Harvard CS 182: Artificial Intelligence September 27, 2021

Lecture 7: Integer Programming

Lecturer: Ariel Procaccia *Author: Zuzanna Skoczylas*

1 Introduction

In this lecture we cover *integer programming*, which solves linear programs where the set of feasible solution includes only integer values.

2 Integer Programming

2.1 Definition

Definition 1 (Feasibility Problem) *Find* $x_1, \ldots, x_l$ *such that*

$$\forall i \in [k], \sum_{j=1}^{l} a_{ij} x_j \leq b_i$$
$$\forall j \in [l], x_j \in \mathbb{Z}.$$

Notation $i \in [k]$ means i is in the set $\{1, 2, \ldots, k\}$. In other words, the Integer Programming Feasibility Problem asks to find values for all x_i such that all k constraints are satisfied.

Definition 2 (Optimization Problem)

$$\max \sum_{j=1}^{l} c_j x_j$$

such that

$$\forall i \in [k], \sum_{j=1}^{l} a_{ij} x_j \leq b_i$$
$$\forall j \in [l], x_j \in \mathbb{Z}.$$

In other words, the Integer Programming Optimization Problem asks to find values for all x_i such that all k constraints are satisfied and the optimization function is maximized.

For both problems it is possible to express constraints of the form $a_{ij}x_j \leq b_i$ by multiplying coefficients by -1. Furthermore, the optimization problem can also be looking for the minimum of the function; this again can be achieved by multiplying coefficients by -1.

The optimal solutions for a linear program is always at least as large as for the corresponding integer program. This is because all solutions to IP are valid for LP, but not the other way around.

2.2 Convexity

Integer Programming is not convex because the feasible set is not convex. The only possible solutions are discrete points. You can imagine drawing them spatially and notice that there is space between the points, which implies that set of feasible solutions is not convex.

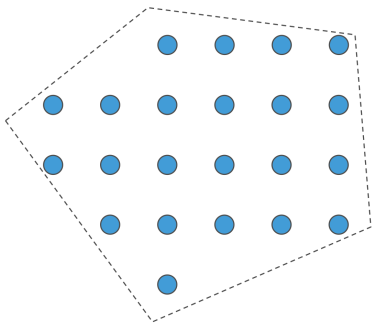

Figure 1: IP Is Not Convex

3 Integer Programming Feasibility Examples

3.1 Sudoku

Sudoku can be defined as an integer program where the player has to find a feasible solution. Define variable x^{ij} to be the value in the entry (i, j). Furthermore, define $x_k^{ij} = 1$ iff $x^{ij} = k$ and 0 otherwise. Consider the set S_t for $t \in [27]$ to be the set of all rows, columns and 3x3 squares.

Example 1 (Sudoku) *Find* $x_1^{11}, \ldots, x_9^{99}$ *such that*

$$\forall t \in [27], \forall k \in [9], \sum_{(i,j) \in S_t} x_k^{ij} = 1$$
$$\forall i, j \in [9], \sum_{k \in [9]} x_k^{ij} = 1$$
$$\forall i, j, k \in [9], \sum_{k \in [9]} x_k^{ij} \in \{0, 1\}$$

The first constraint imposes that each set S_t has exactly one of each of the numbers $1, \ldots, 9$. The second constraint implies that each cell is assigned exactly one number. Lastly, the third constraint imposes that each value x_k^{ij} is 0 or 1 .

3.2 Fair division

Definition 3 (Envy-Free) *Suppose we have players* $N = \{1, \ldots, n\}$ *and items* $M = \{1, \ldots, M\}$*, and player* i *has value* v_{ij} *for item* j*. Partition the items into bundles* $A_1, \ldots, A_n$*. We say that the* $A_1, \ldots, A_n$ *is envy-free iff*

$$\forall i, i', \sum_{j \in A_i} v_{ij} \geq \sum_{j \in A_{i'}} v_{ij}.$$

We can define envy-free division as an IP.

Example 2 (Fair division) *Find* $x_{11}, \ldots, x_{nm}$ *such that*

$$\forall i \in N, \forall i' \in N, \sum_{j \in M} v_{ij} x_{ij} \geq \sum_{j \in M} v_{ij} x_{i'j}$$
$$\forall j \in M, \sum_{i \in N} x_{ij} = 1$$
$$\forall i \in N, j \in M, x_{ij} \in \{0, 1\}$$

The first constraint is the envy-freeness constraint. The second constraint requires that each item be owned by precisely one player. Lastly, the third constraint imposes that each value x^{ij} is 0 or 1.

4 Integer Programming Optimization Examples

4.1 MMS Allocation

Let us revisit the fair division setting. The maximin share guarantee of player i is defined as

$$\max_{X_1,\ldots,X_n} \min_{k\in[n]} \sum_{j\in X_k} v_{ik}.$$

In words, the maximin share guarantee of player i is the value they could guarantee if they divided the items into n bundles $X_1, \ldots, X_n$, but then received the least valuable bundle.

Example 3 (MMS Guarantee) *For player i MMS Guarentee(MMS(i)) is*

$$\begin{aligned}
&\max D \\
&\forall k \in N, \sum_{j\in M} v_{ij} y_{jk} \geq D \\
&\forall j \in M, \sum_{k=1}^{n} y_{jk} = 1 \\
&\forall j \in M, k \in N, y_{jk} \in \{0,1\}
\end{aligned}$$

Here the binary variables y_{jk} are 1 if and only if item j is allocated to bundle k. This implements the maximin objective because we maximize D, where D is the minimum value of any bundle. The reason D is the minimum is because of the first constraint that assures that the value of each bundle is at least D. In trying to maximize D, we will set it to be exactly equal to the least valuable bundle.

An MMS Allocation is an allocation such that for each player i $v_i(A_i) \geq MMS(i)$.

Example 4 (MMS Allocation) *Find $x_{11}, \ldots, x_{nm}$ such that*

$$\begin{aligned}
&\forall i \in N, \sum_{j\in M} v_{ij} x_{ij} \geq MMS(i) \\
&\forall j \in M, \sum_{i\in N} x_{ij} = 1 \\
&\forall j \in M, i \in N, x_{ij} \in \{0,1\}
\end{aligned}$$

4.2 Kidney Exchange

Frequently, patients who need kidneys and their potential donors are not compatible. However, they could potentially find another pair of people with the same problem and swap. Even better, there can be a longer cycle of donations such that everyone will receive a

kidney. This problem can be modeled using the Cycle-Cover problem on a directed graph where each vertex corresponds to a donor-patient pair, and there is a directed edge from u to v is the donor of u is compatible with the patient of v.

Definition 4 (Cycle-Cover) *Given a directed graph G and $L \in N$, find a collection of disjoint cycles of length $\leq L$ in G that maximizes the number of covered vertices.*

Solving CC is easy for $L = 2$ or an unbounded value of L, but hard for any constant $L \geq 3$.

Example 5 (Cycle-Cover) *We can model this problem as an IP where for each cycle c of length $l_c \leq L$, variable $x_c \in \{0, 1\}, x_c = 1$ iff c is included in the cover.*

$$max \sum_c x_c l_c$$
$$\forall v \in V, \sum_{c:v \in c} x_c \geq 1$$
$$\forall c, x_c \in \{0, 1\}$$

UNOS, a non-profit scientific and educational organization that administers the only organ procurement and transplantation network in the United States, is using an IP-based algorithm for their kidney exchange.

5 Branch and Bound

We know that Linear Programming takes polynomial time and gives us an "admissible heuristic," so we can relax the IP constraints and use a search tree to assign variables one by one, solving the LP at every node.

The IP problem is:

$$\max \sum_{j=1}^{l} c_j x_j$$

such that

$$\forall i \in [k], \sum_{j=1}^{l} a_{ij} x_j \leq b_i$$
$$\forall j \in [l], x_j \in \{0, 1\}$$

In the corresponding LP problem, only the last condition is changed:

$$\forall j \in [l], x_j \in [0, 1]$$

The high-level algorithm is the following:

1. Initialize a queue to hold a partial solution with none of the variables of the problem assigned.
2. Loop until the queue is empty:
   (a) Take a node N off the queue
   (b) If N represents a single candidate solution x and $f(x) > B$, then x is the best solution so far. Record it and set $B \leftarrow f(x)$.
   (c) Else, branch on N (consider both options 0 and 1) to produce new nodes N_i. For each of these:
      i. if $bound(N_i) < B$, do nothing. Since the upper bound on this node is smaller than the best solution found so far, it will never lead to the optimal solution, and can be discarded.
      ii. Else, store N_i on the queue.

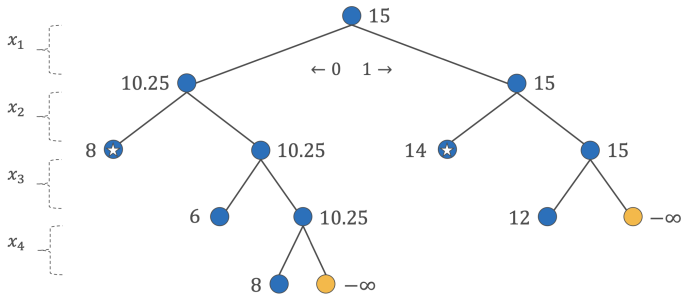

Figure 2: Branch and Bound Lecture Example

In the example, starred nodes are those where the LP solution happens to be integral. Pruning doesn't only happen when the node has an integral solution, but also when its LP solution (which is an upper bound) is lower than the best known solution so far.

In particular, after the algorithm has assigned 0 to x_1 and x_2, the LP yields an integral solution with value 8; we can store it as the best feasible solution so far. Now, after assigning 0 to x_1, 1 to x_2, and 0 to x_3, we get an LP solution with value 6. Since this is an upper bound on any integral assignment to the other variables, there is no point continuing down this branch, as the current incumbent with value 8 is superior.