Introduction to Decomposition
– *Using projections to solve problems*

Thomas Stidsen

thst@man.dtu.dk

DTU-Management
Technical University of Denmark

DTU

Outline

- General information
- MIP/ILP examples
- Practical example
- Algorithms: Branch and Bound
- Software tools:
  - GAMS
  - CPLEX

General Information

This course is about mathematical decomposition techniques used to make hard (MIP) problems solvable. By decomposition we mean that one (large/hard) problem is decomposed into a number (2 or more) smaller more manageable problems.

The Lecturer: Thomas Stidsen

- Name: Thomas Stidsen: tks@imm.dtu.dk
- Nationality: Danish.
- Languages: Danish and English.
- Education: Ph.D. OR/mathematics and Msc. in Computer Science
- Teaching: Teaching in Linear Programming (42112) and Large Scale Optimization using Decomposition (42132), Optimization using Meta-heuristics (42133) and Advanced OR (42134).
- Research: Optimization of telecommunication networks

Literature

The course has a website:
http://www2.imm.dtu.dk/courses/02717/
On the website is the lecture plan which contains all the lectures download-able as pdf files. I will change the lectures during the course, but the version on the day of the lecture should be correct.
Regarding course material:

- We will use copies (handed out after the lecture).
- Later I may hand out some more material.

Projects

There will be two different projects:

- Benders Decomposition algorithm: 10/3 - 24/3
- Column Generation: 7/4 - 28/4
- The projects are specified in the lecture plan, there is 2 weeks for each project and there will be a questioning hour in the week in the middle.

The three algorithms

We will focus on three types of decomposition algorithms:

- Benders decomposition algorithm
- Dantzig-Wolfe decomposition/column generation
- Lagrange decomposition

Why consider decomposition algorithms ?

- Some problems cannot be solved in any other way
- Decomposition of the problems may reveal hidden features in the problem
- Heuristics may be developed based on decomposition algorithms

Example problems

There are a number of example problems in chapter 1.3:

- The Knapsack problem
- Location Models
- Set Partitioning/Set Cover problem

Knapsack problem

The knapsack problem is an "‘easy"’ NP-hard problem which shows up in many different contexts:

- Often the problem arises after column generation.
- Examples of use: Capital budgeting, robbery ....

Knapsack

objective: maximize

$$\sum_i c_i \cdot x_i$$

s.t.

$$\sum_i w_i \cdot x_i \leq W$$

$$x_i \in \{0, 1\}$$

Facility Location

A classical problem: Where to put distribution centers/factories with capacity given a certain number of customers:

- given m customers and n **possible** factory positions

Facility Location

- $c_{i,j}$ the cost of supplying customer j (out of the m) from a possible plant in location i (out of the n)
- f_i cost of establishing a plant at position i and s_i capacity of plant i
- d_j requiered ammount of delivery to customer j
- y_i is the binary variable specifying if a plant is located in position i
- $x_{i,j}$ is the variable specifying howmuch supply customer j should be supplied from plant i

Mathematical Formulation of location problem

objective: minimise

$$\sum_i \sum_j c_{i,j} \cdot x_{i,j} + \sum_i f_i \cdot y_i$$

s.t.

$$\sum_i x_{i,j} = d_j \qquad \forall j$$

$$\sum_j x_{i,j} - s_i \cdot y_i \leq 0 \qquad \forall i$$

$$x_{i,j} \geq 0 \qquad y_i \in \{0,1\}$$

A little diversion ...

- The facility location problem is NP-hard ...
- ... but a "'‘relatively"' easy NP-hard problem ...
- Chosen because of it’s simplicity ....:
  - A solution is uniquely defined by the binary solution vector which defines which factories are established
  - It is easy to ensure feasibility of a solution (just ensure that at least one factory is open ...)
  - The capacitated version of the problem is significantly more complex ...

Crew Scheduling (Set Partioning/Set Covering)

- A classic problem which is **widely** used to solve many different problems.
- Very often the problem arises after column generation.
- Examples of use: Crew scheduling, Truck scheduling, Routing

Set Partioning formulation

objective: minimise

$$\sum_i c_i \cdot x_i$$

s.t.

$$\sum_i a_i^j \cdot x_i \;=\; 1 \qquad \forall j$$

$$x_i \in \{0, 1\}$$

Set Covering formulation

objective: minimise

$$\sum_i c_i \cdot x_i$$

s.t.

$$\sum_i a_i^j \cdot x_i \geq 1 \qquad \forall j$$

$$x_i \in \{0, 1\}$$

Set-Covering/-Partitioning comments

Set Partioning As a practical example with the ILP/MIP problems we will consider the Set-Partioning/-Partitioning problems:

- A classic problem which is widely used to solve many different problems.
- Very often the problem arises after column generation.
- Examples of use: Crew scheduling, Truck scheduling, Routing

Combinatorial Explosion

	10	50	100	300	1000
$5N$	50	250	500	1500	5000
$Nlog_2N$	33	282	665	2469	9966
N^2	100	2500	10.000	90.000	7 digits
N^3	1000	125.000	7 digits	8 d.	10 d.
2^n	1024	16 d.	31 d.	91 d.	302 d.
$N!$	7 d.	65 d.	161 d.	623 d.	“incompreh.”
N^N	11 d.	85 d.	201 d.	744 d.	“incompreh.”

Smart enumeration

- In an **explicit** enumeration all possible solutions are investigated in order to find the optimal solution.
- In an **implicit** enumeration all possible solutions are in the **worst-case** investigated in order to find the optimal solution.

The Branch & Bound Algorithm

while SET_OF_BRANCHES $\neq \emptyset$ **do**
  Select branch $B \in$ SET_OF_BRANCHES
  SOLVE B
  if B infeasible : **FATHOM**
  elseif B integer : **FATHOM**
  elseif B worse than incumbent : **FATHOM**
  else branch
end while

Bounding

The **ESSENTIAL** point in a branch & bound algorithm is to get a good (and fast) bounding ... Different methods are used:

- Linear Programming relaxation ... (old fashion B & B)
- Cutting plan: Branch & Cut Algorithm
- Lagrangian Relaxation: ?
- Column Generation: Branch & Price

Solvers

During the decomposition part of the course you will need to solve LP problems. For that you could use:

- CPLEX
- GAMS (modelling language which gives access to CPLEX)

I will suggest that you use CPLEX ...

CPLEX program

```
minimize
        y1 + 3y2
subject to
c1:    5y1 + 1y2 >= 3
c2:    3y1 + 3y2 >= 5
bounds
          42 <= y1  <= 50
binary
          y2
integer
          y1
end
```

Another CPLEX program

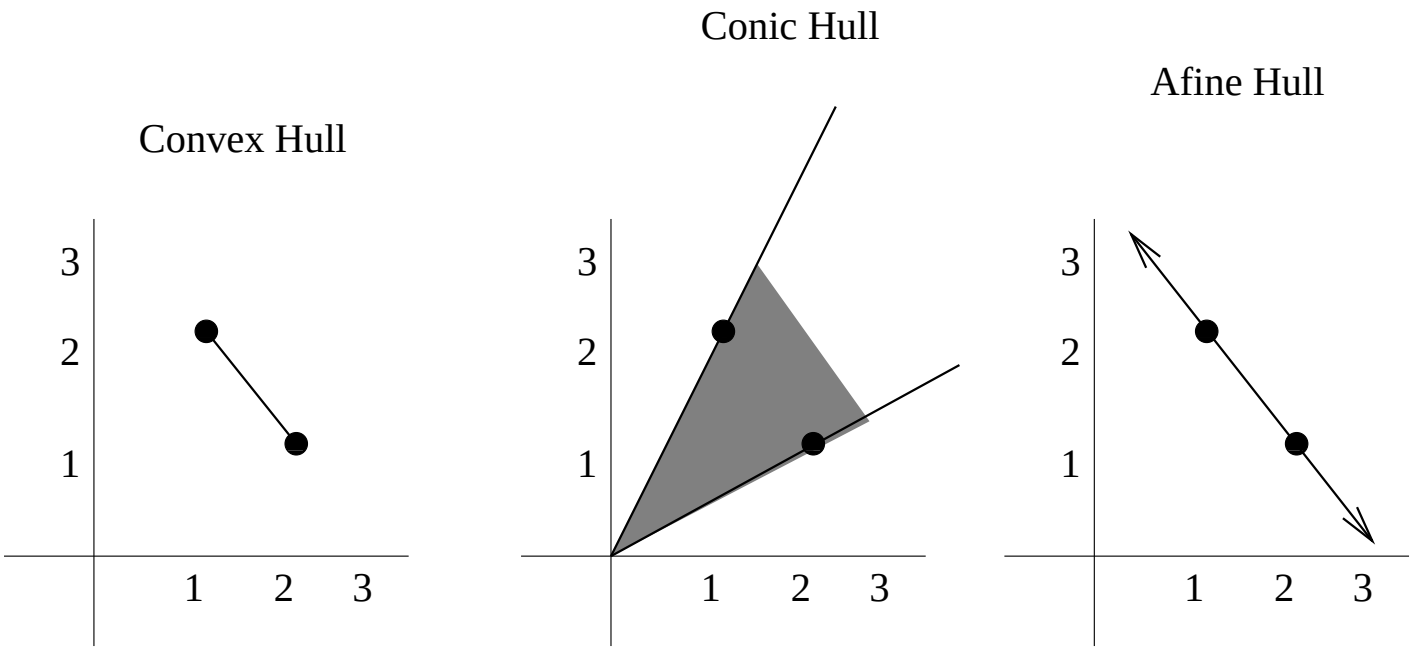
```
\Problem name: SET_COV.lp

Minimize
 R000: C001
Subject To
 R001: C001 - C002 - C003 - C004 - C005 - C006
           - C007 - C008 - C009 - C010 - C011  = 0
 R002: C002 + C003 + C004 + C006 + C007 + C009 + C010 + C011 >= 1
 R003: C002 + C003 + C004 + C006 + C007 + C010 >= 1
 R004: C002 + C004 + C006 + C008 + C009 >= 1
 R005: C002 + C004 + C005 + C007 + C010 + C011 >= 1
 R006: C002 + C004 + C005 + C006 + C007 + C009 + C010 + C011 >= 1
Bounds
      C001 Free
 0 <= C002 <= 1
 0 <= C003 <= 1
 0 <= C004 <= 1
 0 <= C005 <= 1
 0 <= C006 <= 1
 0 <= C007 <= 1
```

Appendix A2, I

Contains many definitions in a very compact form: **Convex Hull**,**Conic Hull** and **Affine Hull**, defined by:

- $\lambda_1 x^1 + \lambda_2 x^2 + \ldots + \lambda_k x^k$, where $\sum_{i=1}^{k} \lambda_i = 1$ and $\lambda_i \geq 0$ (convex hull)
- $\lambda_1 x^1 + \lambda_2 x^2 + \ldots + \lambda_k x^k$, where $\lambda_i \geq 0$ (conic hull)
- $\lambda_1 x^1 + \lambda_2 x^2 + \ldots + \lambda_k x^k$, where $\sum_{i=1}^{k} \lambda_i = 1$ (affine hull)

Appendix A2, II

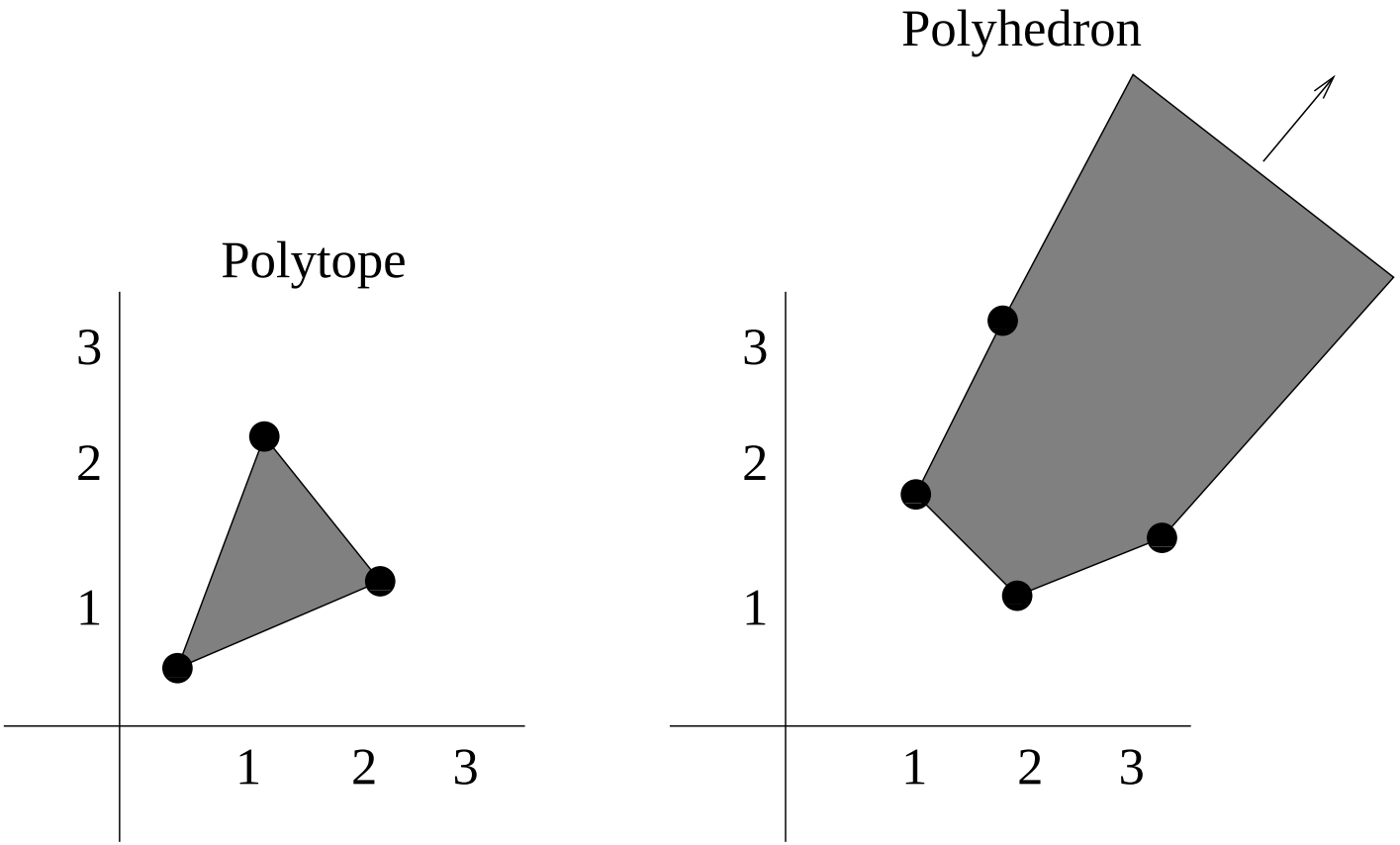


Polytope and Polyhedron definition I

We will talk a lot in this course about two special geometrical structures:

- The polytope
- The polyhedron

Polytope and Polyhedron definition II



Polytope and Polyhedron definition III

For both constructs:

- They are defined in 2 dimensional spaces, 3 dimensional spaces and n dimensional spaces
- They consists of corner points and phases (in 2 dimensional space, lines).
- The difference (OR definition !) is that the polytope is bounded the polyhedron is "infinite"
- Notice that these definitions are OR definitions (different definitions can be found at e.g. wikipedia ...).