Integer Programming – revenge of the MILPs

Guillaume Dalle (ENPC, CERMICS)

REOP - Class 7 (24/11/2021)

Contents

1 Homework

Exercise: Dynamic programming for the knapsack problem

The knapsack problem consists in filling a bag by choosing from a set of items $i \in \{1, ..., n\}$ with weights w_i and prices p_i. Its decision version asks whether there is a subset with total weight $\leq W_{\max}$ but total price $\geq P_{\min}$. Propose a dynamic programming algorithm to compute the value function

$$P(n, W) = \text{maximum price for a bag of weight} \leq W \text{ with the } n \text{ first items}$$

Is it enough to prove that the knapsack problem is in **P**?

Exercise (Meunier, PL12): 2-approximation of VERTEX COVER with an LP

1. Find a MILP formulation for the minimum vertex cover problem
2. Propose a way to build a vertex cover from a solution to the linear relaxation of the previous MILP.
3. Prove that the solution you obtain at at most twice as large as the minimum vertex cover.

Exercise 10.4

Exercise 10.5

2 Solving Integer Linear Programs

2.1 Integral polyhedra

A polyhedron P is integral if $P = \text{conv}(P \cap \mathbb{Z}^n)$. In particular, all its vertices have integer coordinates, which means that

- any LP with P as feasible set has one integral optimal solution
- the simplex algorithm finds such a solution

In these cases, solving a MILP is just as easy as solving its continuous relaxation.

2.1.1 Total Unimodularity

A matrix A is Totally Unimodular (TU) if the determinant of all its square submatrices (obtained by removing rows and/or columns) is in $\{-1, 0, 1\}$. In particular, all of its coefficients must be in $\{-1, 0, 1\}$ too.

Theorem 9.8 - variant (Importance of TU matrices): If A is TU and $b \in \mathbb{Z}^m$, then $P = \{Ax = b, x \geq 0\}$ is integral.

Proof: For any vertex x of P, there is a basis B such that x is the basic feasible solution associated with B. This implies that $x_N = 0$ and $x_B = A_B^{-1}b \geq 0$. Since A is TU and A_B is invertible, the determinant of A_B is in $\{-1, 1\}$. By the comatrix formula, $A_B^{-1} = \frac{1}{\det A_B}\mathrm{com}(A)^\top$, which means A_B^{-1} has integer coefficients. Since b does too, we conclude that $x \in \mathbb{Z}^n$.

Proposition 9.10 (Poincaré criterion for TU): Let $A \in \{-1, 0, 1\}^{m \times n}$. If A contains at most one 1 and one -1 per column, then A is TU.

Proof: By recursion.

Corollary 9.11 (Integrality of the flow polyhedron): The incidence matrix of a direted graph is TU.

How to build new TU matrices? If A is TU, then the following matrices are too: $A^\top$, $-A$, $(A, \pm A)$, (A, e_j), any matrix obtained multiplying a row or column of A by -1.

2.1.2 Examples

Reminder: a matching M in a graph G is a subset of pairwise disjoint edges.

Exercise (Maximum weight matching)

We consider a bipartite graph $G = (L \cup R, E)$ and a weight function $w : E \to \mathbb{R}$. The maximum weight matching problem (or assignment problem) looks for a matching M that maximizes $\sum_{e \in M} w_e$.

1. Show that it can be modeled as a flow problem.
2. Propose two polynomial algorithms to solve it.

Remarks:

- There are other algorithms to find maximum matchings in bipartite graphs, such as the Hungarian (Kuhn-Munkres) algorithm. For non-bipartite graphs, this is necessary since the flow reformulation does not apply.
- There are other matching problems with very different solution methods: the stable matching problem and stable roommates problem are two examples.

Exercise (Meunier, PLNE6)

A company has a set J of possible jobs to do. Each job $j \in J$ is characterized by a start date s_j, an end date e_j and a benefit b_j. At every time, the number of active jobs cannot exceed c. The company must choose the subset $X \subset J$ of jobs that bring maximum benefit while remaining feasible.

1. Formulate this problem as an Integer Linear Program.
2. An interval matrix is a matrix (a_{ij}) with coefficients in $\{0, 1\}$ such that if $a_{kj} = a_{lj} = 1$ for some couple $k \leq l$, then $a_{ij} = 1$ for all $i \in [k, l]$. Using the fact that interval matrices are TU, prove that the simplex algorithm can be used to solve this problem.

2.2 Branch & Bound

Branch & Bound speeds up naive enumeration by discarding useless cases. It is still exponentially slow in the worst case, but it works very well in practice.

Remark: B&B is a general principle which applies beyond Integer Programming (see course notes for a more abstract formulation of the algorithm).

2.2.1 Principle of the algorithm

B&B builds a binary search tree where each node corresponds to a region of the solution space. The root node contains the initial problem, say (P), and its children are obtained by splitting the domain of an integer variable x_j in two. The left child contains the problem $(P) \cap [x_j \leq k]$, and the right child contains the problem $(P) \cap [x_j \geq k+1]$ Then, we keep going like this, splitting nodes and diving further into the tree.

Luckily, we usually don't have to enumerate all the nodes, because we can prune some of them:

- Prune by infeasibility: If the continuous relaxation at a node has no solution, then we don't need to keep looking, because there will be no integer solution either in this region of the space.
- Prune by optimality: If the relaxation at a node returns an integer solution, then we don't need to keep looking because we already know the best candidate in this region of the space.
- Prune by bound (the most important): If the relaxation at a node has a value higher than the best solution found sofar (in the whole tree), then we don't need to keep looking because no matter what we find in this region of the space, it will be worse than the best candidate we already have.

If neither of these three bounding techniques applies, then we must split the current node. To do that, we usually choose x_j and k based on a fractional component of the optimal solution x^* to the continuous relaxation: the additional constraints are $x_j \leq \lfloor x_j^* \rfloor$ and $x_j \geq \lceil x_j^* \rceil$.

Practical illustration in Julia

Open this url https://gdalle.github.io/IntroJulia/notebooks/branch_bound.jl.html and complete the code cells to manually apply B&B on a very simple example.

2.2.2 Improving efficiency

Tuning Branch & Bound is often necessary, and commercial solvers include many complicated heuristics meant to speed up the search. Among the most important design choices, we can list:

- Choice of the variable to branch on
- Choice of the next node to explore

In addition, several methods can be used to obtain tighter relaxations, which lead to better lower bounds and more efficient pruning:

- Reformulations
- Decomposition techniques
- Valid inequalities $\implies$ branch and cut

Even though B&B is guaranteed to return an optimal solution with enough time, we often decide to stop once a desired accuracy is reached.

Exercise 10.3 (approximate B&B)

3 Preparing for the KIRO