Not exam context

Enumerative Methods ⇒ write all possible soln.

in clever ways → general → MILP

① Branch and Bound Alg.

B.B. is a "divide & conquer" algorithm

It is applicable to general MILPs.

Basic idea is to Break the problem up into a series of subproblems.

In the process of solving these subproblems, we assume that we have access to a lower bound for these subproblems that can be computed cheaply. (cheap lower bound)

→ 筛选子问题

The purpose of these lower bound is to quickly eliminate sub-prob.

Consider the following problem: (general prob.)

$$\min_{x} \quad c^T x$$
$$\text{s.t} \quad x \in F$$

→ for example, F could be a feasible region of a MILP.

Suppose we can partition set F into a finite collection of subsets $\{F_1, \ldots, F_n\}$. We solve the following sub-problem separately.

$$\min \ c^T x$$

s.t $x \in F_i$

At the end, we compute the soln from these sub-problems and we pick the best one!

In the process of solving the above sub-problems, we assume that we have access of the sub-problems that can be computed CHEAPLY.

$$\boxed{b(F_i)} \leq c^T x \quad \text{for all} \quad x \in F_i$$

↳ lower bound we want.

For instance, if F is the feasible region of MILP,

then a POPULAR lower bound is to compute the LR obtained by ignoring the integer constraints.

↳ cheap lower bound

⇓

MILP → LP

[Rmk]: This is the most typical way of obtaining a lower Bound, but it is NOT the ONLY WAY!

At the same time, we keep track of an upper bound of the opt. soln$^?$. To get an upper bound, one simply plug in feasible soln$^?$ to the objective.

For instance, by solving one of the sub-problems to optimality.

Let's call this Upper Bound U.

if at any point of time, we encounter

$b(F_i) \geq u$

We can ignore sub-problems F_i. Because we want to find the minimum, the F_i sub-prob. will not improve our soln.

Pseudo Algorithm

(1) Select an active sub-problem F_i from your current list of sub-problems

(2) i) → if the problem is infeasible → delete F_i from list! ($b(F_i) = +\infty$)

→ if not

ii) Compute $b(F_i)$

→ if $b(F_i) \geq u$ → delete F_i

(3) i) Proceed to solve F_i.

ii) If this is difficult to solve, then we may further sub-divide F_i into sub-problems, which we add to our list.

Rmk:

① many ways of performing search

Two EXTREMES are { Breadth-First-Search; Depth-First-Search

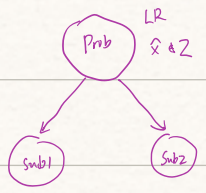
② Typical Way of obtaining sub-problems for MILPs is as follows:

Suppose in the Process of solving a LR, we obtain

a soln where $\hat{x}$ is required to be integer, but it is not an integer. Then we create 2 sub-problems:

(i) same problem + constraint $X \geq \lceil \hat{x} \rceil$

(ii) same problem + constraint $X \leq \lceil \hat{x} \rceil$



③ when we solve the above problem

= "some problem + constraint $X \geq \lceil \hat{x} \rceil$"

This differs from the previous problem by one inequality. So if you solved the previous LR by simplex, you can use this soln as the starting point to the LR of new problems! and you can perform the dual simplex.

④ If you stop mid way, you are still able to report a LB and an UB

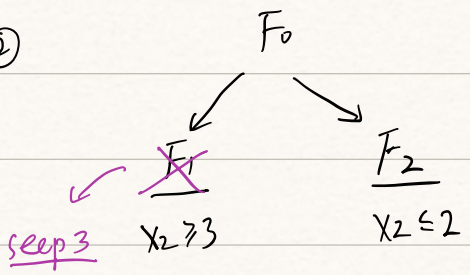
Example:

$$\begin{aligned} \min \quad & x_1 - 2x_2 \\ \text{s.t} \quad & -4x_1 + 6x_2 \leq 9 \\ & x_1 + x_2 \leq 4 \qquad (1) \\ & x_1, x_2 \geq 0 \\ & x_1, x_2 \in \mathbb{Z} \end{aligned}$$

$F_0 \rightarrow U = +\infty$

(also, $(0,0) \in$ Feasible $\Rightarrow U = 0$)

① LR of (1) $\Rightarrow \begin{cases} x_1 = 1.5 \\ x_2 = 2.5 \end{cases} \Rightarrow$ lower Bound of (1) $b(F_0) = -3.5$

② 

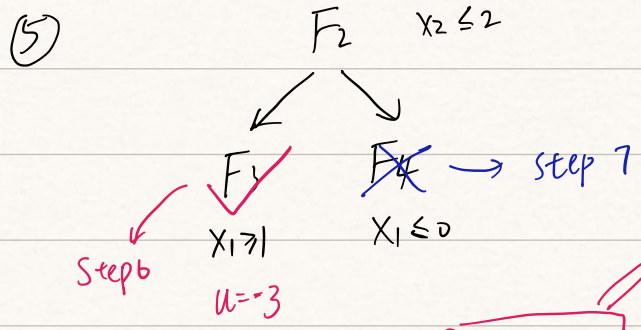
since $x_2 = 2.5$

③ $F_1 \rightarrow LR \rightarrow$ infeasible $\Rightarrow$ $b(F_1) = +\infty \Rightarrow$ delete F_1

④ $F_2 \rightarrow LR \rightarrow \begin{cases} x_1 = 0.75 \\ x_2 = 2 \end{cases} \Rightarrow b(F_2) = -3.25$

Not good

$x_2 = 2$ → good

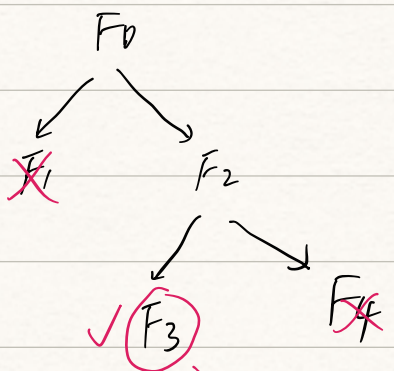
⑤ 

opt of F_3 → solve F_3 completely → haven't solve F_0 yet!

⑥ $F_3 \rightarrow LR \rightarrow \begin{cases} x_1 = 1 \\ x_2 = 2 \end{cases}$ $b(F_3) = -3 \rightarrow$ Update $u = -3$

⑦ $F_4 \rightarrow LR \rightarrow \begin{cases} x_1 = 0 \\ x_2 = 1.5 \end{cases}$ $b(F_4) = -3$

→ Since $b(F_4) \geq u$ ($b(F_4) = u$), then delete F_4 from our list!

⑧ 

→ we obtain the "global" optimal of F_0

↓

$\begin{cases} x_1 = 1 \\ x_2 = 2 \end{cases}$

Branch and CUT

When solving a sub-problem, one can implement a CUT that is valid for the sub-problem:

include an inequality that all soln in the sub-problem satisfy

F_2: $\min\ x_1 - 2x_2$

$s.t.\ -4x_1 + 6x_2 \le 9$

$x_1 + x_2 \le 4$

$x_1, x_2 \ge 0,\ x_1, x_2 \in \mathbb{Z}$

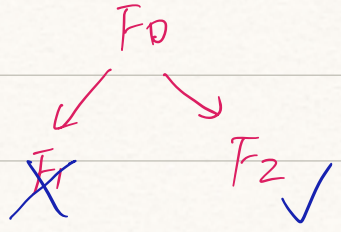
$x_2 \le 2$

valid inequality:

$-x_1 + x_2 \le 1$ ⇒ redundant

all satisfy this inequality (valid)

F_2 → LR → $(x_1, x_2) = (1, 2)$ ⇒ solve F_2 completely!



CUT ⇒ quicker convergence

⇓

Trick: add some redundant inequality to make LR tighter!

Dynamic Programming

The basic idea in DP. is to cast the optimization problem recursively using the optimal sol^n of smaller version of prob.

Caveat: The time taken may still be exponential.

Far more efficient than to naively enumerate all sol^n.

Example [TSP].

$G = (V, E)$ with some weights, say there are n nodes.

Cost C_{ij} for all edges.

Goal: find a tour with smallest cost

Naive strategy: enumerate all soln.

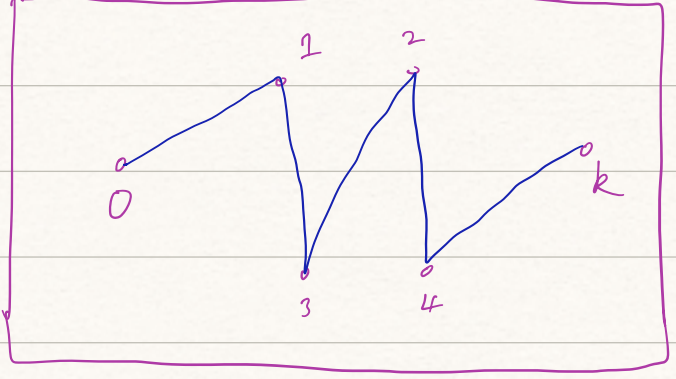
$$O(n!) \sim O\left(\sqrt{n}\left(\frac{n}{e}\right)^n\right)$$

Efficient Strategy: Define a state as (S, k)

where $S \subseteq V$ is a subset of vertices

and $k \in S$ is a vertex in S

Define $C(S, k)$ = min cost of a path from origin 0 (start) and end in k, while visiting all vertices in S



S

$\Rightarrow C(S,k)$

Note: $C(\{0\},0) = 0 \Rightarrow$ stay at origin

$$C(S,k) = \min_{m \in S\setminus\{k\}} \left[C(S\setminus\{k\}, m) + c_{mk} \right]$$

$\Downarrow$

$\Rightarrow$ if is not this path, namely P_1,

and we have $C(P^* \sim k) \leq C(P_1 \sim k)$

Algorithm:

Loop for $i = 1, 2, \ldots, n$

calculate $C(S,k)$ for all subsets $S \subseteq V$

when $|S| = i$, and all choices of $k \in S$

Computational Complexity

There are $O(n \cdot 2^n)$ possible states $\rightarrow (S,k)$

This is because there are 2^n choices of $S \subseteq V$ (n)

and $\leq n$ choices for k

Computing $C(S,k)$ requires $O(n)$ basic operations

$\Rightarrow$ Total cost is $O(n^2 2^n) \Rightarrow$ still exponentially

Bad

but $n^2 2^n << n$

Good

Example: [Knapsack Prob]

Recall the Knapsack Prob.

$$\max \sum_{i=1}^{n} c_i x_i$$
$$s.t \sum_{i=1}^{n} w_i x_i \le B$$
$$x_i \in \{0,1\}$$

背包问题

→ can we find some Recursive structure ?

⇓

should define the states first

Consider the sub-prob corresponding to just the first i vars.

⇓

small scale sub-prob

Notation:

$W_i(u)$ = min weight accumulated to obtain a total value of u using the first i items.

total value: $\sum c_i x_i$

By convention, we will set

$W_i(u) = +\infty$ if not possible to obtain u with the first i item!

(i.e. $\sum c_i < u$)

By convention, let set (start from)

$W_0(0) = 0$

$W_0(u) = +\infty$ since we are unable to attain u without taking any item

we have the recurrence:

$$W_{i+1}(u) = \min\{W_i(u),\ W_i(u - C_{i+1}) + W_{i+1}\}$$

constant # of computation complexity! → very small

BASIC IDEA

① if we do not pick item $i+1$ → $W_i(u)$

② if we pick item $i+1$ → weight W_{i+1} + $W_i(u - C_{i+1})$

State Space: (i, u) $i = 1, 2, \ldots, n$

u for all possible value $(u \le \sum C_i)$ $\le n\, C_{max}$

a TRIVIAL UPPER BOUND of u

total # of states $\le n \cdot n\, C_{max}$

$\le n^2 C_{max}$

In computing $W_i(u)$: constant cost

$\Rightarrow$ computational complexity $O(n^2 C_{max})$

Reasonable algorithm if C_{max} is of Reasonable size

Note: Opt. Sol^n:

$$\hat{u} = \max\{u : W_n(u) \le B\}$$

for n items, it can attain

u-value ($\sum c_i x_i$) with costs (weight) $\leq B$

Note: with some effort, it is also possible to work out the optimal ASSIGNMENT (x_i)

[Rmk]: Give the Best Possible Value $\Rightarrow$ The min weights

$\Downarrow$

Our soln.

An alternative method:

$D_i(u)$ = max value s.t

accumulated weight $\leq w$

$\Downarrow$

One has

$\Rightarrow D_{i+1}(w) = \max\{D_i(u), D_i(w - w_{i+1}) + c_{i+1}\}$

更自然!