ORF 522

Linear Programming and Convex Analysis

Integer Linear Programming

Marco Cuturi

Reminder

- Integer programming formulations
  - Interest of integer programming to model real-life problems
  - Examples of reformulations
  - Relaxations and strong formulations

    - ▷ $X = \{\mathbf{x} \in \mathbb{N}^n \mid A\mathbf{x} \geq \mathbf{b}, A \in \mathbb{Z}^{m\times n}, \mathbf{b} \in \mathbb{Z}^m\}$ feasible set of an IP.
    - ▷ If we were able to defined a matrix $M \in \mathbb{Z}^{p\times n}$ (with $p \gg m$ usually) such that

      $$\langle X \rangle = \{\mathbf{x} \in \mathbb{R}^{\boldsymbol{n}} \mid M\mathbf{x} \geq \mathbf{d}, M \in \mathbb{Z}^{p\times n}, \mathbf{d} \in \mathbb{Z}^p\}$$

      then we would be saved: solution is an extreme point, hence integer.
    - ▷ Very hard to obtain M directly.
    - ▷ Instead look for $M, \mathbf{d}$ such that the two sets are not too different.

Today

- Integer programming algorithms
  - Cutting plane methods
  - Branch & Bound, Branch & Cut
  - Dynamic Programming
- Duality for IP.

Methods

Overview of Methods

- Three main categories of algorithms:

  ○ **Exact** algorithms: guaranteed to find an exact optimum, but may take **exponential time**.
    - ▷ cutting plane methods
    - ▷ branch & bound, branch & cut
    - ▷ dynamic programming (for some problems)

  ○ **Approximation** algorithms: polynomial time with a bound on suboptimality. Only work as specialized solutions that use advanced tricks.

  ○ **Heuristic** algorithms: no theoretical guarantee at all, but acceptable to good practical performance. Usually fall in local-minima.

- We will focus on exact algorithms.

Cutting Plane Methods

Updating recursively the relaxation of an IP

- Remember that for an integer program (IP)

$$\begin{array}{ll} \text{minimize} & \mathbf{c}^T\mathbf{x} \\ \text{subject to} & A\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \in \mathbb{N} \end{array}$$

its **linear programming relaxation** (LPR) is defined as

$$\begin{array}{ll} \text{minimize} & \mathbf{c}^T\mathbf{x} \\ \text{subject to} & A\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \geq 0 \end{array}$$

- The main idea behind **cutting plane algorithms**:

1. Solve (LPR), get an optimal solution $\mathbf{x}^\star$.
2. If $\mathbf{x}^\star$ is integer, stop with that solution for (IP).
3. If not, **add an inequality constraint** to (LPR) that integer solutions of (IP) satisfy but that $\mathbf{x}^\star$ does not. Go back to 1.

Gomory Cutting Plane Algorithm

- Gomory (1958) proposed a way to **generate** such inequalities with the **simplex**.
- Suppose we have an optimum $\mathbf{x}^\star$ of (LPR) with index set $\mathbf{I} = \{i_1, \cdots, i_m\}$.
- We write $\mathbf{O}$ for $\{1, \cdots, n\} \setminus \mathbf{I}$.
- As usual, $\mathbf{x}^\star_{i_k} = (B_\mathbf{I}^{-1}\mathbf{b})_k$ and $\mathbf{x}^\star_j = 0$ when $j \in \mathbf{O}$.
- For **any feasible integer** solution $\mathbf{x}$ of (IP), $A\mathbf{x} = \mathbf{b}$ can be decomposed as

$$B_\mathbf{I}^{-1} A_\mathbf{I} \mathbf{x}_\mathbf{I} + B_\mathbf{I}^{-1} A_\mathbf{O} \mathbf{x}_\mathbf{O} = B_\mathbf{I}^{-1}\mathbf{b}, \text{ or equivalently}$$

$$\mathbf{x}_\mathbf{I} + B_\mathbf{I}^{-1} A_\mathbf{O} \mathbf{x}_\mathbf{O} = B_\mathbf{I}^{-1}\mathbf{b}.$$

- For any $1 \leq j \leq n$, we write $\mathbf{y}_j = B_\mathbf{I}^{-1}\mathbf{a}_j$. For each i_k of $\mathbf{I}$, the kth line of the vector equality above yields

$$x_{i_k} + \sum_{j \in \mathbf{O}} (\mathbf{y}_j)_k \, x_j = (B_\mathbf{I}^{-1}\mathbf{b})_k = \mathbf{x}^\star_{i_k}$$

Non-integer optimal solution and constraint derivation

- There must be an index in $\mathbf{I}$ such that $\mathbf{x}^\star_{i_k}$ is fractional. Suppose it is i_r. Then,

$$x_{i_r} + \sum_{j\in\mathbf{O}} \lfloor(\mathbf{y}_j)_r\rfloor x_j \leq x_{i_r} + \sum_{j\in\mathbf{O}} (\mathbf{y}_j)_r \, x_j = \mathbf{x}^\star_{i_r}$$

- Since the x_j are integers,

$$x_{i_r} + \sum_{j\in\mathbf{O}} \lfloor(\mathbf{y}_j)_r\rfloor x_j \leq \lfloor \mathbf{x}^\star_{i_r} \rfloor$$

- This inequality is valid for all **integer** solutions
- It is invalid for $\mathbf{x}^\star$ since $\mathbf{x}^\star_j = 0$ and $\mathbf{x}^\star_{i_r} \neq \lfloor \mathbf{x}^\star_{i_r} \rfloor$ $\Rightarrow$ what we wanted.
- Practical implementation?
  - add constraints: **dual simplex**.
  - Performance is relatively poor. More structure is needed to improve the cuts.

Branch-and-bound / Branch-and-cut

Branch-and-bound

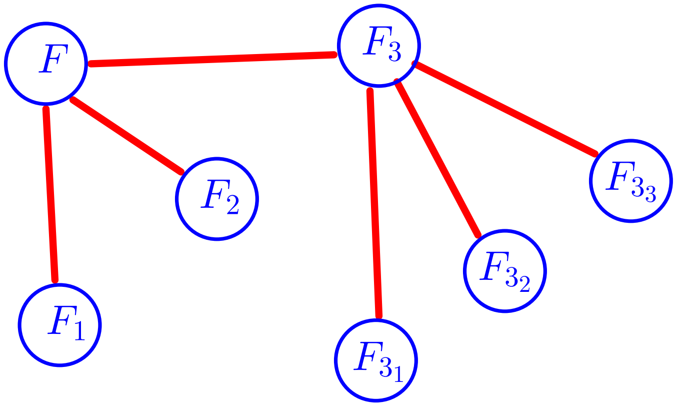
- Family of algorithms proposed in the 60's
- Use the *divide and conquer* approach to tackle an optimization problem.
- Start from a program

$$\begin{array}{ll}\text{minimize} & \mathbf{c}^T\mathbf{x}\\ \text{subject to} & \mathbf{x}\in F\end{array}$$

  divide it into subprograms, where $\bigcup_{i=1}^{k} F_i = F$, to compute for each $i = 1, \cdots, k$

$$\begin{array}{ll}\text{minimize} & \mathbf{c}^T\mathbf{x}\\ \text{subject to} & \mathbf{x}\in F_i,\end{array}$$

- A subprogram on F_i may be equally difficult as on F. Divide again:



Branch-and-bound

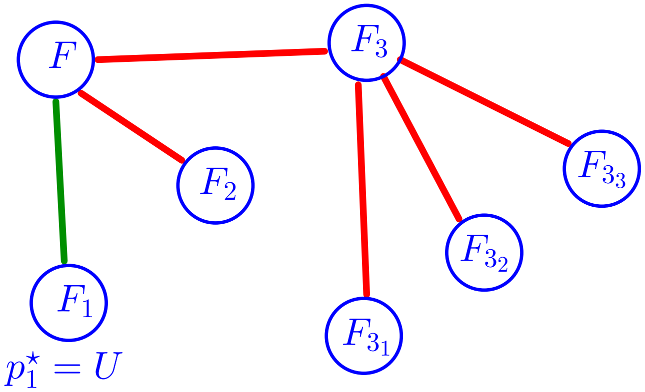
- So far, the ***divide*** part is intuitive.
- the ***conquer*** can be achieved is we have a **cheap** way to estimate a **lower bound** of the objective on F_i, that is $l(F_i)$ such that

$$l(F_i) \leq \min_{\mathbf{x} \in F_i} \mathbf{c}^T \mathbf{x}$$

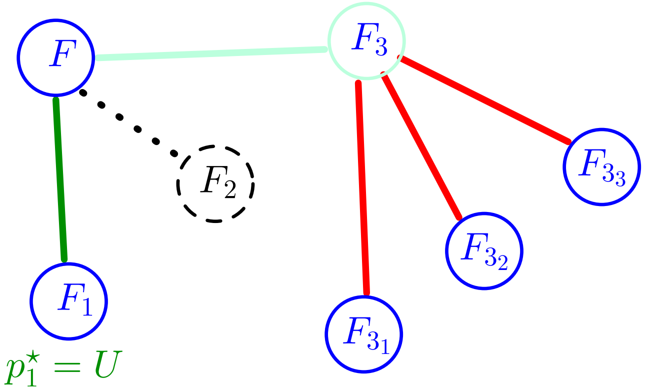
- A lower bound can be typically obtained by using a **relaxation**, or **duality**.

Branch-and-bound: intuitions

- Suppose an **optimum on** F_1 has been computed as $p_1^\star = U$

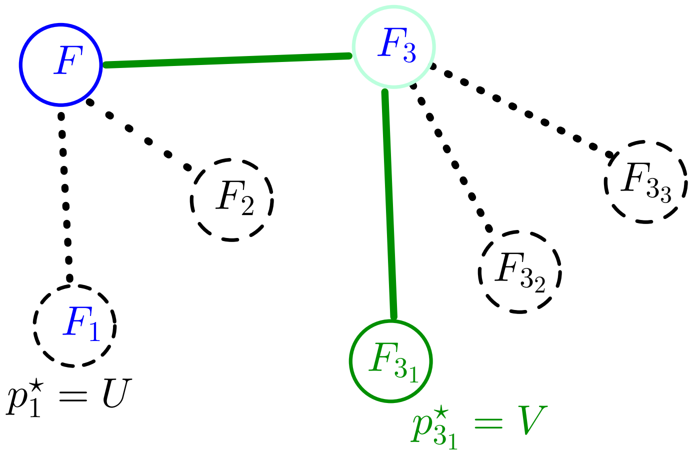


- If $l(F_2) \geq U$ then no need to check F_2 in detail.
- Skip to F_3. Suppose $l(F_3) \leq U$.
- We do not know whether a better point might be in F_3 but need to check



Branch-and-bound: intuitions

- Suppose $l(F_{3_2}) \geq U$ and $l(F_{3_3}) \geq U$. No need to check further.
- Suppose $l(F_{3_1}) \leq U$. Then we compute (expensive) the optimum on F_{3_1}:



- Suppose $p_{3_1}^\star = V < U \Rightarrow$ we have found the optimum.

Branch-and-bound: Generic Algorithm

Algorithm Steps:

1. Select an active subproblem defined on F_i.

2. If F_i is infeasible, delete it.

3. If not, compute $l(F_i)$. If $l(F_i) \geq U$, delete it

4. If $l(F_i) < U$, then either partition F_i either compute the optimum on F_i.

- Only a concept so far: a lot of free parameters.
  - how to choose the *active* subproblem?.
  - how to obtain the lower bound l? LP relaxation, dual.
  - how to define the partitions given a set F?
- Intuition: the tighter the lower bound, the better.

Branch-and-cut: Generic Algorithm

- A mixture of cuts and branch-and-bound

Algorithm Steps:

1. Select an active subproblem defined on F_i.
2. If F_i is infeasible, delete it.
3. If not, compute $l(F_i)$. If $l(F_i) \geq U$, delete it
4. If $l(F_i) = V < U$,
   (a) use cuts to obtain a series of increasing lower bounds $V \leq V_1 \leq V_2 \leq \cdots \leq V_n$.
   (b) n is defined adaptively.
   (c) Partition F_i, select an active subset F_{i_j} and use V_n.

- Even more parameters.. becomes more something of an art.

Dynamic Programming

Dynamic Programming Philosophy

- Dynamic programming is a **family of recursive methods** to solve programs.
- **Cannot be applied** to **all** integer programs unfortunately.
- First, turn a program into a sequence of decisions where each variable is iteratively modified.
- **dynamic programming** works when the *principle of optimality* is satisfied.

principle of optimality (Bellman, 1957)

An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.

Dynamic Programming Philosophy

- **Dynamic programming** is one of the most widely used algorithms class.
- Used to compare complex structures, find optimal allocations, optimal control.
- A non-exhaustive list of examples:
  - to **compare sequences** by considering alignments
    - Speech (Dynamic Time Warping),
    - biological sequence analysis (Smith Waterman),
    - text analysis (Levenshtein distance), detect common subsequences.
  - **Viterbi** algorithm for Hidden Markov Processes.
  - **Selinger** algorithm for query optimization in databases.
  - **Recursive least-squares** in statistics.
  - Bellman-Ford algorithm to compute shortest distance on a graph.
  - **Pricing** of American type Options.
  - **Optimal control** for trading strategies.
- The underlying idea of these algorithms is Bellman's principle of optimality.

Example

Discrete Allocation

- A company has 5 million \$ to **allocate** to 3 different **plants**.
- Each plants has a few projects which have **costs** and **expected revenues**.

(1) has 3 projects:	(2) has 4 projects:	(3) has 2 projects:
(a) do nothing $(\mathbf{0}, \mathbf{0})$. (b) small exp. $(\mathbf{1}, \mathbf{5})$. (c) med. exp. $(\mathbf{2}, \mathbf{6})$.	(a) nothing $(\mathbf{0}, \mathbf{0})$. (b) med. exp. $(\mathbf{2}, \mathbf{8})$. (c) large exp. $(\mathbf{3}, \mathbf{9})$. (d) XL exp. $(\mathbf{4}, \mathbf{12})$.	(a) do nothing $(\mathbf{0}, \mathbf{0})$. (b) small exp. $(\mathbf{1}, \mathbf{4})$.

- How can we maximize expected revenue using at most 5 million \$?
- direct enumeration: $3 \times 4 \times 2 = 24$ possibilities. Some unfeasible.
- Let's find a more clever approach.

Example

- This is a linear integer program after all. Can be written as

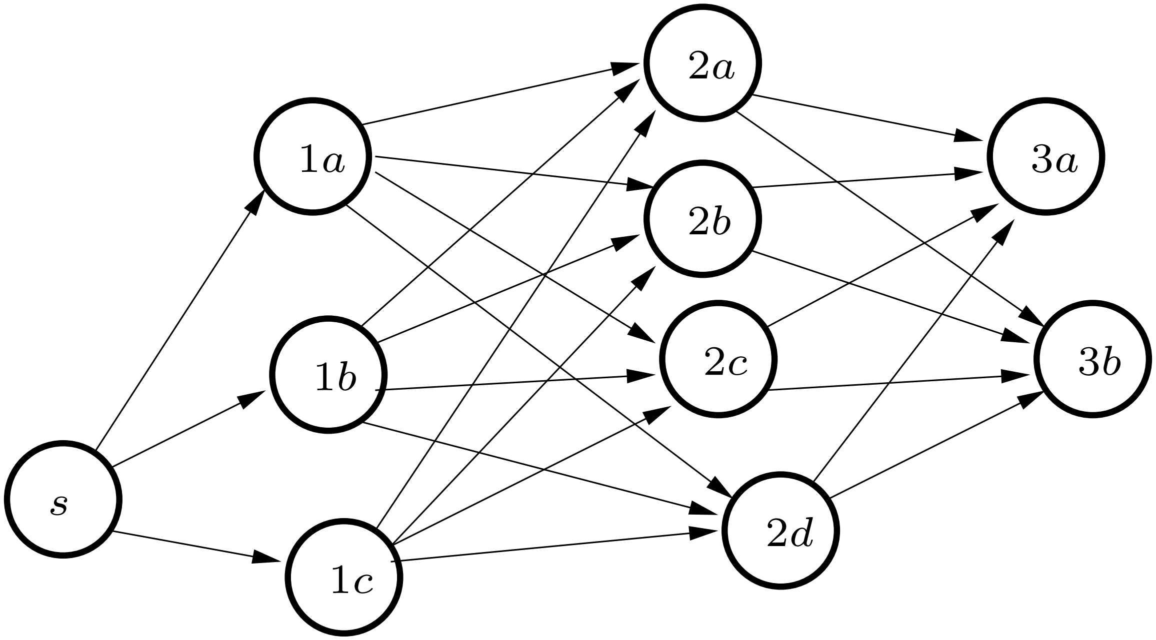
$$\begin{array}{ll} \text{maximize} & \mathbf{c}^T\mathbf{x} \\ \text{subject to} & \mathbf{x} \in F \\ & \mathbf{x} \in \{0,1\}^9 \end{array}$$

- Could use branch/bound, cuts etc.
- An important observation: suppose we have an assignment $\mathbf{x}$.
- This assignment can be seen as a sequential decision.
  - select first a project proposed by (1).
  - with the remaining money, select a project proposed by (2)
  - with the remaining money, select finally a project proposed by (3)

somehow an artifical observation, but crucial

Example

Graphically, start from s, go to either $3a$ or $3b$ through a path.



- how many paths in total?
- how many feasible paths?
- Intuitively, two notable facts:
  - paths can **cross**.
  - when reaching a state with a certain amount of money left, the previously visited states **do not matter** to select the next best decision.

Example

- Starting from (1). With a capital between 0 and 5, we should invest in project:

Capital	Optimal Project	Revenue
0	a	0
1	b	5
2	c	6
3	c	6
4	c	6
5	c	6

- When examining plant (2), **suppose we have 4 Mil$** available. For each choice in (2), there is **only one optimal choice** to invest the remainder in (1).

Project	Cost	Revenue	Remaining Capital	Project (1)	Total revenue
a	**0**	**0**	$4-0=4$	c	$0+6=\mathbf{6}$
b	**2**	**8**	$4-2=2$	c	$8+6=\boldsymbol{14}$
c	**3**	**9**	$4-3=1$	b	$9+5=\boldsymbol{14}$
d	**4**	**12**	$4-4=0$	a	$12+0=\mathbf{12}$

Example

- Computing these numbers not only for 4 Mil $, but **other values 0,1,2,3,5** as well, we come up with a similar table for (2):

Capital	Optimal Project	Revenue **for (1) and (2)**
0	a	0
1	a	5
2	b	8
3	b	13
4	b,c	14
5	d	17

- We can now look at options for (3). With (3) we only assume we start with **5 Mil$**, and invest the remaining in (2), and (1).

Project	Cost	Revenue	Remaining Capital	Project (2)	Total revenue
a	**0**	0	5	d	0+17=**17**
b	**1**	4	4	b,c	4+14=**18**

- Optimum? (3): b, (2): b/c (1): c/b

Summing Up

- Let the different nodes be $X = \{\mathbf{(1,a),(1,b)},\mathbf{(2,a),(2,b),(2,c),(2,d)},\mathbf{(3,a),(3,b)}\}$.
- Let $r(x)$ and $c(x)$, $x \in X$ be their respective revenues and costs.
- Let $f_i(C)$, $i \in \{1,2,3\}$ be the **maximal** revenue achievable when using plants (1) to (i) with **capital** C.
- Then we have the following relationships:

$$f_{\mathbf{1}}(C) = \max_{\{x=(j,s)\in X \mid j=1, c(x)\leq C\}} r(x),$$

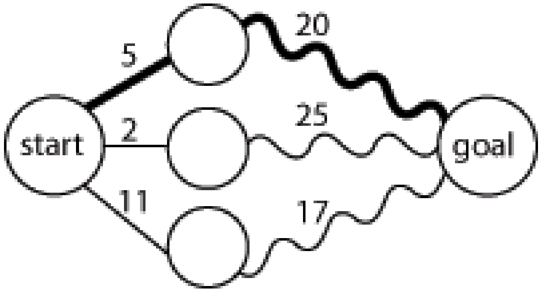
$$\text{for } i = \mathbf{2}, \mathbf{3}, \;\; f_i(C) = \max_{\{x=(j,s)\in X \mid j=i, c(x)\leq C\}} r(x) + f_{i-1}\left(C - c(x)\right).$$

- Computing recursively $f_{\mathbf{1}}, f_{\mathbf{2}}, f_{\mathbf{3}}$ for $C = \{0,1,2,3,4,5\}$ the solution is $f_{\mathbf{3}}(5)$.
- That is what we did exactly in the previous slides.
- Could have computed things in exactly the opposite (backward) way.

Dynamic Programming Implementations

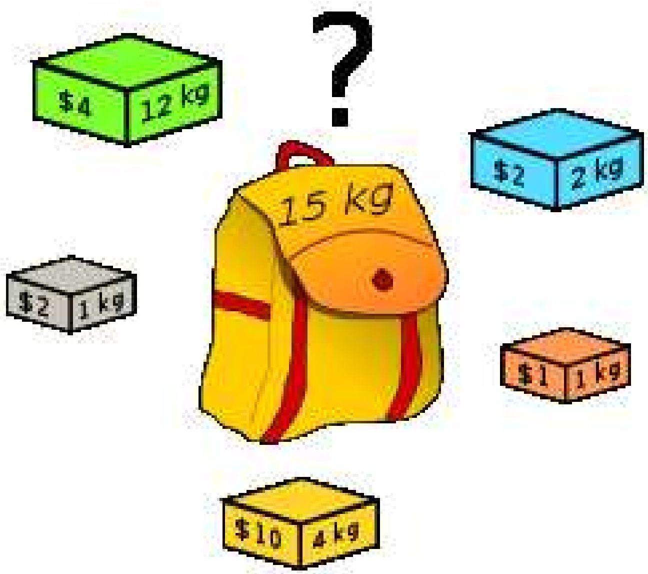
Intuition: DP works for programs which have two properties:

- **optimal substructure**
  - Reminiscent of the labelling algorithm in **Ford-Fulkerson**.



  - Intuitively, additivity of costs plays an important role:
    - ▷ Example: minimizing air travel distance from NY to Johannesburg.
    - ▷ Counterexample: minimizing ticket price from NY to Johannesburg.
- **overlapping subproblems**
  - a naive implementation would re-compute multiple times the same values.
  - Consider for instance the Fibonacci series, $F_{n+2} = F_{n+1} + F_n$.
  - Computing F_{10} involves computing recursively F_9 and F_8. But $F_9 = F_8 + F_7$
  - A recursive implementation would compute en exponential number of times the terms F_1, F_2 etc..

Going back to the Zero-One Knapsack Problem



- n items, jth item has value c_j and weight $w_j \Rightarrow$ vectors $\mathbf{c}$ and $\mathbf{w}$.
- Variable $\mathbf{x} \in \{0,1\}^n$ where $x_j = 1$ means the object is in the knapsack.
- A bound K on the maximum weight that can be carried by the knapsack.

$$\begin{array}{ll} \text{maximize} & \mathbf{c}^T\mathbf{x} \\ \text{subject to} & \mathbf{w}^T\mathbf{x} \leq K \\ & \mathbf{x} \in \{0,1\}^n \end{array}$$

Dynamic Programming Formulation

- *Dynamic programming* is **dynamic**.
- We thus have to make sure objects are ordered.
- Similarly to $f_i(C)$ in previous slides, with $i \leq n$ and $u \in \mathbb{N}$, let $W_i(u)$ be the
  - **least possible weight** that has to be accumulated,
  - in order to **carry value** u,
  - using **only items in** $\{1, \cdots, i\}$.
- Set a few boundary conditions:

$$W_i(u) = \infty \text{ if infeasibility}, \quad W_0(0) = 0, \quad W_0(u) = \infty \text{ for } u > 0.$$

- We then have the recursion:

$$W_{i+1}(u) = \min\left(W_i(u)\,,\, W_i(u - c_{i+1}) + w_{i+1}\right)$$

Dynamic Programming with Knapsack

- Once all numbers $W_i(u)$ are known, the optimal solution is obtained as

$$u^\star = \max\{u | W_n(u) \leq K\}$$

- Compute for each relevant (i, u) the number $W_i(u)$ **recursively**.
- u **is an integer**. can we upperbound it?
    - Suppose

    $$c_{\max} = \max_{i=1,\cdots,n} c_i.$$

    - Then $u \leq nc_{\max}$ for all feasible choices. $W_i(u) = \infty$ for $u > nc_{\max}$.
- On the other hand, $1 \leq i \leq n$.

Hence the total number of pairs (i, u) of interest is $O(n^2 c_{\max})$

- Using the recursion, we can compute all the values of W in $O(n^2 c_{\max})$ time.

Complexity

Theorem 1. *The 0-1 knapsack problem can be solved in time* $O(n^2 c_{\max})$

- Yet NP-hard problem. Contradiction?
- The size of the data required to described a knapsack problem is $O\left(n\left(\log c_{\max} + \log w_{\max}\right) + \log K\right)$.
- Indeed, a number x can be stored in $O(\log(x))$ bits.
- The term $c_{\max}$ is thus **exponential** in the size, **not polynomial**.
- Such algorithms are called **pseudo-polynomial** algorithms.
- For LP's, the bound was $O(n^6 \log(nU)) \Rightarrow$ **polynomial**.

Duality Theory for Integer Programs

Duality Theory: Formulation

- Not just theoretical interest: very important for **branch-and-bound** algorithms.
- Let us start from the beginning with a particular problem

$$\begin{array}{ll} \text{minimize} & \mathbf{c}^T\mathbf{x} \\ \text{subject to} & A\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \in \mathsf{X}, \end{array}$$

  where $\mathsf{X} = \{\mathbf{x} \in \mathbb{N} \,|\, D\mathbf{x} \geq \mathbf{d}\}$. Suppose z_{IP} **is its optimum**.

- We assume that optimizations on X can be done effectively ($e.g.$ network flow constraints). $(A, \mathbf{b})$ are more difficult to handle.
- Introduce dual variables μ (*Lagrange multipliers*) for all constraints of A.
- The *Lagrange dual* ***function*** of μ is then

$$Z(\mu) = \begin{array}{ll} \text{minimize} & \mathbf{c}^T\mathbf{x} + \mu^T(\mathbf{b} - A\mathbf{x}) \\ \text{subject to} & \mathbf{x} \in \mathsf{X}, \end{array}$$

Duality Theory for Integer Programs

Lemma 1. *For all* $\mu \geq 0$, $Z(\mu) \leq z_{IP}$

- Classic duality.
- Introduce now the *Lagrange dual* ***problem***:

$$\begin{array}{ll} \text{maximize} & Z(\mu) \\ \text{subject to} & \mu \geq 0. \end{array}$$

- Write z_D for the optimum, $\max_{\mu \geq 0} Z(\mu)$.
- Remember that X is a discrete set. Suppose $X = \{\mathbf{x}_1, \cdots, \mathbf{x}_k\}$.

$$Z(\mu) = \min_{1 \leq i \leq k} \mathbf{c}^T \mathbf{x}_i + \mu^T (\mathbf{b} - A\mathbf{x}_i)$$

- Z is the minimum of a finite collection of linear functions of μ, hence concave/piecewise linear.

Duality Theory for Integer Programs

Lemma 2. *We have weak duality:* $z_D \leq z_{IP}$

- Again, classic duality.
- unfortunately, no strong duality result.

- highlights usefulness for branch-and-bound.
- how does z_D compare relatively to the relaxation z_{LP}?

- Let us explore further the dual problem with two important theorems

Duality Theory for Integer Programs

Theorem 2. *Suppose* $D, \mathbf{d}$ *have integer entries and* $\{\mathbf{x} \in \mathbb{R}^n | D\mathbf{x} \geq \mathbf{d}\} \neq \emptyset$. *Then* $X = \{\mathbf{x} \in \mathbb{N} \,|\, D\mathbf{x} \geq \mathbf{d}\}$ *is such that* $\langle X \rangle$ *is a polyhedron in* $\mathbb{R}^n$.

- The case where X is finite is covered in the Weyl-Minkowski theorem
- Counterexamples exit when X is infinite, in this case the result holds.

Duality Theory for Integer Programs

Theorem 3. *The optimum* z_D *of the Lagrange dual problem is equal to*

$$\begin{array}{ll} \textit{minimize} & \mathbf{c}^T\mathbf{x} \\ \textit{subject to} & A\mathbf{x} \le \mathbf{b} \\ & \mathbf{x} \in \langle \boldsymbol{X} \rangle, \end{array}$$

- Compare with the LP relaxation

$$\begin{array}{ll} \text{minimize} & \mathbf{c}^T\mathbf{x} \\ \text{subject to} & A\mathbf{x} \le \mathbf{b} \\ & D\mathbf{x} \ge \mathbf{d}, \\ & \mathbf{x} \ge 0. \end{array}$$

- $\langle X \rangle \subset \{\mathbf{x} \ge 0,\ D\mathbf{x} \ge \mathbf{d}\}$ hence $z_{\text{LP}} \le z_D \le z_{\text{IP}}$.