IO543: Scheduling

Lecture 5

Brian Denton
Associate Professor, IOE

IOE Bld 2893
btdenton@umich.edu

Today

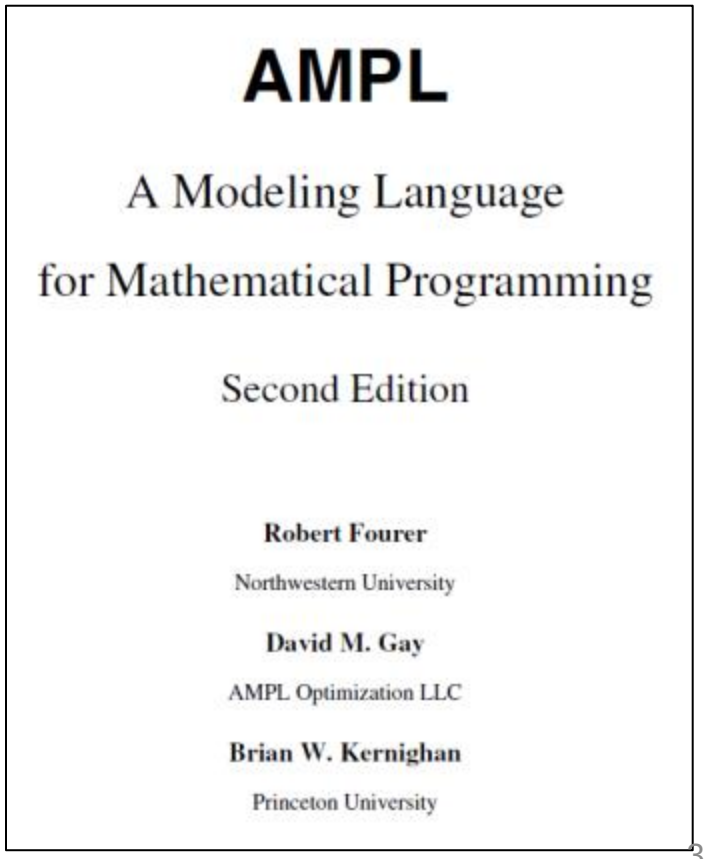
Goals:

- Show $\alpha|\beta|Lmax$ reduces to $\alpha|\beta|\sum T_j$
- Learn a little about AMPL
- Solve $1 \mid r_j \mid L_{\max}$ via branch-and-bound

AMPL

AMPL Resources on IO543 CTools site:

- AMPL software for Mac and Windows systems
- AMPL book
- IOE510 Tutorial on AMPL


AMPL

A Modeling Language

for Mathematical Programming

Second Edition

Robert Fourer

Northwestern University

David M. Gay

AMPL Optimization LLC

Brian W. Kernighan

Princeton University

AMPL

AMPL is a modeling language that can be used to solve many types of problems

- Linear programs
- Integer programs
- Nonlinear programs

AMPL separates the **problem** from the **instance**

AMPL: Example

knapsack.mod

```
set N;

param p {j in N};
param d;
param w {j in N};

var X {j in N} binary;

maximize Total_Jobs: sum {j in N} w[j] * X[j];

subject to Due_Date: sum {j in N} (p[j]) * X[j] <= d;
```

AMPL: Example

knapsack.dat

```
data;
#item index definition
set N := item1, item2, item3, item4, item5, item6, item7, item8, item9, item10;
#item sizes, p, and weights w
param:     p   w  :=
  item1   1    1
  item2   1    2
  item3   1    3
  item4   1    4
  item5   1    5
  item6   1    6
  item7   1    7
  item8   1    8
  item9   1    9
  item10   1    10;
#knapsack size, d
param d := 4;
```

AMPL: Example

knapsack.run

```
model .\Knapsack\knapsack.mod;
data .\Knapsack\knapsack.dat;
option solver cplex; #change default solver to cplex
solve; #solve the knapsack problem
display X; #display the optimal decision variables
```

This is an AMPL script stored in directory .\Knapsack in the AMPL executable directory. To run this....

1) Start AMPL
2) To solve the model type: include .\Knapsack\knapsack.run;
3) To reset AMPL for another problem type: reset;

Limited student versoin available at: http://ampl.com/

Full version available via CAEN Labs

Getting Started

Install AMPL and use it to solve the following instance of $1 \mid \mid \sum U_j$

Jobs	1	2	3	4	5	6	7	8	9	10
w_j	10	9	8	7	6	5	4	3	2	1
p_j	5	7	12	3	8	7	5	1	2	4

Due Date = 30 days

Maximum Lateness: $L_{\max}$

<u>Definition:</u> **Lateness** is the difference between the completion time and the due date.

- The **maximum lateness** is:

$$L_{max} = \max_{i \in \{1,\ldots,n\}} L_i \text{ where } L_i = C_i - d_i$$

Maximum Lateness: L_{max}

Proposition: $1 \mid \mid L_{max}$ is solved by sorting jobs in or order of increasing due dates, known as the **earliest due date (EDD)** first rule

Proof: complete in class

- What is the optimal schedule?

jobs	1	2	3	4
p_j	4	2	6	5
d_j	8	12	11	10

Release dates $1 \mid r_j \mid L_{\max}$

- The problem $1 \mid r_j \mid L_{\max}$ restricts the job start times to be no earlier than the jobs release date, r_j

- This problem is important because it is commonly **used as a heuristic** for more complex scheduling problems (e.g. flow shops, job shops)

Release dates $1 \mid r_j \mid L_{\max}$

<u>Theorem (3.2.4 Pinedo):</u> The problem $1 \mid r_j \mid L_{\max}$ is NP-Hard.

Proof Idea: 3-PARTITION reduces to $1 \mid r_j \mid L_{\max}$.

Branch & Bound for $1 \mid r_j \mid L_{\max}$

Branching

- Level 0: single (root) node and no jobs have been scheduled
- Level 1: *n* nodes such that job *j* is scheduled first at node *j*
- Level *k*: jobs in the first *k* positions have been specified

Note: We will use “best bound” as the strategy for picking which node to branch from next.

Branch & Bound for $1 \mid r_j \mid L_{\max}$

Fathoming Nodes

At a given node at level k:

- Let *J* be the set of unscheduled jobs indexed by j_k
- Let *t* be the time when jobs in J can start
- Only create a node for job j_k if

$$r_{j_k} < \min_{l \in J}(\max(t, r_l) + p_l)$$

Branch & Bound for $1 \mid r_j \mid L_{\max}$

Bounding

- $1 \mid r_j, prmp \mid L_{\max}$ is a **relaxation** of $1 \mid r_j \mid L_{\max}$
- The **Preemptive Earliest Due Date (EDD)** rule is optimal for $1 \mid r_j, prmp \mid L_{\max}$ and therefore provides a lower bound (LB)
- If Preemptive EDD gives a non-preemptive schedule then it is a **candidate schedule**

Branch & Bound for $1 \mid r_j \mid L_{\max}$

Preemptive Earliest Due Date Rule:

1) Sort jobs in order of increasing due date

2) Select among jobs with $r_j \leq t$ the one with the earliest due date

3) Process j until time t' , the time at which a job j′ has an earlier due date and $r_{j'} \leq t'$; Set $t =$ t′ and return to step 2.

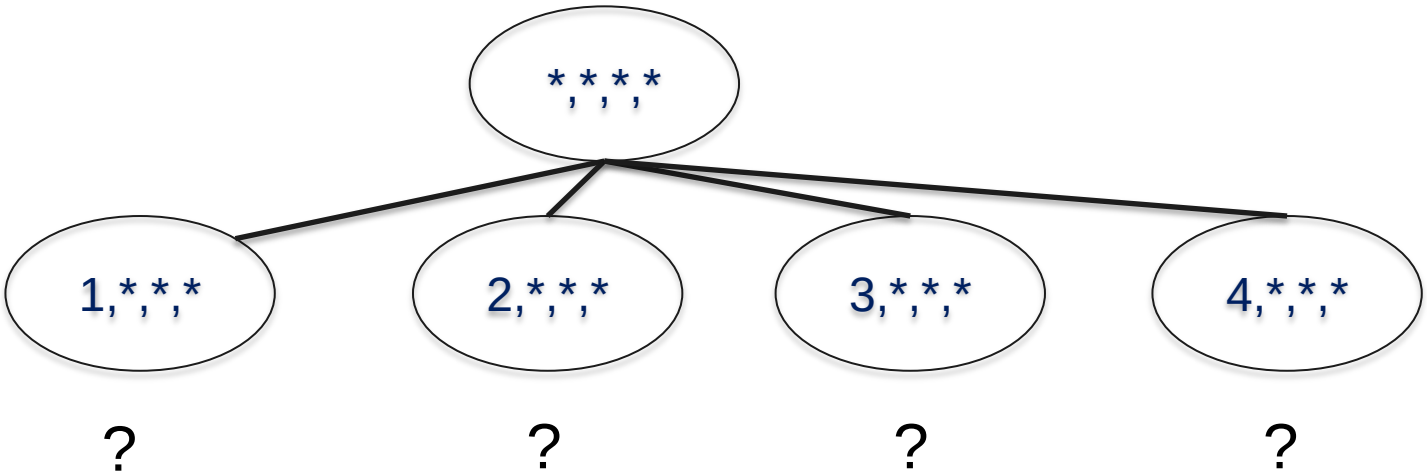
Example

Use Branch-and Bound to solve the following instance of $1 \mid r_j \mid L_{\max}$:

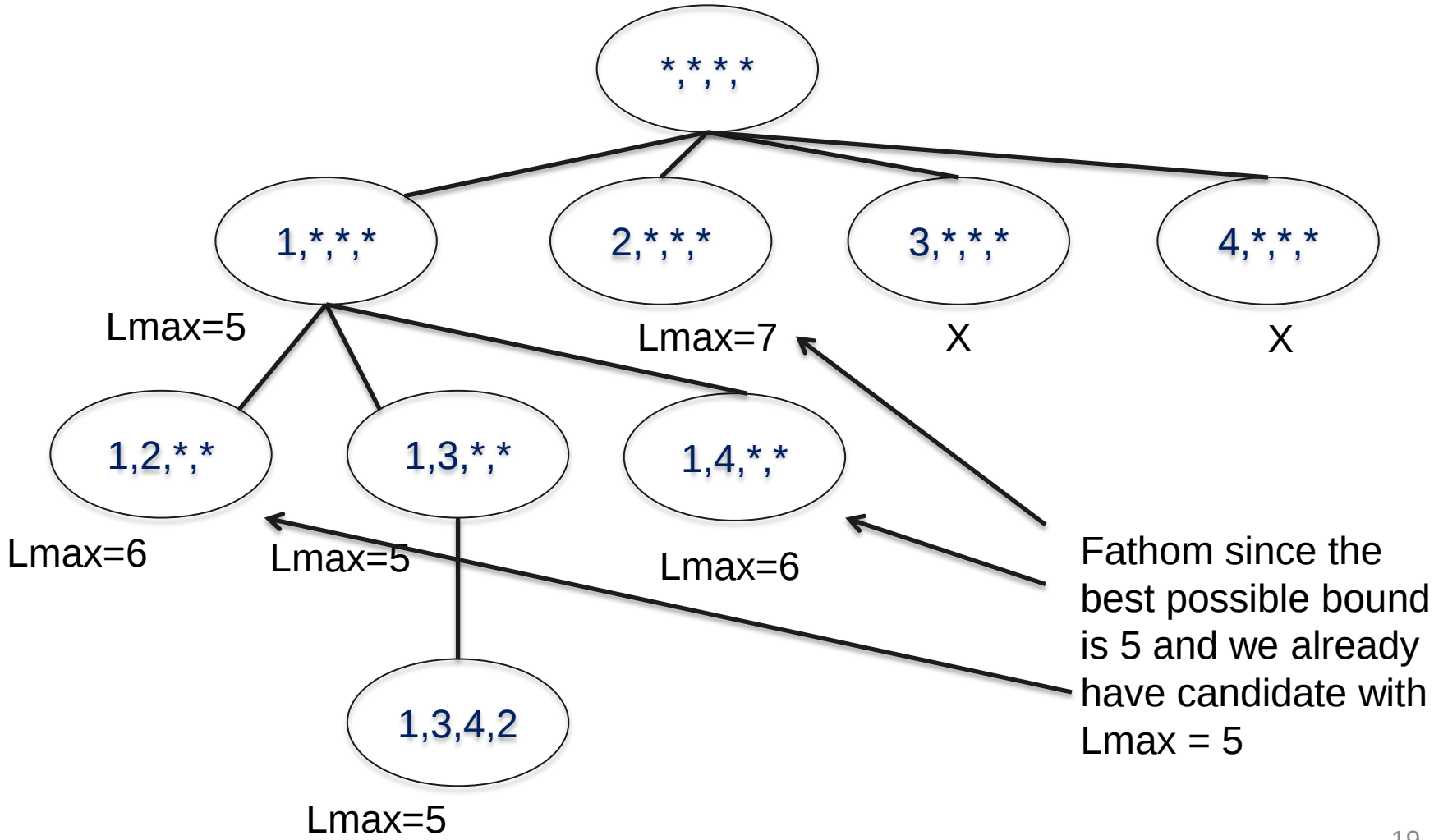
jobs	1	2	3	4
p_j	4	2	6	5
r_j	0	1	3	5
d_j	8	12	11	10

Start by solving level 0: $1 \mid r_j, prmp \mid L_{\max}$

Branch-and-Bound for $1 \mid r_j \mid L_{\max}$



Branch-and-Bound for $1 \mid r_j \mid L_{\max}$



Next Time

For next time read sections 3.4 of Pinedo.

- we will discuss Dynamic Programming (DP) and use it to show EDD is optimal for $1\,|prec|L_{max}$