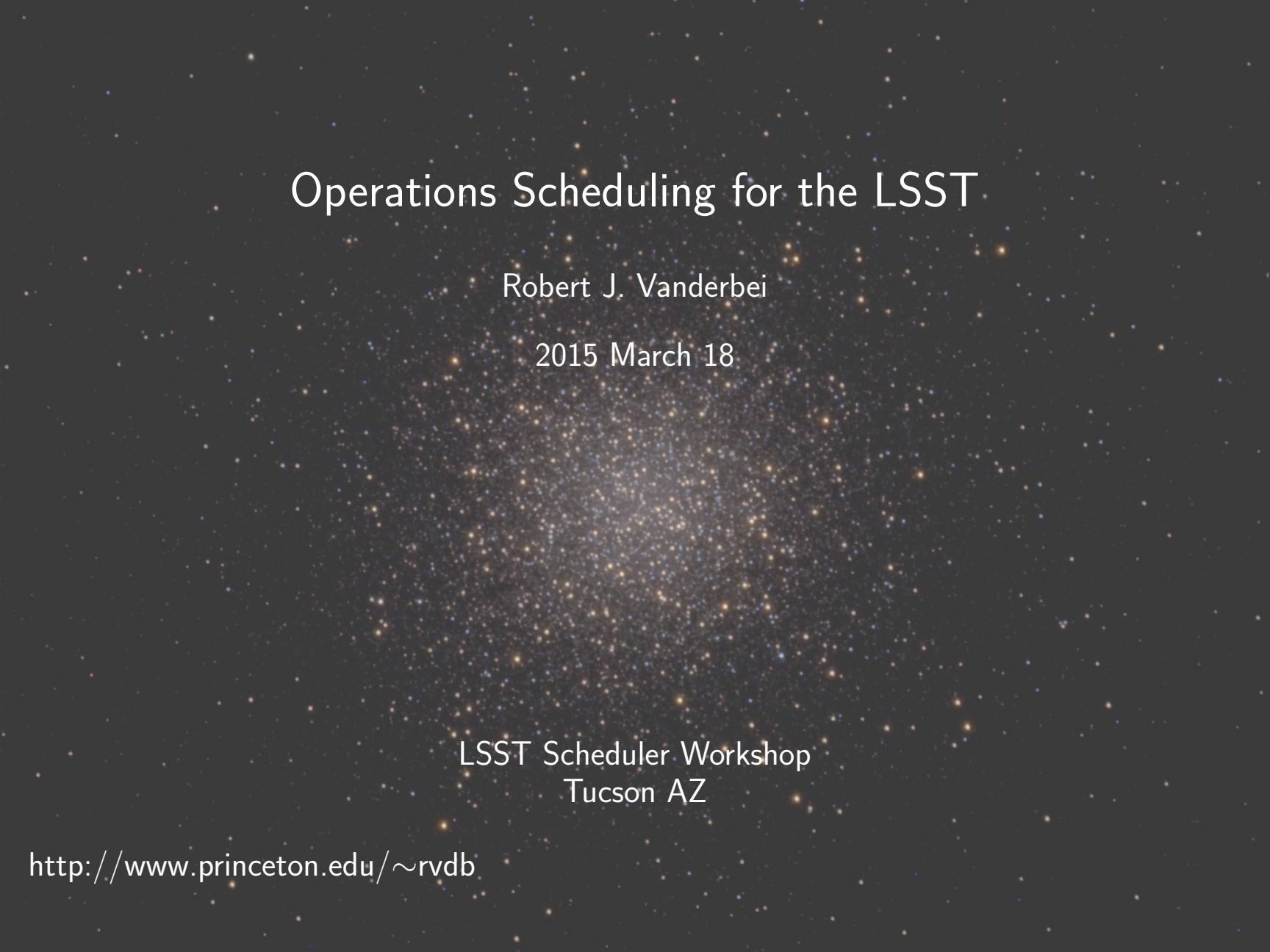


Operations Scheduling for the LSST

Robert J. Vanderbei

2015 March 18

LSST Scheduler Workshop
Tucson AZ

http://www.princeton.edu/∼rvdb

Simple Optimization Problem—Unconstrained

$$\max_{x \in \mathbb{R}^n} \; f(x)$$

The function $f(x)$ is called the *objective function*.

If f is smooth, take the gradient and set it to zero. Solve for x.

If f is strictly concave, there is only one unique place x^* where the gradient vanishes. *It's optimal!*

Here's a *trivial* optimization problem. Suppose that $f(x)$ is a linear function: $f(x) = c^T x$ for some vector c. Then, solution is infinity if $c \neq 0$ and it is zero otherwise.

Slightly Harder—Linear Objective and Constraints

These problems are called linear programming (LP) problems.

Here's an LP in *standard form*:

$$\begin{aligned} \text{maximize} \quad & \sum_{j=1}^{n} c_j x_j \\ \text{subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j \leq b_i \qquad i = 1, 2, \ldots, m \\ & x_j \geq 0 \qquad j = 1, 2, \ldots, n, \end{aligned}$$

In general, constraints can be equalities or inequalities.

If all constraints are equalities, then the problem is again *trivial*.

Linear programming problems can be solved quickly.

If $m \approx n$, then one can expect to solve an LP in about n^3 time.

If the matrix $A = \{a_{ij}\}$ of coefficients is *sparse* then one can expect to solve the problem more quickly.

It is not uncommon to solve sparse problems with a million variables in a few hours.

Going Nonlinear

$$\begin{array}{lll} \text{maximize} & f(x) & \\ \text{subject to} & g_i(x) \leq 0 & i = 1, 2, \ldots, m_{\text{ineq}} \\ & h_i(x) = 0 & j = 1, 2, \ldots, m_{\text{eq}}, \end{array}$$

If $f(x)$ is concave, each $g_i(x)$ is convex, and each $h_i(x)$ is affine (constant plus linear), then the problem is called *convex*.

For convex problems, a locally optimal solution is globally optimal.

Convex problems are about the *same difficulty as LP*.

If the problem is not convex, then there can be local optima that are not globally optimal.

Sometimes this is okay since the solution found will generally be close to an initial estimate provided by the user.

It's not uncommon for the user to have a good idea for an initial estimate.

Finding locally optimal solutions to nonconvex problems is not any harder than solving an LP—about n^3 complexity.

Global vs. Local Solutions

As explained on the previous slide, local solutions near to a supplied starting guess can be found "quickly".

Finding the *globally optimal solution* is *vastly more time consuming*.

In practice, global optimization is limited to cases involving just a small number of variables.

Going Integer

In management-type (and other) problems, it is often the case that an LP has an extra condition that some or all of the variables must be *integer valued*.

Such problems are called *integer programming* (IP) problems:

$$\begin{aligned}
\text{maximize} \quad & \sum_{j=1}^{n} c_j x_j \\
\text{subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j \leq b_i && i = 1, 2, \ldots, m \\
& x_j \geq 0 && j = 1, 2, \ldots, n \\
& x_j \in \mathbb{Z} \quad \leftarrow \textit{integers} && j = 1, 2, \ldots, n.
\end{aligned}$$

The *worst case* complexity of these problems is *exponential*—i.e., 2^n.

That's much worse than the n^3 for LP.

However, because these problems are so common and so important, lots of work has gone into developing solvers that work pretty well on *average*.

In practice, it is often possible to solve an IP with a few thousand variables in a few hours on a modern computer. But, it depends greatly on the problem instance.

Example: Traveling Salesman Problem

A salesman wishes to visit each of n cities and return home.

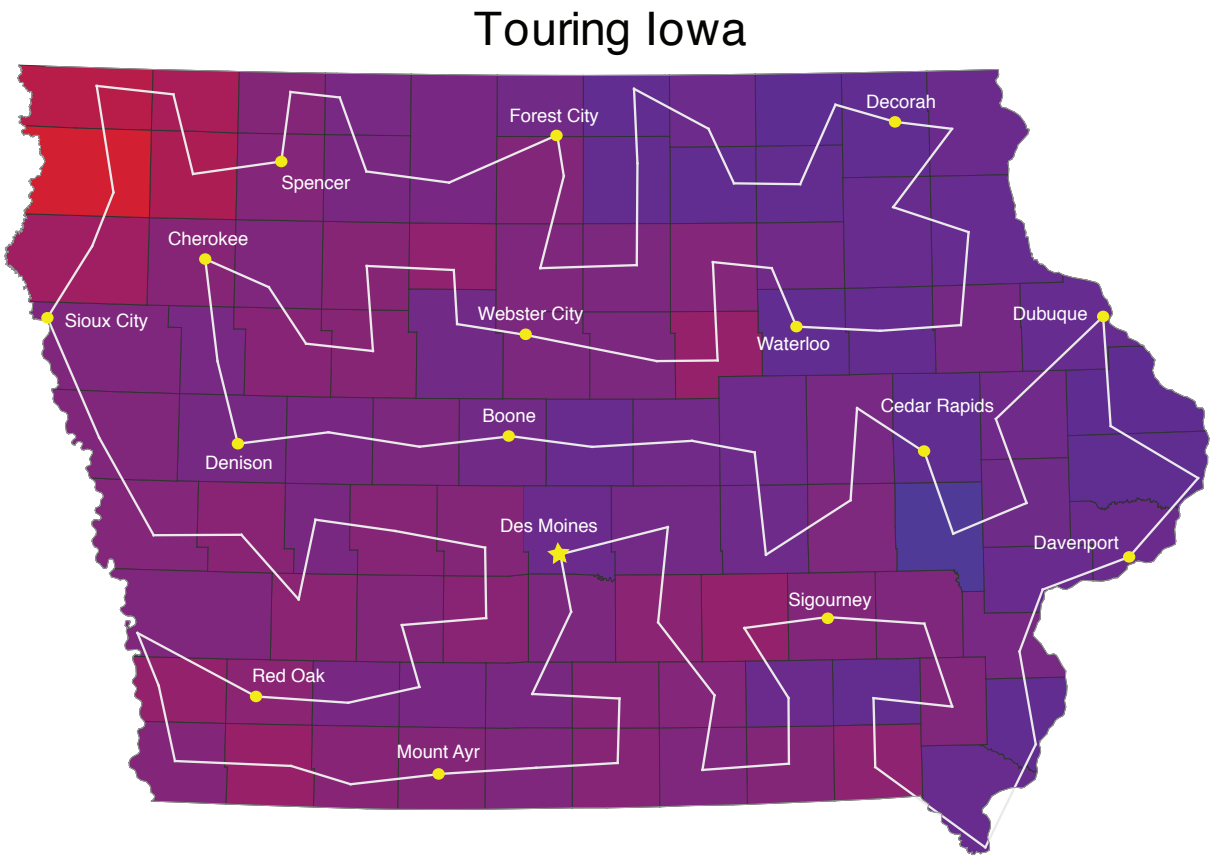
Let x_{ij} be one if city j is visited immediately after city i (zero otherwise).

Let t_i denote where in the order of visits city i is visited (i.e., $t_i = 3$ means that the i-th city is the third city visited on the tour).

Letting d_{ij} denote the distance from i to j, the integer programming problem is:

$$\begin{aligned}
\text{minimize} \quad & \sum_{i,j} d_{ij} x_{ij} \\
\text{subject to} \quad & \sum_{j} x_{ij} = 1, && i = 0, 1, \ldots, n-1, \\
& \sum_{i} x_{ij} = 1, && j = 0, 1, \ldots, n-1, \\
& x_{ij} \in \{0, 1\}, \leftarrow \textit{integers} && i, j = 0, 1, \ldots, n-1, \\
& t_j \geq t_i + 1 - n(1 - x_{ij}), && i \geq 0, j \geq 1, i \neq j, \\
& t_0 = 0, \\
& t_i \in \{0, 1, 2, \ldots\} \leftarrow \textit{more integers} && i = 0, 1, \ldots, n-1.
\end{aligned}$$

Campaign Tour Through Iowa



The TSP went viral (again) on the web a few days ago:

http://www.washingtonpost.com/blogs/wonkblog/wp/2015/03/05/what-if-americas-zip-codes-were-one-big-game-of-connect-the-dots/

http://www.washingtonpost.com/blogs/wonkblog/wp/2015/03/10/a-data-genius-computes-the-ultimate-american-road-trip/

Bang-Bang Solutions

Often the integrality constraints are in fact 0 / 1 constraints; i.e., the variables are *binary*.

In this case, sometimes the *LP-relaxation* automatically produces a binary solution.

Such solutions are called *bang-bang* solutions in the "control" literature.

My favorite example is the *pupil-mask* design problem for producing a star psf that has a very dark section very close to the central lobe of the psf.

This problem is infinite dimensional because the pupil of the telescope is a continuum.

Discretizing the pupil into pixels makes it into a very large (but finite dimensional) linear optimization problem.

Each pixel is then allowed to be some level of "gray" varying between black (opaque) and white (transparent). In other words, we are designing an *apodized filter*.

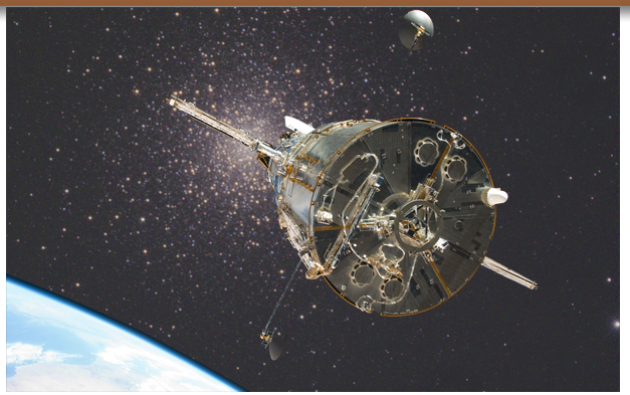
Unfortunately, an apodized filter cannot be made with sufficient precision to achieve the contrast necessary to image exoplanets.

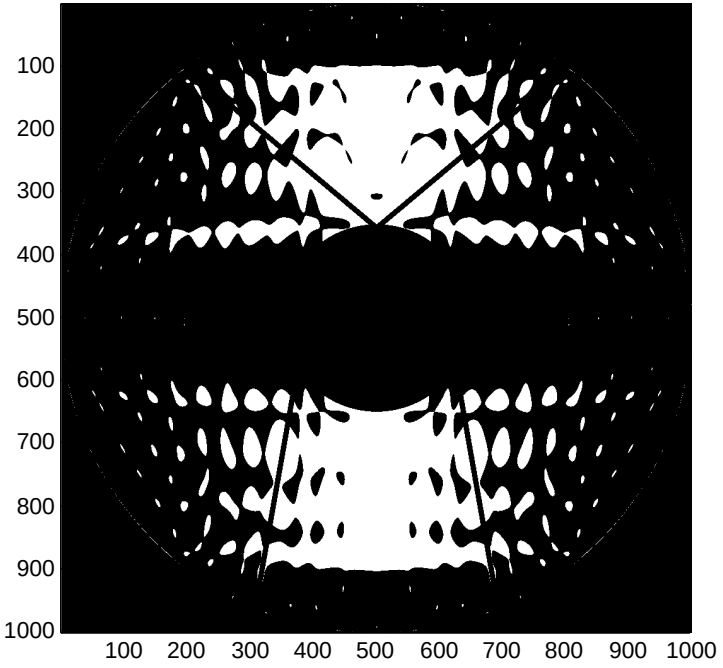
But, a binary mask would work.

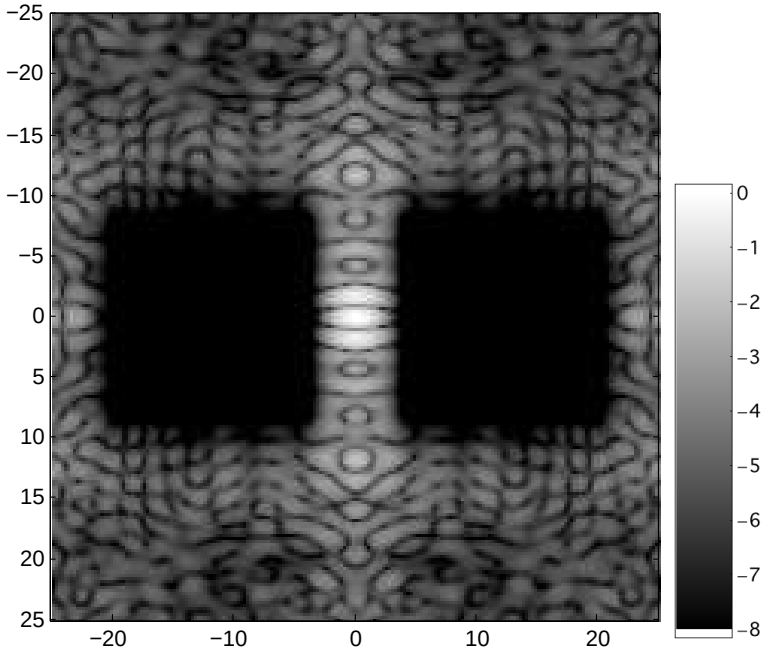
It turns out that the LP-relaxation automatically produces a binary mask...

AFTA/WFIRST Pupil Mask

Originally, five design concepts were proposed.
Our shaped pupil concept (shown here) was selected.
The high-contrast imaging system is being built.
The telescope will launch sometime mid 2020's.







First Thoughts on Scheduling LSST

Let x_{jk} be a binary variable whose value is 1 if target j is the k-th object visited.
Let t_j denote the time at which target j is visited.
Let u_j denote the net "outflow" for target j.

$$\text{minimize} \quad \sum_{i,j} d_{ij}(t_j)x_{ij} \qquad \leftarrow \textit{time/desirability of going from } i \textit{ to } j \textit{ (nonlinear!)}$$

$$\text{subject to} \quad \sum_{i \neq j} x_{ij} - \sum_{k \neq j} x_{jk} = u_j, \qquad j = 1, \ldots, n-1,$$

$$\sum_j u_j = 1, \qquad j = 0, 1, \ldots, n-1,$$

$$\sum_{i,j} x_{ij} \geq \textit{num\_targets}, \qquad j = 0, 1, \ldots, n-1,$$

$$u_0 = -1,$$

$$t_j = \sum_i (t_i + \Delta t_{ij})x_{ij}$$

$$t_0 = 0$$

$$x_{ij} \text{ and } u_j \in \{0, 1\}, \leftarrow \textit{integers} \qquad i, j = 0, 1, \ldots, n-1,$$

plus constraints about cadence and weather...

This looks like a (generalized) *network flow* problem. Their solutions are often *bang-bang*.

One Objective vs. Many

In most real-world problems, it is not absolutely clear what is the most appropriate objective function.

In portfolio selection (from finance), Markowitz won the Nobel prize for introducing the important trade-off between minimizing risk and maximizing reward.

In the high-contrast imaging problem, there are many competing objectives: one could

- maximize the amount of light that gets past the mask, or
- minimize the inner-working angle in the image plane, or
- minimize the amount of light allowed in the so-called dark sector, or
- maximize the size of the dark sector itself.

One can (and I did) devote many years of their life to understanding the trade-offs between such alternative objectives.

I suspect that the LSST scheduling problem also involves competing objectives—*there won't be just one right answer*.

Optimal vs. Just Pretty Good

Optimization problems arising from physics (such as optimal control or mask-design for high-contrast imaging) must be solved precisely or the solution has no physical meaning.

However, for operational problems, such as how to schedule the operation of LSST, optimality is less important.

What matters is finding a "good" solution, which means a solution that seems as good or better than what one might expect a human to arrive at.

It is often possible to use simple/fast heuristics to produce a good solution much more quickly than one could produce an "optimal" solution.

Example 1. *Traveling Salesman Problem (TSP)*. It's really easy to find very good solutions. It's famously hard to prove a solution is optimal.

Example 2. *Territory Assignment Problem*. Similar to TSP...

Territory Assignment Problem

Large pharmaceutical companies employ hundreds of highly paid sales reps.

It's important to assign roughly the same number of doctors to each sales rep.

It's also important to make each sales rep's "territory" as compact/convex as possible.

The "original" formulation of this problem was an integer programming optimization problem. It took about a *month* to solve one instance of the problem.

When asked to help make things go faster, my first question was: "Do you require a solution to the IP or would some other approach be acceptable?"

Answer: other approaches are acceptable as long as the solution is "good".

I reasoned that if the USA were one-dimensional instead of two, the problem would be easy: just divide the number of doctors by the number of sales reps, round that ratio to the nearest integer and then make compact clumps of this size starting at one end and extending to the other. The only reason the real problem is hard is because the USA is a two-dimensional set.

I implemented code using a *space-filling-curve* to map the two-dimensional space into a one-dimensional space and then employed the 1D solution described above.

Because the space-filling-curve is very kinky, the "1D" solution isn't "good" enough. I then used a Kernighan-Lin-style "two-swap" algorithm to improve the solution.

I got "excellent" solutions in just a *few minutes* of cpu time.

I predict that such smart heuristics will prove to be the best way to go for LSST.

How Can I Help?

- I'm familiar with the various algorithms (and codes) for optimization.
- I'm familiar with interfacing optimization software with other software.
- I'd be happy to help compare/assess various solution approaches.
- I might have some good ideas for fast heuristics.

Thank You!

Example: Airline Equipment Scheduling

Given:

- A set of *flight legs* (e.g. Newark to Chicago departing 7:45am).
- A set of aircraft.

Problem: which specific aircraft should fly which flight legs?

Model:

- Generate a set of feasible *routes* (i.e., a collection of legs which taken together can be flown by one airplane).
- Assign a cost to each route (e.g. 1).
- Pick a minimum cost collection of routes that exactly covers all of the legs.

Let:

$$
\begin{aligned}
x_j &= \begin{cases} 1 & \text{if route } j \text{ is selected,} \\ 0 & \text{otherwise} \end{cases} \\
a_{ij} &= \begin{cases} 1 & \text{if leg } i \text{ is part of route } j, \\ 0 & \text{otherwise} \end{cases} \\
c_j &= \text{cost of using route } j.
\end{aligned}
$$

An Integer Programming Problem:

$$
\begin{aligned}
\text{minimize} \quad & \sum_{j=1}^{n} c_j x_j \\
\text{subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j = 1 \qquad i = 1, 2, \ldots, m, \\
& x_j \in \{0, 1\} \qquad j = 1, 2, \ldots, n.
\end{aligned}
$$

An example of *set-partitioning problems*.