints are either equality constraints or $x_j \geqslant 0$
- Terminology for linear programs:
  - An assignment $\overline{x}$ of the variables x_i, $1 \leqslant i \leqslant n$ can be a feasible solution (satisfies all constraints) or an infeasible solution (violates at least one constraint)
  - A solution $\overline{x}$ has the objective value $c^T\overline{x}$
  - A solution $\overline{x}$ whose $c^T\overline{x}$ is the maximum of all the feasible solutions is an optimal solution and its $c^T\overline{x}$ is the optimal objective value
  - If a linear program does not have any feasible solutions then it is infeasible, otherwise it is feasible
  - If a linear program does not have a finite optimal objective value then it is unbounded

- Conversion to standard form:
  - To convert a minimization instance into a maximization instance: negate all the coefficients in the objective function
  - If some variable x_j does not have the constraint $x_j \geqslant 0$:
    - Replace every occurrence of x_j with $x'_j - x''_j$
    - Add the constraints $x'_j \geqslant 0$ and $x''_j \geqslant 0$
  - If we have an equality constraint $g(x_1, \ldots, x_n) = b$: replace it with two constraints $g(x_1, \ldots, x_n) \geqslant b$ and $g(x_1, \ldots, x_n) \leqslant b$
  - If we have a constraint $g(x_1, \ldots, x_n) \geqslant b$: we multiply both sides with (-1) (and we flip the comparison, and we distribute the (-1) into the sum)

- Conversion to standard form:
  - To convert a minimization instance into a maximization instance: negate all the coefficients in the objective function
  - If some variable x_j does not have the constraint $x_j \geqslant 0$:
    - Replace every occurrence of x_j with $x_j' - x_j''$
    - Add the constraints $x_j' \geqslant 0$ and $x_j'' \geqslant 0$
  - If we have an equality constraint $g(x_1, \ldots, x_n) = b$: replace it with two constraints $g(x_1, \ldots, x_n) \geqslant b$ and $g(x_1, \ldots, x_n) \leqslant b$
  - If we have a constraint $g(x_1, \ldots, x_n) \geqslant b$: we multiply both sides with (-1) (and we flip the comparison, and we distribute the (-1) into the sum)
- Conversion of standard form into slack form:
  - Each constraint $\sum_{j=1}^{n} a_{ij} x_j \leqslant b_i$ is rewritten as the two constraints

$$s = b_i - \sum_{j=1}^{n} a_{ij} x_j \qquad\qquad s \geqslant 0$$

  - s is a new variable (slack or basic variable as opposed to a nonbasic variable)…that just does not happen to appear in the objective function

- Problem: Given a graph $G = (V, E)$ with a weight function $w : E \to \mathbb{R}$, a source vertex $s \in V$ and a target vertex $t \in V$, find d_t, the weight of the shortest path from s to t
- Linear program: maximize d_t subject to the following constraints:

$$d_s = 0 \qquad\qquad d_v \leqslant d_u + w(u, v) \text{ for each } (u, v) \in E$$

  - d_u is the weight of the path from s to u

- Problem: Given a graph $G = (V, E)$ with a weight function $w : E \to \mathbb{R}$, a source vertex $s \in V$ and a target vertex $t \in V$, find d_t, the weight of the shortest path from s to t
- Linear program: maximize d_t subject to the following constraints:

$$d_s = 0 \qquad\qquad d_v \leqslant d_u + w(u, v) \text{ for each } (u, v) \in E$$

  - d_u is the weight of the path from s to u
  - Note: $\overline{d_v} = \min_{u \text{ s.t. } (u,v)\in E}\{\overline{d_u} + w(u, v)\}$ so $\overline{d_v}$ is the maximal value smaller than all the values in $\{\overline{d_u} + w(u, v)\}$ so we need to maximize $\overline{d_v}$ (the minimization nature of the problem is given by the constraints)

- Flow network: graph $G = (V, E)$ with a capacity function $c : E \to \mathbb{R}^+$ and two designated vertices $s, t \in V$ such that $(u, v) \in E \Rightarrow (v, u) \notin E$
  - Convenient abuse of notation: $c(u, v) = 0$ whenever $(u, v) \notin E$
- Flow in G: function $f : V \times V \to \mathbb{R}$ satisfying the following constraints
  - Capacity constraint: $0 \leqslant f(u, v) \leqslant c(u, v)$ for all $u, v \in V$
  - Flow conservation: $\sum_{v \in V} f(v, u) = \sum_{v \in V} f(u, v)$ for all $u \in V \setminus \{s, t\}$
- Problem: Given a flow network, find a flow that maximizes $f(s, t)$
- Linear program: maximize $\sum_{v \in V} f_{sv} - \sum_{v \in V} f_{vs}$ subject to the following constraints:

$$\begin{aligned} f_{uv} &\leqslant c(u, v) && \text{for each} && u, v \in V \\ \sum_{v \in V} f_{vu} &= \sum_{v \in V} f_{uv} && \text{for each} && u \in V \setminus \{s, t\} \\ f_{uv} &\geqslant 0 && \text{for each} && u, v \in V \end{aligned}$$

- Input: a flow network $G = (V, E)$ with capacity function c
- Additional input: Commodities $K_1, K_2, \ldots, K_k$
  - $K_i = (s_i, t_i, d_i)$ where s_i/t_i are the source/target vertices for K_i, and d_i is the demand (desired flow value) for K_i
  - Aggregate flow: $f_{uv} = \sum_{i=1}^{k} f_{iuv}$, where f_{iuv} is the flow for K_i from u to v
- Problem: Given a flow network and k commodities, determine whether there exists an aggregate flow f such that $f_{uv} \leqslant c(u, v)$ for all $(u, v) \in E$

Multicommodity flow

- Input: a flow network $G = (V, E)$ with capacity function c
- Additional input: Commodities $K_1, K_2, \ldots, K_k$
  - $K_i = (s_i, t_i, d_i)$ where s_i/t_i are the source/target vertices for K_i, and d_i is the demand (desired flow value) for K_i
  - Aggregate flow: $f_{uv} = \sum_{i=1}^{k} f_{iuv}$, where f_{iuv} is the flow for K_i from u to v
- Problem: Given a flow network and k commodities, determine whether there exists an aggregate flow f such that $f_{uv} \leqslant c(u, v)$ for all $(u, v) \in E$
- Linear program: maximize 0 subject to the following constraints:

$$\begin{aligned}
\sum_{i=1}^{k} f_{iuv} \leqslant c(u,v) &\quad \text{for each} \quad u, v \in V \\
\sum_{v \in V} f_{iuv} - \sum_{\in V} f_{ivu} = 0 &\quad \text{for each} \quad u \in V \setminus \{s, t\} \\
\sum_{v \in V} f_{is_iv} = \sum_{v \in V} f_{ivs_i} = d_i &\quad \text{for each} \quad 1 \leqslant i \leqslant k \\
f_{iuv} \geqslant 0 &\quad \text{for each} \quad u, v \in V \text{ and } 1 \leqslant i \leqslant k
\end{aligned}$$

- Solving multicommodity flow as a linear programming problem is the only known efficient algorithm for this problem

- A slack form is a tuple (N, B, A, b, c, ν) which denotes the following linear program

$$\begin{aligned} z &= \nu + \sum_{j \in N} c_j x_j \\ x_i &= b_i - \sum_{j \in N} a_{ij} x_j \qquad \text{for } i \in B \end{aligned}$$

  with the implicit understanding that $x_i \geqslant 0$ for all $i \in N \cup B$
  - N contains the indices of all nonbasic variables, $|N| = n$
  - B contains the indices of all basic variable, $|B| = m$
  - $N \cup B = \{1, 2, \ldots, n + m\}$

One iteration = pivot operation

1. Set all the nonbasic variables to 0, solve for the basic variables = basic solution
2. Select a nonbasic variable x_e with positive coefficient (entering variable) in the objective function and increase its value as much as possible without constraint violation
3. The above increase makes one basic variable x_l zero (the leaving variable)
4. Reformulate the constraints such that x_e becomes basic and x_l nonbasic
5. If all the coefficients c_j are negative then the current basic solution is the optimal solution, otherwise repeat from Step 1

Algorithm PIVOT$((N, B, A, b, c, \nu), l, e)$ **returns** $(N', B', A', b', c', \nu')$

- Compute the coefficients of the equation for new basic variable x_e:
  1. $b'_e \leftarrow b_l / a_{le}$
  2. **for each** $j \in N \setminus \{e\}$ **do** $a'_{ej} \leftarrow a_{lj} / a_{le}$
  3. $a'_{el} \leftarrow 1 / a_{le}$
- Compute the coefficients of the remaining constraints:
  1. **for each** $i \in B \setminus \{l\}$ **do**
     1. $b'_i \leftarrow b_i - a_{ie} b'_e$
     2. **for each** $j \in N \setminus \{e\}$ **do** $a'_{ij} \leftarrow a_{ij} - a_{ie} a'_{ej}$
     3. $a'_{il} \leftarrow -a_{ie} a'_{el}$
- Compute the objective function:
  1. $\nu' \leftarrow \nu + c_e b'_e$
  2. **for each** $j \in N \setminus \{e\}$ **do** $c'_j \leftarrow c_j - c_e a'_{ej}$
  3. $c'_l \leftarrow -c_e a'_{el}$
- Compute the new basic and nonbasic variables:
  1. $N' \leftarrow N \setminus \{e\} \cup \{l\}$
  2. $B' \leftarrow B \setminus \{l\} \cup \{e\}$

Algorithm SIMPLEX(A, b, c) **returns** $(\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_n)$

1. $(N, B, A, b, c, \nu) \leftarrow$ INITIALIZE-SIMPLEX(A, b, c)
2. let Δ be a vector of size n
3. **while** some index $j \in N$ has $c_j > 0$
    1. choose an index $e \in N$ such that $c_e > 0$
    2. **for each** $i \in B$ **do**
       if $a_{ie} > 0$ **then** $\Delta_i \leftarrow b_i/a_{ie}$ **else** $\Delta_i \leftarrow \infty$
    3. choose $l \in B$ such that Δ_l is minimum over Δ_i
    4. **if** $\Delta_l = \infty$ **then return** "unbounded"
    5. **else** $(N, B, A, b, c, \nu) \leftarrow$ PIVOT$((N, B, A, b, c, \nu), l, e)$
4. **for** $i \leftarrow 1$ **to** n **do**
    1. **if** $i \in B$ **then** $\overline{x}_i \leftarrow b_i$ **else** $\overline{x}_i \leftarrow 0$

- Input is a linear program in standard form
- INITIALIZE-SIMPLEX returns a slack form for which the initial basic solution is feasible (or a suitable message if the linear program is infeasible)

THE INITIALIZE-SIMPLEX ALGORITHM

Lemma

Let L be a program in standard form and x_0 a new variable. Let L_{aux} be:

Maximize $-x_0$
subject to $\sum_{j=1}^{n} a_{ij}x_j - x_0 \leqslant b_i$ *for* $1 \leqslant i \leqslant m$ *and* $x_j \geqslant 0$ *for* $0 \leqslant j \leqslant n$

Then L is feasible iff the optimal objective value for L_{aux} is 0.

INITIALIZE-SIMPLEX then works as follows:

- Let b_k be the minimum b_i
- If $b_k \geqslant 0$ then the initial solution is feasible so convert to slack and return
- Form L_{aux} as in the lemma, convert it to the slack form (N, B, A, b, c, ν)
- Let $l = n + k$ and perform PIVOT((N, B, A, b, c, ν), l, 0); the basic solution is now feasible for L_{aux}
- Use the while loop of SIMPLEX to find an optimal solution for L_{aux}; return "infeasible" if $\overline{x}_0 \neq 0$
- Remove x_0 from the constraints, restore the original objective function for L, but replace basic variables with the right hand side of its constraint
- Return this final slack form

PROPERTIES OF THE SIMPLEX ALGORITHM

Theorem

If SIMPLEX *fails to terminate in at most* $\binom{n+m}{m}$ *iterations then it cycles.*

So SIMPLEX *either reports that the linear program is unbounded or terminates with a feasible solution in at most* $\binom{n+m}{m}$ *iterations*

Theorem (Fundamental theorem of linear programming)

Any linear program L given in standard form either:

1. *has an optimal solution with a finite objective value,*
2. *is infeasible, or*
3. *is unbounded.*

If L is infeasible or unbounded, then SIMPLEX *returns "infeasible" or "unbounded", respectively. Otherwise* SIMPLEX *returns an optimal solution with a finite value.*