The Marriage of Dynamic Programming and Integer Programming

John F. Raffensperger, Ph.D.
Department of Management, Private Bag 4800
University of Canterbury, Christchurch, New Zealand
j.raffensperger@mang.canterbury.ac.nz

Abstract

Dynamic programming (DP) has long been used to solve optimisation problems. Though sometimes suffering from the curse of dimensionality, DP has a dowry of integrality. Integer programming (IP) often suffers interminable branch and bound, but brings the gift of flexibility. DP tends to be hard to program, but IP allows us to easily write models that are hard to solve. Until recently, DP and IP have only courted in specialised algorithms, usually column generation, where the IP (or a linear program) was a master and the DP was a subproblem.

Recently, R. Kipp Martin showed how DP and IP can be married, using his technique of variable redefinition. A dynamic program can often be written as a network linear program (LP). When viewed this way, we can use the underlying DP network structure to reformulate difficult IPs into new models that have better bounds and solve more quickly. The marriage of DP and IP allows the strengths of both types of algorithms to be combined. There are applications wherever DPs are used, allowing the DP modeler to solve more interesting problems than with DP alone. Though I'll leave out many technical details, this paper is a practical tutorial in Martin's variable redefinition. I'll discuss lot sizing, cutting stock, scheduling, and vehicle routing.

1 Introduction

Suppose we want to solve *IP*: min cx, $Ax \geq b$, $x \geq 0$, x integral. A common first step is to solve *LP*: min cx, $Ax \geq b$, $x \geq 0$. Now it is well known that the optimal values have the following relationship: $v(LP) \leq v(IP)$. The tightness of *IP* is defined as $|v(LP) - v(IP)|/v(LP)$. This is also known as the *duality gap*. If this is small, we probably have a good formulation. If this is large, branch and bound is likely to take a long time.

To handle a duality gap, we can try to solve our way through it, to prove optimality. This can be bad with a large gap, where branch and bound is likely to take a long time. So we can quit without proving optimality, say, at 1%. Unfortunately, managers don't like this. If the current value is, say, $20 million with a gap of 1%, a manager may reasonably insist that the analyst continue the solution process further.

Another option is to find a new formulation of *IP* so *LP* has a better bound. For example, consider two well known formulations of the simple facility location problem:

$L1$: min $\sum_i f_i y_i + \sum_i \sum_j c_{ij} x_{ij}$,	$L2$: min $\sum_i f_i y_i + \sum_i \sum_j c_{ij} x_{ij}$,
$\sum_j x_{ij} = 1$, for all i,	$\sum_j x_{ij} = 1$, for all i,
$\sum_i x_{ij} = n \cdot y_j$ for all i,	$x_{ij} \leq y_i$ for all i, j, for all i,
$x_{ij}, y_j \in \{0,1\}$ for all i, j.	$x_{ij}, y_j \in \{0,1\}$ for all i, j.

Both formulations have $I^2 + I$ variables. However, $L1$ has $2 \cdot I$ rows, where as $L2$ has $I^2 + I$ rows. For 50 locations, $L1$ has 2,550 variables and 100 rows. $L2$ has 2,550 variables, but 2,550 rows. Is this smart? Yes! When x_{ij} is set to 1, $L1$ sets y_j to a fraction, on the order of $1/n$, charges a fractional fixed cost, and thus tends to have a large duality gap. But when $L2$ sets y_j to 1, it charges all the fixed cost, and has a very tight bound in the LP relaxation. In fact, $L2$ is often naturally integer. So we see from this old example that, in general, more constraints help an IP.

Some solvers can improve constraint coefficients and variable bounds automatically (e.g. LINDO's TITAN command, [9]). These automatic improvements are often sufficient to make the difference between a reasonable and unreasonable solution time.

However, sometimes it pays to reformulate the *entire* model. Martin [6] and Martin, Rardin and Campbell [7] showed how to reformulate IPs based on a polyhedral characterisation of a DP hypergraph. The new formulations are *much* tighter than the old. Their theory imposes the most mild restriction on the underlying hypergraph: the decision hypergraph must be directed and acyclic. Interestingly, the DP characterisation is therefore totally dual integral (not totally unimodular), but this is sufficient to dramatically improve the tightness of the formulation.

We show next four examples of variable redefinition. The first, the capacitated lot sizing problem, is due to Eppen and Martin [2]. The second, the cutting stock problem, is due to Dyckhoff [1]. The third example, the tank scheduling problem, is due to Raffensperger [8]. The last, which the reader may find amusing, is original here. It is a formulation for the travelling salesman problem that is solvable in strongly polynomial time in the number of variables. We leave the bad news for the end.

2 The lot sizing problem: old and new

The well-known capacitated lot sizing problem is usually formulated as follows:

Parameters:
T, P = number of months, products.
k_t = capacity available in month t.
d_{pt} = demand for product p in month t.
s_{pt} = set-up cost of product p in month t.
c_{pt} = variable cost of product p, month t.
h_{pt} = holding cost of product p, month t.

Decision variables:
$y_{pt} = 1$ if the machine is set up for product p in month t, else 0.
x_{pt} = production of product p in month t.
I_{pt} = inventory of product p in month t.

$LS1$: minimise $\sum_{p=1}^{P}\sum_{t=1}^{T}(c_{pt}x_{pt} + h_{pt}I_{pt} + s_{pt}y_{pt})$ subject to:

$$\sum_{p=1}^{P} x_{pt} \le k_t, \qquad t = 1,\ldots, T,$$
$$I_{p,t-1} + x_{pt} - I_{pt} = d_{pt}, \qquad p = 1, \ldots, P,\ t = 1,\ldots, T,$$
$$x_{pt} \le M_{pt} y_{pt}, \qquad p = 1, \ldots, P,\ t = 1,\ldots, T,$$
$$x_{pt}, I_{pt} \ge 0,\ y_{pt} \in \{0,1\}, \qquad p = 1, \ldots, P,\ t = 1,\ldots, T.$$

Unfortunately, this formulation is big, loose, and difficult to solve. In practice, people have used Dantzig-Wolfe decomposition, often called column generation. The reason is because there is a convenient subproblem – the uncapacitated lot sizing problem, which can be easily solved. The master model at iteration L is given by:

M: minimise $\sum_{l=1}^{k}\sum_{p=1}^{P}\sum_{t=1}^{T}\theta_{lp}\,(c_{pt}x_{lpt} + h_{pt}I_{lpt} + s_{pt}y_{lpt})$,

$$\sum_{l=1}^{k}\sum_{p=1}^{P}\theta_{lp}\, x_{lpt} \le k_t, \qquad t = 1,\ldots, T, \qquad \text{(dual price } \lambda_t)$$
$$\sum_{l=1}^{k}\theta_{lp} = 1, \qquad p = 1, \ldots, P, \qquad \text{(dual price } \pi_p)$$
$$\theta_{lp} \ge 0, \qquad l = 1,\ldots, L,\ p = 1,\ldots,P.$$

Note x_{lpt}, I_{lpt}, and y_{lpt} are constant in M. A column θ_{lp} corresponds to a schedule for one product p at iteration l. For each product p =1,…,P, the subproblem is given by:

S_p: minimise $\sum_{t=1}^{T}(c_{pt}x_{pt} + h_{pt}I_{pt} + s_{pt}y_{pt}) + \sum_{t=1}^{T}\lambda_t x_{pt}, + \pi_p$

$$I_{p,t-1} + x_{pt} - I_{pt} = d_{pt}, \qquad t = 1,\ldots, T,$$
$$x_{pt} \le M_{pt} y_{pt}, \qquad t = 1,\ldots, T,$$

$x_{pt}, I_{pt} \geq 0, y_{pt} \in \{0,1\}, \qquad t = 1,\ldots, T.$

This subproblem can be solved with IP or with the Wagner-Whitin DP algorithm. (Why not solve the LP relaxation? Because we end up with a worse bound in the resulting master program, and fractional solutions too.) This is the typical way DP is used with IP – as part of a column generation algorithm.

Eppen and Martin [2] showed how to reformulate *LS*1 into a much tighter model. They did this using the LP dual of the Wagner-Whitin DP. Let's see how.

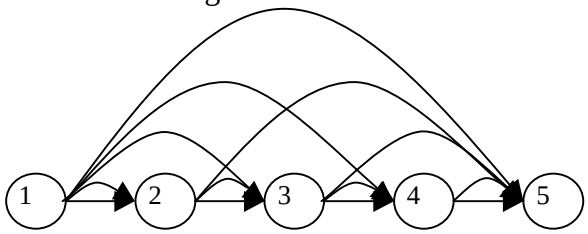

Figure 1 Network diagram for lot sizing dynamic program.

The Wagner-Whitin DP can be written as a shortest path network, as in Figure 1. If we observe that this shortest path network can be rewritten as an LP, with the arcs as variables and the nodes as constraints, we can produce a new formulation for the lot sizing problem. We have a cost parameter: $c_{ijp} = \sum_{t=i}^{j-1} h_t(\sum_{u=t+1}^{j} d_u) + c_i(\sum_{u=i}^{j} d_u)$. The decision variables and model are:

z_{ijp} = fraction of demand produced in period i for demand in periods i to j.
$v_{ip} = 1$ if we produce product p in period i, else 0.

*LS*2: minimise $\sum_{p=1}^{P}\sum_{t=1}^{T}(\sum_{t=1}^{T} c_{ijp}z_{ijp} + s_{tp}v_{tp})$
Capacity: $\sum_{p=1}^{P}\sum_{t=i}^{T}(\sum_{l=i}^{t} d_l)z_{itp} \leq k_i$ for $i=1,\ldots,T$.
Produce each product: $\sum_{j=1}^{T} z_{1,j,p} = 1$ for $p = 1,\ldots, P$.
If production ends in period i, it must begin in period i+1:
$\sum_{t=1}^{i-1} z_{t,i-1,p} - \sum_{j=1}^{T} z_{ijp} = 0$ for $i =1,\ldots, T$, $p=1,\ldots, P$.
Set up forcing: $\sum_{j=i}^{T} z_{i,j,p} \leq v_{t,p}$ for $i=1,\ldots,T$, $p=1,\ldots, P$.
Non-negativity and integrality: $z_{i,j,p} \geq 0$, $v_{t,p} \in \{0,1\}$.

This formulation is tighter than the original and will solve faster. The LP relaxation of *LS*2 has an optimal objective value equal to the that of *M*, when we solve the subproblem *S* with DP or as an integer program. Variable redefinition produced a tight model that could be solved directly with IP. No column generation was needed. Let's see how this works with some more examples.

3 The cutting stock problem

3.1 Old and new

The venerable cutting stock problem made its debut with Gilmore and Gomory [3], who solved it with column generation. The subproblem is a one-constraint integer knapsack problem, which can be solved conveniently with a DP. The master that Gilmore and Gomory used is as follows. Variable x_j is the number to make of one pattern, as shown in Figure 2. Parameter a_{ij} is the number of pieces of length i in pattern j.

Master *CS*1: minimise $\sum_j x_j$,	Subproblem *S*: minimise $\sum_i \pi_i a_i$
$\sum_j a_{ij}x_j \geq d_i$ for each demand i,	$\sum_i l_i a_i \leq L$,
x_j integer.	a_i integer.

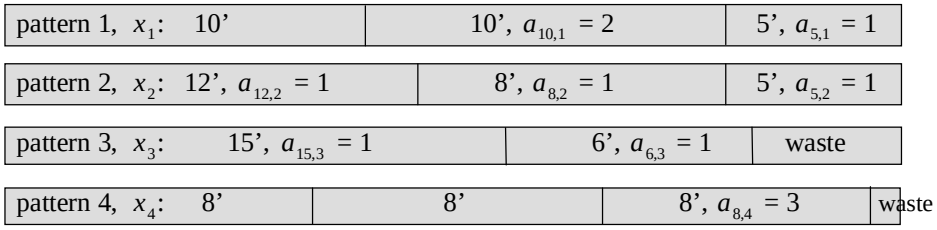

Figure 2 A variable corresponds to a pattern.

It would be nice to have a formulation that avoided column generation. To do so, we need to put all the information about patterns in the model. How can we get all patterns in the model? The trick is to think about the DP. As we did with the Wagner-Whitin DP, let's examine this DP's network. There is an arc (t,u) for each constraint coefficient $l_i = t - u$. The cost of arc (t,u) is the objective coefficient of variable a_i. As with the Wagner-Whitin DP, this is just a shortest path problem, and we could write a network LP for it. As shown in Figure 3, a variable corresponds to a cut.

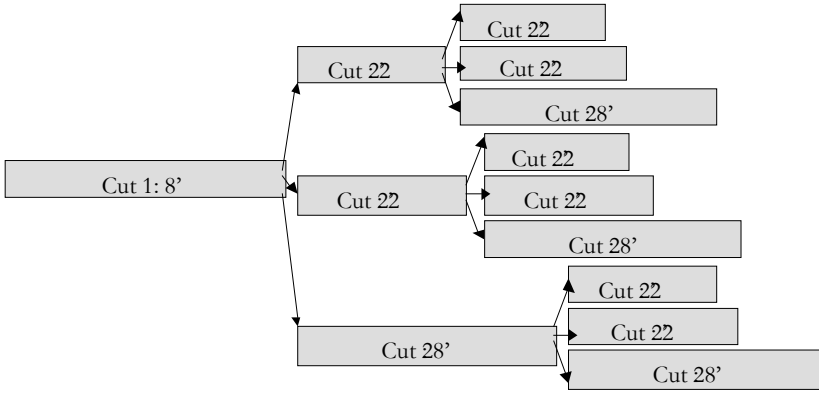

Figure 3 A variable in *CS*2 corresponds to a cut.

Note that many redundant arcs can be removed in a wise implementation of the DP. For example, note that Figure 4 contains the two paths, {0,5},{5,25}, and {0,20},{20,25}. However, only one of these is needed. So a smart DP will eliminate needless states and arcs. This cleverness extends to our reformulation of the integer program. Just as we eliminate needless arcs in the DP, we can eliminate needless arcs in the network formulation.

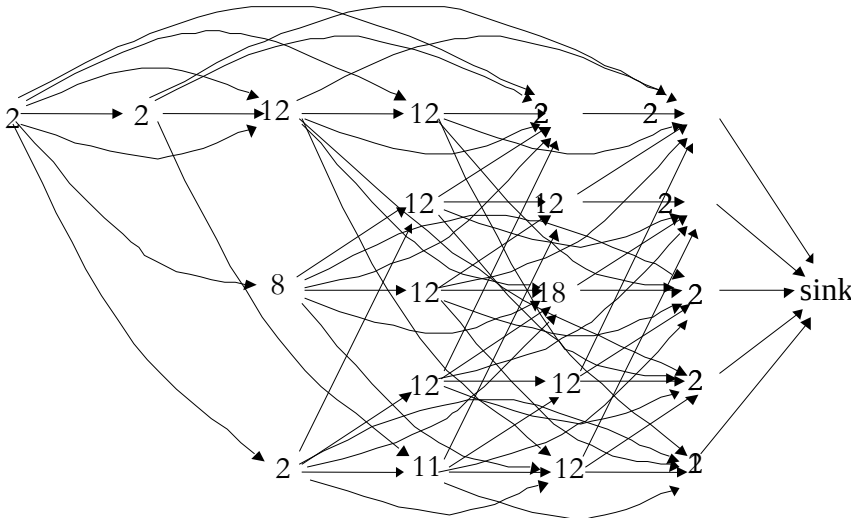

Figure 4 The complete network for a knapsack DP

What is the network formulation for the DP? It is a binary shortest path problem. But this binary formulation suggests a reformulation of the original problem. We need a general integer formulation, and now we can easily write it.

Parameters are L = the length of the uncut bar, d_i = the demand for length i and l_i = the lengths demanded, $i = 1, \ldots, K$. Variables are $y_{i,j}$ = the number of pieces of length $j - i$, cut at i inches from the end, and $w_{i,L}$ = the number of pieces waste, length $L–i$, cut at i inches from the end.

$CS2$: minimise $\sum_j w_{j,L}$
$\sum_h y_{h,i} - \sum_j y_{i,j} = 0$, for all $i \le L - l_K$,
$\sum_h y_{h,i} - \sum_j y_{i,j} - w_{i,L} = 0$, all $L - l_K < i \le L$,
$\sum_k y_{i,j} \ge d_k$, for each product k,
$y_{i,j}$ integer.

$CS2$ is due to Dyckhoff [1], but the theory came later from Martin [6].

When $CS1$ and $CS2$ are solved as LPs, and if the knapsack subproblem S is solved as LP, we have $v(CS1) \le v(CS2)$. When $CS1$ and $CS2$ are solved as LPs, and if the knapsack subproblem is solved as an integer program or with a DP, then $v(CS1) = v(CS2)$. However, after we solve $CS1$ to optimality as an LP (which we must for column generation to work), even if the subproblem S is an integer program, we have $v(CS1) \ge v(CS2)$. The reason: $CS1$ is missing columns, but $CS2$ contains all possible patterns.

We get a better objective value, but also the solution time is *much* better with $CS2$. Why? With $CS1$, we must solve the subproblem many times, possibly hundreds of times, just to solve the LP relaxation. With care, we can avoid doing too much work in the master at each iteration. But with $CS2$, we solve the DP exactly once, to generate the LP formulation. Then we solve the LP once, and we have a very good solution. If we wish, we can go into branch and bound immediately. In short, if the underlying network structure is not too big, $CS2$ is simply the better formulation. However, there is another reason to use $CS2$. We can use it to solve *new* problems that would be hard with $CS1$.

3.2 The Best Stocklength Problem

Our cutting stock firm, named Manuka Metals to motivate it, currently buys only 25’ lengths, which are then cut into the demanded lengths. Suppose Manuka Metals could ask the supplier for S special lengths, for extra cost. That is, we can pay extra to buy 17’ lengths, or 5.5’ lengths, or whatever we wish, which we would then cut into the lengths demanded. Manuka Metals doesn’t want to buy many special stocklengths, probably just one or two, to keep inventory down. Which special lengths should they order?

Assume L (= 25’ here) is the biggest possible length. Assuming only integral demand lengths, we could solve this by doing Gilmore-Gomory column generation L times. Let’s not. Instead, we can modify $CS2$ to solve this. Recall from the above that $w_{i,L}$ = pieces waste of length $L - i$, starting i inches from the end. We can now have any stock length. We must add more of these variables: $w_{i,j}$ = pieces waste of length $j - i$, starting i inches from the end of a length that is j inches long, where $i \le j < i + l_k$, so waste ≤ smallest product.

Arc $w_{i,j}$ may end at any possible stock length, not just the maximum. So arc $w_{i,i}$ is the number of patterns cut to length of i, with no waste. We add one of these variables for each node in the DP network. We also need a binary variable to decide whether we choose a particular special stocklength: z_j=1 if we choose stock length L_j , else 0.

BSL: minimise $\sum_{k=1}^{K} \sum_t \sum_{j:\, t < j < t + lk} (j - t) w_{t,j}$
$\sum_h y_{h,i} - \sum_j y_{i,j} - \sum_{j:\, t < j < t + lk} = 0$, all $L - l_K < i \le L$,
$\sum_k y_{i,j} \ge d_k$, for each product k,= 1,…,K.
$\sum_t w_{t,j} \le M z_j$, for all j.
$\sum_j z_j \le S$,

$y_{i,j}$ integer, z_j binary.

Once solved with branch and bound, this formulation will give the true optimum, which would be very difficult with column generation. Another model we can now solve easily: the dynamic cutting stock problem. Like the lot sizing problem, we may have demand over several periods, and so we will want to hold inventory. With the new formulation, it is easy to add a subscript for time, and add inventory variables. So we see that variable redefinition allows us tighter formulations than before, and it also allows us to solve models we could not before.

3.2 Problems with DP-based formulations

The lot size and cutting stock models worked well because we have good polynomial or pseudo-polynomial time DP algorithms. When variable redefinition was applied, the concise DP resulted in a reformulation with a concise number of variables. Martin studied only polynomial time DPs, because they will produce a polynomial number of variables in the IP, but his theory applies to any acyclic decision hypergraph.

For many important problems, the DP suffers from the "curse of dimensionality," though this curse is not as oppressive as it was with the old computers. The corresponding LP network would also be big. For these problems, reformulating based on the DP can result in a huge number of variables. So really hard problems stay hard.

Sometimes there is a way out: we can cleverly reduce the state space of the DP. With the knapsack for the cutting stock problem, we saw that we can erase redundant arcs. In the next example, the tank scheduling problem, we shall see that this reduction can make or break the model, even though the DP is indeed dimensionally cursed.

4 The tank scheduling problem

A military tank battalion commander selects and schedules training exercises for his four tank companies, to train the companies in a set of skills. Their exercises can be repeated. The skills they learn have *precedents* – soldiers learn to drive a tank before shooting, for example. There are capacity constraints which depend on the time period – only so many tanks can get on the playing field at once before they bump into each other. The problem is to find a schedule of exercises for each of the 4 tank companies, to train each company fully in its required skills, within capacity, as quickly as possible.

This problem is similar to job shop scheduling, only not so tidy. Avoiding formalities (we will point out the crucial parts), here is a simplified formulation:

$$TS1\text{: minimise } \sum_{t=1}^{T} tz_t \tag{1}$$

$$z_t \geq z_{rt},\ r = 1, \ldots, R,\ t = 1, \ldots, T, \tag{2}$$

$$z_{rt} - \sum_{k=1}^{K} \sum_{u=t-d_k+1}^{t} x_{rku} = 0,\ r = 1, \ldots, R,\ t = 1, \ldots, T, \tag{3}$$

$$\sum_{t=1}^{T} \sum_{k=1}^{K} q_{rki} x_{rkt} \geq p_{ri},\ r = 1, \ldots, R,\ i = 1, \ldots, I, \tag{4}$$

$$\sum_{r=1}^{R} \sum_{k=1}^{K} \sum_{u=t-d_k+1}^{t} c_{kw} x_{rku} \leq C_{wt},\ w = 1, \ldots, W,\ t = 1, \ldots, T, \tag{5}$$

$$\sum_{u=1}^{t-1} \sum_{\{k|q_{rkj}=1\}} q_{rki} x_{r,k,u-d_k+1} \geq p_{ri} x_{rkt}, \text{ for } \{i,j\text{: } i \rightarrow j\}, \text{ for } \{l\text{: } q_{rlj}>0\},\ r=1,\ldots,R,\ t=1,\ldots,T, \tag{6}$$

$z_t, z_{rt}, x_{rkt} \in \{0,1\}$.

Depending on the time horizon T and the number of entities R, there are 20,000 to 500,000 binary variables. So *TS*1 is a monster.

Constraint set (4) is a multidimensional knapsack. Set (5) has the capacity constraints. Set (6), the precedence constraints, is the bad part of this formulation. There may be *millions* of precedence constraints, since the number of these is $O(I^2KRT)$. This "standard formulation" is intractable. However, if we drop the capacity constraints, set (5), the problem decomposes by entity r. For one entity (say, one company of the battalion's four tank companies), we can use DP to find a schedule. The states of the DP are the skills and the time period.

Now this DP is truly cursed. It is a general integer multidimensional knapsack, with typically 3,000 variables and 25 to 30 constraints, which in general takes an exponential amount of time to solve. However, the precedents – so bad before – greatly *reduce* the state space, because they give a partial ordering to our multidimensional knapsack.

Figure 5 is a sample DP network. (Time period states are left off for simplicity.) Note that exercise activities A, B, and D are available immediately, but exercise C is not available until skill 1 has reached 3. So there is a precedence from skill 1 to skill 3, and this reduces the number of arcs and nodes in the DP. Besides precedence constraints, the DP can easily handle other useful constraints such as time windows.

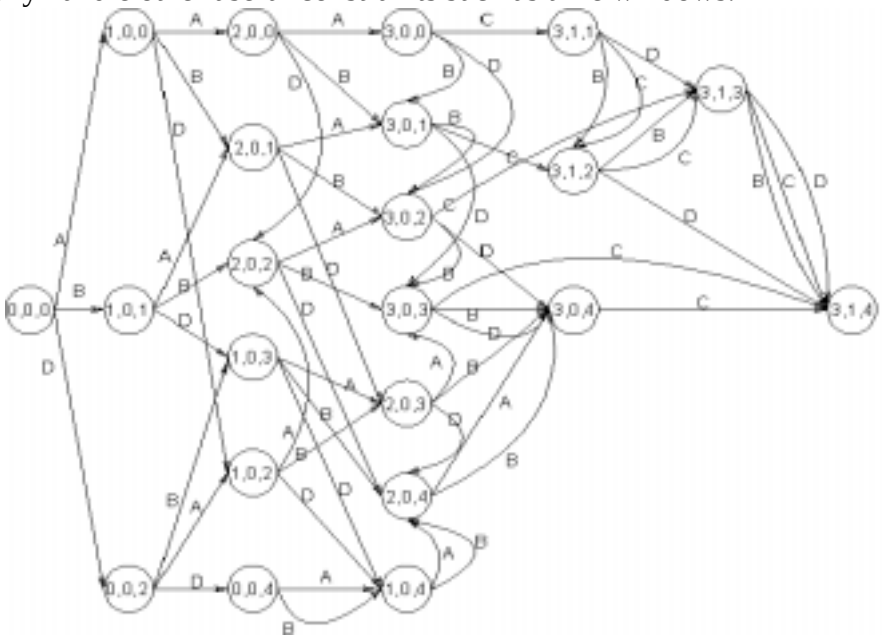

Figure 5 DP network for one entity of the tank scheduling problem

Now we have a network. Let's reformulate the monster *TS*1 in terms of the DP. Let $\theta_{k,\pi,\rho,t}$ = the number of entities with skill vector $\boldsymbol{\pi}$ in period t, who do exercise activity k, resulting in skill vector $\boldsymbol{\rho}$, in period $t+d_k$.

*TS*2: minimise $\sum_{k=1}^{K}\sum_{\boldsymbol{\pi}}\sum_{\boldsymbol{\rho}}\sum_{t=1}^{T} D_{k,t}\theta_{k,\boldsymbol{\pi},\boldsymbol{\rho},t}$, (7)

$$\sum_{k=1}^{K}\sum_{\boldsymbol{\pi}}\theta_{k,s_r,\boldsymbol{\rho},0} = S(s_r) \text{ for all } s_r, \quad (8)$$

$$\sum_{k=1}^{K}\sum_{\boldsymbol{\pi}}\theta_{k,\boldsymbol{\pi},\boldsymbol{\rho},t-d_k+1} = \sum_{k=1}^{K}\sum_{\boldsymbol{\rho}}\theta_{k,\boldsymbol{\rho},\boldsymbol{\rho},t} \text{ for all } \boldsymbol{\rho}, t, \quad (9)$$

$$\sum_{k=1}^{K}\sum_{\boldsymbol{\rho}}\sum_{t=1}^{T-d_k+1}\theta_{k,\boldsymbol{\rho},p_r,t} = R(p_r) \text{ for all } p_r, \quad (10)$$

$$\sum_{k=1}^{K}\sum_{\boldsymbol{\pi},\boldsymbol{\rho}}\sum_{u=t-d_k+1}^{t} c_{k,w}\theta_{k,\boldsymbol{\pi},\boldsymbol{\rho},u} \le C_{w,t} \text{ for all } w,t, \quad (11)$$

$$\theta_{k,\boldsymbol{\pi},\boldsymbol{\rho},t} \in Z^+ \text{ for all } k,\boldsymbol{\pi},\boldsymbol{\rho},t. \quad (12)$$

Again, we have left out many tangential details, but the point is that *TS*2 is network, except for the capacity constraints, set (11). In spirit, this is very much like the cutting stock problem – a knapsack reformulated as mostly network, with general integer variables. Restricting the DP made this succeed. *TS*2 satisfies all the precedents, because these were handled explicitly in the DP during model creation. In fact, this model was *still* too large for a single formulation, so a specialised column generation algorithm was used. Interestingly, entities (the tank companies) are similar or even identical, so we can solve all *R* subproblems in one pass of the DP. There is a demand node for each distinct group of entities. But this is going beyond our story.

The bounds on this model were often perfectly tight. In fact, when the LP was solved, the resulting solution was often naturally integer. So we see that variable redefinition tightens a formulation, and we do not need a polynomial time DP. It is important that we have a DP that is reasonable. But what if the DP is unreasonable?

5 The travelling salesman problem

A standard formulation for the Travelling Salesman Problem is as follows:

TSP1: minimise $\sum_i \sum_j c_{ij} x_{ij}$
$\sum_i x_{ij} = 1$ for all j.
$\sum_j x_{ij} = 1$ for all i.
$\sum_{i,j \in S} x_{ij} \le |S| - 1$ for every subset S.
$x_{ij} \in \{0, 1\}$ for all i, j.

There are exponentially many subsets S, so this formulation is huge. In practice, TSP1 is often solved with a few subtour constraints, resulting in infeasible solutions. Then subtour constraints are added, and it is solved again. Also, solutions may be fractional, requiring branch and bound.

But it is easy to modify the formulation to be more general. We can add a constraint for fuel capacity by adding the constraint $\sum_i \sum_j f_{ij} x_{ij} \le F$. But a multi-vehicle travelling salesman problem, even with identical trucks, requires a complete reformulation.

So how do we apply variable redefinition to the TSP? First let me say that, once we do, we will get a *naturally integer network formulation* for the TSP that can be solved in polynomial time in the number of variables. What is the bad news? Keep reading.

The TSP can be solved by DP (Held and Karp [4]), though it has the curse of dimensionality. For n cities, it requires time $O(2^n)$. Not only is this hard, but it is hard to modify this for side constraints, e.g. fuel capacity. And it is difficult to modify for the multi-vehicle TSP (Lawler, et al [5], p. 436). But carry on we will. The network diagram is in Figure 6.

Now we write this as a network LP. Note α, β, γ are sets of cities.

$x_{\alpha,i,\beta,j}$ =1 if the car has done city set α, is at city i, and goes to city j. So $\beta = \alpha \cup j$.

TSP2: Min $\sum_\alpha \sum_i \sum_j c_{ij} x_{\alpha,i,\beta,j}$
$\sum_j x_{\varnothing,1,\beta,j} = 1$,
$\sum_\alpha \sum_i x_{\alpha,i,\beta,j} - \sum_\gamma \sum_k x_{\beta,j,\gamma,k} = 0$ for each set β and city j.
$x_{\alpha,i,\beta,j} \ge 0$ for all α, i, j.

TSP2 is clearly a naturally integral network LP. Big networks can be solved quickly. Unfortunately, big comes very soon. For n cities, the number of variables is $\sum_{i=1}^{n-1} i(n-i+1) \binom{n}{i}$. For 20 cities, this is about 5 million. For 40 cities, this is about 10^{13}.

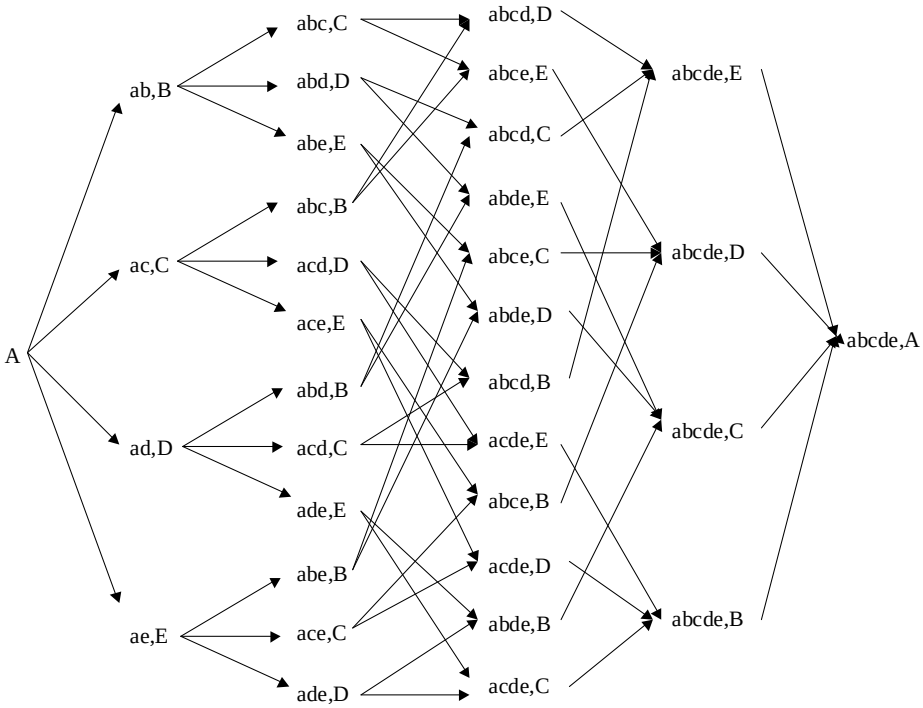

Figure 6 The DP network for a 5-city TSP.

Modifying TSP2 for more general problems is done easily. For example, fuel capacity is trivial to add to the model: $\sum_\alpha \sum_i \sum_j f_{ij} x_{\alpha,i,\beta,j} \le F$. More interestingly, we can easily modify the formulation for the multi-vehicle TSP: $x_{\alpha,i,\beta,j}$ = number of cars that have done city set α, are at city i, then go to city j, and $y_{\alpha,i}$ = number of cars that have done city set α, are at city i, then go to city 1.

MVTSP: Min $\sum_\alpha \sum_i \sum_j c_{ij} x_{\alpha,i,\beta,j} + \sum_\alpha \sum_i c_{i1} y_{\alpha,i}$

$\sum_j x_{\varnothing,1,\beta,j} = k$, the supply of vehicles.

$\sum_\alpha \sum_i x_{\alpha,i,\beta,j} - \sum_\gamma \sum_k x_{\beta,j,\gamma,k} - y_{\beta,j} = 0$ for each set β and city j.

$\sum_\alpha \sum_i x_{\alpha,i,\beta,j} = 1$ for each city j, the demand for vehicles.

$x_{\alpha,i,\beta,j}, y_{\alpha,i} \in Z^+$ for all α, i, j.

This easy to solve, if you can create it! Like the knapsack and the tank scheduling problems, we can reduce the state space, but we leave that for another paper.

5 Implications for dynamic programmers and integer programs

5.1 Implications for dynamic programmers

Your DP is probably not as big as that for the TSP, which is rather like a worst case. So you can probably add a bit of code to put your DP into CPlex or LINDO. Think of the DP as a matrix generator for a larger model. Here is pseudo-code that shows how. The parts in bold are the parts added to the DP in order to do the matrix generation.

```
CreateNode (∅,1);
appendConstraintToCplex (1);
For node i = 1 to n:
{         For decision x = 1 to X:
          If arcMakesSense (node i, x)
          {     Node j = CreateNode (i, x);
                appendConstraintToCplex (j);
                appendVariableToCplex (i, j);
                n = n + 1;
    }     }
```

After this, the network is stored in CPlex as an LP formulation. We are not solving a DP so much as we are finding all the nodes and arcs. It is not necessary to do primal retrieval. Now we can use DP to solve new problems by adding side constraints. The LP relaxation will be very tight, so they will solve quickly. Furthermore, as Martin, Rardin, and Campbell [7] point out, we can use LP output for sensitivity analysis.

5.2 Implications for integer programs

Consider whether your difficult formulation can be recast with DP. PhD students are cranking out by the bushel column generation algorithms with DP subproblems. Many of these would benefit from the use of variable redefinition.

The issues in reformulating are to be sure solutions are feasible to the original problem, and to be sure we have not cut off the optimal solution. Martin [6] showed how to do this with a linear transformation: the user must find a linear transformation from the old model to the new good model. Unfortunately, this makes reformulating as hard as writing a theorem, but there is a proof the formulation is correct.

Generally, users formulate models intuitively, and in my experience, this is satisfactory for the use of variable redefinition. Just make sure you've got it formulated properly! Keep the old formulation around awhile for validation purposes.

Martin, Rardin and Campbell [7] studied LP reformulations based on polynomial time DPs. However, if we can improve the DP for an exponential DP, we may be able to use variable redefinition more generally. We have seen several examples of variable redefinition. This article contributed an extension to Dyckhoff's cutting stock problem as well as an amusing formulation for the travelling salesman problem. Hopefully, the reader has a better understanding of the purpose and value of variable redefinition.

References

[1] Dyckhoff, Harald, "A New Linear Programming Approach to the Cutting Stock Problem," *Operations Research*, v.29, no. 6, Nov-Dec. 1981.

[2] Eppen, Gary, and R.Kipp Martin, "Solving Multi-Item Capacitated Lot-Sizing Problems Using Variable Redefinition," *Operations Research*, v35, n6, Nov-Dec 1987.

[3] Gilmore, P.C. and R.E. Gomory, "A Linear Programming Approach to the Cutting Stock Problem," *Operations Research*, v9, 1961, pp. 849-859.

[4] Held, M., and R.M. Karp, "A dynamic programming approach to sequencing problems," *SIAM J. Appl. Math*, v10, pp. 196-210.

[5] Lawler, E.L., J.K. Lenstra, A.H.G. Rinnooy Kan, D.B Shmoys, *The Traveling Salesman Problem*, John Wiley & Sons, Inc., Chichester, Great Britain, 1985.

[6] Martin, R. Kipp "Generating Alternative Mixed-Integer Programming Models Using Variable Redefinition," *Operations Research*, v35, n6, Nov-Dec 1987.

[7] Martin, R.Kipp, Ronald L. Rardin, Brian A. Campbell, "Polyhedral Characterization of Discrete Dynamic Programming," *Operations Res.*, v38, n1, Jan-Feb 1990.

[8] Raffensperger, John F., *Measuring and Improving the Readiness of Emergency Organizations*, PhD dissertation, University of Chicago, 1997.

[9] Schrage, Linus, *LINDO User's Manual*, LINDO Systems, Inc., Chicago, IL.