Module 13: Go With The Flow: Network Models

Reading Material: 13.1 – Introduction

In this module, supply chain modeling and integer programming are merged, in a sense, to cover some practical and useful core models related to networks. Maybe it's like the commercial in which two people, one with peanut butter and the other with chocolate, run into each other to create chocolate peanut butter cups. Anyway, building on the alternative format we looked at back in Module 11 (termed From-To format), we study how we can create linear programming (LP) models that find shortest routes and maximum flow through capacitated networks.

Each day we use explicit (or implicit) shortest route models – for instance, sending an e-mail uses Dijkstra's shortest route algorithm in routing data packets (note that it is not named after Lenny Dykstra, former Philadelphia Phillies and New York Mets baseball player). If you use MapQuest or a GPS system in your car, you also are likely are using a shortest route algorithm. This module shows how to model such situations using LP.

A second core network model involves modeling flows in a capacitated network. For instance, one might be interested in analyzing car traffic, sending oil products through pipelines, or sending packets of data through the backbone fiber network – any scenario in which one is interested in maximizing flow through a network that has link capacity constraints.

There are other generic network formulations one might cover here, but the two selected for presentation are the two most useful, practical applications. The concepts learned are applicable in more complex situations as well, which means these core models can help one develop sophisticated vehicle routing problems, water system management tools, and a host of other potential, practical "killer" applications.

An interesting side note: the two core models we focus on – shortest route and maximal flow – can be solved by use of specialized "greedy" (i.e., by hand) algorithms that work amazingly well and are easily programmable. Many textbooks still teach them. We will stick with the tried and true approach of mastering an Excel Solver model for these problems so that we are capable of adding other realistic components to the basic model to develop the killer applications we WILL derive from our core knowledge to later save the world.

Now, to the inevitable in-class examples.

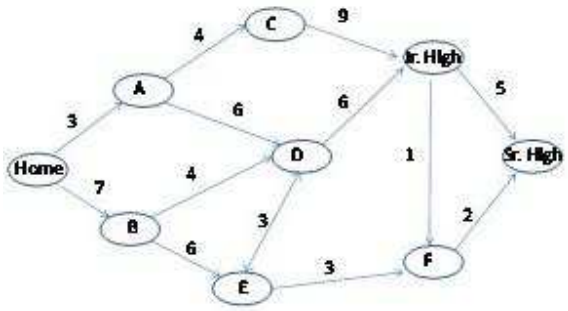
Reading Material: 13.2 – Example 1: The Case of the Clutch Fearing Commuter – Alternative title: Trevor's Tumultuous Tour)

13.2.1 Problem Description

Note: All characters appearing in this example are fictitious. Any resemblance to real persons, living or dead, is purely coincidental.

Trevor is a newly minted, 16-year-old driver whose father has given him the classic "let no good deed go unpunished" opportunity. His dad has acquired a used car for his use in driving to school, baseball practice, running errands, cruising with the girls on the Cross Country team . . . but the car is a manual transmission. (Note: This car is NOT Trevor's, just an extra car that can be used as behavior warrants. It is also a spare in case dad's daily driver, previously purchased for his older brother and then totaled in its first week of operation, goes "belly up.")

Trevor realizes that he cannot complain too much lest his father pull the plug on his immense generosity. Being born into an analytical family, Trevor charts possible ways to drive to the high school and measures each leg of his journey on the degree to which stops may have to be made that could require him to be able to use the clutch (which he wishes to minimize). The roadmap below shows the links in Trevor's network – each link is uni-directional EXCEPT for the link between Node D and Node E. The values represent the potential for "uneven" stops that would require the manual neophyte to perform under immense pressure. Trevor wishes to find the route with the least pressure (i.e., the shortest route) to the high school from home.



13.2.2 Model Preliminaries

Shortest route problems appear everywhere in life – unless we live with Tom Hanks on a deserted island, we cannot escape them. To model this scenario, the From-To format discussed previously in the Supply Chain module will be employed. It was shown there as an alternative way of formulating transportation and transshipment models. Of course, those scenarios are simply special cases of generalized network models.

To review, the decisions in the model will be link-based. Every link in the model corresponds to a decision variable. In the shortest route problem, each link is represented by an (implicit) 0/1 decision variable – the value of the variable is 1 if the link is on the shortest path, 0 if otherwise.

As before, there is a constraint corresponding to each node, implementing the mantra IN=OUT. Theses constraints implement "If the shortest route comes in to the node, it must also leave the node" and, of course, the reverse – "If the shortest route does NOT come in to the node, then it cannot leave the node).

The algebra of this formulation was discussed in Module 11 and will not be reviewed here.

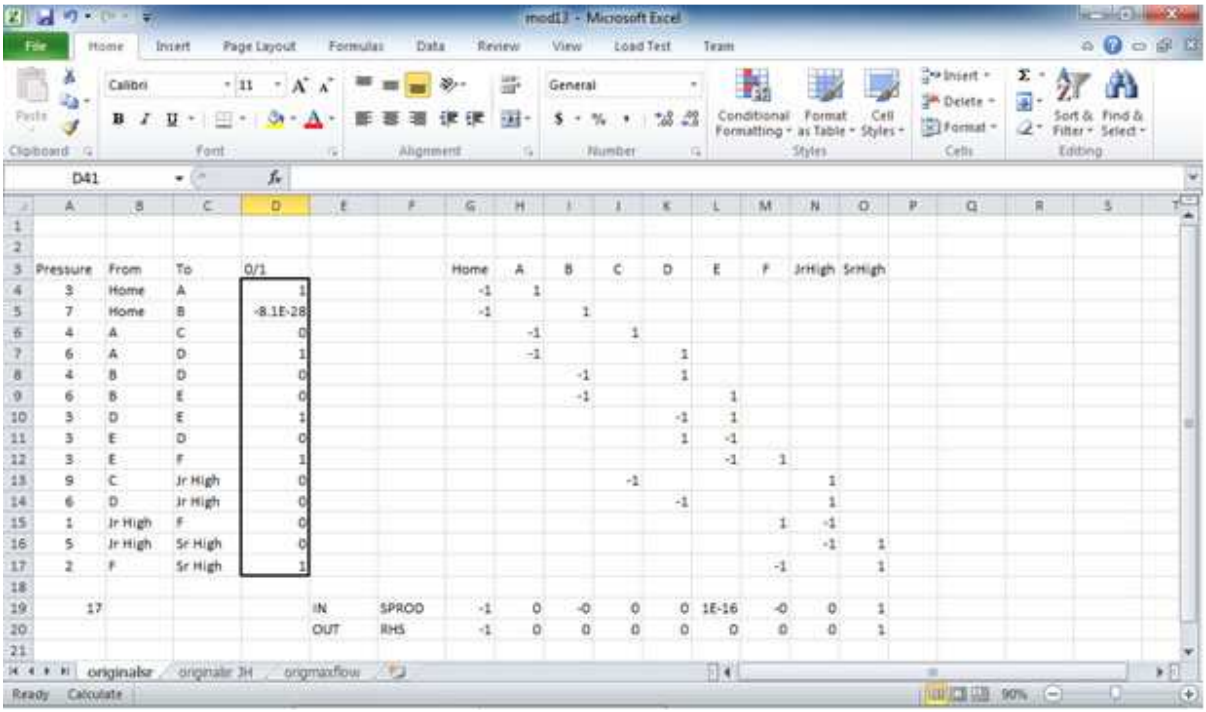
Restating – in Trevor's scenario, there are 14 links (decisions variables) of interest – note that the connection between D and E allows travel in either direction. Bi-directional capabilities are modeled using two unique decision variables. There are nine nodes, including the start node (HOME) and the destination node (Sr. HIGH). Thus, there will be nine constraints. The objective will be to minimize the pressure metric assigned to each link in the road network.

As previously covered in Module 11, the nine constraints have different right-hand side (RHS) values depending on whether they are start nodes, intermediate nodes, or the destination node. The start node constraint must ensure that the shortest path begins there – it will have a RHS of –1 given the way constraints are coded. The destination node will have a RHS of +1, implying that the shortest route comes in to the last node.

All intermediate nodes will have a RHS of 0 – because the sum of the values of the decision variables representing the links coming into the node will need to equal the sum of the value of the decision variables leaving the node – IN=OUT. The Excel Implementation and solution of the Shortest Route model follows.

13.2.3 Excel and Solver Implementation

Figures 13.1 and 13.2 show the implementation of Trevor's shortest route problem. Figure 13.1 is the Excel spreadsheet in To-From form, and Figure 13.2 is the Solver model.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1																			
2																			
3	Pressure	From	To	0/1			Home	A	B	C	D	E	F	JrHigh	SrHigh				
4	3	Home	A	1			-1	1											
5	7	Home	B	-8.1E-28			-1		1										
6	4	A	C	0				-1		1									
7	6	A	D	1				-1			1								
8	4	B	D	0					-1		1								
9	6	B	E	0					-1			1							
10	3	D	E	1							-1	1							
11	3	E	D	0							1	-1							
12	3	E	F	1								-1	1						
13	9	C	Jr High	0						-1				1					
14	6	D	Jr High	0							-1			1					
15	1	Jr High	F	0									1	-1					
16	5	Jr High	Sr High	0										-1	1				
17	2	F	Sr High	1									-1		1				
18																			
19	17				IN	SPROD	-1	0	-0	0	0	1E-16	-0	0	1				
20					OUT	RHS	-1	0	0	0	0	0	0	0	1				
21																			

Figure 13.1

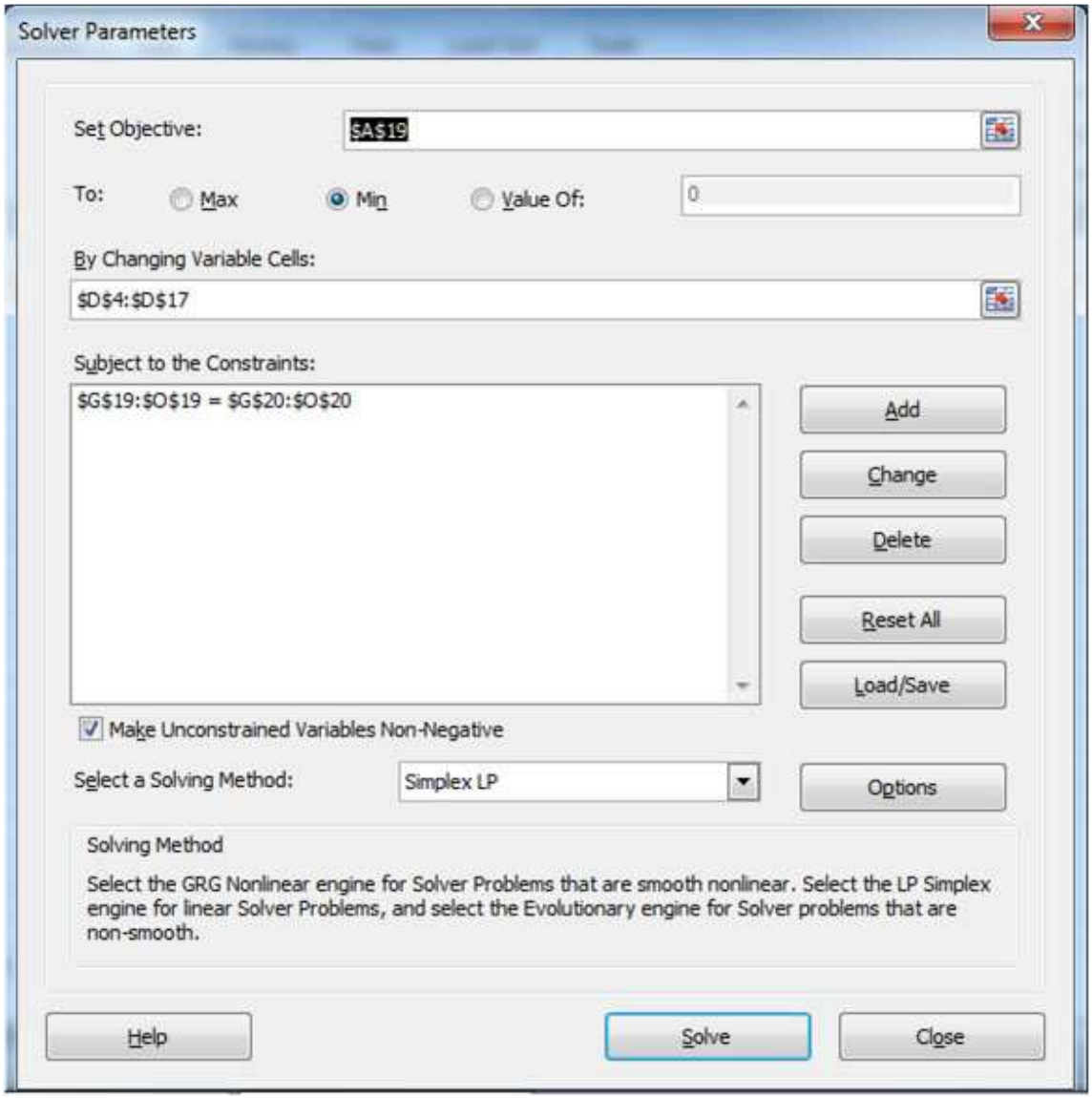


Figure 13.2

Columns B and C are descriptors for the links in the network. Column B is the FROM node of a link, and Column C is the TO node of the link. Column D is the placeholder for the decision variables – again, in the solution, a 1 indicates the link is on the shortest route; a 0 implies that it is NOT on the shortest route.

Column A is the pressure associated with each link. Obviously, in general, the criteria could be distance, time, and other measures. Cell A19 is the target cell – the sumproduct of Column A times Column D. This calculation tells us the length of the shortest path through the network. The explicit formula is A19: =SUMPRODUCT(A4:A17,$D4:$D17).

Columns G through O is the constraint block where the nine constraints are implemented – one for each node. Row 3 holds the relevant node labels. Recall the mechanism/rule for creating the left-hand side (LHS) coefficients for the constraints. In a cell in the constraint block, if the FROM node of the row (link) matches the Column (node), a –1 is entered. If the TO node of the row (Link) matches the Column (node), a +1 is entered. If there is not a match with either the TO or FROM node, nothing is entered (an implied 0). Again, this rule creates the proper algebraic constraints for IN=OUT as long as RHS values are coded properly (Row 20).

To finalize the constraints, the LHS (Row 19) and the RHS (Row 20) are created. Row 19 is a similar sumproduct to that of Cell A19. In fact, as shown earlier, one can copy A19's formula from Columns G through O. If retyping, Cell G19's formula would be: =SUMPRODUCT(G4:G17,$D4:$D17). Then, it too could be copied through Cell O19.

Row 20 is coded (based on the discussion in Module 11) as follows for the shortest route model – the start node (HOME) has a value of –1, the end/destination node (Sr. High) has a value of +1, and all the other intermediate nodes have a value of 0. This implements IN=OUT.

In the Solver dialogue box, the only family of constraints is G19:O19 = G20:O20.

Note that we did NOT code Column D as binary . . . they will automatically be 0/1 based on the algebra of the core constraints. Add them as binary if it bothers you. I tend to add integer constraints only after I am certain the easier, continuous model works okay. Just a habit, not a necessary rule. One might find it necessary to add binary constraints once side conditions are added to the model.

The model is solved, and the results indicate that the shortest route for Trevor to travel from home to the high school, as measured in pressure, is 17 units. The 1s in Column D indicate the shortest path – home to A, A to D, D to E, E to F, and then F to the high school.

That's all, folks! Now, let us add one complication to illustrate side constraint use.

13.2.4 Complications: Dropping off the Neighbor

Consider the following scenario: How does the solution change if you make Trevor take his neighbor to the junior high before going to the high school? (In other words, you force the shortest path to go to the junior high).

Consider planning your traditional driving family summer vacation – you want to find the shortest route between Tulsa and Phoenix, but along the way, you might want to stop at the Four Corners, or the World's Biggest Ball of String, or whatever. Thus, you'd need to augment a shortest route model by adding other requirements.

To do this for Trevor, we can choose to physically implement the junior high visit side constraint anywhere on the spreadsheet. I know sometimes in class just to be a smart-aleck, I use row 123

or something like that to prove my point. So the choice of WHERE to implement the constraint in Excel is arbitrary.

Algebraically, to force the path to the junior high, the shortest path must be forced to either use a “TO” link involving the junior high, or a “FROM” link involving the junior high. Doing one actually automatically does the other, because we have IN=OUT in our core model.

In Figure 13.3, the model sums the links coming “TO” the junior high and a constraint forces their sum to be equal to 1. This will force the path to travel to the junior high. Cells D13 and D14 represents the two links entering the junior high. So, we add the constraint:

D13 + D14 = 1

And then, resolve the model with all the original core constraints.

Cell B23 implements this (the formula is in fact D13+D14), as is shown in Figure 13.3 and Figure 13.4.

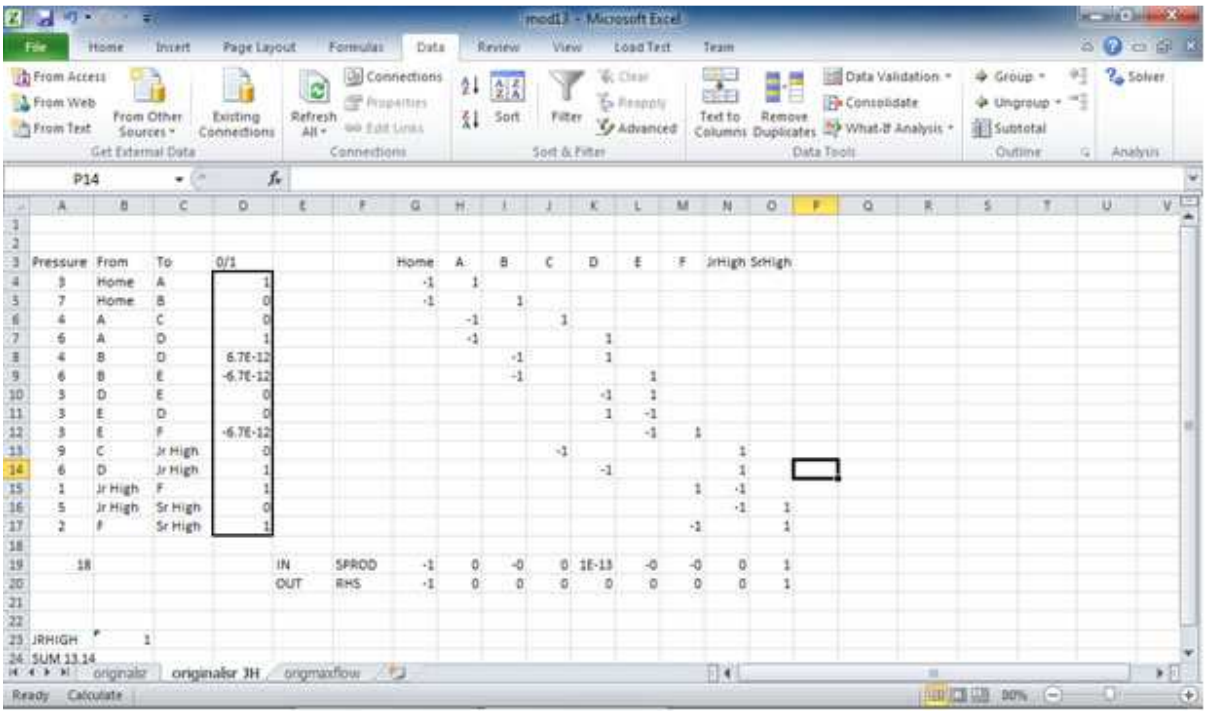


Figure 13.3

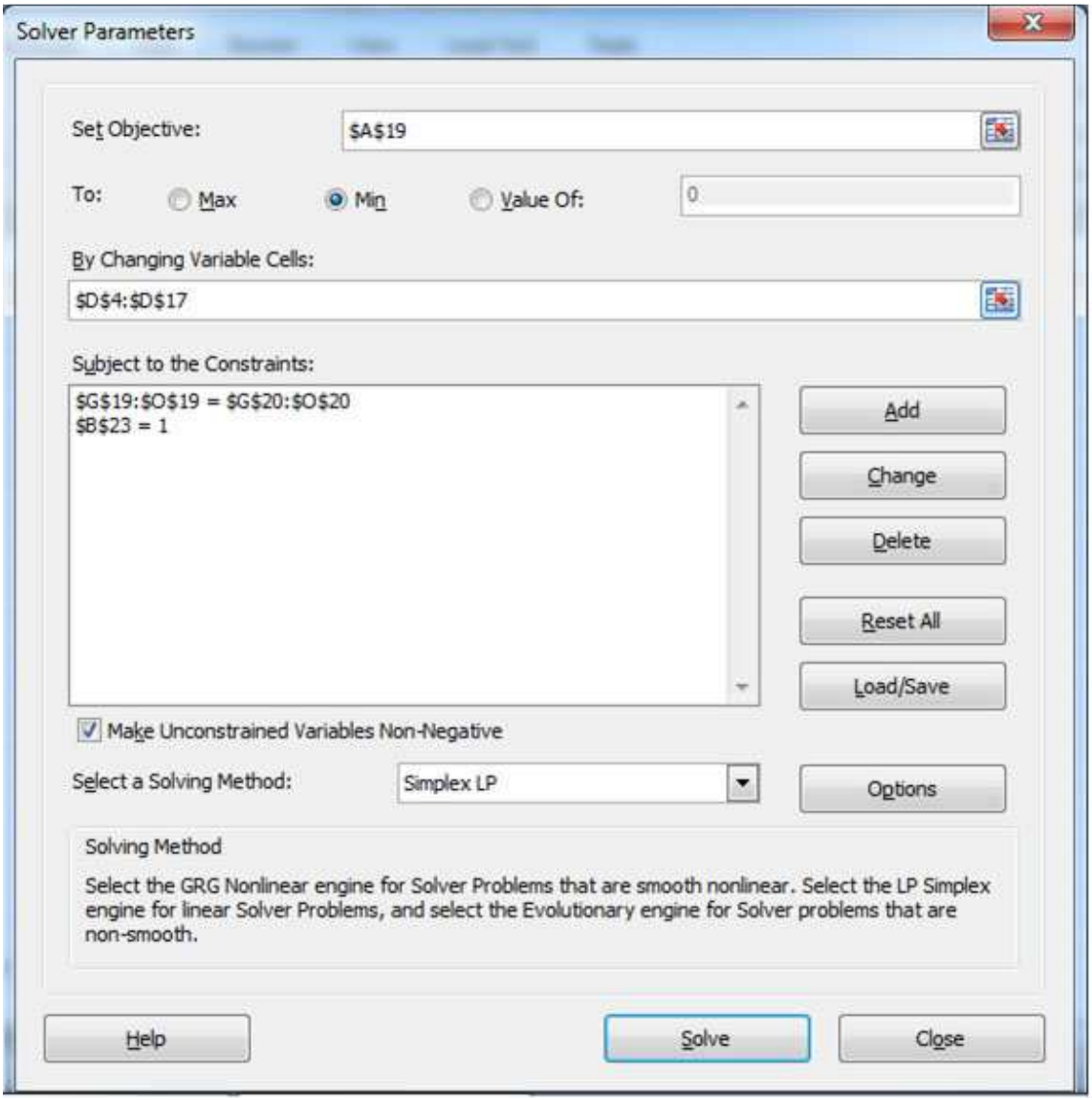


Figure 13.4

The optimal path now travels through the junior high (Home to A to D to Junior High), before going to F, before reaching the Final Destination. This new optimal path is only 1 unit more difficult in terms of pressure (18 units versus 17).

Again, one can add all kinds of additional constraints to the core model – but the key starting point to finding the shortest route between two points in a network is the core model illustrated in this example.

Reading Material: 13.3 – Example 2: Colin's Crazy Cohorts

To illustrate modeling flow through a capacitated network, the same underlying road network used above will be utilized, but with subtle modifications.

13.3.1 Problem Description

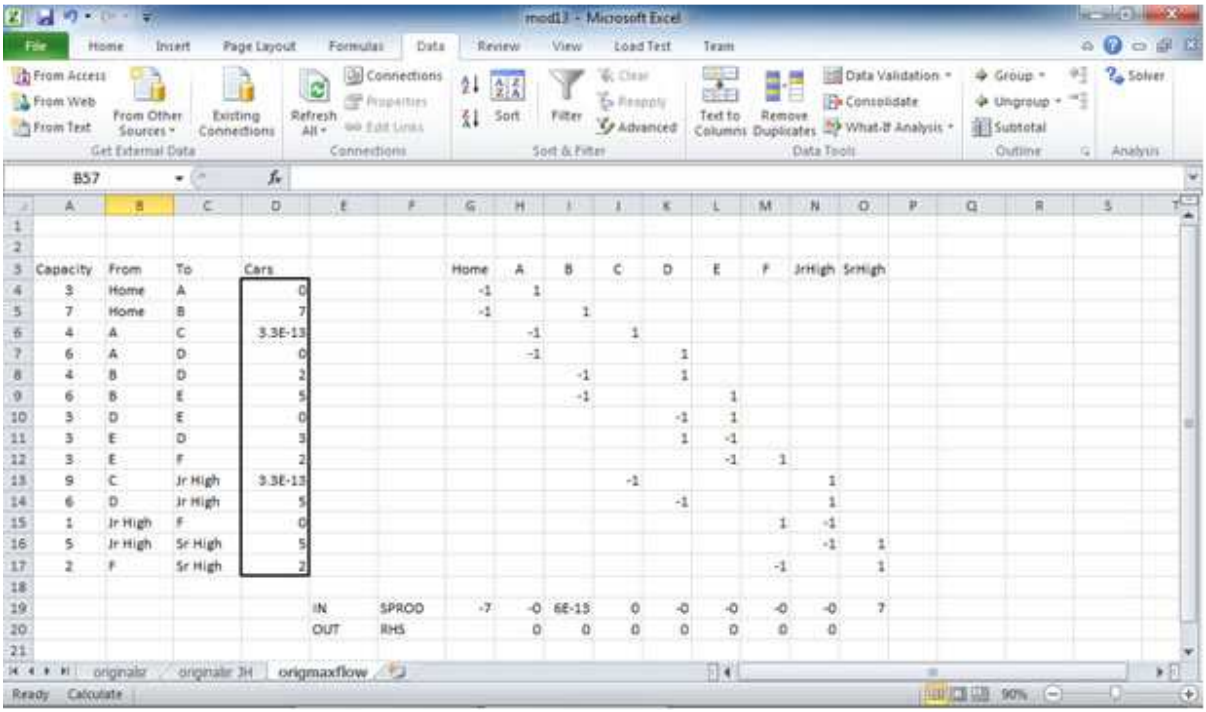
Note: All characters appearing in this example are fictitious. Any resemblance to real persons, living or dead, is purely coincidental.

Trevor's brother Colin typically hosts get-togethers at his house. Sometimes, at a moment's notice, the attendees need to get to the high school ASAP (the kids are still young, and use the parking lot as a congregating place). The Stillwater Police maintain diligence in catching speeders throughout the road network of Stillwater (especially as it relates to young drivers). Consider that the values associated with each link are the "capacity" or the number of Colin's friends who can safely traverse the roads represented by those links and not get pulled over. Thus, three friends can safely go from HOME to Node A. Considering this situation, find the most friends/vehicles that can safely go from HOME to the senior high through Stillwater's street network.

13.3.2 Model Solution: Excel and the Solver

This example might seem more contrived, but by borrowing the same network structure as before, I think it helps us understand the similarities and differences in the two core network models. In some ways, the maximal flow problem described earlier (find the maximum flow through Stillwater's street network from Colin's house to the senior high) is a constrained transshipment model, with one source and one destination. I mentioned this not to wow you with terminology, but to point out that the transportation and network models all share similar characteristics. Thus, when you are saving the world in developing your killer applications, core modeling concepts can be mixed and matched together to create the exact type of model you need.

Figure 13.5 shows the Excel Spreadsheet that implements the maximum flow model. Columns B, C, and D are the same as before, except for the values one will find in column D. The decision variables now represent the amount of flow (in vehicles) between the two nodes – before, it was a 0/1 indicator of whether a link was on the shortest path. So Column D should be a continuous decision variable (one could obviously restrict it to integer numbers in this case, but in general, if flow is oil or water, fractional values of flow is reasonable).



Capacity	From	To	Cars			Home	A	B	C	D	E	F	JrHigh	SrHigh
3	Home	A	0			-1	1							
7	Home	B	7			-1		1						
4	A	C	3.3E-13				-1		1					
6	A	D	0				-1			1				
4	B	D	2					-1		1				
6	B	E	5					-1			1			
3	D	E	0							-1	1			
3	E	D	3							1	-1			
3	E	F	2								-1	1		
9	C	Jr High	3.3E-13						-1				1	
6	D	Jr High	5							-1			1	
1	Jr High	F	0									1	-1	
5	Jr High	Sr High	5										-1	1
2	F	Sr High	2									-1		1
				IN	SPROD	-7	-0	6E-15	0	-0	-0	-0	-0	7
				OUT	RHS		0	0	0	0	0	0	0	

Figure 13.5

Column A is the capacity of each link – thus, from Home to A, at most, the model should allow no more than three cars to travel that route. So, one has a new family of constraints – the link capacity requirements. This is easily implemented by Column D <= Column A.

The table in Column G through Column O – the –1, +1, 0 representation – is still exactly the same, because the network being analyzed is the same network.

Row 19 and Row 20 are basically the same as before, as well, with subtle differences. Row 19 is still the sumproduct function you would expect. In G19, you would have ==SUMPRODUCT(G4:G17,$D4:$D17). One could then copy it all the way through O19.

For maximum flow problems, one only constrains the intermediate nodes with IN=OUT. The objective of the model is to find the maximum amount that can reach the destination node – thus, it is not constrained. And because we don't know how many units can be sent through the network ahead of time, the start/source node(s) also cannot be constrained. Therefore, the IN=OUT family of constraints is H19:N19 = H20:N20, with 0 as the RHS value in Row 20.

As mentioned earlier, the objective is to find the maximum number of vehicles one can get to the senior high. That amount is calculated in cell O19. Thus, O19 is the target cell to be maximized.

Figure 13.6 restates the Solver model used to create the maximum flow model. The solution shows that the maximum number of vehicles can travel safely from Colin's house to the senior

high is seven. You can trace the flow of the cars by looking at the decision variables (this can be good practice in general to validate your model by ensuring flow is conserved).

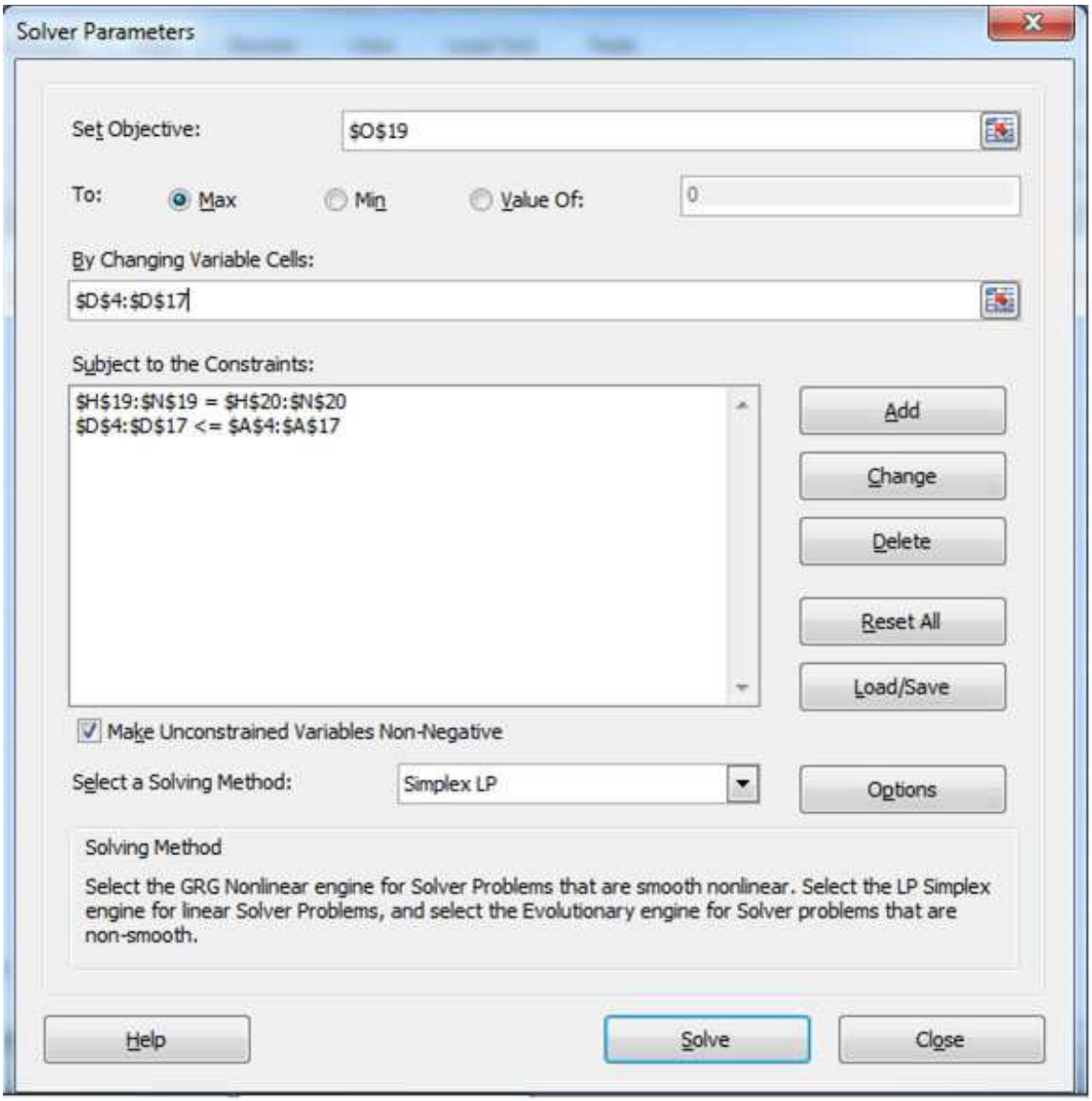


Figure 13.6

Seven vehicles go from Home to node B. At node B, two vehicles travel to D and the other five go to E. At node E, three vehicles travel to node D (recall that is a bi-direction node) and two vehicles to node F. The five vehicles that travel to node D go to the junior high, then in turn go to the senior high. Likewise, the two vehicles at node F then travel to the senior high, for a total of seven vehicles.

Maximum flow problems, without side constraints of any kind, tend to be “underconstrained,” implying there are multiple optimal solutions. Again, that is not bad, just the nature of the

algebra of the model. I mention this because you may set up this model similarly and get different individual link flows, but the maximum vehicles getting to the senior high will still be seven.

Because of the multiple optimal solutions, one might be interested in applying secondary objectives to find a better solution. Having a formal way of dealing with multiple objectives might be nice and useful. Glad you agree. The next couple of modules look at (among other items) dealing explicitly with multiple objectives, including the formal technique of goal programming, which is LP on steroids.

13.4 - Example 3 - Package Delivery – a Solution "Detour"

13.4.1 Introduction

Pardon the pun, but this section is a good detour to take after examining shortest route models and before we look at different solution approaches that allow consideration of multiple objectives (goal programming) in the next module. Here, we explore a fairly straight forward modeling 'trick' called construction methods that can be useful in attacking large scale LP models, and not just in the network realm of applications (though that is how we illustrate it here).

The example we will use to illustrate this is similar (yet different!) to the model in Section 13.2 in which we forced a shortest path to visit a specific intermediate location on the way to its final location.

Example: You are a delivery driver for BP Express Delivery. You are presently in Stillwater, OK (STW), and need to deliver packages to customers in Tulsa, Bartlesville, and Ponca City (in any order), then return the vehicle to the main office in Oklahoma City (OKC). Your task is to find the shortest route that starts in Stillwater and ends in OKC that visits Tulsa, Bartlesville and Ponca City exactly once in an optimal fashion.

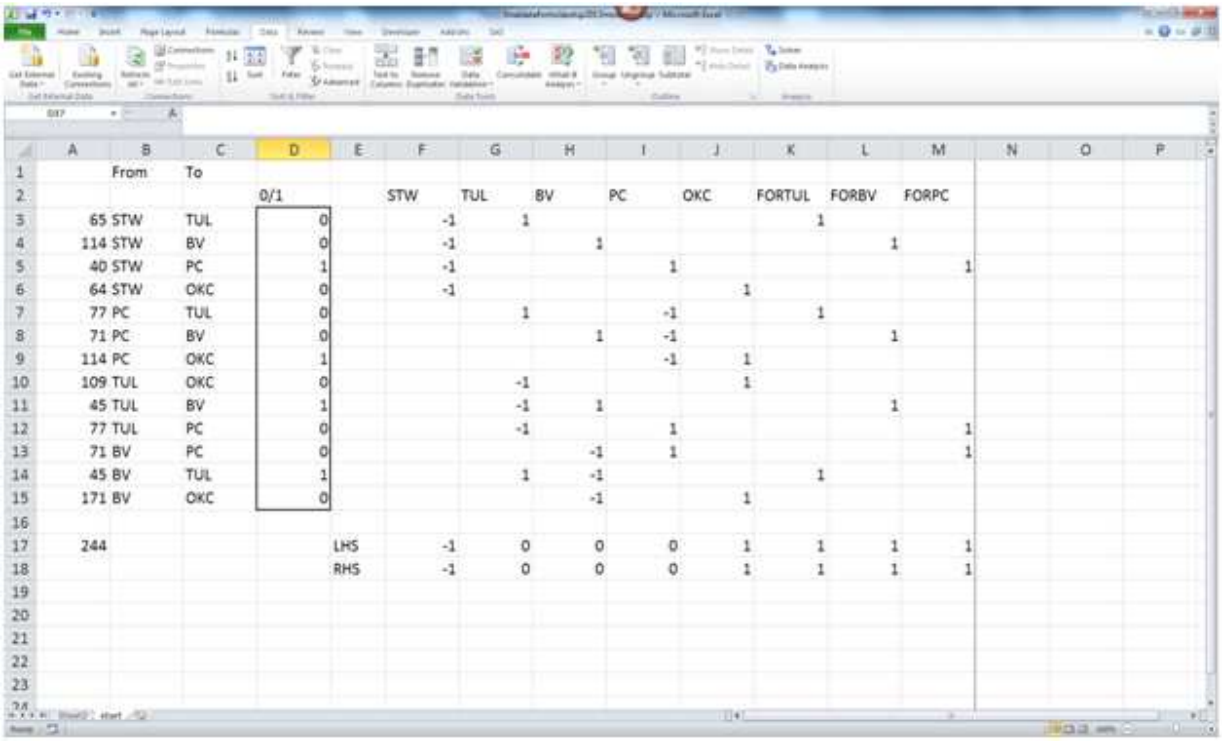
Yes, we could probably find a solution by hand or trial and error, but like a number of examples in this book, small examples allow us to 'touch' the modeling learning objectives intimately and learn the important concepts with minimal distractions from a larger problem context.

13.4.2 Setting up the Core Model

The strategy to attack this situation – first, we set up our core shortest route (SR) model. Additionally, as we did in the supplement example toward the end of Module 13.2.2, we add constraints that force the model to visit each city by adding constraints that sum all entering arcs into a city and force that sum to be equal to 1. We allow bidirectional traversal of

the links between cities, with the obvious exception of STW as the starting point and OKC as the destination.

Figure 13.7 is a screen capture of the solved core model. It is a SR model with the only constraints row 17 = row 18. Note that columns K-M force 'a path' through Tulsa, Bartlesville and Ponca City.



		From	To											
				0/1		STW	TUL	BV	PC	OKC	FORTUL	FORBV	FORPC	
3	65	STW	TUL	0		-1	1				1			
4	114	STW	BV	0		-1		1				1		
5	40	STW	PC	1		-1			1				1	
6	64	STW	OKC	0		-1				1				
7	77	PC	TUL	0			1		-1		1			
8	71	PC	BV	0				1	-1			1		
9	114	PC	OKC	1					-1	1				
10	109	TUL	OKC	0			-1			1				
11	45	TUL	BV	1			-1	1				1		
12	77	TUL	PC	0			-1		1				1	
13	71	BV	PC	0				-1	1				1	
14	45	BV	TUL	1			1	-1			1			
15	171	BV	OKC	0				-1		1				
16														
17	244				LHS	-1	0	0	0	1	1	1	1	
18					RHS	-1	0	0	0	1	1	1	1	

Figure 13.7 - Solution Iteration #1

Obviously, there is something wrong with the solution. The solution claims the shortest route is from STW to PC, then PC to OKC. There are two other links 'used' in the shortest route – TUL to BV and BV to TUL. The solution to the specified model is feasible from an LP modeling standpoint – it just isn't implementable (or logical!). We have a sub-tour – a disconnected path in our solution.

This phenomenon can happen when enough 'side constraints' are added to the shortest route model (the 'forced visitation' constraints) that the actual omission of contiguous link constraints are exposed (bet you didn't realize they were omitted before !). To insure that no 'broken paths' occur in a solution, one would need to enter a number of constraints "ahead of time". These constraints would be numerous – make sure that all 2 city subtours are not allowed, then making sure all 3 city subtours not allowed, etc. Painful perhaps.

We are better off from an efficiency standpoint solving the model knowing we do not have these constraints included, getting a 'super-optimal' solution with subtour(s) (such as what

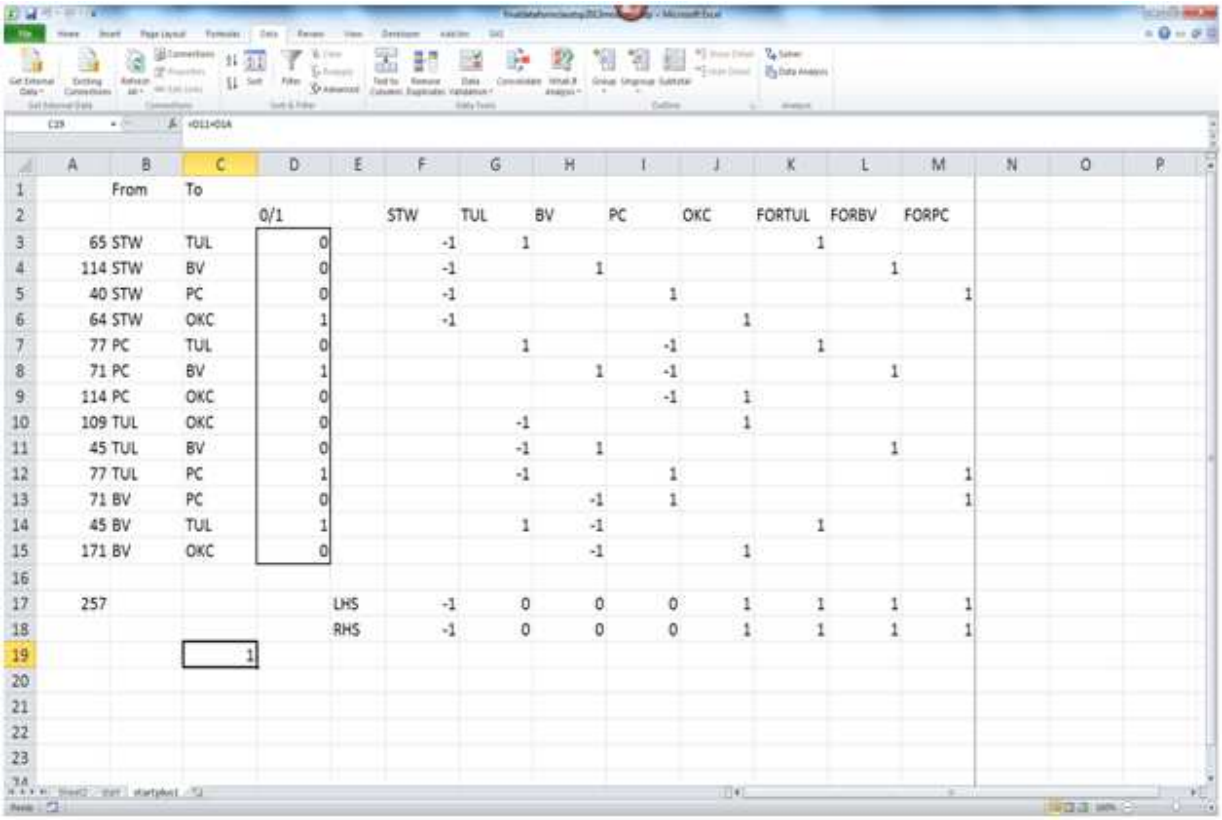
happened), then iteratively adding constraints that remove visible subtours, and eventually, the process gets to the optimal solution. (Adding more and more constraints to the 'super-optimal' solution model will lead us to an optimal solution once we have a connected path and thus, "pass the smell test").

Let's see how this works with our example.

13.4.3 The first additional sub-tour constraints

Next step: Get rid of the subtour between Tulsa and Bartlesville. How? Force the sum of the two decision variables that were both selected to be less than or equal to 1. This makes the subtour itself impossible, but allows the shortest path to include either the link from Tulsa to Bartlesville or the link from Bartlesville to Tulsa, or neither of the two links in the optimal solution.

We can add this constraint anywhere, and you see it added arbitrarily in cell c19. The formula in c19 is: d11 + d14. We will add a constraint in the solver: c19 <=1. Figure 13.8 below shows the results of this additional constraint.



	A	B	C	D	E	F	G	H	I	J	K	L	M
1		From	To										
2				0/1		STW	TUL	BV	PC	OKC	FORTUL	FORBV	FORPC
3	65	STW	TUL	0		-1	1				1		
4	114	STW	BV	0		-1		1				1	
5	40	STW	PC	0		-1			1				1
6	64	STW	OKC	1		-1				1			
7	77	PC	TUL	0			1		-1		1		
8	71	PC	BV	1				1	-1			1	
9	114	PC	OKC	0					-1	1			
10	109	TUL	OKC	0			-1			1			
11	45	TUL	BV	0			-1	1				1	
12	77	TUL	PC	1			-1		1				1
13	71	BV	PC	0				-1	1				1
14	45	BV	TUL	1			1	-1			1		
15	171	BV	OKC	0				-1		1			
16													
17	257				LHS	-1	0	0	0	1	1	1	1
18					RHS	-1	0	0	0	1	1	1	1
19			1										

.

Figure 13.8 - Solution Iteration #2

Isn't that an interesting solution?! Now our shortest path has increased from 244 to 257 miles – is it feasible? Well, the LP model solution is feasible, but the solution again is not implementable. You probably were yelling and screaming at me the whole time to NOT include the link from STW to OKC because that didn't make sense to have even in the original model. See, you were right! It's on the shortest path. But what I really wanted to illustrate was the other funny business going on – the sub tour of PC to BV, BV to TUL and TUL back to PC.

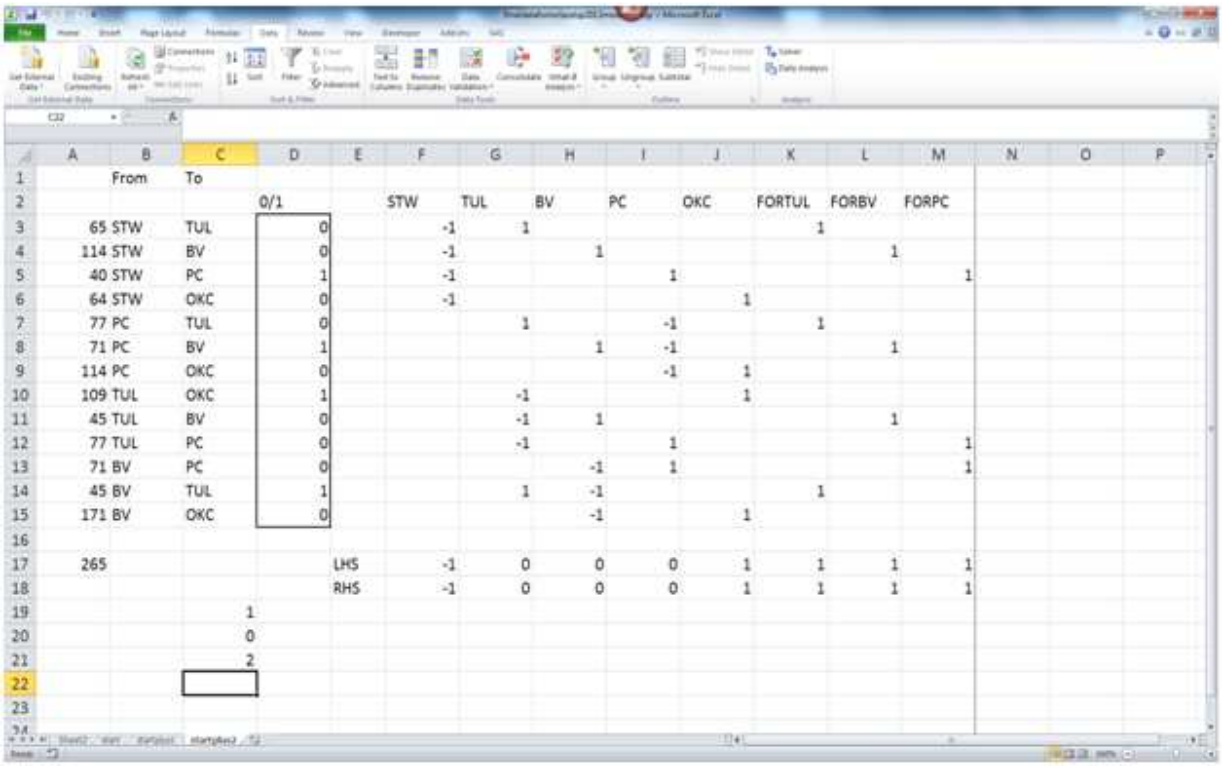
We will add constraints in the next iteration to remove both of these undesirable link combinations.

13.4.4: Next iteration: Adding the 3-city subtour constraint

First, we will add a constraint that forces STW to OKC link =0.

Second, what do we do with the tripartite subtour? We're going to add a constraint that forces the sum of those three links to be <=2 – again, this will remove the 3-city subtour, but allow 2 of the 3 links to be used in a new and improved shortest route. (Note: it is possible, in general, that we'll have to add a subtour constraint for the 'reverse direction' of these three cities as well – but let's just do this one by one and maybe other factors will not make it necessary!).

So, we add both of these new constraints in cells c20 and c21, and add the corresponding constraints to the solver (c20 = 0, c21 <=2). The resolved model is shown below in Figure 13.9.



		From	To											
				0/1		STW	TUL	BV	PC	OKC	FORTUL	FORBV	FORPC	
3	65	STW	TUL	0		-1	1				1			
4	114	STW	BV	0		-1		1				1		
5	40	STW	PC	1		-1			1				1	
6	64	STW	OKC	0		-1				1				
7	77	PC	TUL	0			1		-1		1			
8	71	PC	BV	1				1	-1			1		
9	114	PC	OKC	0					-1	1				
10	109	TUL	OKC	1			-1			1				
11	45	TUL	BV	0			-1	1				1		
12	77	TUL	PC	0			-1		1				1	
13	71	BV	PC	0				-1	1				1	
14	45	BV	TUL	1			1	-1			1			
15	171	BV	OKC	0				-1		1				
16														
17	265				LHS	-1	0	0	0	1	1	1	1	
18					RHS	-1	0	0	0	1	1	1	1	
19				1										
20				0										
21				2										
22														
23														

Figure 13.9 - Final Solution

13.4.5 Are we there yet?

I believe our solution is feasible from an LP standpoint and passes the smell test from all other perspectives. It tells us that we should go from Stillwater to Ponca City to Bartlesville to Tulsa and then to OKC, traversing 265 miles. Since we started from what I've called a 'super-optimal' solution (better than what is really possible), now that we've reached a feasible AND implementable solution, we have found the optimal solution … and we are done!

So let's summarize. Why didn't we add all possible constraints for subtours up-front? Consider a problem with 50 1-way links. That would be 100 decision variables assuming bidirectionality. So, we would start with 50 constraints that deal with subtours back and forth to the same city. Then – how many 3 city combinations would we have to do (with the 3 links <=2) to cover all possible combinations? Then, 4 city combinations <= 3, etc. Get the picture? A lifetime could pass by and we'd still be trying to write all the combination constraints – instead, I would argue it is a better, more efficient strategy to iteratively add to an 'underspecified' model, then try to handle all combinations of possible combinations up front, most of which would simply be unnecessary effort.

Now don't misinterpret this - the only constraints we are adding do not dictate WHAT the solution WILL possess – we are dictating combinations of links that the solution will NOT possess. This is an important distinction. If we force certain aspects of a solution, then look at

other requirements in sequential order, we are actually imposing a priority order to multiple objectives in the solution process. We do this formally in Module 14 – goal programming – but it is intentional. We never want to accidentally impose order when there is none as it will lead to an unintended non-optimal solution.

Finally, this construction approach is a commonly used concept in large scale killer applications. Models do get too big for conventional solution approaches and so we as modelers sometimes must be creative. Another common approach to overcome ginormous model size can be termed a column generation approach – when our decision variables are added in the same, iterative fashion – because we face too many choices (decisions). Thus, we try to get a feasible solution, and then add more choices strategically to try to improve upon it. This is a little harder to simply illustrate but there are plenty of examples in the practice literature and someday maybe they'll be a section 13.5 that shows you a useful, manageable example of such. Of course, maybe Maxwell the Geico pig will be riding in first class too.

Reading Material: 13.5 – Final Comments on Network Models

As mentioned in the introduction, manual programmable approaches that solve these core network models (shortest route and maximum flow) exist. Add some additional constraints to the model, though, and these specialized algorithms do not work. Thus, the more general and useful approach is to learn LP modeling basics for these problem domains. I would be more than happy to help you learn these by-hand algorithms if you have an interest.

There are more specific types of network problems not examined here – minimal spanning trees for one, along with other types of general network flow models. The punch line – once you understand the IN=OUT mantra, the alternative ways of models (To-From versus Row-Column, etc.), you have the ammunition to be able to model any scenario that is network oriented. From time to time, you may hear me mention my experience in creating a system that optimized water systems. From scratch, I wrote a network simplex solution algorithm that was not trivially easy (about 4,000 to 5,000 lines of code). I wish we had access to spreadsheet modeling at the time. We could have "ruled the world" (apologies to Cold Play). So much less effort could have accomplished so much more – just with the concepts presented in this text. Truly, by mastering the network modeling concepts and the other types of models, one really does have the ability and the exposure to relevant scenarios to not only model the world, but also to save it!

We start looking at different ways of codifying model objectives in the next few sections, as well as introducing some interesting model/modeler partnering that is often times necessary in practice.