 Notes by Stephen Chenney

- Algorithms
  1. Problem specific aspects
  2. Cutting Plane method
  3. Recent developments
- Airline Crew Scheduling
  1. The Problem
  2. Set Covering and Partitioning
  3. Generating Crew Pairings
  4. TRIP
  5. New system

1 Algorithms (continued)

1.1 Problem Specific Aspects

There are many places in the branch-and-bound and implicit-enumeration algorithms in which domain specific knowledge can help the algorithms. Here we mention some of these.

- In branch-and-bound domain specific knowledge can be used to choose the next variable to branch on in the search algorithms, and which branch to take first. For example based on the domain we might know that certain variables are "more" important than others, or that solutions are more likely to lie to one side or the other of the branch. The implicit enumeration algorithm is similar—domain specific knowledge can help choose a variable to fix and which branch to try first.
- Domain specific knowledge can be used to help generate cuts. In the cutting plane algorithm (later in these notes), a constraint must be added at each step. Domain knowledge can be used to choose a more restrictive constraint, moving you toward the solution faster. Domain knowledge also determines if approximations are acceptable, which allows for more aggressive cutting planes.
- Domain specific heuristics can be used for predicting upper bounds for more aggressive pruning of the tree. Knowing an upper bound a-priori means that many branches can be eliminated based on their linear program's cost even before any integer solution is found.

- Domain specific knowledge can be used to help quickly check for feasibility. For instance, knowing which constraints are more likely to be unsatisfiable suggests an ordering for testing constraints.
- Domain specific knowledge can help solve the linear program or specify that a particular approximate solution is good enough. See the Airline Crew Scheduling problem later in these notes for an example of both these cases.

1.2 Cutting Plane Algorithm

The basic idea is to add a new constraint (geometrically, a new hyperplane) at each step. This plane will cut out non-integer solutions and move the solution closer to an integer solution. Figure 56 shows the intuition. A new plane is added at each step to cut off the corner of the feasible region containing the linear program solution. In Figure 57 the limiting case is shown, where the new constraints approximate a curve bounding the acceptable integer solutions.

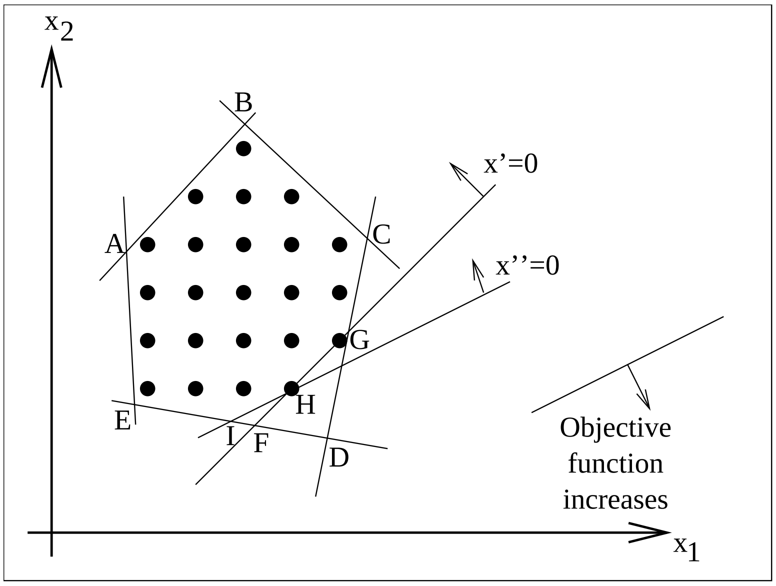

Figure 56: The Basic Approach. Initially, the solution to the linear program is at D. This is not an integer solution, so constraint x' is added, moving the linear solution to F. Then constraint x" is added, giving H as the linear, and integer, solution.

The cutting plane algorithm is:

function $IP(\mathbf{A}, c)$
$x = LP(\mathbf{A}, c)$
while not(integer(x) or infeasible(x))
begin
  add constraint to $\mathbf{A}$
  $x = LP(\mathbf{A}, c)$
end

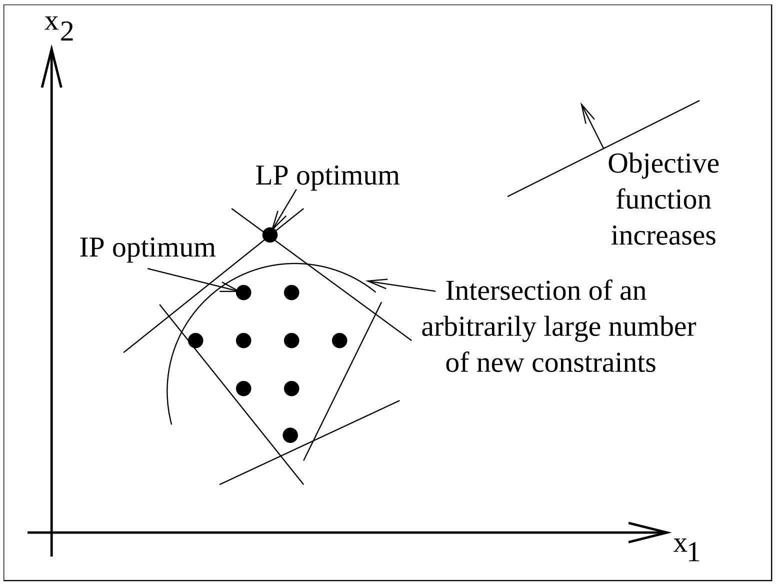

Figure 57: Curve Phenomenon in the Basic Approach

Under some circumstances (problem specific) it is acceptable to add constraints that cut off the optimal solution if it makes finding a good solution faster.

1.3 Recent Developments

- Good formulations—heuristics and theory. The goal is to get an LP solution as close as possible to the IP solution. Examples: Disaggregation and adding constraints (cuts).
- Preprocessing—Automatic methods for reformulation. There's some interesting graph theory involved.
- Cut generation (branch and cut). The idea is to add cuts during the branch-and-bound.
- Column generation—improve formulation by introducing an exponential number of variables. Solve an integer subproblem.

2 Airline Crew Scheduling (A Case Study)

2.1 The Problem

American Airlines, and all other airlines, must schedule crew (pilots and flight attendants) on flight segments to minimize costs.

- over 25,000 pilots and flight attendants.
- over 1.5 billion dollars per year in crew cost.

This process assumes that flight segments are already fixed. In other words, the airline has already decided which planes will fly where and when. Obviously, determining the aircraft schedules is an optimization problem in itself, and the 2 problems are not independent. But treating them independently makes them tractable and gives good answers.

In these notes 2 methods are covered:

- 1970-1992: TRIP (Trip Reevaluation and Improvement Program) This is a local optimization approach which starts with a (hopefully good) guess.
- 1992-present?: Global optimization, approximately.

2.2 Set Covering and Partitioning

Before describing airline crew scheduling we briefly discuss the general problem of set covering and partitioning. Set partitioning will be used in the crew scheduling algorithm.
Given m sets and n items.

$$\begin{aligned} A_{ij} &= \begin{cases} 1, & \text{if set } j \text{ includes item } i \quad \text{Columns = sets} \\ 0, & \text{otherwise} \quad\quad\quad\quad\quad\;\; \text{Rows = items} \end{cases} \\ c_j &= \text{cost of set } j \\ x_j &= \begin{cases} 1, & \text{if set } j \text{ is included} \\ 0, & \text{otherwise} \end{cases} \end{aligned}$$

Set Covering:

$$\begin{array}{ll} \text{minimize} & cx \\ \text{subject to} & Ax \geq 1, x \text{ binary} \end{array}$$

Set Partitioning:

$$\begin{array}{ll} \text{minimize} & cx \\ \text{subject to} & Ax = 1, x \text{ binary} \end{array}$$

As a standard application, consider a distribution problem with some possible warehouse locations (the sets) and some areas (the items) that must be served. There is a cost associated with building and operating each warehouse, and we want to choose the warehouses to build such that all the areas are covered and the cost of building and operating is minimal.

For example:

set	members	cost
s_1	$\{a, c, d\}$	.5
s_2	$\{b, c\}$	.2
s_3	$\{b, e\}$	.3
s_4	$\{a, d\}$	.1
s_5	$\{c, e\}$	.2
s_6	$\{b, c, e\}$	.6
s_7	$\{c, d\}$	.2

$$A = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix}$$

The best cover is $\{s_2, s_4, s_5\} = .5$. The best partition is $\{s_4, s_6\} = .7$.

2.3 Generating Crew Pairings

A *crew pairing* is a set of flights a crew member will fly, starting and finishing at their *base*. Pairings are broken into duty periods (one per day, separated by overnight rests) which are broken into segments (individual flights). An example, with base at DFW, is:

Duty period 1

Sign in: 8:00

DFW 9:00 - 10:00 AUS (segment 1)

AUS 11:00 - 13:00 ORD (segment 2)

ORD 14:00 - 10:00 SFO (segment 3)

Sign out: 15:15

Overlay is SFO

Duty period 2

Sign in: 8:00

SFO 8:00 - 9:00 LAX (segment 4)

LAX 10:00 - 11:45 SAN (segment 5)

SAN 13:00 - 19:30 DFW (segment 6)

Sign out: 19:45

National pairings typically last 2-3 days (*i.e.*, 2-3 duty periods). International pairings may last for a week, because there are generally fewer, longer flights. A crew member will work 4 or 5 pairings per month. Pairings are actually assembled into groups, called bidlines, which is yet another optimization process. Recent work has focussed on the bidline problem, but we will not discuss it here. A crew member bids for bidlines each month. Bidlines are then distributed according to seniority based rank.

The optimization problem is: given all the segments that need to be covered for a month, find the optimal grouping of segments into pairings, according to the constraints and costs we will define next.

The constraints, apart from the obvious ones (such as segments in a pairing can't overlap in time) are related to union and Federal Aviation Agency (FAA) rules. For example:

- Maximum 8 hours of flying time per duty period
- Maximum 12 hours total duty time (which is flying plus time between flights).
- There is a minimum lay-over time, which depends on the number of hours flown in the previous duty period.
- There is a minimum time between flight in the duty period, so that the crew can move from plane to plane.

The cost of a pairing can include both direct and indirect costs (such as employee satisfaction). Example contributors to the cost are:

- Total duty period time - obviously related to the salary paid.
- Time away from base - also related to salary paid, but also part of employee satisfaction.
- Number and location of overlays - hotels (at least) are paid for by the airline, and some cities are more expensive than others.
- Number of time-zone changes - an employee satisfaction, and maybe union requirement.
- Cost of changing planes.

The overall goal is to **cover all segments with a valid set of pairings that minimize cost**. We must also consider the number of crew available at crew bases (this will also be a factor in scheduling flights in the first place). The problem is simplified in part because flights are roughly the same each day, but it is made more difficult because flight schedules change at the end of the month. For this reason, in part, pairings are generally done on a monthly basis.

One possible approach is to consider all valid pairings and generate costs for each. Then solve it as a set covering problem:

$$\begin{array}{ll} \text{minimize} & cx \\ \text{subject to} & Ax \geq 1, x \text{ binary} \end{array}$$

$$\begin{aligned} A_{ij} &= \begin{cases} 1, & \text{if pairing } j \text{ covers segment } i \\ 0, & \text{otherwise} \end{cases} \\ c_j &= \text{cost of pairing } j \\ x_j &= \begin{cases} 1, & \text{if pairing } j \text{ is included} \\ 0, & \text{otherwise} \end{cases} \end{aligned}$$

Problem: There are billions of possible pairings. For the methods we will discuss, a subset of possible pairings is considered and in some way optimized. For example, with the segments:

1. DFW 900-1200 LGA
2. LGA 1300-1500 ORD
3. ORD 1600-1800 RDU
4. ORD 1700-1900 DFW
5. RDU 1900-2100 LGA
6. RDU 1900-2100 DFW
7. LGA 1400-1600 ORD
8. DFW 1600-1800 RDU

we might generate the possible pairings (this is not an exhaustive list):

1. DFW 9-12 LGA 14-16 ORD 17-19 DFW
2. LGA 13-15 ORD 16-18 RDU 19-21 LGA
3. ORD 16-18 RDU 19-21 DFW 9-12 LGA 13-15 ORD
4. DFW 16-18 RDU 19-21 DFW
5. DFW 16-18 RDU 19-21 LGA 14-16 ORD 17-19 DFW

which corresponds to the matrix

$$A = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

where each column is a pairing (set) and each row is a segment (element that needs to be covered). Someone can then price each pairing for us giving us a cost vector

$$c = [c_1 c_2 c_3 c_4 c_5]$$

2.4 TRIP

TRIP (Trip Reevaluation and Improvement Program) was in use until around 1992. As the name suggests, it reevaluates a previous solution and then attempts to improve it. The initial solution is generally taken to be that of the previous month, modified to adjust for schedule changes. We then repeat the following steps until no improvement is found:

1. Select a small set of pairings from the current solution. The pairings selected should come from roughly the same region (covering similar cities). We will generally choose 5-10.

2. Generate all the valid pairings that cover the same segments. There will typically be a few thousand. Note that if we chose pairings from disjoint cities in the previous step, we would be able to generate no new pairings for this step.

3. Generate the cost of each new pairing.

4. Optimize over these pairings by solving the set partitioning problem. This will generate a new set of pairings that cover the same segments but at minimal cost.

The advantage of this method is that it solves small sub-problems. The big disadvantage is that it only does local optimization. If finding a better solution requires optimizing over a large number of possible pairings and segments, we won't find it.

2.5 New System

Described by Anbil, Tanga and Johnson in 1992 (IBM Systems Journal, 31(1)). The system may have been in use since then, although we don't know if it has been replaced by a better technique. A later paper on the bidline problem suggests it hasn't. The basic algorithm is:

1. Generate a large pool of pairings (around 6 million).

2. Solve an LP approximation using a specialized technique.

3. Uses branch-and-bound for the IP solution, but with heuristic pruning.

Each linear program solved takes around 1 hour. The algorithm does not guarantee the best solution for the chosen pool because of pruning, but it does generate significantly better solutions than the previous method.

2.5.1 Generating a Pool of Pairings

This is speculative, since the authors say very little about it.
We need to generate 6 million initial pairings out of billions of possible pairings.

1. - Node: time and airport
   - Edge: (directed) flight segment, wait time, or overlay
   - Edge weight: "excess cost" of edge. The excess cost is the amount that edge would cost above the minimum it could cost. For example, there might be a fixed cost for a wait independent of its length (just to get on and off the plane) and we don't include this, and a variable cost based on the length, and we do include this.

2. Find the 6 millions shortest paths in the graph that start and end at a crew base.

Nodes in the graph will generally have many edges coming in and going out. Incoming edges will be wait times and overlays ending at that time, and flights arriving at that time. Outgoing edges will be overlays or wait times starting at that node, and flights leaving at that time.

The algorithm given is a heuristic that prefers short time away from base.

2.5.2 Solving the LP Approximation

An LP problem with 6 million columns is too big to solve directly, so instead the following is done (referred to as the SPRINT approach, Figure 58):

1. A = select a small set of m columns (each column is a pairing). $m \approx 5000$
2. Repeat until the optimal is found:
   - Optimize based on A
   - Use the basis generated to "price" the remaining variables. Then set A to the m new "best" columns. ($i.e.$, minimum $r = c_b B^{-1} N - c_n$)

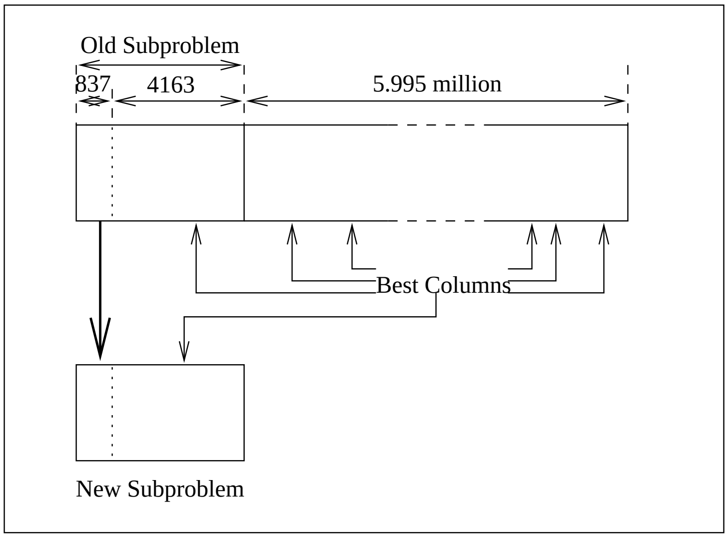

Figure 58: The SPRINT approach to solving the 6 million column problem. The number of rows is 837, which also gives the number of columns in the basis. The basis is always kept for the next round. The remaining $5000 - 837 = 4163$ columns from the subproblem are combined with all other columns to select the best ones for the next round.

2.5.3 Using the LP for the IP

In practice, the variable values for the optimal LP solution are spread between 0 and 1. This says that the optimal linear solution is to use pieces of many pairings to cover all the segments—it's like saying we want part of a crew person. Obviously, we need to either use all or none of a pairing.

The intuition for generating the IP solution is to look for follow on segments. A follow on for a segment is simply the segment that follows it in the final set of pairings. The heuristic we will use for pruning is to assume that a pair of adjacent segments that appear in many good pairings should be a follow on in the final solution.

So we run the following algorithm to select exactly which pairings to use:

- Solve the LP approximation across 6 million columns (as just discussed).
- Select approximately 10K pairings with the best reduced costs.
- Repeat until all segments have a follow on (or end at base):
  1. For all columns (pairings) with nonzero values in the linear program solution, consider all adjacent segments (s_i, s_j) in the itineraries (this includes wraparound so the last segment in the itinerary is followed by the first). Note that if there are n rows (segments) then at most n columns will have nonzero values in the linear program solution (i.e. only the basis can have nonzero values).
  2. For each follow-on (s_i, s_j) being considered weight it with the sum of the values of the linear program solution for the column (pairing) it appears. For example if (2,3) are adjacent in pairing 23 which has value .6 and pairing 95 which has value .2, then its total weight will be .8. Select the follow-on with the largest weight. The intuition is that this follow-on occurs in many good segments, so we will assume it belongs in the final solution. This is similar to choosing a variable to fix in the implicit enumeration algorithm.
  3. Fix (s'_i, s'_j) and throw out all the pairings that include $(s'_i, s_k), k \neq j$. This is the pruning step.
  4. Resolve the LP (across the 10K pairings minus those that have been deleted).
  5. If the solution becomes infeasible, add columns from the original 6 million that satisfy all the follow-ons we have fixed.

As an example, recall our previous set of possible pairings, and the associated matrix

$$A = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

An LP solution is $x_1 = x_2 = x_3 = x_4 = x_5 = 1/2$.
The pairings, expressed in term of the order of their segments s_i were:

$$x_1 = (s_1, s_7, s_4)$$
$$x_2 = (s_2, s_3, s_5)$$
$$x_3 = (s_3, s_6, s_1, s_2)$$
$$x_4 = (s_8, s_6)$$
$$x_5(s_8, s_5, s_7, s_4)$$

The follow-ons (including wraparound) and their weights are:

Follow-on Pair	Summed Weight
$(1,2)$	1/2
$(1,7)$	1/2
$(2,3)$	1
$(3,5)$	1/2
$(3,6)$	1/2
$(4,1)$	1/2
$(4,8)$	1/2
$(5,2)$	1/2
$(5,7)$	1/2
$(6,1)$	1/2
$(6,8)$	1/2
$(7,4)$	1
$(8,5)$	1/2
$(8,6)$	1/2

The follow-ons $(2,3)$ and $(7,4)$ are the only ones that appear twice. We fix these two follow-ons (whenever multiple follow-ons have weight 1, we fix them all). Fixing these follow-ons does not eliminate any pairings. For the next step we will have to fix the other follow ons in turn, and see which is best, or add more pairings.

2.5.4 Additional Constraints

We must also somehow account for the number of crew available at each base. To do so, we add constraints on the maximum hours available from each base. This is easy to do in the formulation of the linear-program by adding a row for each base and then for each pairing (column) we place the number of hours it requires in the row of its home (where it starts and ends). We then use this as an inequality constraint so the sum of each base-row is less than the total number of hours available at that base.

We also need to patch between months. To do this we separately schedule the first 2 days of each month with additional constraints put in (referring to the previous months ending). Finally, we must have a scheme to manage cancelled or delayed flights. Currently this is done by hand. Every base has a small set of reserve crew which can be put on duty if required.

2.5.5 Conclusions

- Special purpose techniques were used.
- The optimization process was largely separated from the cost and constraints rules.
- A 6 million variable LP was solved as a sub-step, plus a large number of smaller LPs.
- It is currently difficult to find out exactly how much money is saved (older papers were far more forthcoming).

- Current and future work:
  - Putting pairings together into bidlines and optimizing jointly. This is really the same as making the pairings a month long, with lay-over holidays as part of the "pairings" and additional constraints.
  - Handling real-time constraints (cancellations, rerouting).
  - Joint flight and crew scheduling. This is a much bigger problem.