Container Loading

Container Loading problem: The constraint is that the **container loaded** have total weight ≤ the cargo weight capacity, and the objective function is to find a largest set of **containers** to **load**.

Dr. Bibhudatta Sahoo

Communication & Computing Group

Department of CSE, NIT Rourkela

Email: bdsahu@nitrkl.ac.in, 9937324437, 2462358

Solving optimization problem with Greedy Algorithm

- **Optimization Problem**: Given an Optimization Function and a set of constraints, find an optimal solution.
- **Feasible Solution**: [Solution that satisfies the constraints] Solutions to such a problem that satisfy the **constraints** are called **feasible solutions**.
- **Optimal Solution**: A feasible solution for which the optimization function has the best possible value.

Minimum Spanning Trees

Consider the spanning tree problem, where we are given an undirected graph G with edge weights $w_{u,v}$ for every pair of vertices u, v.

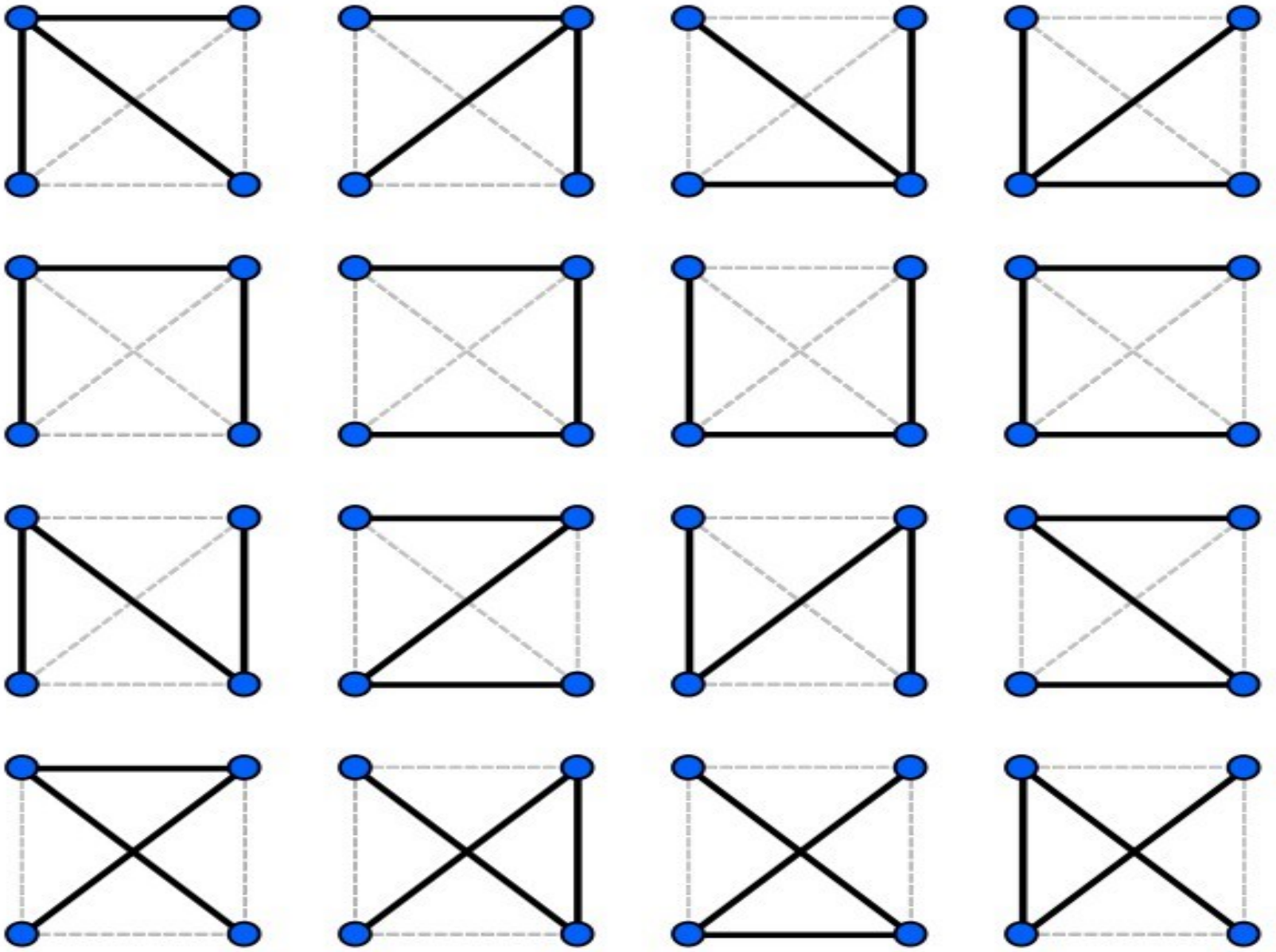
An integer linear program that solves the minimum spanning tree problem is as follows:

$$\text{Minimize} \sum_{(u,v)\in E} w_{u,v}x_{u,v}$$

$$\text{subject to} \sum_{\{u,v\}\in E:u\in L,v\in R} x_{u,v} \geq 1 \quad \text{for all partitions of } V \text{ into disjoint nonempty sets } L, R$$

$$x_{u,v} \in \{0,1\}, \quad \forall (u,v) \in E$$

Possible spanning tree ....



Optimization Problem

- A problem in which some function (called the optimization or objective function) is to be optimized (usually minimized or maximized) subject to some constraints.

Minimum Spanning Trees

Consider the spanning tree problem, where we are given an undirected graph G with edge weights $w_{u,v}$ for every pair of vertices u, v.

An integer linear program that solves the minimum spanning tree problem is as follows:

$$\text{Minimize} \sum_{(u,v)\in E} w_{u,v}x_{u,v}$$

$$\text{subject to} \sum_{\{u,v\}\in E:u\in L,v\in R} x_{u,v} \geq 1 \quad \text{for all partitions of } V \text{ into disjoint nonempty sets } L, R$$

$$x_{u,v} \in \{0,1\}, \quad \forall (u,v) \in E$$

Feasible And Optimal Solutions

- A feasible solution is a solution that satisfies the constraints.

- An optimal solution is a feasible solution that optimizes the objective/optimization function.

Greedy Method

- Solve problem by making a sequence of decisions.
- Decisions are made one by one in some order.
- Each decision is made using a greedy criterion.
- A decision, once made, is (usually) not changed later.

The greedy method

- It is one way to construct a **feasible solution** for such optimization problems, and, sometimes, it leads to an optimal one.
- When applying this method, we construct a solution in **stages**.
- At each stage, we make a decision that appears to be the best at that time, according to a certain **greedy criterion**.
- Such a decision will not be changed in later stages. Hence, each decision should assume the feasibility.
- Sometimes, such a locally optimal solution does lead to an **overall optimal** one.

Container Loading problem

- A large ship is to be loaded with containers of cargos. Different containers, although of equal size, will have different weights. Objective is to load the ship with **maximum number of containers** without exceeding the cargo weight capacity


EIMSKIP

Container Loading problem

- A large ship is to be loaded with cargo.
- The cargo is containerized, and all containers are of the same size.
- Different containers may have different weights.
- The constraint is that the container loaded have total weight ≤ the cargo weight capacity, and the objective function is to find a largest set of containers to load.
- Container **i** has weight $\mathbf{w_i}$.
- The cargo weight capacity of the ship is **c** (and every $w_i \leq c$).
- Load the ship with **maximum number of containers without exceeding the cargo weight capacity**.

Container Loading problem as optimization problem

Find values $x_i \in \{0, 1\}$ such that $\sum_{i=1}^{n} w_i \; x_i \leq c$ and the optimization function $\sum_{i=1}^{n} x_i$ is maximized.

Homework: What is a feasible, and optimal, solution in this case? Come up with an algorithm to find an optimal solution, when possible.

Container Loading problem

Let w_i be the weight of the i^{th} container, $i \in [1, n]$, and the capacity of the ship is c, we want to find out a way to load the ship with the maximum number of containers, without tipping over the ship.

Let $x_i \in \{0, 1\}$. If $x_i = 1$, we will load the i^{th} container, otherwise, we will not load it.

We wish to assign values to x_i's such that $\sum_{i=1}^{n} x_i w_i \leq c$, and $\sum_{i=1}^{n} x_i$ is maximized.

Greedy Solution

- Load containers in increasing order of weight until we get to a container that does not fit.
- Does this greedy algorithm always load the maximum number of containers?

Example: Container loading

w1	w2	w3	w4	w5	w6	w7	w8	C
100	200	50	90	150	50	20	80	400
1	1	1	0	0	1	0	0	400

Is it an optimal solution? No! Why? One can take out object 2 and add objects 7 and 8. That would be a better solution.

Is a feasible solution optimal if one cannot trade two objects for one (in the solution) while maintaining feasibility?

w1	w2	w3	w4	w5	w6	w7	C
60	60	60	60	80	80	80	240
0	0	0	0	1	1	1	240

Example: Container loading

w1	w2	w3	w4	w5	w6	w7	w8	C
100	200	50	90	150	50	20	80	400
1	1	1	0	0	1	0	0	400

Load ship in stages, one container per stage. At each stage we need to decide which container to load.

Greedy criterion: From the remaining containers, select the one with least weight.

w1	w2	w3	w4	w5	w6	w7	w8	C
20	50	50	80	90	100	150	200	400
1	1	1	1	1	1	0	0	390

Container Loading problem as optimization problem

Find values $x_i \in \{0, 1\}$ such that $\sum_{i=1}^{n} w_i \ x_i \leq c$ and the optimization function $\sum_{i=1}^{n} x_i$ is maximized.

Algorithm ContainerLoading

1. Algorithm **ContainerLoading** (c, w, capacity, noofcont, x)
2. //Set x[i]=1 if and only if container c[i], i ≥ 1 is loaded
3. **Sort(c, noofcont);**
4. n := noofcont;
5. for i := 1 to n do
6. x[i] :=0;
7. // select container in order of weight
8. i:=1;
9. while (i ≤ n && w[i] ≤ capacity)
10. { x[i] := 1;
11. capacity := capacity – w[i];
12. i:= i+1;
13. }

- **Theorem**: The greedy algorithm **ContainerLoading** generates an **optimal set** of containers to load.
- **Proof:** No matter which feasible solution (Y) you start with, it is possible to transform it to the solution generated by the algorithm without decreasing the objective function value
- Assume without loss of generality (wlog) $w_1 \leq w_2 \leq \ldots \leq w_n$;
- Let $X = (x_1, x_2, \ldots x_n)$ be the solution generated by the algorithm
- Let $Y = (y_1, y_2, \ldots y_n)$ be any feasible solution such that

$$\sum_{i=1}^{n} w_i y_i \leq c.$$

- Transform Y to X in several steps without decreasing the objective function value.

- From the way the algorithm works, there is a $k \in [0, n]$ s.t. $x_i = 1$ for $i \leq k$, and $x_i = 0$ for $i > k$. (i.e., X = 1, 1, ..., 1, 0, 0, ..., 0).
- Transformation: Let j be the smallest integer in $[1, n]$, s.t. $x_j \neq y_j$.
- So either:

  (1) No such j exists in which case $Y = X$.

  (2) $j \leq k$ (as otherwise Y is not feasible).

  So, $x_j = 1$ and $y_j = 0$.

  Change y_j to 1
- If Y is infeasible then there is an l in $[j + 1, n]$ s.t. $y_l = 1$, y_l is changed to 0, and the new Y is feasible (because $w_j \leq w_l$).
- No matter what the new Y is, it has at least as many 1s (or more) as before.
- Apply the transformation until you get $Y = X$.

Example: Container loading

w1	w2	w3	w4	w5	w6	w7	w8	C
100	200	50	90	150	50	20	80	400
1	1	1	0	0	1	0	0	400

Is it an optimal solution? No! Why? One can take out object 2 and add objects 7 and 8. That would be a better solution.

Is a feasible solution optimal if one cannot trade two objects for one (in the solution) while maintaining feasibility?

w1	w2	w3	w4	w5	w6	w7	C
60	60	60	60	80	80	80	240
0	0	0	0	1	1	1	240

Example for Proof

Solution Generated by our algorithm (X)

w1	w2	w3	w4	w5	w6	w7	w8	C
20	50	50	80	90	100	150	200	400
1	1	1	1	1	1	0	0	390

Consider the following feasible solution (Y)

w1	w2	w3	w4	w5	w6	w7	w8	C
20	50	50	80	90	100	150	200	400
0	1	0	0	0	0	1	1	490

(j in the proof is 1, then 3, 4, 5, 6).

w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	c
20	50	50	80	90	100	150	200	400
1	1	0	0	0	0	0	1	270
1	1	1	0	0	0	0	1	320
1	1	1	1	0	0	0	1	400
1	1	1	1	1	0	0	0	290
1	1	1	1	1	1	0	0	390

- Remaining objects are stored in a heap ordered with respect to their weight (smallest on top of the heap).
- Algorithm talks O(n log n) time (n deletes from a heap with initially n objects, creating the heap takes O(n) time
- Algorithm takes O(n) time if optimal sol has very few (n/ log n) objects.

Is it possible to have a linear time algorithm

- Assume all weights are different, if there are repeated weights then a similar algorithm exists for the solution of the problem.
- We use an algorithm (which we will describe when we cover divide-and-conquer algorithms) that finds the middle object of n objects (i.e., find the element such that there are exactly $\lceil n/2 \rceil$ objects smaller or equal to it) in O(n) time.

- Our algorithm works by doing a binary Search type of search on the unsorted weights (W).
- Let S be the smallest $\lceil n/2 \rceil$ objects (in W) and let t be their total weight. There are three cases:

  (1) If $t > c$ then search for a solution in S only with the same capacity c.

  (2) If $t = c$, then add all the objects in S to the solution and end the procedure;

  (3) Else, add all the elements in S to the solution and set the the remaining capacity to $c - t$ and now try to add as many objects as possible from $W - S$.

- Repeat the above step until there are no objects left.

Time complexity is
$c_1 n + c_1 n/2 + c_1 n/4 + \ldots = c_2 n$

Example 1

- Suppose that W is: $\{6, 10, 8, 15, 22, 19, 5, 9\}$, and $c = 25$.

The middle object is 9, $S = \{6, 8, 5, 9\}$ and $t = 28$. The objects in S do not fit.

- The new W is: $\{6, 8, 9, 5\}$, and $c = 25$.

The middle object is 6, $S = \{6, 5\}$, and $t = 11$. The objects in S fit and are added to the solution.

- The new W is: $\{8, 9\}$, and $c = 14$.

The middle object is 8, $S = \{8\}$, and $t = 8$. The object in S fit and are added to the solution.

- The ne W is: $\{9\}$, and $c = 6$.

The middle object is 9, $S = \{9\}$, and $t = 9$. The object in S does not fit.

- There are no objects left and we are done. The solution are the objects with weight 5, 6 and 8.

Example 2

- Suppose W is: $\{6, 10, 8, 15, 22, 19, 5, 9\}$, and $c = 53$.

The middle object is 9, $S = \{6, 8, 5, 9\}$, and $t = 28$. The objects in S fit and are added to the solution.

- The new W is: $\{10, 15, 22, 19\}$, and $c = 25$.

The middle object is 15, $S = \{10, 15\}$, and $t = 25$. The objects in S fit exactly and the algorithm finishes.

- The solution are the objects with weight 6, 8, 5, 9, 10, and 15.

Thanks for Your Attention!

Exercises

Three dimension container loading

PROBLEM DESCRIPTION

- We have a container of dimensions (L,W,H) that has to be filled with a set of n boxes, B_i, $i = 1, \ldots, n$. Each box has dimensions (l_i, w_i, h_i) in cm. and weight in kg. and a destination or drop number d_i. The objective is to find an orthogonal packing of the boxes so as to maximize the container volume utilization, subject to these constraints:

- ***Full support*:**

- The base of each box has to be placed on the floor of the container or completely on top of other boxes.

- ***Allowed orientation:***

- Each box has a set of allowed orientations due to its content or its structure. The orientations are denoted by o_{ij}, where $i=1,\ldots,n$ represents the box and $j=1,2,3$ the orientation, that is, the dimension of the box that can be placed upright: $o_{i1}=1$ if dimension l of box i can be upright and 0 if it may not, $o_{i2}=1$ if dimension w can be upright and 0 if it may not, and $o_{i3}=1$ if dimension h can be upright and 0 if it may not. At least one of these parameters must be 1 for each box i.

Three dimension container loading

- ***Load-bearing strength:***

- Each box placed in one allowed orientation can bear a maximum weight m_{ij} on top of it, expressed in gr/cm^2, where $i=1,\ldots,n$ represents the box and $j=1,2,3$ the orientation. There are several ways in which the weight of a box is transmitted downwards onto the boxes supporting it from below. In practice, it depends on the stiffness of the supporting surface: the more rigid it is, the more the weight is spread over the whole surface. When using soft materials like cardboard, the weight of the box above is essentially transmitted down onto the contact area only. We will assume this latter situation and therefore when a box k is put on top of another box i, its weight w_k will be divided by its base area to calculate the pressure, p_k in gr/cm^2, exerted on the supporting box. If this pressure p_k exceeds the load-bearing capacity of the box below, m_{ij} , the packing is not feasible. Otherwise, the remaining load-bearing capacity of box i will be reduced by p_k .

- ~~***Multi-drop*:**~~

- ~~We admit the three possible definitions of multi-drop constraints described in the previous section. We have designed an algorithm able to deal with all kinds of multi-drop constraints. Depending on the type of constraint being considered, some steps of the algorithm will be changed, as will be explained in the next section.~~

Advantages and disadvantages [Greedy Algorithm]

- It is quite easy to **come up with a greedy algorithm** (or even multiple greedy algorithms) for a problem.
- **Analyzing the run time for greedy algorithms will generally be much easier** than for other techniques (like Divide and conquer). For the Divide and conquer technique, it is not clear whether the technique is fast or slow. This is because at each level of recursion the size of gets smaller and the number of sub-problems increases.
- The difficult part is that for greedy algorithms **you have to work much harder to understand correctness issues**. Even with the correct algorithm, it is hard to prove why it is correct. Proving that a greedy algorithm is correct is more of an art than a science. It involves a lot of creativity.