Integer Linear Programming (ILP)

- Linear programming problems at which values of the computed variables must be integers are called **Integer Linear Programming (ILP)** problems.
- If only some of the variables must be integers, we call them **Mixed Integer Linear Programming** problems.
- There is a huge number of problems that could be phrased as ILP. (include many NP-hard problems, where no polynomial-time algorithms exist)
- A few libraries could handle them, including CPLEX.
- Running time could vary a lot, and could be extremely slow for some instances.
- Yet extremely useful for instances when actual running time is acceptable.
- Also useful for comparing fast heuristic to global optimum.

Big difference between LP and ILP:
- If we could express a problem as an LP, practically we could consider it solved: LP is very fast in practice for any realistic input. Libraries are easy to use.
- If we could express a problem as an ILP, the libraries are still easy to use, but running time varies a lot. Not always we will live long enough to see the program terminates.

Application: Bipartite Matching.

- A graph $G(V, E)$ is called bipartite if $V$ can be partitioned into two sets $V = A \cup B$, and each edge of $E$ connects a vertex of $A$ to a vertex of $B$. We sometimes denote these graphs by $G(A \cup B, E)$.
- Example: The set $A = \{a_1, a_2, a_3, a_4\}$ is a set of instructors, the set $B = \{b_1, b_2, b_3, b_4\}$ is the set of courses. There is an edge $(a_i, b_j)$ in $E$ iff instructor $a_i$ could teach course $b_j$.
- A matching is a set of edges $M$ of $E$, where each vertex of $A$ is adjacent to at most one vertex of $B$, and vice versa.
- (in the example, each instructor will teach at most one course, and vice versa)

**Maximum-cardinality matching**: Find a matching with as many edges as possible.

- This problem could be solved with in $O(nm)$ time using Ford-Fulkerson algorithm. Faster algorithms exist as well. However, we will use it as an example to the ease of using ILP.

ILP for Max-Cardinality Bipartite Matching.

- For every edge $e_j \in E$ we define a **Boolean** variable $x_j$.
  - When specifying ILP, to specify that it is a boolean var, we write $0 \leq x_j \leq 1$ and $x_j$ is an int.
  - $x_j = 1$ if $e_j$ participates in the matching $M$.
  - Similarly $x_j = 0$ if $e_j$ does not participate in the matching $M$.
- The goal is to maximize the number of edges in $M$, while keeping $M$ a proper matching. So the ILP is:

  Cost function: maximize $\sum_{j=1}^{m} x_j$

Subject to:

1. $0 \leq x_j \leq 1$ \hspace{1em} for every $x_i$
2. $x_i$ is an integer, \hspace{1em} for every $x_i$
3. for every $a \in A$ $\sum_{e_j \text{ starts at } a} x_j \leq 1$ \hspace{1em} //each instructor teaches at most one course
4. for every $b \in B$ $\sum_{e_j \text{ leads to } b} x_j \leq 1$ \hspace{1em} //each course will be taught by at most one instructor

In the example, at most one of the edges $(a_2, b_1)$ and $(a_3, b_1)$ will be in $M$, since $x_2 + x_3 \leq 1$.
Vertex Cover and ILP

- Given: A graph $G(V,E)$. A subset $C \subseteq V$ is a vertex cover if every edge $(u,v) \in E$ we have either $u \in C$ or $v \in C$ or both.
- Finding the min-cardinality Vertex Cover is NP-Hard
- ILP for this problem: We will use boolean variables $x_1, \ldots, x_n$ (one per vertex).
- $v_i \in C$ if $x_i = 1$ (for $i = 1 \ldots n$)
- Formalizations of ILP:
- Variables: $x_1, \ldots, x_n$
  - Cost function: $\text{minimize } \sum_{i=1}^{n} x_i$
  - Subject to:
    1. $0 \leq x_i \leq 1$ for every $x_i$
    2. $x_i + x_j \geq 1 \ \forall (v_i, v_j) \in E$ // either $v_i$ or $v_j$ are in $C$
    3. Each $x_i$ is an integer,
Flow in Networks

**Definition.** A *flow network* is a directed graph $G = (V, E)$ with two distinguished vertices: a *source* $s$ and a *sink* $t$. Each edge $(u, v) \in E$ has a nonnegative *capacity* $c(u, v)$.

- The capacity limits the number of gallons/seconds (or MB/second, or vehicle/second) that the edge could tolerate.
- **Goal**— push as much flow as possible, from $s$ to $t$.

**Example:**

![Flow Network Diagram](image)

The maximum-flow problem

**Maximum-flow problem:** Given a flow network $G$, find a flow of maximum value on $G$.

In the example, the value of the flow equals $1 + 2 = 3$.

Goal: Assign a flow to every edge, (legally), so the value of the flow is maximize.
Lemma:

The value of the flow equals to the sum of flows entering $t$.

\[
\sum_{(w, v) \in E} f(w, v) = \sum_{(s, w) \in E} f(s, w)
\]

Lemma: The value of the flow equals to the sum of flows entering $t$

LP could solve flow problems (but values might be non-integers)

Given: The graph, special vertices $s, t \in E$, and the capacities $c(u, v)$ for every edge $(u, v) \in E$.

Define a linear programming problem (LP) that will find the maximum (legit) flow:

- **Unknown variables**: $f(u, v)$ for every edge $(u, v) \in E$.
  - If you find the usage of the parenthesis confusing, we could just name the vertices $V = \{v_1, \ldots, v_n\}$ and the variables are $f_{u,v}$ for every edge $(v_i, v_j) \in E$.

- **Cost Function**: Maximizes the flow from $s$.
  \[
  \max \sum_{(u, v) \in E} f(u, v)
  \]

- **Capacity constraint**: $0 \leq f(u, v) + f(v, t) \leq c(u, v)$ for every edge $(u, v) \in E$.

- **Flow conservation**: For every vertex $u \in V$, which is not the source nor the sink, the flow arriving into $u$ must be equal to the total flow that leaves $u$. Formally:
  \[
  \sum_{(w, v) \in E} f(w, v) - \sum_{(v, w) \in E} f(v, w) = 0
  \]

A 2-Approximation for Vertex Cover (Bar-Yehuda Algorithm)

The unweighted case

**Algorithm** VertexCoverApprox($G$)

**Input** graph $G$

**Output** a vertex cover $C$ for $G$

$C_{app} \leftarrow \text{empty set}$

$H \leftarrow E$

/* $H$ – what is left to be covered */

while $H$ has edges (not empty)

$(u, v) \leftarrow \text{An edge of } H$

Add both $u$ and $v$ to $C_{app}$

for each edge $f$ of $H$ incident to $u$ or $v$

Remove $f$ from $H$

/* No need to cover $f$ again */

**Comment**: The approximation algorithm for vertex cover in an unweighted graph (all edges have the same weight). This version is much easier, and does not require linear programming.

- Obviously $C_{app}$ that the algorithm produces is a cover. So $C_{app} \leq C_{opt}$.
- Let $C_{opt}$ be an optimal vertex cover.
- Every edge $(u,v)$ chosen by the algorithm has both $u$ and $v$ in $C_{app}$.
- But $(u,v)$ must be covered by at least one vertex of $C_{app}$, so $C_{app}$ contains either $u$ or $v$ or both.
- That is, for every single vertex in $C_{app}$ there are at most two vertices in $C_{app}$.
- We proves that $C_{app} \leq 2 \cdot C_{opt}$.
- That is, $C_{app}$ is a 2-approximation of $C_{opt}$.
- Running time: $O(|E|)$ if the graph is stored as an adjacency list.
**Min-Weight Vertex Cover: Exact and Approximated solutions:**

- Sometimes the LP (instead of the ILP) could help us finding good approximations.
- Given a graph \( G(V,E) \), a subset \( U \subseteq V \) is a vertex cover if every edge \((v, w) \in E\). at least one of its endpoints is in \( U \).
- Each vertex \( v \in V \) is given with a cost (or weight) \( w(v) > 0 \). (in contrast, in the unweighted case, we could assume that the cost is the same for all vertices).
- We also define the cost of \( C \), is defined as the sum of costs of vertices in \( C \).

**Finding the min-cardinality or min-weight Vertex Cover is NP-hard:**

- We phrase the problem as an ILP. However, if running time is too large, we will compromise on LP plus some other manipulations.

**Let's start with ILP for this problem:**

- The variables are \( x_i = \{0, 1\} \) is the number of variables.
- All the variables are integers and between 0 and 1.
- The vertex \( v_i \) is in the cover \( C \) if \( x_i = 1 \) for \( i = 1 \ldots n \).
- \( x_i \) set to \( 0 \) if \( i \notin C \).
- Set \( \bar{x} = \{x_1, \ldots, x_n\} \).

**Example:**

\[ C^{opt} \in \{v_1, v_4\}, \quad \text{and } \bar{x}(C^{opt}) = 4 + 8 = $12 \]

**ILP: Cost function:**

-\[ \text{minimize } \bar{x} \cdot \bar{t} = \sum_{i=1}^{n} x_i t_i \]

-\[ x_i + \bar{t}_i \geq 1 \quad \forall (v_i, v_j) \in E. \]

-\[ 0 \leq x_i \leq 1 \quad \forall i \in V. \]

**Example:**

\[ \bar{x}(C^{opt}) = \{v_1, v_4\}, \quad \text{and } \bar{x}(C^{opt}) = 4 + 8 = $12 \]

**Claim:**

1. \( C^{opt} \) is a vertex cover. To prove this fact, think about an edge \((v_i, v_j) \in E \). It cannot be that both \( x_i < 0.5 \) and \( x_j < 0.5 \), otherwise, they are not solution to the LP.
2. \( \bar{x} \cdot \bar{t} \leq \bar{x} \cdot \bar{t}^{opt} \). To see this fact, remember that to obtained \( \bar{x} \cdot \bar{t}^{opt} \) we required the same constraint as in the LP, but add more requirement (integrity), so the cost we pay could only increase.
3. \( \bar{x} \cdot \bar{t} \geq \bar{x} \cdot \bar{t}^{opt} \) for every \( t_i \). Proof: There are two cases, if \( x_i \neq 0 \) then the claim is trivially correct. If \( x_i = 0 \) then \( x_i \leq 0.5 \), happened because \( x_i \geq 0.5 \). QED.
4. Putting it together: \( \bar{x} \cdot \bar{t} \leq \bar{x} \cdot \bar{t}^{opt} \leq \bar{x} \cdot \bar{t}^{opt} \).

In words:

The approximation costs: \( \leq 2 \cdot opt \),

We say that \( C^{opt} \) is a 2-approximation of the weighted vertex cover.