# Stanford MS&E/CS 331: Integer programming formulations

Ellen Vitercik*

September 30, 2025

A goal of this week's lectures is to pin down a common language for discussing optimization problems, especially in the discrete setting. This common framework allows us to talk about optimization at a useful level of abstraction, focusing on two central paradigms: integer programming (IP) and constraint satisfaction programming (CSP). Grounding our discussions in IP and CSP enables us to design machine learning methods that yield broadly applicable tools for solving diverse optimization problems.

Integer programming has many applications across science and engineering, including scheduling, routing, planning, manufacturing, and finance. This lecture will cover how to formulate discrete optimization problems as integer programs.

At a high level, there are three basic components of an optimization problem:

1. **Decision variables**: these variables describe choices that are under our control.
2. **Objective function**: this is the criterion we want to minimize (for example, minimizing cost) or maximize (for example, maximizing profit).
3. **Constraints**: these are limitations restricting our choices for the decision variables.

An *integer linear program* (the focus of this module) is an optimization problem where the objective function is linear, each constraint is a linear inequality or equality, and some decision variables must be integer-valued, which typically makes the optimization problem NP-hard.

## 1 Examples

We will start with a variety of different examples before discussing integer programming more abstractly.

### 1.1 Minimum vertex cover (MVC)

A *vertex cover* of a graph $G = (V, E)$ is a subset $S \subseteq V$ such that every edge $(i, j) \in E$ is incident to a vertex in $S$, i.e., $i \in S$, $j \in S$, or both. In the MVC problem, the goal is to find the smallest vertex cover.

We will begin by identifying the three basic components of this optimization problem:

*These notes are course material and have not undergone formal peer review. Please feel free to send me any typos or comments.

1. **Decision variables:** for each vertex $i \in V$, we define the decision variable

$$x_i = \begin{cases} 1 & \text{if } i \text{ is in the vertex cover} \\ 0 & \text{else.} \end{cases}$$

2. **Objective function:** since our goal is to minimize the size of the vertex cover, our objective function is to minimize

$$\sum_{i \in V} x_i,$$

which is a linear function.

3. **Constraints:** We must design the constraints so that if an assignment of the decision variables $x_1, \ldots, x_{|V|}$ satisfies the constraints, then $\{i : x_i = 1\}$ is a vertex cover. To do so, we will add the constraint $x_i + x_j \geq 1$ for all edges $(i, j) \in E$. This ensures that for every edge, $x_i = 1$ and/or $x_j = 1$.

Putting these ingredients together, we have the MVC integer program:

$$\begin{array}{lll} \text{minimize} & \displaystyle\sum_{i \in V} x_i & \\ \text{subject to} & x_i + x_j \geq 1 & \text{for all } (i, j) \in E \\ & x_i \in \{0, 1\} & \text{for all } i \in V. \end{array}$$

### 1.2 Maximum independent set (MIS)

The maximum independent set integer program is very similar to the MVC integer program. A set $S \subseteq V$ is an *independent set* if no vertices in $S$ are connected by an edge. In the MIS problem, the goal is to find the largest independent set. At this point, I'd recommend trying to write the MIS integer program yourself before reading further.

As before, we will identify the three basic components of this integer program:

1. **Decision variables:** for each vertex $i \in V$, we define the decision variable

$$x_i = \begin{cases} 1 & \text{if } i \text{ is in the independent set} \\ 0 & \text{else.} \end{cases}$$

2. **Objective function:** Since we aim to maximize the size of the independent set, our goal will be to maximize

$$\sum_{i \in V} x_i.$$

3. **Constraints:** Finally, we must define the constraints so that if $x_1, \ldots, x_{|V|}$ satisfy the constraints, then $\{i : x_i = 1\}$ is an independent set. To this end, we add the constraint $x_i + x_j \leq 1$ for all $(i, j) \in E$. This constraint ensures that for every edge, either $x_i = 1$, $x_j = 1$, or $x_i = x_j = 0$.

Putting these pieces together, we get the MIS integer program: MIS integer program:

$$\begin{array}{lll} \text{maximize} & \displaystyle\sum_{i \in V} x_i & \\ \text{subject to} & x_i + x_j \leq 1 & \text{for all } (i, j) \in E \\ & x_i \in \{0, 1\} & \text{for all } i \in V. \end{array}$$

### 1.3 Warehouse location

We wrap up this section with an integer program for a more practical problem [1]. The manager of a company that produces some goods must decide which of $n$ warehouses to open to meet the demands of $m$ customers. Her decision depends on the following values:

- If the manager chooses to open warehouse $i \in [n]$, she must pay a fixed cost $f_i \geq 0$.
- The company has committed to meeting the demand $d_j \geq 0$ of each consumer $j \in [m]$. This is the number of units of the company's product that the consumer demands.
- Finally, there is a transportation cost of $c_{ij} \geq 0$ to ship each unit of the good from warehouse $i$ to customer $j$.

The manager's goal is to minimize their total operating and transportation costs while ensuring that all of the customers' demands are fulfilled. We now identify the three basic components of this optimization problem:

1. **Decision variables:** there are two types of decision variables. For each warehouse $i \in [n]$, we define the decision variable

$$y_i = \begin{cases} 1 & \text{if warehouse } i \text{ is opened} \\ 0 & \text{else.} \end{cases}$$

   Moreover, we define the decision variable $x_{ij}$ to be the number of units that are sent from warehouse $i$ to customer $j$. For simplicity, this amount may be fractional, so we will only require that $x_{ij} \geq 0$.

2. **Objective function:** the goal is to minimize the total transportation and opening costs, i.e.,

$$\underbrace{\sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}x_{ij}}_{\text{Transportation costs}} + \underbrace{\sum_{i=1}^{n} f_i y_i}_{\text{Opening costs}} .$$

3. **Constraints:** there are several categories of constraints. First, we require that $x_{ij} \geq 0$ and $y_i \in \{0, 1\}$. Second, for each customer $j \in [m]$, the total amount of goods sent to them—across all $n$ warehouses—must equal their demand, meaning that

$$\sum_{i=1}^{n} x_{ij} = d_j.$$

   Finally, goods can only be shipped from a warehouse if that warehouse is open—a relationship we must enforce between the $x_{ij}$ and $y_i$ variables. If $y_i = 0$, warehouse $i$ is not opened, so it cannot ship any goods to any customers, meaning that

$$y_i = 0 \qquad \Rightarrow \qquad \sum_{j=1}^{m} x_{ij} = 0. \tag{1}$$

Meanwhile, if $y_i = 1$, warehouse $i$ can ship any number of units to the customers, and the total amount it ships should only be constrained by the total demand. In other words,

$$y_i = 1 \qquad \Rightarrow \qquad \sum_{j=1}^{m} x_{ij} \leq \sum_{j=1}^{m} d_j. \tag{2}$$

We can encode Equations (1) and (2) with the following constraint:

$$\sum_{j=1}^{m} x_{ij} \leq y_i \sum_{j=1}^{m} d_j.$$

Putting these pieces together, we obtain the warehouse location integer program:

$$\begin{array}{lll}
\text{maximize} & \displaystyle\sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}x_{ij} + \sum_{i=1}^{n} f_i y_i & \\
\text{subject to} & \displaystyle\sum_{i=1}^{n} x_{ij} = d_j & \text{for all } j \in [m] \\
& \displaystyle\sum_{j=1}^{m} x_{ij} \leq y_i \sum_{j=1}^{m} d_j & \text{for all } i \in [n] \\
& x_{ij} \geq 0 & \text{for all } i \in [n], j \in [m] \\
& y_i \in \{0, 1\} & \text{for all } i \in [n].
\end{array}$$

## 2 General form of an integer program

In general, an integer program can be written in the following general form:

$$\begin{array}{lll}
\underset{x_1,\ldots,x_n}{\text{maximize}} & \displaystyle\sum_{j=1}^{n} c_j x_j & \\
\text{subject to} & \displaystyle\sum_{j=1}^{n} a_{ij} x_j \leq b_i & \text{for all } i \in [m] \\
& x_j \geq 0 & \text{for all } j \in [n] \\
& x_j \in \mathbb{Z} & \text{for some or all } j \in [n].
\end{array}$$

An equality constraint

$$\sum_{j=1}^{n} a_{ij} x_j = b_i$$

can be written using two inequality constraints:

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i \qquad \text{and} \qquad -\sum_{j=1}^{n} a_{ij} x_j \leq -b_i.$$

Moreover, if we aim to minimize a linear objective $\sum c_j x_j$, we can simply maximize $-\sum c_j x_j$.

It is typical to write integer programs using vector and matrix notation, with $\boldsymbol{b} = (b_1, \ldots, b_m)$, $\boldsymbol{c} = (c_1, \ldots, c_n)$, and

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix}.$$

The integer program is written as

$$\begin{array}{ll} \text{maximize} & \boldsymbol{c}^T \boldsymbol{x} \\ \text{subject to} & A\boldsymbol{x} \leq \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0} \\ & x_j \in \mathbb{Z} \quad \text{for some or all } j \in [n]. \end{array} \tag{3}$$

## 3 Linear programming

If we ignore the integrality constraint in Equation (3), we obtain the integer program's *linear programming relaxation*:

$$\begin{array}{ll} \text{maximize} & \boldsymbol{c}^T \boldsymbol{x} \\ \text{subject to} & A\boldsymbol{x} \leq \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0}. \end{array} \tag{4}$$

Unlike integer programs, linear programs are efficiently solvable. The following is an important fact that underpins how IP solvers operate.

**Fact 3.1.** *Let $\boldsymbol{x}^*_{IP}$ be the optimal solution to Equation (3) and let $\boldsymbol{x}^*_{LP}$ be the optimal solution to Equation (4). Then $\boldsymbol{c}^T \boldsymbol{x}^*_{LP} \geq \boldsymbol{c}^T \boldsymbol{x}^*_{IP}$.*

This fact follows from the observation that we can only improve the solution to this maximization problem by removing the integrality constraints.

## 4 Integer programming solvers

Modern integer programming solvers such as Gurobi are powered by the branch-and-bound algorithm [2]. This framework is used in real-world, high-stakes applications: the NFL relies on it to generate full season schedules, airlines use it for flight and crew planning, and power grid operators depend on it for deciding which generators to run and when.

Branch-and-bound organizes the problem into a search tree of subproblems (created by fixing or bounding variables). At each node, the solver computes a bound by solving the LP relaxation of that subproblem. If the LP solution is fractional, the solver *branches*—fixing a variable to 0 or 1—and continues exploring. Each relaxation provides a bound on the best possible solution in that subtree. By maintaining the best incumbent (the best integer solution found so far), the solver can prune subtrees whose bounds cannot improve on it, dramatically shrinking the search space.

In practice, solvers also employ a range of accelerators: heuristics to quickly generate good incumbents, warm starts that exploit existing solutions or predictions, and extensive parameter tuning, including cutting planes, heuristics, and parallelism.

## References

[1] Stephen P Bradley, Arnoldo C Hax, and Thomas L Magnanti. *Applied mathematical programming*. Addison-Wesley, 1977.

[2] Ailsa H Land and Alison G Doig. An automatic method of solving discrete programming problems. *Econometrica*, pages 497–520, 1960.