SAPIENZA
UNIVERSITÀ DI ROMA

"Sapienza" University of Rome
Faculty of Information Engineering,
Informatics and Statistics
Department of Computer Science

Optimization

Lecture notes integrated with the book "Understanding and Using Linear Programming.",
J. Matoušek, B. Gärtner

Author
Simone Bianco

March 19, 2024

Contents

Information and Contacts

Personal notes and summaries collected as part of the *Optimization* course offered by the degree in Computer Science of the University of Rome "La Sapienza".

Further information and notes can be found at the following link:
https://github.com/Exyss/university-notes. Anyone can feel free to report inaccuracies, improvements or requests through the Issue system provided by GitHub itself or by contacting the author privately:

- Email: **bianco.simone@outlook.it**
- LinkedIn: **Simone Bianco**

The notes are constantly being updated, so please check if the changes have already been made in the most recent version.

Suggested prerequisites:

Preventive learning of material related to the *Linear Algebra* and *Algorithms 2* course is recommended

Licence:

1 Flow networks

1.1 Networks and flows

Definition 1: Graph

A **graph** is a mathematical structure $G = (V, E)$, where V is the set of **vertices** in G and $E \subseteq V \times V$ is the set of **edges** that link two vertices in G.

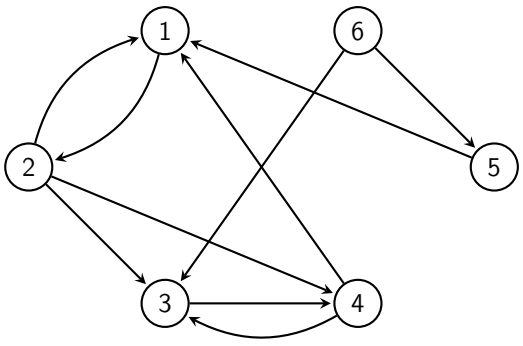
We will assume that each graph is *simple*, meaning that there are no multiple edges between the same nodes and no loops (that being an edge from a vertex to itself).

From now on, we will assume that $|V(G)| = n$ and $|E(G)| = m$.

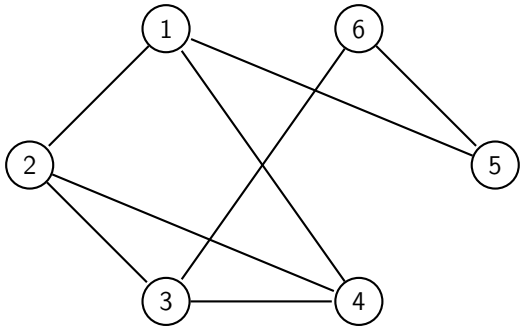
A graph can be **directed** or **undirected**. In a directed graph we consider the edges $(u, v) \in E(G)$ and $(v, u) \in E(G)$ as two different edges, while in an undirected graph they represent the same edge.

Example:

Directed graph



Undirected graph

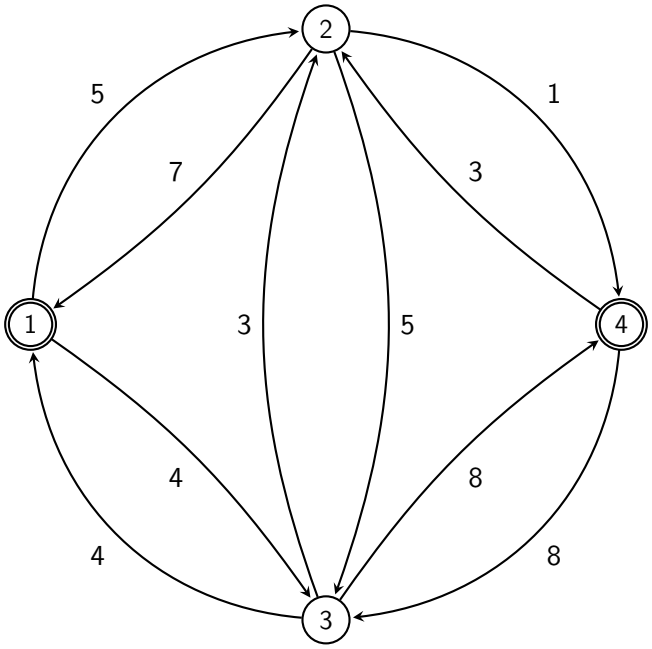


Definition 2: Network

A **network** is a mathematical structure $N = (G, s, t, c)$ where:

- $G = (V, E)$ is a directed graph
- s and t are two vertices of G ($s, t \in V(G)$), respectively called the **source** and the **sink**
- $c : E(G) \to \mathbb{R}^+$ is a weight function on the edges called **capacity**
- $(u, v) \in E(G) \implies (v, u) \in E(G)$

Example:



The numbers on the edges represent the capacities of the edges, while the nodes 1 and 4 are chosen respectively as the source and the sink of the network

Essentially, a network effectively describes a *water system* made up of *pipes* (the edges) that can transport a maximum amount of fluid inside them (the capacity of the edges). In fact, the last property of a network defines the idea of a bi-directional *flow of water* inside the pipes. In particular, we note that each pipe can have a different capacity based on the direction of the flow.

Definition 3: Flow

Given a network $N = (G, s, t, c)$, a **flow** is a weight function $f : E(G) \to \mathbb{R}$ on the edges defined by the following properties:

- **Skew-symmetric**: $\forall (u, v) \in E(G) \quad f(u, v) = -f(v, u)$, meaning that the incoming flow in an edge is the inverse of the outgoing flow in the corresponding opposite edge
- **Capacity bounded**: $\forall (u, v) \in E(G) \quad f(u, v) \leq c(u, v)$, meaning that the flow can't be greater than the supported capacity
- **Conservation of flow**: $\forall v \in V(G) - \{s, t\}$ it holds that

$$\sum_{\substack{(u,v)\in E(G):\\ f(u,v)>0}} f(u, v) = \sum_{\substack{(v,w)\in E(G):\\ f(v,w)>0}} f(v, w)$$

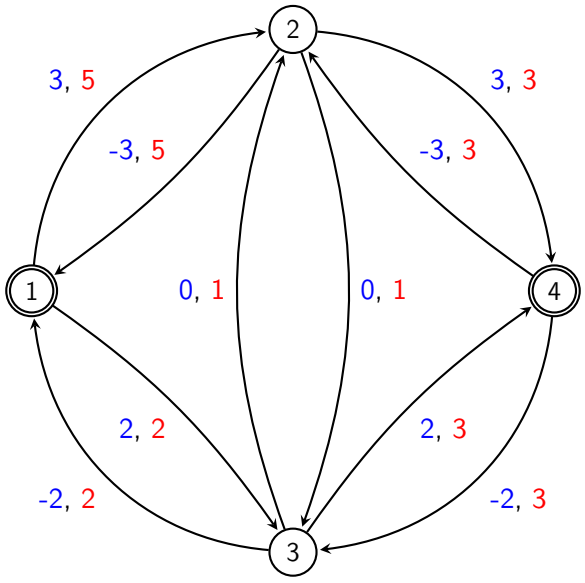
  meaning that the incoming flow in v is the same as the outgoing flow from v

Definition 4: Flow value

Given a network $N = (G, s, t, c)$ and a flow f on N, we define the **value of** f, noted by $\text{val}(f)$, as the sum of the flow outgoing from the source s or the sum of the flow incoming to the sink t:

$$\text{val}(f) := \sum_{(s,u)\in E(G)} f(s, u) = \sum_{(v,t)\in E(G)} f(v, t)$$

Example:



For each edge, the numbers in blue and red respectively represent its flow and its capacity. The flow value of the given flow is 5.

Observation 1: Nullification of flow for middle edges

Given a network $N = (G, s, t, c)$ and a flow f defined on G, for each vertex different from s and t it holds that:

$$\sum_{\substack{(u,v)\in E(G):\\ f(u,v)>0}} f(u,v) = \sum_{\substack{(v,w)\in E(G):\\ f(v,w)>0}} f(v,w) \iff \sum_{(u,v)\in E(G)} f(u,v) = 0$$

Proof.

Due to the conservation of flow and the skew-symmetric properties, for all nodes $v \neq s, t$ we know that:

$$\sum_{\substack{(u,v)\in E(G):\\ f(u,v)>0}} f(u,v) = \sum_{\substack{(v,w)\in E(G):\\ f(v,w)>0}} f(v,w) = \sum_{\substack{(v,w)\in E(G):\\ f(v,w)>0}} -f(w,v)$$

which is possible if only if:

$$\sum_{\substack{(u,v)\in E(G):\\ f(u,v)>0}} f(u,v) = - \sum_{\substack{(v,w)\in E(G):\\ f(v,w)>0}} f(w,v) \iff$$

$$\sum_{\substack{(u,v)\in E(G):\\ f(u,v)>0}} f(u,v) + \sum_{\substack{(v,w)\in E(G):\\ f(v,w)>0}} f(w,v) = 0$$

Again, by the skew-symmetric property we get that:

$$\sum_{\substack{(u,v)\in E(G):\\ f(u,v)>0}} f(u,v) + \sum_{\substack{(v,w)\in E(G):\\ f(v,w)>0}} f(w,v) = 0 \iff$$

$$\sum_{\substack{(u,v)\in E(G):\\ f(u,v)>0}} f(u,v) + \sum_{\substack{(w,v)\in E(G):\\ f(w,v)<0}} f(w,v) = 0$$

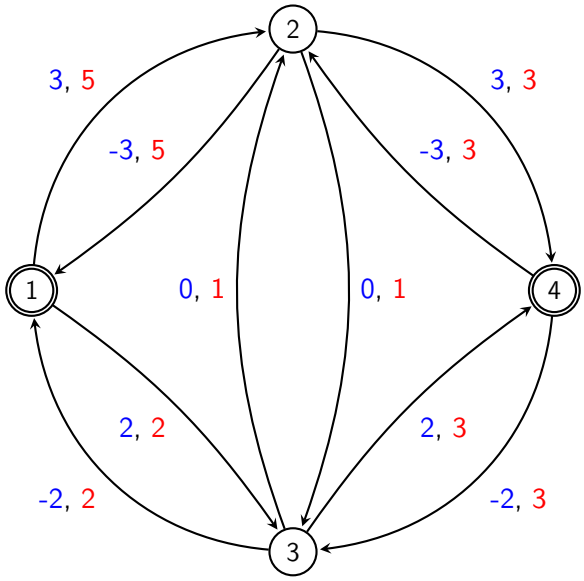
Then, by adding each edge incoming in v that has no flow we conclude that:

$$\sum_{\substack{(u,v)\in E(G):\\ f(u,v)>0}} f(u,v) + \sum_{\substack{(w,v)\in E(G):\\ f(w,v)<0}} f(w,v) + \sum_{\substack{(x,v)\in E(G):\\ f(x,v)=0}} f(x,v) = 0 \iff \sum_{(u,v)\in E(G)} f(u,v) = 0$$

□

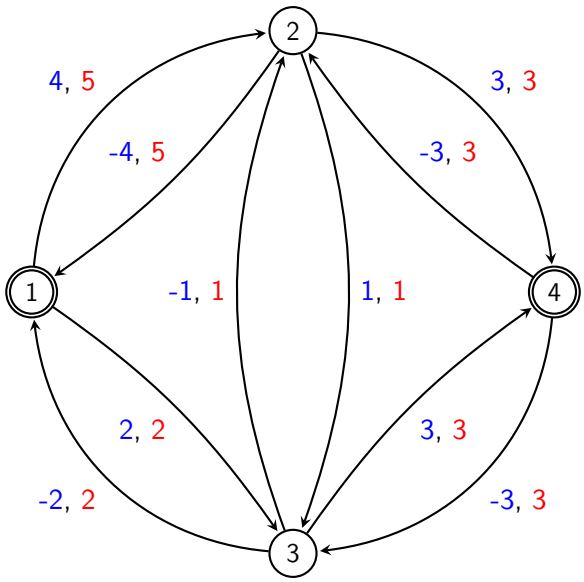
1.2 Residual graphs, flow increase and st-cuts

Consider the network shown in the last example of the previous section.



We notice that some *pipes* aren't completely *"filled up"*, meaning that their capacity could support a bigger flow. In particular, due to conservation of flow, not all pipes can be filled to the maximum capacity. In fact, we know that flow outgoing from the source must still be equal to the incoming flow of the sink.

Thus, we can increase the flow value by 1 only by using the remaining capacities in the path $1 \to 2 \to 3 \to 4$



The flow value of the new flow is 6.

Definition 5: Residual capacity

Given a network $N = (G, s, t, c)$ and a flow f on N, the **residual capacity** is a weight function $r : E(G) \to \mathbb{R}^+$ defined as:

$$r(u, v) = c(u, v) - f(u, v)$$

Observation 2

Given a network $N = (G, s, t, c)$ and a flow f on N, we note that:

$$r(u, v) + r(v, u) = c(u, v) + c(v, u)$$

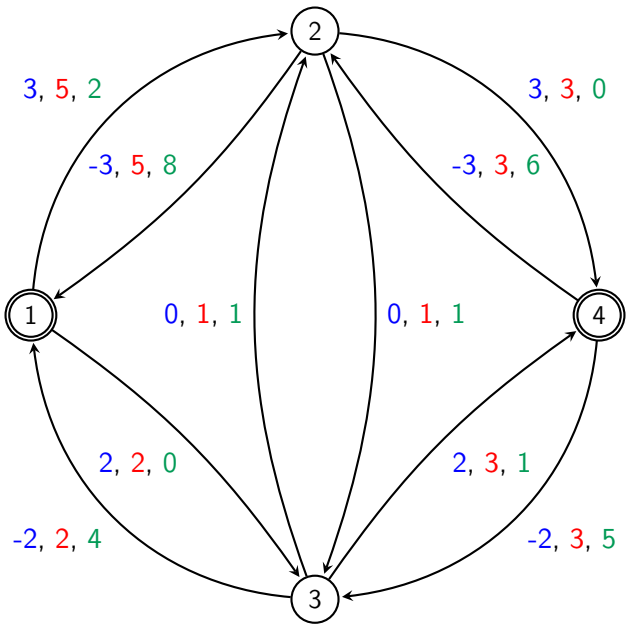
Definition 6: Residual graph

Given a network $N = (G, s, t, c)$ and a flow f on N, we define $R \subseteq G$ as the **residual graph** of G if:

$$(u, v) \in E(R) \iff r(u, v) > 0$$

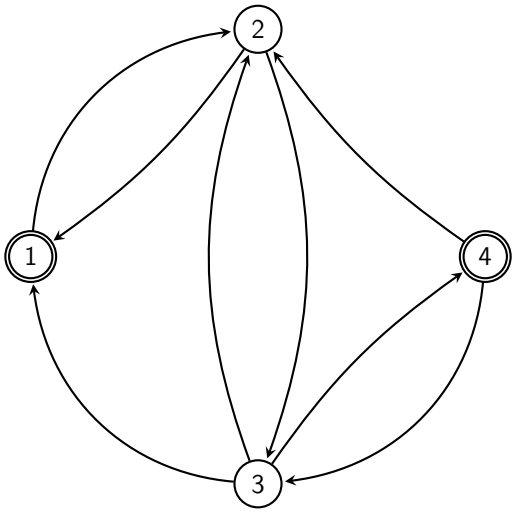
Example:

Consider the first graph previously shown with flow value 5. We now add the residual capacities obtained through that flow.



For each edge, the number in green represents its residual capacity

Thus, the residual graph is the following:



Proposition 1: Flow-augmenting path

Given a network $N = (G, s, t, c)$ and a flow f on N, let $R \subseteq G$ be the residual graph of G on f and let P be a direct path $s \to t$ in R.

Given $\alpha := \min_{(u,v)\in E(P)} r(u, v)$, we define $f' : E(G) \to R$ as:

$$f'(u,v) = \begin{cases} f(u,v) + \alpha & \text{if } (u,v) \in E(P) \\ f(u,v) - \alpha & \text{if } (v,u) \in E(P) \\ f(u,v) & \text{otherwise} \end{cases}$$

The function f' is a flow for N such that $\text{val}(f') = \text{val}(f) + \alpha$.

Proof.

Suppose that $(u, v) \in E(P)$:

- By using the skew-symmetric property of f, we get that:

$$f'(u,v) = f(u,v) + \alpha \implies -f'(u,v) = -f(u,v) - \alpha = f(v,u) - \alpha$$

  Also, since $(u, v) \in E(P)$, for (v, u) we get that

$$f'(v,u) = f(v,u) - \alpha = -f'(u,v)$$

- By assumption, we know that $f'(u, v) = f(u, v) + \alpha$. Thus, by definition of α we get that:

$$\alpha \leq r(u,v) = c(u,v) - f(u,v) \implies$$
$$f'(u,v) = f(u,v) + \alpha \leq f(u,v) + c(u,v) - f(u,v) = c(u,v)$$

If $(v, u) \in E(P)$, we can get the same result by repeating the same steps. Otherwise, the flow remains unchanged, meaning that the property is already satisfied. Thus, we conclude that f' is *skew-symmetric* and *capacity bounded*.

Given a vertex $x \in E(G)$, if $x \notin E(P)$ then the flos of its edges are unchanged. If $x \in E(P)$, we procede by splitting the edges in G through P:

$$\sum_{\substack{(u,x)\in E(G):\\ f'(u,x)>0}} f'(u,x) = \sum_{\substack{(u,x)\in E(G):\\ f'(u,x)>0,\\ (u,x)\in E(P)}} f'(u,x) + \sum_{\substack{(u,x)\in E(G):\\ f'(u,x)>0,\\ (x,u)\in E(P)}} f'(u,x) =$$

$$\sum_{\substack{(u,x)\in E(G):\\ f'(u,x)>0,\\ (u,x)\in E(P)}} [f(u,x)+\alpha] + \sum_{\substack{(u,x)\in E(G):\\ f'(u,x)>0,\\ (x,u)\in E(P)}} [f(u,x)-\alpha]$$

We notice that the amount of vertices in these sums is the same, implying that each time α gets summed in the first sum it also gets subtracted from the other sum:

$$\sum_{\substack{(u,x)\in E(G):\\ f'(u,x)>0,\\ (u,x)\in E(P)}} [f(u,x)+\alpha] + \sum_{\substack{(u,x)\in E(G):\\ f'(u,x)>0,\\ (x,u)\in E(P)}} [f(u,x)-\alpha] =$$

$$\sum_{\substack{(u,x)\in E(G):\\ f'(u,x)>0,\\ (u,x)\in E(P)}} f(u,x) + \sum_{\substack{(u,x)\in E(G):\\ f'(u,x)>0,\\ (x,u)\in E(P)}} f(u,x) = \sum_{\substack{(u,x)\in E(G):\\ f'(u,x)>0}} f(u,x)$$

By using the same argument, we get that:

$$\sum_{\substack{(x,w)\in E(G):\\ f'(x,w)>0}} f'(x,w) = \sum_{\substack{(x,w)\in E(G):\\ f'(x,w)>0}} f(x,w)$$

Finally, through the the *conservation of flow* of f, we conclude that f' also satisfies such property:

$$\sum_{\substack{(u,x)\in E(G):\\ f'(u,x)>0}} f'(u,x) = \sum_{\substack{(u,x)\in E(G):\\ f'(u,x)>0}} f(u,x) = \sum_{\substack{(x,w)\in E(G):\\ f'(x,w)>0}} f(x,w) = \sum_{\substack{(x,w)\in E(G):\\ f'(x,w)>0}} f'(x,w)$$

□

Definition 7: $st-$cut

Given a network $N = (G, s, t, c)$ and a flow f on N, we define a ***st*-cut of** G as a subset $\mathcal{U} \subseteq V(G)$ that makes a partition on $V(G)$ such that $s \in \mathcal{U}, t \notin \mathcal{U}$.

Additionally, the vertices inside of $\mathcal{U}$ are called the *s-part* of the cut, while the vertices outside of $\mathcal{U}$ are called the *t-part* of the cut.

Proposition 2: Flow of an st-cut

Given a network $N = (G, s, t, c)$, a flow f on N and an st-cut $\mathcal{U} \subseteq G$, we have that:

$$\text{val}(f) = \sum_{\substack{(u,v)\in E(G):\\ u\in\mathcal{U}, v\notin\mathcal{U}}} f(u,v)$$

Proof.

Consider the total sum of the flows of all the vertices in $\mathcal{U}$:

$$\sum_{x\in\mathcal{U}} \sum_{(u,v)\in E(G)} f(u,v)$$

By definition, we know that $s \in \mathcal{U}$. We can separate the flows outgoing from s from the rest of the flows:

$$\sum_{x\in\mathcal{U}} \sum_{(u,v)\in E(G)} f(u,v) = \sum_{(s,w)\in E(G)} f(s,w) + \sum_{x\in\mathcal{U}-\{s\}} \sum_{(u,v)\in E(G)} f(u,v)$$

Due to the Nullification of flow for middle edges, for all vertices $u, v \neq s$ we know that the total flow is qual to 0, implying that:

$$\sum_{(s,w)\in E(G)} f(s,w) + \sum_{x\in\mathcal{U}-\{s\}} \sum_{(u,v)\in E(G)} f(u,v) = \sum_{(s,w)\in E(G)} f(s,w) + 0 = \text{val}(f)$$

We now consider again the initial total sum. We notice that:

$$\sum_{x\in\mathcal{U}} \sum_{(u,v)\in E(G)} f(u,v) = \sum_{\substack{(u,v)\in E(G):\\ u\in\mathcal{U}}} f(u,v)$$

Additionally, for any (u, v) if $u, v \in \mathcal{U}$ then $f(u, v)$ cancels out with $f(v, u)$, implying that:

$$\sum_{\substack{(u,v)\in E(G):\\ u\in\mathcal{U}}} f(u,v) = \sum_{\substack{(u,v)\in E(G):\\ u\in\mathcal{U}, v\notin\mathcal{U}}} f(u,v)$$

By combining the shown equalities, we conclude the proof.

□

Definition 8: Capacity of an st-cut

Given a network $N = (G, s, t, c)$, a flow f on N and an st-cut $\mathcal{U} \subseteq G$, we define the **capacity of an st− cut on** G, noted by $c(V(G)\backslash\mathcal{U})$, as the sum of the capacities of the edges outgoing from *s-part* to the *t-part*.

$$c(V(G)\backslash\mathcal{U}) := \sum_{\substack{(u,v)\in E(G):\\ x\in\mathcal{U}, u\notin\mathcal{U}}} c(u,v)$$

Lemma 1

Given a network $N = (G, s, t, c)$, the maximum value of a flow on G at most the minimal capacity of an st-cut on G:

$$\max_{f\,:\,\text{flow on } G}(\text{val}(f)) \leq \min_{\mathcal{U}\,:\,st-\text{cut on } G}(c(V(G)\backslash\mathcal{U}))$$

Proof.

Let f be the flow on G that maximizes $\text{val}(f)$ and let $\mathcal{U} \subseteq G$ be the st-cut on G that minimizes capacity. By the Flow of an st-cut and by the *capacity bounded* property of f, we get that:

$$\text{val}(f) = \sum_{\substack{(u,v)\in E(G):\\ u\in\mathcal{U}, v\notin\mathcal{U}}} f(u,v) \leq \sum_{\substack{(u,v)\in E(G):\\ u\in\mathcal{U}, v\notin\mathcal{U}}} c(u,v) = c(V(G)\backslash\mathcal{U})$$

□

1.3 The Ford-Fulkerson algorithm

Algorithm 1: The Ford-Fulkerson algorithm

Given a network $N = (G, s, t, c)$, we define the following algorithm:

function FordFulkerson(G)
  Start with the trivial flow f with all 0s
  while True **do**
    Compute the residual graph $R \subseteq G$ on flow f
    Find a path P in R from $s \to t$
    if P does not exist **then**
      Return f
    else
      Increase f through the value α obtained by P
    end if
  end while
end function

We note that the FordFulkerson(N) terminates only if the augment value eventually becomes 0, implying that there is no flow-augmenting path to be used. Thus, if the capacities defined by c are all integers, the algorithm always terminates.

Moreover, the flow-augmenting path of each iteration of FordFulkerson(N) can be found with a simple DFS search, requiring $O(n+m)$, which is also the cost needed for computing the residual graph of each iteration. Thus, the **computational complexity** of the algorithm is $O(k(n + m))$, where k is the maximum number of iterations of the while loop.

It's easy to notice that this value k **depends too much on the shape of the graph** G. However, due to the *capacity bounded* property, in the worst case we have that k equals the maximum flow value. We also notice that, technically, the cost of this algorithm is **exponential**: the number of bits required to store k are $\log_2 k$, but the cost relies on $2^{\log_2 k} = k$ iterations.

Lemma 2

Given a network $N = (G, s, t, c)$, if the algorithm FordFulkerson(N) terminates then it returns a flow f such that there exists an st-cut $\mathcal{U} \subseteq G$ for which we have that $\text{val}(f) = c(V(G) \backslash \mathcal{U})$

Proof.

Let R be the residual graph of G on f = FordFulkerson(N) and let r be the residual capacity function obtained through f. We define $\mathcal{U} \subseteq G$ as:

$$\mathcal{U} = \{x \in V(G) \mid \exists s \to x \text{ in } G'\}$$

Since the algorithm terminates when there is no path from $s \to t$, we know that $t \notin \mathcal{U}$, implying that $\mathcal{U}$ is an st-cut of G. Thus, by the Flow of an st-cut, we have that:

$$\text{val}(f) = \sum_{\substack{(x,v)\in E(G):\\ x\in\mathcal{U}, v\notin\mathcal{U}}} f(x,u)$$

By way of contradiction, we suppose that $\exists (x,v) \in E(G') \subseteq E(G)$ such that $x \in \mathcal{U}, v \notin \mathcal{U}$. By definition of $\mathcal{U}$, we know that $s \to x$ in G', so by adding (x,v) to the path we get that $s \to x \to y$, which contradicts $y \notin \mathcal{U}$.

Thus, such edge can't exists, implying that $\forall (x,v) \in E(G)$ such that $x \in \mathcal{U}, v \notin \mathcal{U}$ it holds that $(x,v) \notin E(G')$, which by definition of residue graph implies that:

$$\forall (x,v) \in E(G) \text{ s.t. } x \in \mathcal{U}, v \notin \mathcal{U} \quad f(x,v) = c(x,v)$$

concluding that:

$$\text{val}(f) = \sum_{\substack{(x,v)\in E(G):\\ x\in\mathcal{U}, v\notin\mathcal{U}}} f(x,u) = \sum_{\substack{(x,v)\in E(G):\\ x\in\mathcal{U}, v\notin\mathcal{U}}} c(x,u) = c(V(G)\backslash\mathcal{U})$$

□

1.3.1 The Max-flow/Min-cut theorem

Theorem 1: Max-flow/Min-cut theorem

Given a network $N = (G, s, t, c)$, the maximum value of a flow on G at most the minimal capacity of an st-cut on G:

$$\max_{f\,:\,\text{flow on } G}(\text{val}(f)) = \min_{\mathcal{U}\,:\,st-\text{cut on } G}(c(V(G)\backslash\mathcal{U}))$$

Proof.

By the Lemma 1, we already know that:

$$\max_{f\,:\,\text{flow on } G}(\text{val}(f)) \leq \min_{\mathcal{U}\,:\,st-\text{cut on } G}(c(V(G)\backslash\mathcal{U}))$$

We now consider $f' = \textsf{FordFulkerson}(N)$. By the Lemma 2, we know that there exists an st-cut $\mathcal{U}' \subseteq G$ such that $\text{val}(f') = c(V(G)\backslash\mathcal{U}')$.

Thus, it's easy to conclude that:

$$\max_{f\,:\,\text{flow on } G}(\text{val}(f)) \geq \text{val}(f') = c(V(G)\backslash\mathcal{U}) \geq \min_{\mathcal{U}\,:\,st-\text{cut on } G}(c(V(G)\backslash\mathcal{U}))$$

□

Corollary 1: Optimality of Ford-Fulkerson

The Ford-Fulkerson algorithm returns a flow with **maximum value**

1.4 The Edmonds-Karp algorithm

Algorithm 2: The Edmonds-Karp algorithm

Given a network $N = (G, s, t, c)$, we define the following algorithm:

function EDMONDSKARP(N)
  Start with the trivial flow f with all 0s
  while True **do**
    Compute the residual graph $R \subseteq G$ on flow f
    Find the shortest path P in R from $s \to t$
    if P does not exist **then**
      Return f
    else
      Increase f through the value α obtained by P
    end if
  end while
end function

Lemma 3

Given a network $N = (G, s, t, c)$, if the algorithm EdmondsKarp(N) terminates then it returns a flow f such that there exists an st-cut $\mathcal{U} \subseteq G$ for which we have that $\text{val}(f) = c(V(G) \backslash \mathcal{U})$

(*proof identical to* Lemma 2)

Corollary 2: Optimality of Edmonds-Karp

The Edmonds-Karp algorithm returns a flow with **maximum value**

The Edmonds-Keep algorithm looks identical to the Ford-Fulkerson algorithm, except for the type of path searched at each iteration. The idea behind finding the shortest path instead of a random path is to **limit** the number of iterations made by the algorithm by making them rely on the number of nodes instead of the maximum flow value. Thus, the **computational complexity** of the algorithm effectively becomes $O(mn(n + m))$, meaning that it's not an exponential algorithm. The following statements will be necessary to justify this result.

Observation 3

Given a graph G and two vertices $x, y \in V(G)$, let P be the shortest path from $x \to y$. Then, for each node $z_i \in V(P)$ the path P contains the shortest path P_i from $x \to z_i$

Proof.

For each node $z_1, \dots, z_k \in V(P)$ (x and y included), let P_i be the shortest path from $x \to z_i$. By way of contradiction, suppose that $P_i \not\subseteq P$.

Given an index $i \in [1, k]$, let $P_x, P_y \subseteq P$ be the sub-paths that partition P by z_i, meaning that $x, z_1, \dots, z_i \in V(P_x)$ and $z_{i+1}, \dots, z_k, y \in V(P_y)$.

Since by assumption P_i is shorter that P_x, we get that the path $P_i \cup P_y$ is a path from $x \to y$ that is shorter than P, which is a contradiction. Thus, we conclude that for each node $z_1, \dots, z_k \in V(P)$ it holds that $P_i \subseteq P$.

□

Proposition 3: Disappearing and appearing edges

Given a network $N = (G, s, t, c)$ and the computation $\mathsf{EdmondsKarp}(N)$, let:

- $f_0, \dots, f_k$ be the series of flows computed, where f_0 is the trivial flow
- $R_0, \dots, R_k$ be the series of residue graphs computed, where R_0 is obtained with flow f_i
- $P_0, \dots, P_k$ be the series of shortest paths from $s \to t$ computed on R_i

Then, $\forall u \in V(G)$ and $\forall i \in [1, k]$ it holds that:

$$(u, v) \in E(R_i), (u, v) \notin E(R_{i+1}) \implies (u, v) \in E(P_i)$$

and that:

$$(u, v) \notin E(R_i), (u, v) \in E(R_{i+1}) \implies (v, u) \in E(P_i)$$

Proof.

If $(x, y) \in E(R_i)$ but $(x, y) \notin E(R_{i+1})$, the edge got removed by the flow-increase of path P_i. This can only happen only if $c(x, y) = f_{i+1}(x, y) = f_i(x, y) + \alpha_i$, where α_i is the flow increase obtained by P_i, meaning that $(x, y) \in P_i$.

Instead, if $(x, y) \notin E(R_i)$ but $(x, y) \in E(R_{i+1})$, the edge got added by the flow-increase of path P_i. This can happen only if $f_i(x, y) \geq f_{i+1}(y, x)$, which in turn can happen only if the flow of the opposite edge (x, y) got increased, meaning that $(y, x) \in E(P_i)$.

□

Lemma 4: Monotone increasing distance in Edmonds-Keep

Given a network $N = (G, s, t, c)$ and the computation $\mathsf{EdmondsKarp}(N)$, let:

- $f_0, \ldots, f_k$ be the series of flows computed, where f_0 is the trivial flow
- $R_0, \ldots, R_k$ be the series of residue graphs computed, where R_0 is obtained with flow f_i
- $P_i, \ldots, P_k$ be the series of shortest paths from $s \to t$ computed, where $P_i \subseteq R_i$

Then, $\forall u \in V(G)$ and $\forall i \in [1, k]$ it holds that:

$$\text{dist}_{R_i}(s, u) \leq \text{dist}_{R_{i+1}}(s, u)$$

Proof.

By way of contradiction, we assume that there exist some vertices for which the statement doesnt hold, meaning that $\exists u_1, \ldots, u_k \in V(G)$ such that $\text{dist}_{R_i}(s, u_j) > \text{dist}_{R_{i+1}}(s, u_j)$.

Let v be the vertex picked from $u_1, \ldots, u_k$ with minimal distance from s in R_i, meaning that:

$$\text{dist}_{R_i}(s, v) = \min_{j=1}^{k} \text{dist}_{R_i}(s, u_j)$$

Consider the shortest path P in R_{i+1} from $s \to v$, that being the path with length $\text{dist}_{R_{i+1}}(s, v)$. Let $u \in V(P)$ be the vertex that precedes v in P, meaning that $(u, v) \in E(P)$.

Since v is the vertex from $u_1, \ldots, u_k$ with the shortest distance and since $\text{dist}_{R_{i+1}}(s, u) = \text{dist}_{R_{i+1}}(s, v) - 1$ due to it being the previous vertex of v in P, it must hold that $u \notin \{u_1, \ldots, u_k\}$ because otherwise u would be the one with the shortest distance.

Thus, we get that $\text{dist}_{R_i}(s, u) \leq \text{dist}_{R_{i+1}}(s, u)$, which implies that:

$$\text{dist}_{R_i}(s, u) \leq \text{dist}_{R_{i+1}}(s, u) = \text{dist}_{R_{i+1}}(s, v) - 1$$

Suppose now that $(u, v) \in E(R_i)$. Given the shortest path P' in $R_i i$ from $s \to u$ can be extended with (u, v), we get that $\text{dist}_{R_i i}(s, v) \leq \text{dist}_{R_i i}(s, u) + 1$. However, this would imply that:

$$\text{dist}_{R_i}(s, v) \leq \text{dist}_{R_i}(s, u) + 1 \leq \text{dist}_{R_{i+1}}(s, u) + 1 \leq= \text{dist}_{R_{i+1}}(s, v)$$

which contradicts the initial assumption. Thus, it can't be that $(u, v) \in E(R_i)$.

Moreover, since $(u, v) \in E(P) \subseteq E(R_{i+1})$ and $(u, v) \notin E(R_i)$, by Proposition 3 we know that:

$$(u, v) \notin E(R_i), (u, v) \in E(R_{i+1}) \implies (v, u) \in E(P_i)$$

Since P_i is the shortest path $s \to t$ in R_i and $u, v \in V(P_i)$, by Observation 3 we know that P_i also contains the shortest paths from $s \to u$ and $s \to v$ in R_i. In particular, the path from $s \to u$ also contains the path $s \to v$, so $\text{dist}_{R_i}(s, v) = \text{dist}_{R_i}(s, u) - 1$.

Combining this with the previous results and the initial assumption, we get that:

$$\text{dist}_{R_i}(s, v) = \text{dist}_{R_i}(s, u) - 1 \leq \text{dist}_{R_{i+1}}(s, u) - 1 = \text{dist}_{R_{i+1}}(s, v) - 2 < \text{dist}_{R_i}(s, v) - 2$$

implying that $0 < -2$, which is impossible, meaning that such vertices $u_1, \ldots, u_k$ can't exists.

□

Theorem 2: Total iterations of Edmonds-Karp

The total number of iterations done by the Edmonds-Karp algorithm is $O(mn)$.

This also implies that the algorithm always terminates.

Proof.

Given a network $N = (G, s, t, c)$ and the computation $\mathsf{EdmondsKarp}(N)$, let:

- $f_0, \ldots, f_k$ be the series of flows computed, where f_0 is the trivial flow
- $R_0, \ldots, R_k$ be the series of residue graphs computed, where R_0 is obtained with flow f_i
- $P_i, \ldots, P_k$ be the series of shortest paths from $s \to t$ computed, where $P_i \subseteq R_i$

In the following steps, we define an edge (u, v) as *critical* for R_i if $(u, v) \in E(R_i)$ but $(u, v) \notin E(R_{i+1})$. In particular, by Proposition 3 we know that if an edge is critical for R_i then $(u, v) \in P_i$. Moreover, for each shortest path P_i we know that there is at least one critical edge inside it, that being the edge which defines the flow augmentation value α_i for P_i.

Given an edge $(u, v) \in E(G)$, let $\pi(1) < \pi(2) < \ldots < \pi(\ell) \in [1, k]$ be the indices such that (u, v) is critical for $\pi(i)$. By Observation 3, we know that $\forall i \in [1, k]$ it holds that:

$$(u, v) \in E(P_{\pi(i)}) \implies \text{dist}_{R_{\pi(i)}}(s, v) = \text{dist}_{R_{\pi(i)+1}}(s, u) + 1$$

meaning that $(u, v) \in E(R_{\pi(i)+1})$.

If (u, v) is critical for both $R_{\pi(i)}$ and $R_{\pi(j)}$ (where $i < j$, meaning that (u, v) disappears both times), then there must be an index h such that $\pi(i) < h < \pi(j)$ where (u, v) reappears, meaning that $(u, v) \notin E(G_h)$ and $h \in E(G_{h+1})$.

Again, by Proposition 3, we know that:

$$(u, v) \notin E(G_h), h \in E(G_{h+1}) \implies (v, u) \in E(P_h) \implies \text{dist}_{R_h}(s, u) = \text{dist}_{R_h}(s, v) + 1$$

Then, since $\pi(i) < h < \pi(j)$, by the Monotone increasing distance in Edmonds-Keep we know that:

$$\text{dist}_{R_{\pi(j)}}(s,u) \geq \text{dist}_{R_h}(s,u) = \text{dist}_{R_h}(s,v) + 1 \geq \text{dist}_{R_{\pi(i)}}(s,u) + 2$$

Thus, between $R_{\pi(i)}$ and $R_{\pi(j)}$ the vertex u's distance increases by at least 2. Thus, since u's final distance can be at most $n-1$, meaning that $\text{dist}_{R_k}(s,u) \leq n$, starting from 0 the distance can be increased by 2 at most $\frac{n}{2}$ times, meaning that $\ell = \frac{n}{2}$.

Since each flow-augmenting shortest path computed by the algorithm implies the existence of a a critical path and since since the number of times each edge can be critical is at most $\ell = \frac{n}{2}$, the number of paths computable is at most $m \times \frac{n}{2}$, which is in $O(mn)$.

□

1.5 Applications of the Max-flow/Min-cut theorem

Considering the previous definitions, it's easy to see that for each undirected graph there is an associated network with chosen capacities by simply "splitting" each undirected edge (u,v) into two directed edges $(u,v),(v,u)$. Viceversa, by inverting this transformation process, we can associate an undirected graph to each network. In particular, given an undirected graph G, we denote with $\overrightarrow{G}$ the directed graph underling the network $N = (\overrightarrow{G}, s, t, c)$ associated to G.

This natural association can be used to prove theorems on undirected graphs through the use of the Max-flow/Min-cut theorem. In particular, we'll consider the problem of **finding the maximum number of pairwise edge-disjoint paths** between two nodes in an undirected graph.

Lemma 5

Let G be an undirected graph and let $N = (\overrightarrow{G}, s, t, c)$ be the network associated to G such that all edges have capacity set to 1. If G has at most k pairwise edge-disjoint paths $s \to t$ then the maximum value of a flow f on N is k.

Proof.

Since G has at most k pairwise edge-disjoint paths $s \to t$, the associated graph $\overrightarrow{G}$ will have k pairwise edge-disjoint paths $s \to t$ and k pairwise edge-disjoint paths $t \to s$ thanks to the "splitting" process.

Then, for each path $s \to t$ in $\overrightarrow{G}$ we can set $f(u,v) = 1$ and $f(v,u) = -1$ for each edge (u,v) in the path. Since each path is disjoint, sending flow to one of them doesn't influence the other paths, implying that one unit of flow can be sent on each one of the k paths and thus that the maximum value of f is k.

□

Definition 9: Support of a flow

Given a network $N = (\overrightarrow{G}, s, t, c)$ and a flow f on N, we define the **support of** f as the subgraph $\overrightarrow{W} \subseteq \overrightarrow{G}$ such that:

$$E(\overrightarrow{W}) = \{(u, v) \in E(\overrightarrow{G}) \mid f(u, v) \neq 0\}$$

Lemma 6

Let G be an undirected graph and let $N = (\overrightarrow{G}, s, t, c)$ be the network associated to G such that all edges have capacity set to 1. If the maximum value of a flow f on N is k then the undirected graph W associated to the support graph $\overrightarrow{W}$ of f has at most k pairwise edge-disjoint paths $s \to t$.

Proof.

Let f be a flow on N with maximum value, $\overrightarrow{W} \subseteq \overrightarrow{G}$ be the support graph of f and W the undirected graph associated to $\overrightarrow{W}$. By strong induction on the number m of edges in W, meaning that $|E(W)| = m$, we show that W has maximum k pairwise edge-disjoint paths $s \to t$.

If $m = 0$, then $\forall (u, v) \in E(G)$ it holds that $f(u, v) = 0$, implying that $\operatorname{val} f = 0$ and that the number of pairwise edge-disjoint paths $s \to t$ in W is 0. We now assume by strong induction that the statement holds for all graphs whose support graph has at most m edges.

Suppose now that $|E(W)| = m + 1$. In $\overrightarrow{W}$, let $P \subseteq \overrightarrow{W}$ be the subgraph such that:

- $V(P) = \{v_0, \ldots, v_h\}$
- $v_0 = s$
- $\forall i \in [0, h-1] \;\; (v_i, v_{i+1}) \in E(\overrightarrow{W})$
- $\forall i \in [0, h-1] \;\; f(v_i, v_{i+1}) = 1$
- h is as big as possible

In particular, we observe that this subgraph describes the longest possible path starting from s in the whole graph. Such a subgraph can always be found since v_h can be equal to s, giving us the trivial zero-edged path $s \to s$.

In case $v_h = t$, the subgraph P corresponds to the longest path $s \to t$ in $\overrightarrow{W}$ and thus the longest path in W. Then, considering f' such that:

$$f'(x, y) = \begin{cases} 0 & \text{if } (x, y) \in E(P) \text{ or } (x, y) \in E(P) \\ f(x, y) & \text{otherwise} \end{cases}$$

its easy to see that f' is a flow on the network $(\overrightarrow{W}, s, t, c)$.

Let $\overrightarrow{W'} \subseteq \overrightarrow{W}$ be the support graph of f'. By definition of f', we get that $\overrightarrow{W'} = \overrightarrow{W} - E(P)$. Since P contains an edge outgoing from s, in f' the flow value of that edge was set to 0, implying that $\operatorname{val} f' = \operatorname{val} f - 1 = k - 1$.

Since $|E(W')| < |E(W)|$, by inductive hypothesis the graph W' has at most $\operatorname{val} f' = k-1$ pairwise edge-disjoint paths $s \to t$. Moreover, since $\overrightarrow{W'} = \overrightarrow{W} - E(P)$, the path P is disjoint from these $k-1$ paths, concluding that W has k pairwise edge-disjoint paths.

In case $v_h \neq t$, instead, since $f(v_{h-1}, v_h) = 1$, there must exists an edge $(u, v_h) \in E(\overrightarrow{W})$ with flow value $f(u, v_h) = -1$ in order to preserve the conservation of flow for v_h. Moreover, this also implies that $f(v_h, u) = 1$.

By way of contradiction, suppose that $u \notin V(P)$. Then, since $f(v_h, u) = 1$, we could extend this path P with (v_h, u), which contradicts the fact that h was chosen as big as possible, meaning that the only possibility is that $u \in V(P)$. In particular, since by definition of P we have that $f(v_{h-1}, v_h) = 1$ and for u we have that $f(u, v_h) = -1$, we know that $u \neq v_{h-1}$.

Given $i \in [0, k-2]$ such that $u = v_i$, let C be the directed cycle $v_i, \ldots, v_h, u$. Then, considering f'' such that:

$$f''(x,y) = \begin{cases} 0 & \text{if } (x,y) \in E(C) \text{ or } (x,y) \in E(C) \\ f(x,y) & \text{otherwise} \end{cases}$$

its easy to see that f'' is a flow on the network $(\overrightarrow{W}, s, t, c)$.

If $i \neq 0$ then $v_i \neq s$, implying that $\operatorname{val} f'' = \operatorname{val} f = k$. Otherwise, if $i = 0$, we have that $f''(s, v1) = f''(v_h, s) = 0$. Then, since previously we had $f(s, v1) = 1$ and $f(v_h, s) = -1$, the value of the flow hasn't changed, concluding that in both cases we have that $\operatorname{val} f'' = \operatorname{val} f$.

Finally, since $|E(W'')| < |E(W)|$, by inductive hypothesis the graph W'' has at most $\operatorname{val} f'' = k$ pairwise edge-disjoint paths $s \to t$, implying that also W has these paths, concluding that in all cases the statement holds for $m + 1$.

□

Corollary 3

Let G be an undirected graph and let $N = (\overrightarrow{G}, s, t, c)$ be the network associated to G such that all edges have capacity set to 1. If the maximum value of a flow f on N is k then G has at most k pairwise edge-disjoint paths $s \to t$.

Proof.

By the previous lemma, we know that $W \subseteq G$ has k edge-disjoint paths. By way of contradiction, suppose that the number of pairwise edge-disjoint paths in G is greater than k. Then, there exists at least another disjoint path P from $s \to t$ in G that isn't in W.

Since P is not in W, by definition of support graph, each edge in P must have a flow equal to 0. Then, we could send another unit of flow in this path, contradicting the fact that f is the maximum flow in G. Thus, the only possibility is that such a path cannot exist.

□

Theorem 3: Menger's theorem

Let G be an undirected graph and let $N = (\overrightarrow{G}, s, t, c)$ be the network associated to G such that all edges have capacity set to 1. The maximum value of a flow f on N is exactly equal to the maximum number of pairwise edge-disjoint paths $s \to t$ in G.

(follows from Lemma 5 and Corollary 3)

2

Linear programming

2.1 Introduction and interpretation

Linear programming, also called *linear optimization*, is a method to achieve the **optimal outcome** (such as maximum profit or lowest cost) in a mathematical model whose requirements and objective are represented by **linear inequalities**. In particular, the objective to be optimized is defined by an **objective function** on variables $x_1, \ldots, x_n$ that must be *maximized* (or *minimized*).

For example, given two variables $x_1, x_2 \in \mathbb{R}$, we want to maximize the sum $x_1 + 2x_2$ while also respecting the following linear constraints:

$$\begin{aligned} x_1 &\geq 0 \\ x_2 &\geq 0 \\ x_1 + 6x_2 &\leq 15 \\ 4x_1 - x_2 &\leq 10 \\ -x_1 + x_2 &\leq 2 \end{aligned}$$

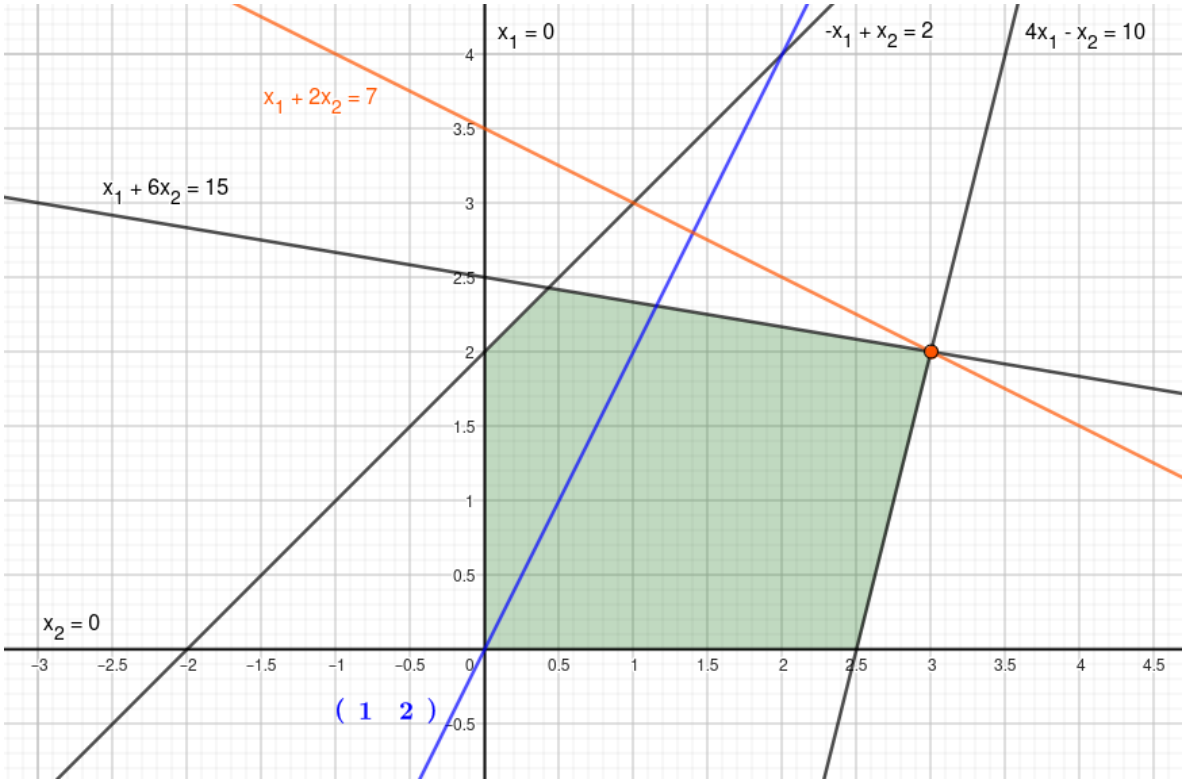
Since we have only two variables, these constraints can be interpreted as lines in a cartesian plane defined by x_1 and x_2: given a constraint $\alpha x_1 + \beta y_1 \leq \gamma$ (or $\alpha x_1 + \beta y_1 \geq \gamma$), the plane gets partitioned by the line $\alpha x_1 + \beta y_1 = \gamma$ and we can consider only the part that respects the constraint.

By applying this process for each constraint, only one part of the graph will respect all the constraints. Any point in this area is a **feasible solution** to the problem. Furthermore, the area that contains the feasible solutions is called the **feasible region**.

In particular, we want to find a feasible solution that maximizes (or minimizes) the objective function. These solutions are called **optimal solutions**. We notice now that the objective function $x_1 + 2x_2$ can also be rewritten as a scalar product:

$$x_1 + 2x_2 = \begin{bmatrix} 1 & 2 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Considering the line described by the vector $\begin{bmatrix}1 & 2\end{bmatrix}$, it's easy to see the optimal solution to the problem is given by the point "furthest" from the origin: the line *perpendicular* to the vector $\begin{bmatrix}1 & 2\end{bmatrix}$ that passes through such point is the optimal solution.



The optimal solution is given by $x_1 + 2x_2 = 7$

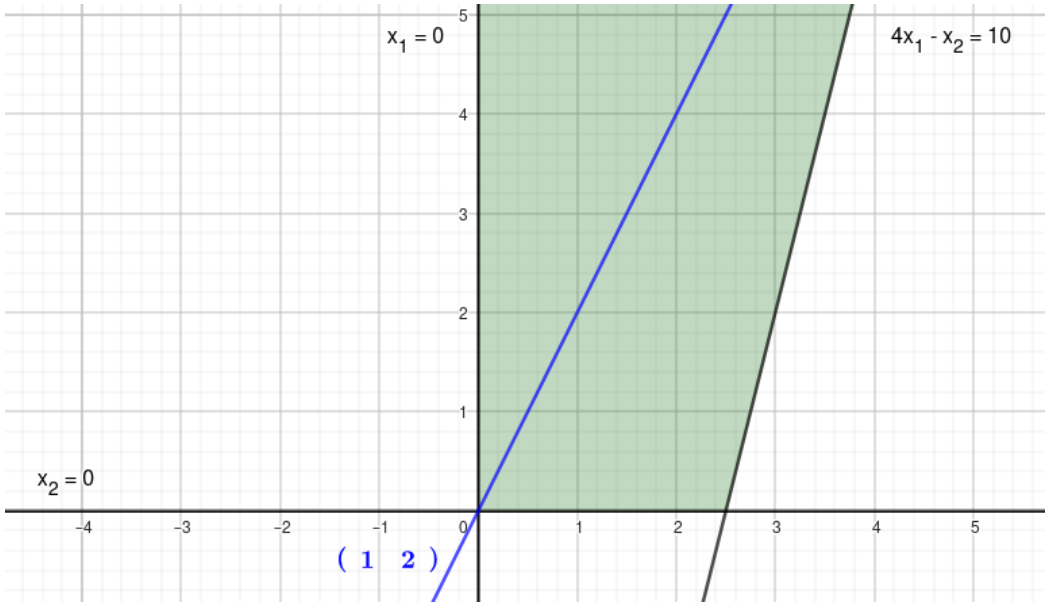
For each linear program, only one of the following cases holds:

1. There is no feasible solution
2. There are infinite feasible solutions but no optimal solution
3. There are infinite feasible solutions and only one optimal solution
4. There are infinite feasible and optimal solutions

This result is (for now) just an informal result. In later sections, we will formally define each concept and justify each result.

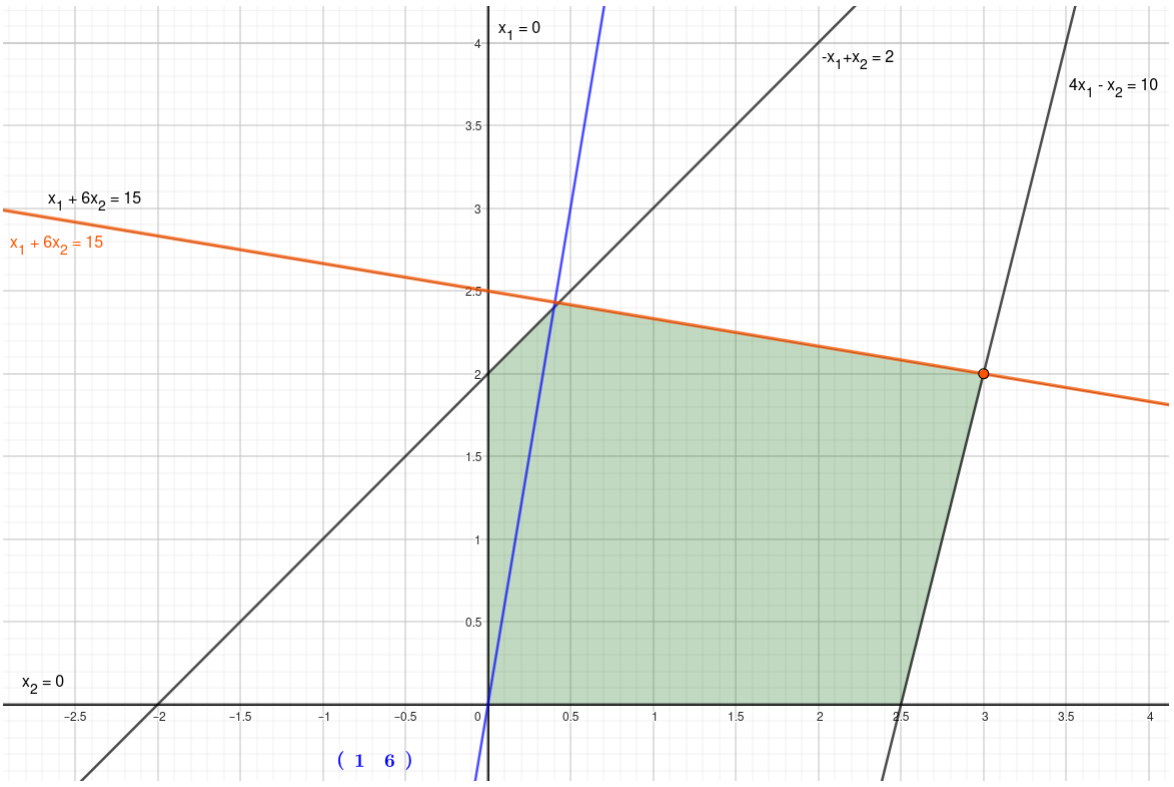
The previous problem clearly is an example of the third case. For the first case, a problem with constraints $x_1 \geq 0$ and $x_1 \leq 1$ trivially has no feasible solution since there is no possible x_1 value that respects the constraints.

The second case, instead, can be viewed as a problem with constraints such that the feasible region isn't bounded "in all directions". For example, if we remove the constraint $x_1 + 6x_2$, the feasible region isn't bounded from above, meaning that there will always be a feasible solution better than the one considered.



Example with no optimal solution

The final case can be obtained if the objective function is equal to a constraint: since the every point on the perpendicular line is an optimal solution and since in $\mathbb{R}^2$ there are infinite points on a line, we get infinite optimal solutions.



Same problem as before with objective function $x_1 + 6x_2 = 15$

Optimization problems are hidden in everyday life. For example, we can resolve the following two problems with linear programming:

- **Diet optimization**:

  Given n types of food, for each i such that $1 \leq n$ we define x_i and c_i respectively as the quantity and the cost of each food type i. Considering a set of nutritional constraints, such as minimum and maximum macro-nutrient intakes, we can define the following optimization problem in order to find the best combination of foods for our diet:

$$\max c_1 x_1 + \ldots + c_n x_n$$

$$\begin{gathered} a_{1,1} x_1 + \ldots + a_{1,n} x_n \leq b_1 \\ \vdots \\ a_{m,1} x_1 + \ldots + a_{m,n} x_n \leq b_m \end{gathered}$$

- **Network flow**:

  Given a network $N = (G, s, t, c)$ and a flow f on N, we can define a variable $x_{u,v}$ for all $(u, v) \in E(G)$ in order to formulate the max flow value problem as an optimization problem and find the optimal flow by setting $f(u, v) = x_{u,v}$. Trivially, each constraint of the network becomes a constraint of the problem.

2.2 Linear programs

In this section we give a proper formal definition of the concepts previously shown. Hence, it is assumed that each interpretation is already well-known. In particular, we will define linear programs only for maximization problems since minimizing a value $f(x)$ is equal to maximizing $-f(x)$.

Definition 10: Linear function

Let $f : \mathbb{R}^n \to \mathbb{R}$. We say that f is a **linear function** if

$$\forall x, y \in \mathbb{R}^n, \forall \alpha, \beta \in \mathbb{R} \quad f(\alpha x + \beta y) = \alpha f(x) + \beta f(y)$$

Definition 11: Linear program

A **linear program**, or *linear optimization problem*, is an optimization problem defined on an linear function $f : \mathbb{R}^n \to \mathbb{R}$ to be maximized, called **objective function**, and constraints on the input vector x:

$$\max f(x) = c_1 x_1 + \dots c_n x_n$$

$$\begin{aligned} a_{1,1}x_1 + \dots + a_{1,n}x_n &\leq b_1 \\ &\vdots \\ a_{m,1}x_1 + \dots + a_{m,n}x_n &\leq b_m \end{aligned}$$

Each vector x that respects the constraints is called a **feasible solution** and the set of feasible solutions is called **feasible region**. Each feasible solution that maximizes the objective function is called **optimal solution**.

In the previous section, we noticed how the objective function can also be rewritten as a scalar product:

$$f(x) = c_1 x_1 + \dots + c_n x_n = \begin{bmatrix} c_1 & \dots & c_n \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$$

Thus, we can say that $f(x) = c^T x$, where $c^T = \begin{bmatrix} c_1 & \dots & c_n \end{bmatrix}$ is the transpose of the **coefficient vector of** f.

Moreover, given an inequality $a_1 x_1 + \dots a_n x_n \geq b$, we know that:

$$a_1 x_1 + \dots + a_n x_n \geq b \iff -a_1 x_1 - \dots - a_n x_n \leq -b$$

meaning that we can substitute the first inequality on the left with the one on the right. Likewise, we want all the variables inside the solution to be non-negative. Thus, given a constraint $x_i \leq 0$, we can consider two new variables z_i, z_i' such that $x_i = z_i - z_i'$

and $z_i, z_i' \geq 0$. Thus, we can substitute the first constraint with the three constraints $z_i - z_i^i - x = 0$ and $z_i, z_i' \geq 0$.

Finally, by considering each constraint of as a row of a matrix A, we can rewrite each constraint to define the following **general formulation of linear programs**:

$$\max c^T x$$

$$Ax \leq b$$
$$x \geq 0$$

Definition 12: Convex set

Let $X \subseteq \mathbb{R}^n$. We say that X is **convex** if $\forall x_1, x_2 \in X$ it holds that all the points in the line segment from x_1 to x_2 are also in X.

Formally, this means that if $\forall x_1, x_2 \in X$ and $\forall \alpha \in [0, 1]$, it holds that the linear combination $\alpha x_1 + (1 - \alpha)x_2$ is also in X.

Proposition 4

The feasible region of every linear program is a convex set.

Proof.

Suppose that the following inequalities are the constraints of a linear program:

$$\begin{aligned} a_{1,1}x_1 + \ldots + a_{1,n}x_n &\leq b_1 \\ &\vdots \\ a_{m,1}x_1 + \ldots + a_{m,n}x_n &\leq b_m \end{aligned}$$

Let X be the feasible region of the linear program. Let $\overline{x} := (\overline{x_1}, \ldots, \overline{x_1})$ and $\overline{y} := (\overline{y_1}, \ldots, \overline{y_1})$ be two feasible solutions, meaning that $\overline{x}, \overline{y} \in X$.

Given $\alpha \in [0, 1]$, we notice that:

$$\begin{aligned} \alpha(a_{1,1}\overline{x_1} + \ldots + a_{1,n}\overline{x_n}) &\leq \alpha b_1 \\ &\vdots \\ \alpha(a_{m,1}\overline{x_1} + \ldots + a_{m,n}\overline{x_n}) &\leq \alpha b_m \end{aligned} \quad \Longrightarrow \quad \begin{aligned} a_{1,1}(\alpha\overline{x_1}) + \ldots + a_{1,n}(\alpha\overline{x_n}) &\leq \alpha b_1 \\ &\vdots \\ a_{m,1}(\alpha\overline{x_1}) + \ldots + a_{m,n}(\alpha\overline{x_n}) &\leq \alpha b_m \end{aligned}$$

Likewise, we can show that:

$$\begin{aligned} a_{1,1}((1-\alpha)\overline{y_1}) + \ldots + a_{1,n}((1-\alpha)\overline{y_n}) &\leq (1-\alpha)b_1 \\ &\vdots \\ a_{m,1}((1-\alpha)\overline{y_1}) + \ldots + a_{m,n}((1-\alpha)\overline{y_n}) &\leq (1-\alpha)b_m \end{aligned}$$

Thus, by summing the two systems of inequalities, we get that:

$$\begin{gathered} a_{1,1}(\alpha\overline{x_1} + (1-\alpha)\overline{y_1}) + \ldots + a_{1,n}(\alpha\overline{x_n} + (1-\alpha)\overline{y_n}) \leq b_1 \\ \vdots \\ a_{m,1}(\alpha\overline{x_1} + (1-\alpha)\overline{y_1}) + \ldots + a_{m,n}(\alpha\overline{x_n} + (1-\alpha)\overline{y_n}) \leq b_m \end{gathered}$$

concluding that $\alpha\overline{x} + (1-\alpha)\overline{y}$ is also a feasible solution and thus that it's in X

□

Proposition 5: Infinite optimal solutions

Given a linear program with objective function $f(x) = c_1x_1 + \ldots + c_nx_n$, if there are two distinct optimal solutions $\overline{x}, \overline{y}$ then there are infinite optimal solutions.

Proof.

Let β be the maximum value of f obtained with the optimal solutions $\overline{x}, \overline{y}$, meaning that $f(\overline{x}) = f(\overline{y}) = \beta$. For each $\alpha \in [0, 1]$, we know that $\alpha\overline{x} + (1-\alpha)\overline{y}$ is a feasible solution due to the feasible region being a convex set.

Moreover, by linearity of f, we notice that:

$$f(\alpha\overline{x} + (1-\alpha)\overline{y}) = \alpha f(\overline{x}) + (1-\alpha)f(\overline{y}) = \alpha\beta + (1-\alpha)\beta = \beta$$

concluding that $\alpha\overline{x} + (1-\alpha)\overline{y}$ is an optimal solution.

□

2.2.1 Standard form of a linear program

Given a linear program, we would like to find a way to define the linear program in an **equational form**. In particular, consider the following linear program:

$$\max c_1 x_1 + \ldots + c_n x_n$$

$$\begin{aligned} a_{1,1}x_1 + \ldots + a_{1,n}x_n &\leq b_1 \\ &\vdots \\ a_{m,1}x_1 + \ldots + a_{m,n}x_n &\leq b_m \end{aligned}$$

where A is the matrix formed by the constraints.

We define m variables $s_1, \ldots, s_m$, called **slack variables**, such that for each $i \in [1, m]$, we define $s_i := b_i - a_{1,1}x_1 + \ldots + a_{1,n}x_n$. By adding the slack variable s_i to the left side of the i-th inequality, each inequality becomes an equality.

Thus, at the cost of adding m variables to the problem, we can reformulate the problem as:

$$\max c_1 x_1 + \ldots + c_n x_n + 0 \cdot s_1 + \ldots + 0 \cdot s_m$$

$$\begin{aligned} a_{1,1}x_1 + \ldots + a_{1,n}x_n + s_1 &= b_1 \\ &\vdots \\ a_{m,1}x_1 + \ldots + a_{m,n}x_n + s_m &= b_m \end{aligned}$$

Given the $m \times m$ identity matrix I_m, we get that:

$$\begin{bmatrix} A \mid I_m \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \\ s_1 \\ \vdots s_m \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_m \end{bmatrix}$$

Then, by setting $A' := (A \mid I_m)$ and $c' := \begin{bmatrix} c_1 & \ldots & c_n & 0 & \ldots & 0 \end{bmatrix}$ we get the following linear program in equational form:

$$\max c'^T x$$

$$\begin{aligned} A'x &= b \\ x &\geq 0 \end{aligned}$$

Definition 13: Standard form

A linear program is said to be in **standard form** (or *equational form*) if it is formulated as:

$$\max c^T x$$

$$Ax = b$$
$$x \geq 0$$

where $c, x \in \mathbb{R}^n$, $A \in \text{Mat}_{m \times n}(\mathbb{R})$ and $b \in \mathbb{R}^m$.

Through *linear algebra*, we can find numerous results involving linear programs in standard form. In particular, a fundamental result is that given the equation $Ax = b$ of a linear program in standard form, the set of feasible solutions is equal to $\ker(A)$, implying that it is **invariant under Gaussian row operations**. Thus, we can reduce the number of rows of A through row elimination in order to obtain the **minimal amount of constraints** that define the problem. For these reasons, we are going to assume that each removable row has already been removed, meaning that the **rows** (but always not the columns) of A are assumed to be **linearly independent**.

Moreover, since we generally obtain a standard form linear program from a non-standard one and the transformation adds m variables to the problem, we're going to always assume that $m \leq n$. Thus, since $\text{rowrank}(A) = m$ and $m \leq n$, by basic linear algebra results we also get that $\text{colrank}(A) = m$.

Observation 4: Assumptions on standard form LPs

Given a linear program in standard form defined by the equation $Ax = b$, we assume that:

- $A \in \text{Mat}_{m \times n}(\mathbb{R})$ and $m \leq n$
- The rows of the matrix A are always linearly independent
- The rank of A is $\text{rank}(A) = \text{colrank}(A) = \text{rowrank}(A) = m$

2.3 Basic feasible solutions

Consider a linear program in standard form:

$$\max c^T x$$

$$Ax = b$$
$$x \geq 0$$

where $A \in \text{Mat}_{m \times n}$ and we denote with $A = (A^1, \ldots, A^n)$ the column decomposition of A. Given a subset of indices $\mathcal{B} \subseteq \{1, \ldots, n\}$ called **basis**, we define the matrix $A_\mathcal{B}$ as the **matrix formed by the columns of** A whose index is inside $\mathcal{B}$, meaning that $A_\mathcal{B} = (A^{i_1}, \ldots, A^{i_k})$ where $i_1, \ldots, i_k \in \mathcal{B}$.

Example:

- Suppose that:
$$A = \begin{bmatrix} 1 & 6 & 3 & -3 & 4 \\ 0 & 1 & 3 & 2 & 4 \end{bmatrix}$$

  If $\mathcal{B} = \{2, 4\}$, the matrix $A_\mathcal{B}$ is equal to:
$$A_\mathcal{B} = \begin{bmatrix} 6 & -3 \\ 1 & 2 \end{bmatrix}$$

Proposition 6

Given the matrix equation $Ax = b$ and a subset of indices $\mathcal{B} \subseteq \{1, \ldots, n\}$, if $|\mathcal{B}| = m$ and $A_\mathcal{B}$ is *non-singular*, meaning that $\det(A_\mathcal{B}) \neq 0$, then there exists a solution $\overline{x} := (x_1, \ldots, x_n) \in \mathbb{R}^n$ such that $A\overline{x} = b$ and $\forall i \notin \mathcal{B}\ x_i = 0$.

Proof.

If $|B| = m$ and $\det(A_\mathcal{B}) \neq 0$ then $A_\mathcal{B}$ is a non-singular $m \times m$ matrix with rank m. Thus, through the *Rouché-Capelli theorem*, since A_B is also a minor of both A and $(A \mid b)$, we get that $\text{rank}(A) = \text{rank}(A \mid b) = m$, implying that subspace of solutions of $Ax = b$ has rank $n - m$.

Moreover, this subspace will always contain a solution such that $\forall i \notin \mathcal{B}\ x_i = 0$ since the columns whose index isn't in $\mathcal{B}$ are linearly dependent by the other columns.

□

Definition 14: Basic feasible solution

Given the equation $Ax = b$ of a standard form linear program, we say that $\overline{x} \in \mathbb{R}^n$ is a **basic feasible solution (BFS)** if $\overline{x}$ is a feasible solution for which there exists an index set $\mathcal{B} \subseteq \{1, \ldots, n\}$ such that:

- $A_{\mathcal{B}}$ is an $m \times m$ non-singular matrix
- $\forall i \notin \mathcal{B} \ \ \overline{x_i} = 0$

Moreover, we say that $\mathcal{B}$ **certifies** $\overline{x}$ and call **basic variable** each variable x_i such that $i \in \mathcal{B}$, while the other variables are called **non-basic variable**.

Example:

- Suppose that a standard form linear program is defined by:

$$A = \begin{bmatrix} 1 & 6 & 3 & -3 & 4 \\ 0 & 1 & 3 & 2 & 4 \end{bmatrix} \qquad b = \begin{bmatrix} 6 \\ 6 \end{bmatrix}$$

If $\mathcal{B} = \{2, 4\}$, the matrix $A_{\mathcal{B}}$ is equal to:

$$A_{\mathcal{B}} = \begin{bmatrix} 6 & -3 \\ 1 & 2 \end{bmatrix}$$

Moreover, since $\det(A_{\mathcal{B}}) = 15 \neq 0$ and $|\mathcal{B}| = 2$, we know that there exists a vector $\overline{x} = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 & x_5 \end{bmatrix}$ such that $A\overline{x} = b$. By solving the following equation

$$\begin{bmatrix} 6 & -3 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} x_2 \\ x_4 \end{bmatrix} = \begin{bmatrix} 6 \\ 6 \end{bmatrix}$$

we get that $x_2 = 2$ and $x_4 = 2$. Finally, by setting all the other variables to 0, we get the vector $\overline{x} = \begin{bmatrix} 0 & 2 & 0 & 2 & 0 \end{bmatrix}$ is a BFS.

An important thing to notice is that $\overline{x}$ is a BFS because it also respects the constraint of non-negative solutions since $\overline{x} \geq 0$, meaning that it is indeed a feasible solution. The method that was just shown doesn't always guarantee that the solution to the "reduced equation" is also a feasible solution.

For example, if we consider $\mathcal{B}' = \{1, 2\}$, we get that $\det(A_{\mathcal{B}'}) = -5 \neq 0$ and $|\mathcal{B}'| = 2$, so we know that there exists a vector $\overline{y} = \begin{bmatrix} y_1 & y_2 & y_3 & y_4 & y_5 \end{bmatrix}$ such that $A\overline{y} = b$. However, by solving the following equation

$$\begin{bmatrix} 1 & 6 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_3 \end{bmatrix} = \begin{bmatrix} 6 \\ 6 \end{bmatrix}$$

we get that $y_1 = -30$ and $y_2 = 6$, the vector $\overline{y} = \begin{bmatrix} -30 & 6 & 0 & 0 & 0 \end{bmatrix}$ doesn't respect the constraint $\overline{y} \geq 0$, meaning that it isn't a feasible solution and by consequence that it isn't a BFS.

Theorem 4

Given a feasible solution $\overline{x} = \begin{bmatrix} x_1 & \dots & x_n \end{bmatrix}$ to a linear program, let $K = \{i \in \{1, \dots, n\} : x_i > 0\}$. It holds that:

$$\overline{x} \text{ is a BFS} \iff \text{Columns of } A_K \text{ are lin. ind.}$$

Proof.

First implication. Suppose that $\overline{x}$ is a BFS and let $\mathcal{B}$ be the basis through which we obtain $\overline{x}$. Every non-basic variable of $\overline{x}$ is set to 0, meaning that $\forall i \notin \mathcal{B}$ we have that $x_i = 0$. However, since for all the other basic variables x_j it holds that $x_j \geq 0$, these variables could also be set to 0.

Thus, generally we have that $K \subseteq \mathcal{B}$. Since $A_\mathcal{B}$ is non-singular, the columns of $A_\mathcal{B}$ are linearly independent. Then, since $i \in K \subseteq \mathcal{B}$, each column of the matrix A_K is also linearly independent.

Second implication. Suppose that the columns of A_K are linearly independent, implying that $\det(A_K) = 0$. If $|K| = m$ then K certifies that $\overline{x}$ is a BFS. If $|K| < m$, instead, we pick $\mathcal{B} \subseteq \{1, \dots, n\}$ such that:

- $K \subseteq \mathcal{B}$
- $A_\mathcal{B}$ has linearly independent columns
- $\mathcal{B}$ is as big as possible

Such set $\mathcal{B}$ always exists since, eventually, $K = \mathcal{B}$ is allowed by definition.

Since the columns of $A_\mathcal{B}$ are also columns of A, since $\mathcal{B}$ is as big as possible and since $A_\mathcal{B}$ is has linearly independent columns, it holds that $\text{rank}(A_\mathcal{B}) = \text{rank}(A) = m$, implying that $|\mathcal{B}| = m$ and thus that $\mathcal{B}$ certifies that $\overline{x}$ is a BFS.

□

Observation 5

A BFS can be certified by more than one basis

Example:

- Suppose that a standard form linear program is defined by:

$$A = \begin{bmatrix} 1 & 6 & 3 & -3 & 4 \\ 0 & 1 & 3 & 2 & 4 \end{bmatrix} \qquad b = \begin{bmatrix} 6 \\ 6 \end{bmatrix}$$

By the previous theorem, the vector $\overline{x} = \begin{bmatrix} 0 & 0 & 2 & 0 & 0 \end{bmatrix}$ is a BFS due to the fact that $K = \{3\}$ and the columns of A_K are clearly linearly independent since it is made of only one column. Moreover, we notice that we can expand the set K to two different basis $\mathcal{B} = \{1, 3\}$ and $\mathcal{B}' = \{2, 3\}$

Definition 15

Given a basis $\mathcal{B}$ of a linear program, we say that $\mathcal{B}$ is **feasible** if it certifies a BFS

Observation 6

Given a feasible basis $\mathcal{B}$, there exists only one BFS $\overline{x}$ certified by $\mathcal{B}$

Proof.

Let $\mathcal{B} = \{i_1, \dots, i_k\}$. Since $\mathcal{B}$ is a feasible basis, the matrix $A_{\mathcal{B}}$ is made of linearly independent columns, implying that there must be only one solution to the equation

$$A_{\mathcal{B}} \cdot \begin{bmatrix} x_{i_1} \\ \vdots \\ x_{i_k} \end{bmatrix} = b$$

□

Proposition 7

A linear program in standard form has at most $\binom{n}{m}$ BFS

Proof.

Since each basis must have cardinality m and since A has n columns, the number of possible choises for a basis is $\binom{n}{m}$. Then, even if all the possible basis are feasible basis, each one of them has a unique BFS, □