CS 524: Introduction to Optimization
Lecture 11 : Critical Paths

Michael Ferris

Computer Sciences Department
University of Wisconsin-Madison

October 9, 2025

Widgetco

We're making widgets

Widgetco is about to introduce a new product. One unit of this product is produced by assembling subassembly 1 and subassembly 2. Before production begins on either subassembly, raw materials must be purchased and workers must be trained. Before the subassemblies can be assembled into the final product, the finished subassembly 2 must be inspected.

	Activity	**Duration**
A	Train Workers	6
B	Purchase Raw Materials	9
C	Make Subassembly 1	8
D	Make Subassembly 2	7
E	Inspect Subassembly 2	10
F	Assemble Subassemblies	12

Your Challenge

Schedule activities to complete assembly in minimum time

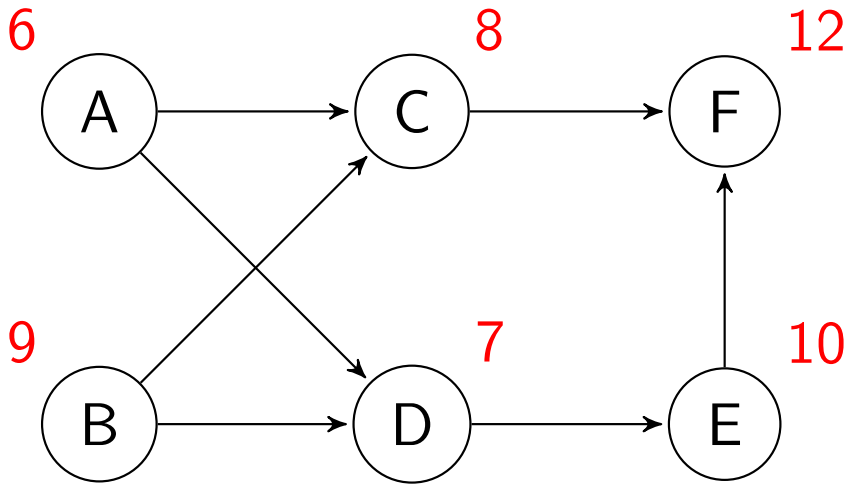
Wash and Go With

- Project Scheduling: PERT (Project Evaluation and Review Technique)
- Often used synonomously with CPM: Critical Path Method

PERT

- I: Set of projects
- $P \subset I \times I$: Precedence relationships
- a_i: Duration of activity $i \in I$

Precedence relationship (P) for Widgetco



Modeling PERT

Variables

- t_i: Time activity starts

Constraints

- j cannot begin before i finishes:

$$t_j \geq t_i + a_i \qquad \forall (i,j) \in P$$

Objective

- Minimize the latest job completion time (makespan).

$$\min \max\{t_1 + a_1, t_2 + a_2 \ldots, t_{|I|} + a_{|I|}\}.$$

Mini-Max

- Minimax will haunt you

$$
\begin{aligned}
T^* = \quad & \min_{z,t} \quad z \\
& \text{s.t.} \quad && z \geq t_i + a_i, && \forall i \in I \\
& && t_j \geq t_i + a_i, && \forall (i,j) \in P \\
& && t_i \geq 0, && \forall i \in I
\end{aligned}
$$

How can we find critical projects, or the critical path?

Critical Path

- An activity is critical if by increasing its duration, the time to complete the project increases.
- How can we find critical projects, or the critical path?
- What do the dual variables in the previous LP mean? Spoiler: positive dual variables λ_{ij} identify a critical path

---

$$
\begin{aligned}
\max \quad & \sum_{(i,j)\in P} a_i \lambda_{ij} + \sum_{i \in I} a_i \pi_i \\
\text{s.t.} \quad & \sum_{i \in I} \pi_i = 1 \\
& -\pi_i - \sum_{j \in I:(i,j)\in P} \lambda_{ij} + \sum_{j \in I:(j,i)\in P} \lambda_{ji} \leq 0, \ \forall i \in I \\
& \pi_i, \lambda_{ij} \geq 0
\end{aligned}
$$

Final Exercise (14widgetco.ipynb)

1. Fix $z = T^*$
2. Solve max $\sum_{i \in I} t_i$: Gives latest possible start times
3. Solve min $\sum_{i \in I} t_i$: Gives earliest possible start times
4. Bottleneck Projects: Which projects have early time = late time?
5. Note relationship to marginal values

Notes

- When there are multiple critical paths (see cell where duration['C'] is reset to 17 for example), then the marginal values only identify *one* critical path, not all of them. However the bottleneck code does find all.
- Can find bottleneck projects with more complex python code and the value of T^* as shown at end of notebook.

Course Project

- The fundamental goal of the course is to equip you with the experience and background necessary to model and implement a real, large-scale optimization problem.
- We will do a class project in which you will define, model, solve, analyze, and explain the solution(s) to a decision problem of your own choosing.
- Typically (and ideally), students come to class having an idea of the project they would like to do.
- We can give some additional ideas (special office hour: Oct 14 3-4pm)

Typical Project Expectations

- Write and submit (Nov 4 at 9AM) a one page pdf outline of project and get it approved by me.
- Respond to annotated questions about project
- Implement an optimization model in `GAMSPy`, describe/output results in a form reasonable for the application. Describe, modify, and implement improvements to the original format if necessary.
- At end of course, turn in a completed application in a notebook, along with all necessary files to run in a zip archive.
- A potential oral defense of project.

Project: essentially a kind of case study

Projects `http://www.neos-guide.org/Case-Studies`

- A "web" description of a problem, understandable to the general public.
- A model and submission that would solve the problem and provide some form of output/visualization.
- Ability to change data of problem, and then resolve to gain more intuition on the model.
- Also see Examples section of GAMSPy user guide.
- Also see the github of GAMSPy model library, specifically the notebooks directory for sources.

- Delivered as a Jupyter notebook.
- Make clear what you have done, and detail your contribution to the "question at hand".

What you need to do

- Each student must:
  1. form a team of at most 2 people from the class
  2. individually submit a one page (12pt font, 1 inch margins) pdf spec of the project
  3. spec for each team member must be identical
- Spec must include 5 sections:
  1. What is the issue being addressed?
  2. Where does the data come from and how will it be obtained?
  3. What is the optimization problem underlying this project?
  4. What are the deliverables?
  5. Other points for me to consider when evaluating.
- I will provide short feedback on this spec.
- Final submission (in the form of a Jupyter notebook and all source files in a zip archive) is due at 12:25 on Dec 12. Each application must use `GAMSPy` and have components that address each of the points noted in the project outline above.

Repeat: typical project expectations

- Write one page (pdf) outline of project (by Nov 8 at NOON) and get it approved by me. Respond to hand scrawled questions about project
- Implement an optimization model in `GAMSPy`, describe/output results in a form reasonable for the application. Describe, modify, and implement improvements to the original format if necessary.
- By Dec 12 at 12:25, submit:
  - a Jupyter notebook, or
  - with prior permission a GAMS/MIRO, GAMS/Matlab or GAMS/R application,
- that includes the description of the problem, the model, how to run it, some visualization and an ability to change data and run again. If you used data or other files, submit these as well as a zip archive so project can be run by me.
- Only if you ran into significant problems with certain aspects of the project, you should detail (in at most 1 page pdf) what the changes you made were, and why they were necessary/time consuming.
- A potential oral defense of project.

Grading

- Projects will be marked out of e.g. 100 (will be scaled so score is correct percentage of final grade)
- Individual projects will get this score.
- For group (of 2) projects, each student will get identical score that will be 0.92 of original project score.

Discrete variables

- In many modeling situations some or all of the variables are constrained to take on only integer values.
- In addition to `positive` and `free variables`, `GAMSPy` has `integer variables` $x \in \mathbb{Z}$ and `binary variables` $x \in \{0, 1\}$.
- Any model that involves discrete variables and (otherwise) linear constraints is a `MIP` and not a linear program. The `model` statement must be modified:

```
mipmod = Model(m,
    equations=m.getEquations(),
    problem=Problem.MIP,
    sense=Sense.MIN,
    objective=Sum(j, x[j]), )
```

- Such problems are much harder to solve as we now see.
- Note that replacing `MIP` by `RMIP` in the above model statement relaxes the integer constraints so the variable are continuous and the resulting model is then a linear program (but is a "relaxation").

Semiint, semicont or sos (other discrete) variables

- Note that `GAMSPy` can use semiint or semicontinuous variables that are also discrete variables. See the "Variables" section of the user guide.
- Semicontinuous is a variable that is either zero, or else in a given range $[L, U]$. Define `semicont` variable `x`, `x.lo[:] = L`, `x.up[:] = U`
- Semiint is the same, except that there is also an integer restriction. It's a variable that is either zero or else an integer in the range $[L, U]$.
- Not all mip solvers allow these variable types so we often just model using binary and integer variables.
- There are also `SOS1` and `SOS2` variables as we shall use later.

Integer Programs are hard!

- IP models can be *very* much more difficult to solve than LP models.

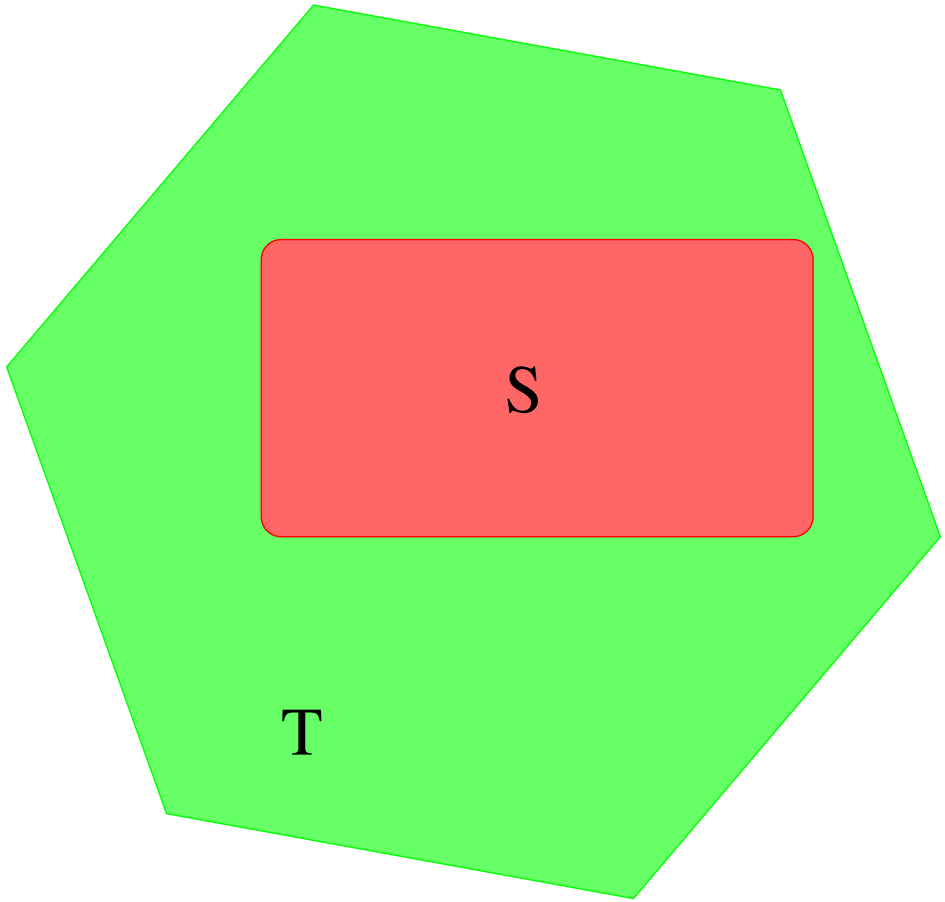
I'm not kidding

IP models can be *very* much more difficult to solve than LP models.

- It is important that you have a handle on...
  1. How to build a problem that is likely to be solved – Proper formulation is important!
  2. The general ideas of how integer programming problems are solved. Branch and Bound.

It's All About Relaxations

- Let $z_S = \max f(x) : x \in S$
- Let $z_T = \max f(x) : x \in T$



Question Time

- What can we say about z_S and z_T
- $z_S \leq z_T$?
- $z_S \geq z_T$?
- It depends?

Obvious, but Important Stuff

$$\boxed{z_T \geq z_S !!!}$$

- What if we get lucky?
- If x_T^* is an optimal solution to $\max f(x) : x \in T$
- And $x_T^* \in S$, then
- x_T^* is an optimal solution to $\max f(x) : x \in S$
- (If we were to replace max by min, then $z_T \leq z_S$. Everything is just flipped!)

LP relaxation

maximize

$$z_{IP} = c^T x$$

subject to

$$\begin{aligned} Ax &\geq b \\ x &\geq 0 \\ x_j &\in \mathbb{Z},\ j \in P \subseteq N. \end{aligned}$$

maximize

$$z_{LP} = c^T x$$

subject to

$$\begin{aligned} Ax &\geq b \\ x &\geq 0 \end{aligned}$$

- $z_{LP} \geq z_{IP}$
- If x^* solves the LP relaxation and x^* satisfies the integrality requirements ($x^* \in S$), then x^* solves IP

$x^* \notin S$?

- Then we branch!
- Partition the problem into smaller pieces.
- $x^* \notin S \Rightarrow \exists k \in P$ such that x_k^* is fractional.
- Create two new (restricted IP) problems...

1. In one problem, add the constraint $x_k \leq \lfloor x_k^* \rfloor$
2. In the other problem, add the constraint $x_k \geq \lceil x_k^* \rceil$

What Does All That Fancy Notation Mean?

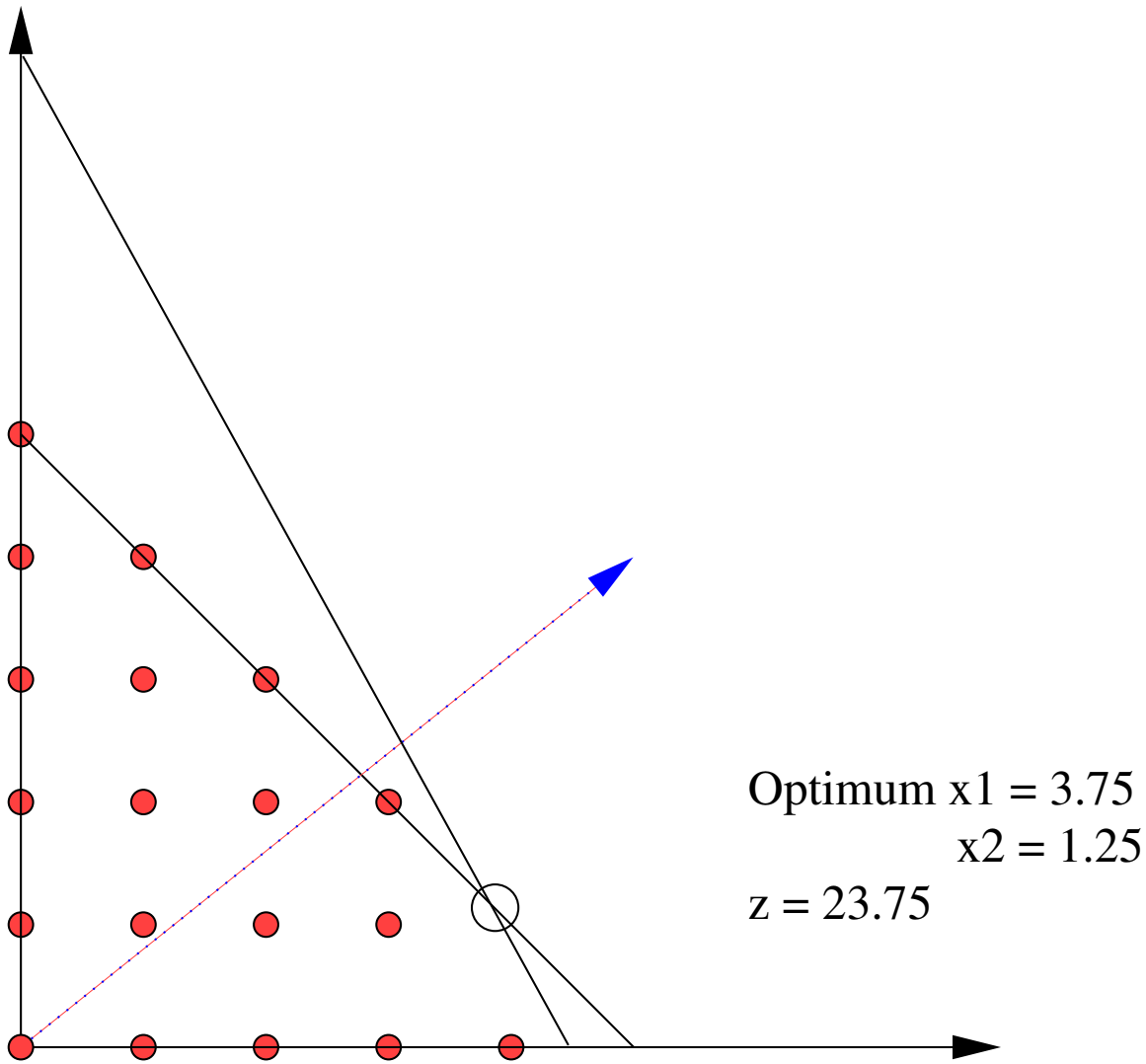
maximize

$$z = 5x_1 + 4x_2$$

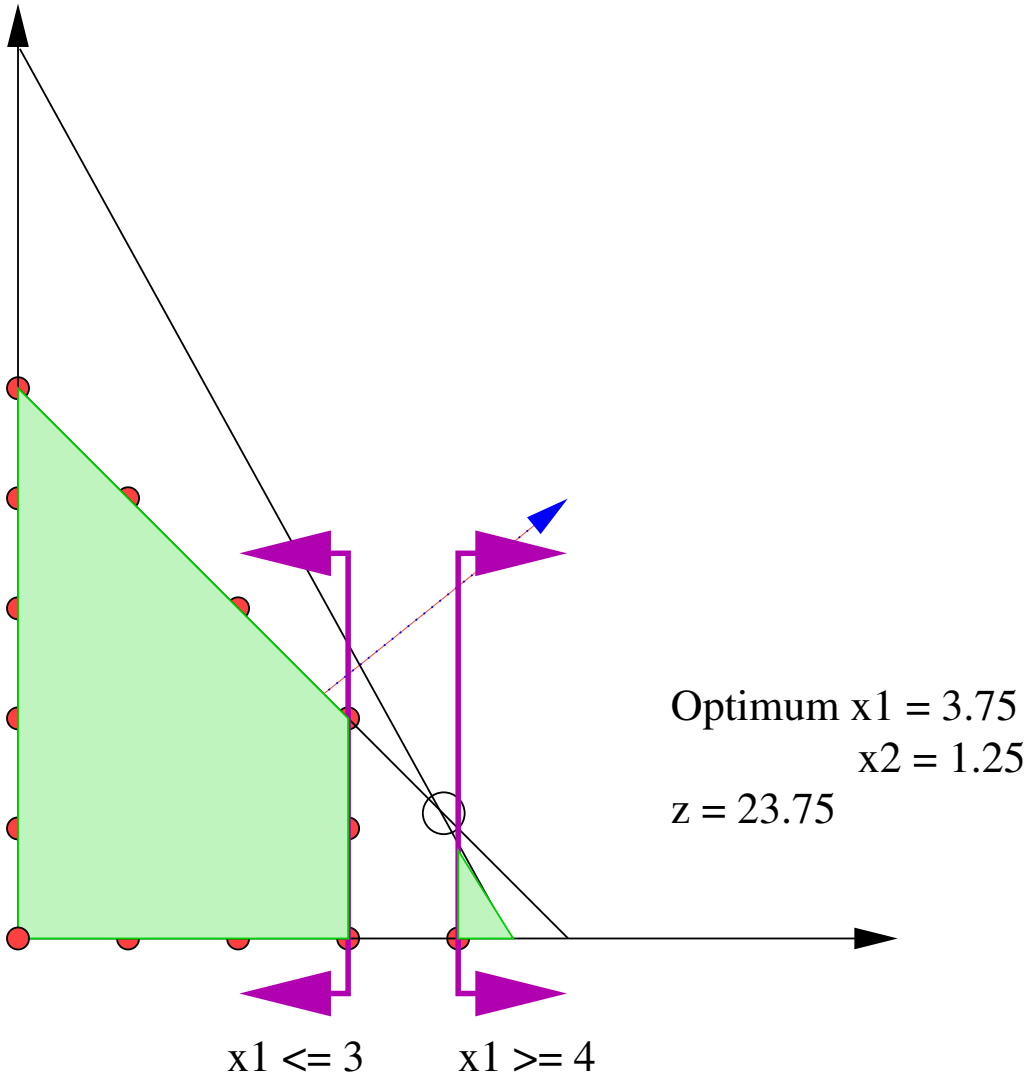
subject to

$$\begin{aligned} x_1 + x_2 &\leq 5 \\ 10x_1 + 6x_2 &\leq 45 \\ x_1, x_2 &\geq 0 \\ x_1, x_2 &\in \mathbb{Z} \text{ (must be integer valued)} \end{aligned}$$

The Feasible Region



Branching



The Branch and Bound Algorithm

- All of the following assumes a *maximization* problem.
- It works equally well for minimization, but you need to replace "lower" by "upper" (and vice verse) everywhere
- Let z_L be a lower bound on optimal objective value. Originally $z_L = -\infty$. The initial list of restricted IP problems to solve consists of just the entire problem.

Branch and Bound (1 of 2)

1. Select a restricted IP problem from the list of open problems.
2. Solve the LP relaxation of the restricted IP problem. If the LP relaxation is infeasible, Go to 1.
3. Otherwise, let x^* be the optimal solution to the LP relaxation and let z^* be its objective value.
4. If x^* satisfies the integer restrictions, then z^* is the optimal value for the restricted IP problem. If $z^* > z_L$, then $z_L := z^*$. (Also keep track of x^* as a candidate solution). Go to 1.

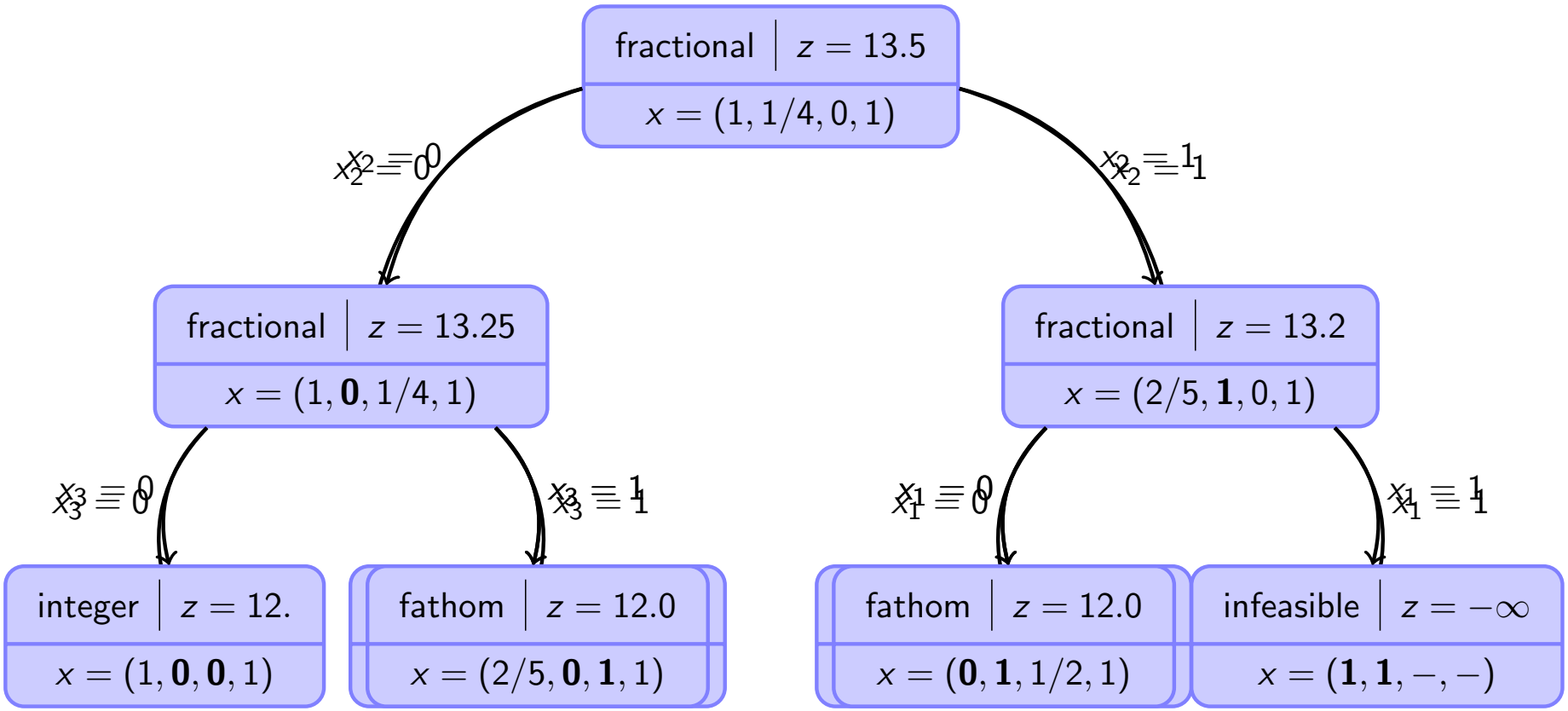
Branch and Bound (2 of 2)

5. Otherwise, x^* does not satisfy the integer restrictions. z^* is an upper bound on the optimal value of the restricted IP problem.
6. If $z^* \leq z_L$, (Fathom) Go to 1. (There is no way that a better optimal solution can be in this restricted region).
7. Otherwise, Divide. Choose some index k such that x_k^* does not satsify the integer restriction. Add two new restricted IP problems to the list:
    1. Include constraint $x_k \leq \lfloor x_k^* \rfloor$
    2. Include constraint $x_k \geq \lceil x_k^* \rceil$
8. Go to 1.

Simple knapsack example (17bandb.ipynb)

$$\max_x 8x_1 + 6x_2 + 5x_3 + 4x_4$$
$$5x_1 + 4x_2 + 4x_3 + 2x_4 \leq 8$$
$$x_1, x_2, x_3, x_4 \in \{0, 1\}$$

Branch and Bound Process (17bandb.ipynb)



- $z_L = -\infty 12.0$, $z_U = 13.513.212.0$
- Can round down objective since must be integer (all entries integer)

Bounds and performance guarantees

- We saw in the description of the algorithm how to get a lower bound z_L on the optimal objective function value
- We can also get an upper bound on the optimal objective value. z_U is the maximum of all of the upper bounds of all of the remaining restriced problems that must be evaluated
- If we only want to solve a problem to within $\alpha\%$ of optimality, we can!!!
- This is really what is great about optimization. Not only do you get a solution, but you can tell your boss that if there is a better solution it can be at most $\alpha\%$ better.

GAMSPy and Tolerances

- `option.relative_optimality_gap = 1.0e-4`
- `option.absolute_optimality_gap = 1.0`

- relative: Stop if $|z_U - z_L| / \max\{|z_L|, |z_U|\} \leq$ `relative_optimality_gap`
- absolute: Stop if $|z_U - z_L| \leq$ `absolute_optimality_gap`

Some Good CPLEX parameters

- mipemphasis (1 for solution finding, 2 for optimality)
- varsel (strong branching 3)
- cuts (-1 no, 5 aggressive),
- probe (-1 no, 3 full),
- lbheur (local branching heuristic),
- preslvnd,
- repeatepresolve,
- subalg,
- nodesel (0 dfs, 2 best estimate),
- heurfreq (-1 for node heuristic off)
- mipstart 1 (use an existing integer solution to start)
- numericalemphasis (1 for extreme caution)

Some Good Gurobi parameters

- cuts -1: auto, 0:off, 3:aggresive
- heuristics (fraction) % of time spent trying to find feasible solutions
- mipfocus: 1 for solution finding, 2 for optimality
- mipstart 1

For further details on options for CPLEX (and Gurobi): Refer to [Klotz and Newman, 2013b] for MIP and [Klotz and Newman, 2013a] for LP.


Klotz, E. and Newman, A. M. (2013a).
Practical guidelines for solving difficult linear programs.
Surveys in Operations Research and Management Science, 18(1-2):1–17.


Klotz, E. and Newman, A. M. (2013b).
Practical guidelines for solving difficult mixed integer linear programs.
Surveys in Operations Research and Management Science, 18(1-2):18–32.