Advanced Algorithms

Floriano Zini

Free University of Bozen-Bolzano
Faculty of Computer Science

Academic Year 2013-2014

Lecture 5 – Greedy algorithms (cont.)

Minimum spanning tree

- **Definition**
  - *Input:* An undirected graph $G = (V, E)$; edge weights w_e
  - *Output:* A tree $T = (V, E')$, with $E' \subseteq E$, that minimizes

$$\text{weight}(T) = \sum_{e \in E'} w_e$$

- **Example**

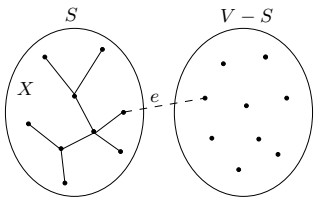
The cost is 1 + 4 + 2 + 4 + 5 =16

Prim's algorithm

- The cut property tells us in general that any algorithm conforming to the following greedy schema is guaranteed to work

```
X = { }  (edges picked so far)
repeat until |X| = |V| − 1:
  pick a set S ⊂ V for which X has no edges between S and V − S
  let e ∈ E be the minimum-weight edge between S and V − S
  X = X ∪ {e}
```

- A popular alternative to Kruskal's algorithm is Prim's
  - the intermediate set of edges *X* always forms a subtree, and *S* is chosen to be the set of this tree's nodes
- On each iteration, the subtree defined by *X grows* by the lightest edge between a node in *S* and a node outside *S*



Huffman encoding

In the **MP3 audio compression scheme**, a sound signal is encoded in three steps

1. It is **digitized** by sampling at regular intervals, yielding a sequence of real numbers $s_1, s_2, \ldots, s_T$
   - For instance, at a rate of 44,100 samples per second, a 50-minute symphony correspond to $T=50*60*44,100 \approx 130$ million measurements
2. Each real-valued sample s_t is **quantized**: approximated by a nearby number from a finite set Γ
   - This set Γ is carefully chosen to exploit human perceptual limitations
3. The resulting string of length T over alphabet Γ is **encoded** in binary
   - In this last step, the **Huffman encoding** is used

Toy example

- $T = 130$ million
- Γ consists of four symbols *A, B, C, D*
- What is the most **economical way** to write this long string in **binary**?
- Obvious choice is to use ***2* bits per symbol**
  - *00* for *A*, *01* for *B*, *10* for C, and *11* for D
  - *260* megabits are needed in total!
- Suppose that **the 4 symbols are nor equally abundant**
- Is there a of **variable-length encoding**, in which just **one** bit is used **for high frequency symbols**, at the expense of **three or more bits for less common symbols**?

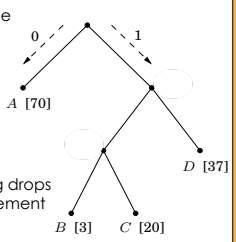
Symbol	Frequency
A	70 million
B	3 million
C	20 million
D	37 million

Toy example (cont.)

- A **danger** is that the resulting **encoding is not uniquely decipherable**
  - For instance, if the codewords are *{0, 01, 11, 001}* for *{A, B, C, D}* the decoding of strings like *001* is ambiguous
    - (*AB* or *D*?)
- We avoid this problem by assuring that **no codeword can be a prefix of another codeword**
- Any **prefix-free encoding** can be represented by a **full binary tree**
  - The **symbols** are at the **leaves**
  - Each **codeword** is generated by a **path from root to leaf, interpreting left as 0 and right as 1**
- **Decoding is unique**
  - Start at the root
  - Read the bitstring from left to right to move downward
  - When a leaf is reached, output the corresponding symbol
  - Return to the root

Symbol	Codeword
A	0
B	100
C	101
D	11

A prefix-free encoding



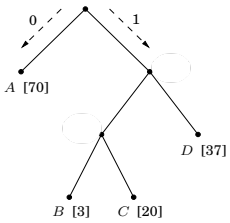
The total size of the binary string drops to 213 megabits, a *17%* improvement

General method

- How do we find the **optimal coding tree**, given the frequencies $f_1, f_2, \ldots, f_n$ of n symbols?
- We want a **tree** whose **leaves correspond to symbols** and which **minimizes the overall length of the encoding**

$$\text{cost of tree} = \sum_{i=1}^{n} f_i \cdot (\text{depth of } i\text{th symbol in tree})$$

The number of bits required for a symbol is exactly its depth in the tree

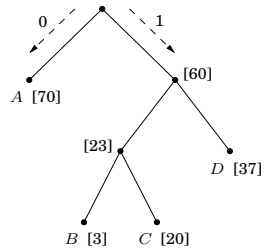


Cost = 70M*1 + 3M*3 + 20M*3 + + 37M*2 = 213M

- This formulation tells us that **the two symbols with the smallest frequencies must be at the bottom of the optimal tree**, as children of the lowest internal node
- Otherwise, swapping these two symbols with whatever is lowest in the tree would improve the encoding

General method (cont.)

- We can define the frequency of any **internal** node to be the sum of the frequencies of its descendant leaves
- **The cost of a tree is the sum of the frequencies of all leaves and internal nodes, except the root**

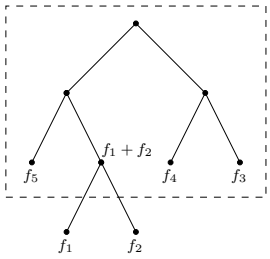


Cost = 70M + 60M + 23M + 37M +
+ 3M + 20M = 213M

General method (cont.)

The general method constructs the tree **greedily**

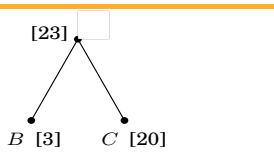
- find the two symbols *i* and *j* with the **smallest frequencies** f_1 and f_2
- make *i* and *j* children of a new node, having frequency f_1+f_2
- any tree in which f_1 and f_2 are sibling-leaves has cost $f_1 + f_2$ plus the cost for a tree with $n-1$ leaves of frequencies $(f_1 +f_2), f_3, f_4, \ldots, f_n$
- Pull f_1 and f_2 off the list of frequencies, insert $(f_1 + f_2)$, and loop
- The resulting algorithm takes ***O(n*log n)*** where *n* is the number of symbols



General method (example)

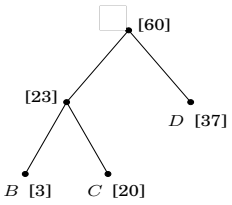
Symbol	Freq.
A	70 M
B	3 M
C	20 M
D	37 M

3, 20



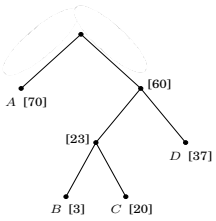
Symbol	Freq.
A	70 M
{B,C}	23 M
D	37 M

23, 37



Symbol	Freq.
A	70 M
{B,C,D}	60 M

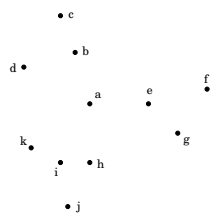
70, 60



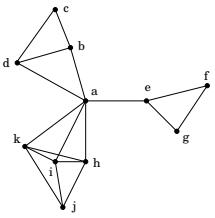
Set cover

- A county is in its early stages of planning and is deciding **where to put schools**
- There are only two **constraints**
  - **each school should be in a town**
  - **no one should have to travel more than 30 miles to reach the closest school**
- What is the **minimum number of schools** needed?

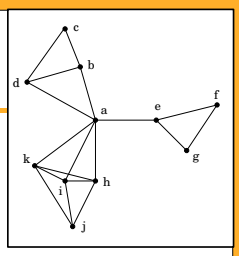
Eleven towns



Towns that are within 30 miles of each other



Set cover (cont.)



This is a typical **set cover problem**

- For each town x, S_x is the set of towns within 30 miles of it
- A school at x will essentially "cover" these other towns
- How many sets S_x must be picked in order to cover all the towns in the county?

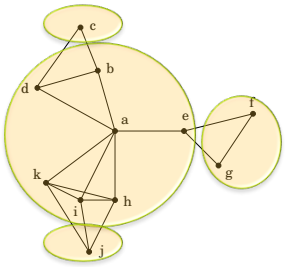
SET COVER

Input:	A set of elements B, sets $S_1, \ldots, S_m \subseteq B$
Output:	A selection of the S_i whose union is B.
Cost:	Number of sets picked.

- In our example, the elements of B are the towns

Set cover (cont.)

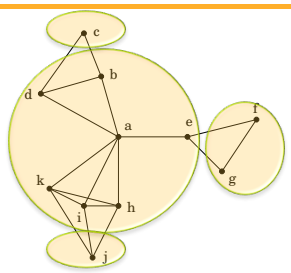
- This problem lends itself immediately to a **greedy solution**:
  - **Repeat until all elements of *B* are covered:**
    Pick the set S_i with the largest number of uncovered elements
- **How does it work on the example?**
  - school at town *a*, since this covers the largest number of other towns *{a, b, d, e, k, i, h}*
  - school at town *f* or *g*, that covers *{f, g}*
  - school at town *c* that covers itself only
  - school at town *j* that covers itself only
  - **4 schools in total**

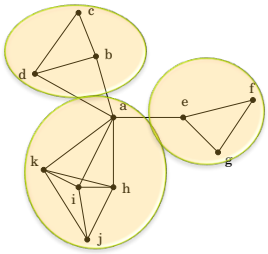


Is the found solution optimal?

NO!

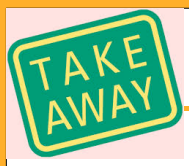
- there is a solution with just three schools
- at *b*, *e*, and *i*





Set cover (cont.)

- **The greedy scheme is not optimal!**
- But luckily, **it isn't too far from optimal**
- **Claim:**
  - Suppose *B* contains *n* elements and that **the optimal cover consists of *k* sets**
  - Then the greedy algorithm will use **at most *k*ln n* sets**



- **Greedy** algorithms build up a solution step by step, always choosing the next step that offers the **most obvious and immediate benefit**
- For many computational problems (e.g., **minimum spanning tree**, **Huffman coding**) they are **optimal**
- For others (e.g., **set cover**), they are quite **close to optimum** and can be profitably used, as the optimal solution is intractable

Lecture 5 – Linear Programming

Examples of LP Problems (1)

A Product Mix Problem

- A manufacturer has **fixed amounts of different resources** such as raw material, labor, and equipment
- These **resources** can be **combined** to produce any one of several **different products**
- The **quantity** of any **resource** required to produce **one unit** of any product is **known**

The decision maker wishes to produce the combination of products that will maximize total income

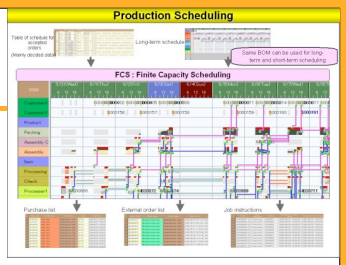
Examples of LP Problems (2)

A Blending Problem

- Blending problems refer to situations in which **a number of components (or commodities) are mixed together to yield one or more products**
- Typically, **different commodities are to be purchased**; each commodity has known characteristics and costs

The problem is to determine how much of each commodity should be purchased and blended with the rest so that the characteristics of the mixture lie within specified bounds and the total cost is minimized

Examples of LP Problems (3)

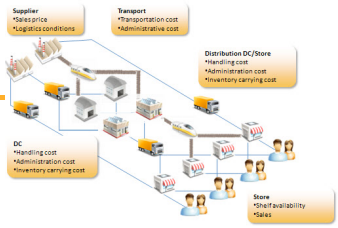


A Production Scheduling Problem

- A manufacturer must supply a given number of items of a certain product each month for the next *n* months
- Items can be **produced** either in **regular time**, subject to a maximum each month, or in **overtime**
  - The cost of **producing** an item during **overtime** is **greater** than during **regular time**
- A **storage cost** is associated with each item not sold at the end of the month

The problem is to determine the production schedule that minimizes the sum of production and storage costs

Examples of LP Problems (4)

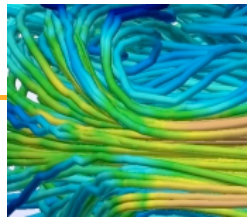


A Transportation Problem

- A **product** is to be **shipped** in the amounts a_1, a_2, ..., a_m from *m* shipping origins and received in amounts b_1, b_2, ..., b_n at each of *n* shipping destinations
- The **cost** of shipping a unit from the i^{th} origin to the j^{th} destination is **known for all combinations of origins and destinations**

The problem is to determine the amount to be shipped from each origin to each destination such that the total cost of transportation is a minimum

Examples of LP Problems (5)



A Flow Capacity Problem

- One or more **commodities** (e.g., traffic, water, information, cash, etc.) are **flowing** from one point to another through a **network** whose **branches** have various **constraints** and **flow capacities**
- The **direction of flow** in each branch and the **capacity** of each branch are **known**

The problem is to determine the maximum flow, or capacity of the network

Linear Programming

- What is it?
  - Tool **for optimal allocation of scarce resources**, among a number of **competing activities**
  - **Powerful** and **general** problem-solving **method**
- Why is it significant?
  - **Many problems are optimization tasks**
    - we search for a solution that satisfy certain constraints and is the best possible w.r.t. some well-defined criterion
  - **Widely applicable** and **widely used** in **industry**
  - Fast **commercial solvers available**: CPLEX, OSL
  - **Ranked among most important scientific advances of 20th century**

Example - Confectionery's problem

- A small **confectionery** produces **cookies** and **cupcakes**
- The production is limited by **scarce resources**: **flour**, **sugar**, and **milk**


- The **receipts** for cookies and cupcakes **differ in the proportion of the resources**

	Sugar (Hg)	**Milk** (Litres)	**Flour** (Hg)	**Profit** ($)
available	480	160	1190	
cookies (1 box)	5	4	35	13
cupcakes (1 box)	15	4	20	23

- Confectionery's problem: **choose product mix to maximize profits**

	Sugar (Hg)	**Milk** (Litres)	**Flour** (Hg)	**Profit** ($)
all cookies (34 boxes)	179	136	1190	442
all cupcakes (32 boxes)	480	128	640	736
20 boxes of cookies 20 boxes of cupcakes	400	160	1100	720
12 boxes of cookies 28 boxes of cupcakes	480	160	980	800
more profitable mix?	?	?	?	> 800?

Linear Programming - Idea

In a linear programming problem we are given a set of **variables**, and we want to **assign real values to them** to:

1. **Satisfy a set of linear equalities and/or linear inequalities involving these variables**

$$\overset{\text{Linear equality}}{f(x_1, x_2, ..., x_n) = b} \qquad \overset{\text{Linear inequality}}{f(x_1, x_2, ..., x_n) \leq b}$$

2. **Maximize or minimize a given linear objective function**

$$f(x_1, x_2, ..., x_n) = a_1 \cdot x_1 + a_2 \cdot x_2 + a_n \cdot x_n = \sum_{i=1}^{n} a_i \cdot x_i$$

Linear Programming - Idea

- **First step** – ***Identification of the problem***
  - Individuate (or choose) the **objective function**
  - Individuate the **available resources**
  - Individuate **available data**
- **Second step** – ***Formulation of the linear programming problem***
  - Define the **variables** (unknowns of the problem)
  - Define the **mathematical relations** among and data (constraints and objective function)
- **Third step – *Solve the problem***

Linear Programming Problem

- Input:
  - A set of **real variables** *X* with arbitrary **bounds**
  - A set of **linear constraints** on *X*
  - A **linear objective function** on *X*
- Output:
  - **An assignment to *X* with optimal value of the objective function**

Example 1 - Confectionery's problem

First step – Identification of the problem

- Individuate (or choose) the objective
  - ***maximizing the revenue***
- Individuate the available resources
  - ***Sugar; Milk; Flour***
- Individuate available data
  - ***unitary sugar/milk/flour requirement both for cookies and cupcakes***
  - ***unitary sell prices***
  - ***availability of sugar, milk and flour***

Example 1 - Confectionery's problem (cont.)

Second step – Formulation of the linear programming problem

- Define the variables
  - ***x_1: numbers of boxes of cookies***
  - ***x_2: number of boxes of cupcakes***
- Define the mathematical relations among variables and data
  - Constraints:
    - Available sugar 480 Hg
      1 box of cookies needs 5 Hg
      1 box of cupcakes needs 15 Hg
      $5 \cdot x_1 + 15 \cdot x_2 \leq 480$
    - Available milk 160 litres
      1 box of cookies needs 4 litres
      1 box of cupcakes needs 4 litres:
      $4 \cdot x_1 + 4 \cdot x_2 \leq 160$
    - Available flour 1190 Hg
      1 box of cookies needs 35 Hg
      1 box of cupcakes needs 20 Hg
      $35 \cdot x_1 + 20 \cdot x_2 \leq 1190$
  - Objective function
    - Maximize the profit
      profit of 1 box of cookies 13$
      profit of 1 box of cupcakes 23$
      $\max 13*x_1+23*x_2$
  - Any other constrain?
    $x_1 \geq 0; x_2 \geq 0$

Example 1 - Confectionery's problem (cont.)

To summarize:

x_1:cookies, x_2: cupcakes

$$\max 13x_1 + 23x_2$$
$$5x_1 + 15x_2 \leq 480$$
$$4x_1 + 4x_2 \leq 160$$
$$35x_1 + 20x_2 \leq 1190$$
$$x_1, x_2 \geq 0$$