Math 3272: Linear Programming[1] Mikhail Lavrov

Lecture 26: The branch-and-bound method

November 17, 2022 *Kennesaw State University*

1 The bin packing problem

Problem 1. *For mysterious reasons, you are carrying crates of potatoes from one end of campus to another. You have fifteen crates to carry: five* 4*-kilogram crates, five* 5*-kilogram crates, and five* 6*-kilogram crates. You can carry multiple crates at a time, but you're not willing to carry more than* 12 *kilograms in one trip.*

What is the minimum number of trips you'll have to make?

This problem is a special case of the **bin packing problem**, usually phrased in terms of packing items (of different sizes) into the smallest possible number of containers (of some fixed size). Here, each of your trips across campus is a "container".

There are multiple different formulations of the bin packing problem as an integer program. Let's begin with an attempt that might be *intuitive* but actually *is a bad idea.*

1.1 Bin packing by planning each trip

The natural choice of variables in this problem is to have variables that answer the questions: "What will you carry on your first trip? What will you carry on your second trip?" and so on. The identities of the crates don't really matter, only their weight. So suppose you want to answer these questions as: "On my first trip, I will carry three 4-kilogram crates. On my second trip, I will carry one 5-kilogram crate and one 6-kilogram crate. On my third trip, ..."

This suggests using nonnegative integer variables $x_{i,j}$ equal to the number of i-kilogram crates carried on the j^{th} trip: $x_{4,1}, x_{5,1}, x_{6,1}, x_{4,2}, x_{5,2}, x_{6,2}, x_{4,3}, \ldots$. Here, i is always one of 4, 5, or 6. It's a bit of a challenge deciding how far j should go: isn't that the question we're trying to answer? A quick way to settle this is to decide that it's definitely possible to solve the problem in 15 trips, so we'll have j go up to 15, and then we'll try to minimize the number of trips we'll actually use.

Three constraints will tell us that we end up carrying all the crates we have:

$$x_{4,1} + x_{4,2} + x_{4,3} + \cdots + x_{4,15} = 5$$
$$x_{5,1} + x_{5,2} + x_{5,3} + \cdots + x_{5,15} = 5$$
$$x_{6,1} + x_{6,2} + x_{6,3} + \cdots + x_{6,15} = 5$$

To make sure that for each j, we don't exceed our weight limit on the j^{th} trip, we could add constraints

$$4x_{4,j} + 5x_{5,j} + 6x_{6,j} \leq 12$$

[1] This document comes from the Math 3272 course webpage: https://facultyweb.kennesaw.edu/mlavrov/courses/3272-fall-2022.php

for each $j = 1, \ldots, 15$. (Wait on this, though; we'll modify this constraint in a bit.)

The tricky part is figuring out how to minimize the number of trips necessary. One way to do this is with a boolean variable y_j for each trip j (that is, an integer variable with $0 \leq y_j \leq 1$) answer the question: did we take trip j at all? To enforce this interpretation of y_j, we can modify the above constraint to

$$4x_{4,j} + 5x_{5,j} + 6x_{6,j} \leq 12y_j$$

for each $j = 1, \ldots, 15$. If $y_j = 1$, we get back our previous constraint; if $y_j = 0$, then this modified constraint forces us to have $x_{4,j} = x_{5,j} = x_{6,j} = 0$. In other words, if we want to make use of the j^{th} trip, we must set $y_j = 1$. Now, we can

$$\text{minimize } y_1 + y_2 + y_3 + \cdots + y_{15}$$

to minimize the number of trips taken.

This is a valid solution! Why don't we like it? Because there is too much symmetry. There are many essentially equivalent solutions that have different representations. If we take one solution and swap the crates we carry on the first and second trip, then we get a different, equally good solution. We could even "renumber our trips" and say, "We will take 10 trips, but they will be trips 3 through 12" by setting $y_3 = y_4 = \cdots = y_{12} = 1$ and $y_1 = y_2 = y_{13} = y_{14} = y_{15} = 0$. This is equivalent to going on 10 trips numbered 1 through 10.

In integer programming, we don't like having many equivalent solutions. Intuitively, this is because we end up having to wade through many equally good options in the hope of finding a better one, which takes a long time. After we see the branch-and-bound method today, we will be able to say more precisely why symmetry is bad.

1.2 The configuration linear program

The **configuration linear program** (or configuration LP) is a different approach to the bin packing problem; the idea can be applied to many other combinatorial optimization problems like it.

Here, we begin by listing the possible **configurations**: the sets (multisets) of crates that can be carried on a single trip. The ones that are worth considering are

$$(4,4,4) \quad (5,5) \quad (6,6) \quad (4,5) \quad (4,6) \quad (5,6).$$

Here $(4,4,4)$, for example, means that you carry three 4-kilogram crates; $(4,6)$ means that you carry a 4-kilogram crate and a 6-kilogram crate. This list leaves out suboptimal configurations like "carry just two 4-kilogram crates"; this simplifies our setup, though we'll have to address this issue later.

Now we can define nonnegative integer variables $x_{444}, x_{55}, x_{66}, x_{45}, x_{46}, x_{56}$ to be the number of trips taken with each configuration of goods. The total number of trips is just the sum of these variables: $x_{444} + x_{55} + x_{66} + x_{45} + x_{46} + x_{56}$; this is the quantity we will want to minimize.

For each size of crate, we ask for the total number of crates taken of that size to be *at least* 5; for example, we add the constraint

$$3x_{444} + x_{45} + x_{46} \geq 5$$

for the 4-kilogram crates. Why at least 5 and not exactly 5? Because in practice, we might want to carry suboptimal configurations we didn't include: if our first two trips just deal with the 4-kilogram crates, the configurations will be $(4, 4, 4)$ and then $(4, 4)$. We will model that as "taking an extra 4-kilogram crate", with two trips that are both $(4, 4, 4)$.

In this case, our entire linear program is:

$$\begin{array}{rrrrrrrrl} \text{minimize} & x_{444} + & x_{55} + & x_{66} + x_{45} + x_{46} + x_{56} & & & & & \\ \text{subject to} & 3x_{444} & & & + x_{45} + x_{46} & & & \geq & 5 \\ & & 2x_{55} & & + x_{45} & & + x_{56} & \geq & 5 \\ & & & 2x_{66} & & + x_{46} + x_{56} & & \geq & 5 \\ & & & & x_{444}, x_{55}, x_{66}, x_{45}, x_{46}, x_{56} & & & \geq & 0 \\ & & & & x_{444}, x_{55}, x_{66}, x_{45}, x_{46}, x_{56} & & & \in & \mathbb{Z} \end{array}$$

Figuring out the configurations takes a bit of work (and in general, there may be many of them, which is a weakness of this method). However, once we get there, solving this integer program will turn out to be easier.

How do we get there? Well, we can begin by ignoring the integer variables, and pretending they can be fractions. This is called "solving the LP relaxation".

In this case, doing that will tell us that we can solve the problem in $6\frac{2}{3}$ trips,[2] by setting $x_{55} = \frac{5}{2}$, $x_{66} = \frac{5}{2}$, and $x_{444} = \frac{5}{3}$.

Of course, this is not an actual workable solution. But it does give us some important information! We now know for certain that *at least* 7 trips will be necessary.

2 The branch-and-bound method

2.1 Branching

The branch-and-bound method is the first complete method we'll see for solving integer programs.

The first ingredient of this method is the branching. The idea is that we'll start by solving the LP relaxation of a problem, and we'll *keep* solving LP relaxations. However, we can't just keep solving the same linear program, or we'll keep getting the same solution. In fact, we should only solve linear programs that somehow exclude that previous fractional solution we got.

To accomplish this, we **branch** on an integer variable which has been given a fractional value. Whenever our optimal solution ends up giving some integer variable x_i a fractional value $f \notin \mathbb{Z}$, we know that one of two things must hold:

- either x_i is actually at most $\lfloor f \rfloor$ (f rounded down),
- or x_i is actually at least $\lceil f \rceil$ (f rounded up).

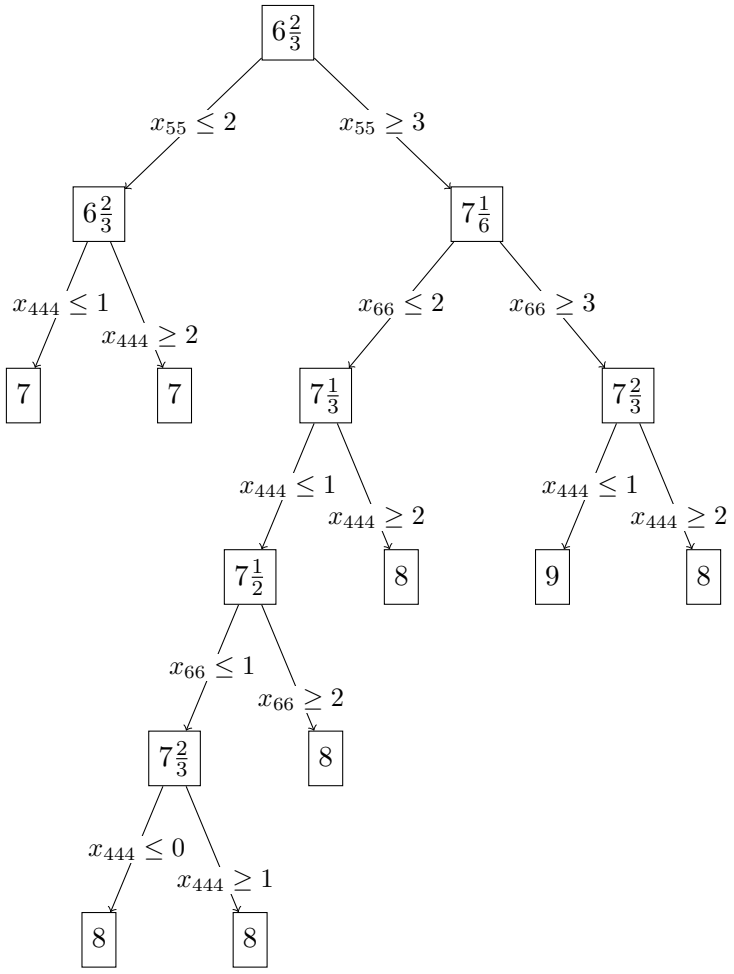
So we can solve two new linear programs: one where we add the constraint $x_i \leq \lfloor f \rfloor$, and one where we add the constraint $x_i \geq \lceil f \rceil$. Both of these exclude our previous solution, because setting x_i to

[2] I am writing the objective value as a mixed fraction, which is rarely done in advanced mathematics, because I want it to be easy to see which integer values are the closest.

f does not satisfy either constraint. However, we do not exclude any solutions where x_i really is an integer, so we have not lost the true optimal solution to our problem.

We do not have to start over from scratch. The dual simplex method is tailor-made for situations where we take a formerly-optimal dictionary and add a constraint that it violates. Even though we get two new linear programs to solve every time we branch, we can hope that we won't have to do too much extra work to solve each one of them.

When we solve the new linear programs, we might still get fractional solutions to each one of them. So we repeat the process, branching from each of those fractional solutions. The number of linear programs can grow quickly. Here is what happens if we start branching from our solution to the potato-crate problem:



In this diagram, each rectangle is labeled with an optimal objective value, and every arrow is labeled with the constraint we added when branching. The rectangles with no branches correspond to places where the linear program obtained an integer optimal solution.

2.2 Pruning

As you can see, it can take a while to end up at integer solutions by branching.

In fact, it is typical to have to deal with many cases in the branch-and-bound method; often, there are exponentially many cases. We saw in the previous lecture that integer programming is very expressive and can handle many different problems; the downside is that it is not usually easy to solve.

However, things are not *as* bad as I've made them seem, because we have only seen half of the branch-and-bound method. In addition to branching, there is **pruning**: discarding branches that are not promising without exploring them.

Here's an example. Suppose that we start solving the potato-crate problem, and get the following results:

1. We solve the original LP and get the the solution

   $$(x_{444}, x_{55}, x_{66}, x_{45}, x_{46}, x_{56}) = (\tfrac{5}{3}, \tfrac{5}{2}, \tfrac{5}{2}, 0, 0, 0)$$

   with objective value $6\frac{2}{3}$. From here, we branch on x_{55}.

2. We try the $x_{55} \geq 3$ branch and get the solution

   $$(x_{444}, x_{55}, x_{66}, x_{45}, x_{46}, x_{56}) = (\tfrac{5}{3}, 3, \tfrac{5}{2}, 0, 0, 0)$$

   with objective value $7\frac{1}{6}$. From here, we branch on on x_{66}.

3. Before doing that, we go back to see what happens with the $x_{55} \leq 2$ branch, and get the solution

   $$(x_{444}, x_{55}, x_{66}, x_{45}, x_{46}, x_{56}) = (\tfrac{5}{3}, 0, 0, 0, 0, 5)$$

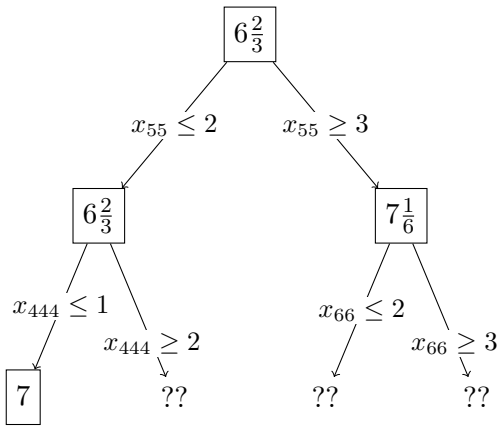
   with objective value $6\frac{2}{3}$, which seems more promising. From here, we branch on x_{444}.

4. We try the $x_{444} \leq 1$ sub-branch of the $x_{55} \leq 2$ branch, and get the solution

   $$(x_{444}, x_{55}, x_{66}, x_{45}, x_{46}, x_{56}) = (1, 0, 0, 1, 1, 4)$$

   with objective value 7. This is our first integer solution!

Here is the diagram of where we've gotten so far:



There are certainly branches we haven't explored yet. But are they worth exploring? We have an integer solution with objective value 7 already: we know we can solve the problem in 7 trips. So the only reason to explore other branches is if they could give us a 6-trip solution.

However, whenever we add more constraints, we can only make the objective value worse. Therefore exploring down from the node labeled $6\frac{2}{3}$ will give us more solutions with objective value at least $6\frac{2}{3}$: if they're integer solutions, the objective value will be at least 7. Exploring down from the node labeled $7\frac{1}{6}$ will be even worse: the smallest integer objective value we can get is at least 8.

So at this point we know we've found an optimal integer solution! We can stop.

The general philosophy is:

1. If ζ^* is the objective value of the best integer solution we've found, we can **prune** nodes that branched from an objective value of ζ^* or worse: we don't explore those sub-branches. They will never give us anything better than what we've already found.
2. In problems where integer solutions have integer objective values, we can do a bit better. For each fractional objective value, treat it as the next-worst integer value when pruning, since that is the best we can do from that branch. (In our example, we prune the other branch leading out of $6\frac{2}{3}$, even though $6\frac{2}{3} < 7$, because there's no integer between $6\frac{2}{3}$ and 7.)

(Sometimes, adding constraints will create an infeasible linear program in one of our sub-branches. These should also be pruned, because adding more constraints to an already-infeasible lineanear program will never yield solutions.)

2.3 Why symmetry is bad

Now that we know how branch-and-bound works, we can say more about why symmetry—that is, having many equivalent integer solutions—is a bad sign in an integer program.

Let's look back at our first formulation of the potato-crate problem, where we had a variable $x_{i,j}$ for the number of i-kilogram crates on the j^{th} trip. A fractional solution might have $x_{5,1} = \frac{4}{5}$: we carry a fraction of a 5-kilogram crate on the first trip. What will happen when we branch on $x_{5,1}$, adding the constraints $x_{5,1} \leq 0$ or $x_{5,1} \geq 1$?

One thing that's likely to happen is that we'll find an equivalent fractional solution with $x_{5,2} = \frac{4}{5}$, instead: we carry the same fraction of a 5-kilogram crate, but on the second trip. We'll have to branch on $x_{5,2}$, then on $x_{5,3}$, and so on through $x_{5,15}$ before we finally start seeing really new solutions. It takes much longer before we get to an integer solution we can actually use to prune our options.

When we have a choice between several integer programs for the same underlying problem, we prefer to pick ones without this type of symmetry: ones where every branch will lead to new solutions.