Sunday, July 21, 2013

Decision Modeling and Optimization in Game Design, Part 3: Allocation and Facility Location Problems

This article is the third in an ongoing weekly series on the use of decision modeling and optimization techniques for game design. The full list of articles includes:

- *Part 1: Introduction (Blogspot) (Gamasutra)*
- *Part 2: Optimization Basics and Unrolling a Simulation (Blogspot) (Gamasutra)*
- *Part 3: Allocation and Facility Location Problems (Blogspot) (Gamasutra)*
- *Part 4: Competitive Balancing Problems (Blogspot) (Gamasutra)*
- *Part 5: Class Assignment Problems (Blogspot) (Gamasutra)*
- *Part 6: Parametric Design Techniques (Blogspot) (Gamasutra)*
- *Part 7: Production Queues (Blogspot) (Gamasutra)*
- *Part 8: Graph Search (Blogspot) (Gamasutra)*
- *Part 9: Modular Level Design (Blogspot) (Gamasutra)*

Spreadsheets for this article can be downloaded here: (SuperTank) (Teleporters part 1) (Teleporters part 2)

SuperTank: Solved!

In the first article in this series, we introduced an example problem for a game called SuperTank. In the second article, we introduced the basic concepts of decision modeling and guided you through a simple example using Excel's Solver tool.

Now, it's time to apply the techniques we learned in Part 2 to the SuperTank problem from Part 1, and prove that we can use them to quickly and easily solve SuperTank. Just to refresh your memory, SuperTank is a game where you're able to fight with a customizable tank. A SuperTank looks something like this:

Each SuperTank can have any number of weapons of five different types, as shown below:

	Damage	Weight	Cost	Critical Slots
Machine Gun	2	1	5	0
Rockets	8	3	12	0
MegaRockets	15	10	16	1
Laser	7	4	9	0
UltraLaser	20	16	18	1

Your SuperTank can hold 50 tons worth of weapons, and you have 100 credits to spend. Your SuperTank also has 3 "critical slots" used by special weapons like MegaRockets and UltraLasers.

You can download the spreadsheet for this example here.

The goal is to pick the weapons that maximize the SuperTank's damage, while staying within the limits of 50 tons, 100 credits, and 3 critical slots. We also assume that this table properly encapsulates everything we need to know, and that factors such as weapon range, refire rate, and accuracy are either irrelevant or are already properly factored into the "Damage" setting for that weapon.

In order to optimize this, we'll first enter the table above into a spreadsheet. Then, just below it, we'll enter a second table that has a set of 5 "quantity" cells to specify the quantity of each of the 5 weapon types.

For now, we'll enter a value of '1' into these cells just to test that they work properly, but these are our decision cells - we will ask Excel Solver to find the correct values of these cells for us (you can tell that they are decision cells due to the yellow coloring, as we are following the formatting guidelines laid out in the second article). To the right of "quantity" cells, we'll add calculation cells that multiply the quantity values in these decision cells by the Damage, Weight, Cost, and Critical Slots values from the table above. Thus, each row of this table will properly add up how much damage, weight, cost, and critical slots all the weapons in each weapon category will use up.

Quantity	Damage x Quantity	Weight x Quantity	Cost x Quantity	Critical Slots x Quantity
1	2	1	5	0
1	8	3	12	0
1	15	10	16	1
1	7	4	9	0
1	20	16	18	1

Finally, we set up a section below that sums up all the quantity, weight, cost, and critical slots values from the table above, and compares those against the maximum weight, cost, and critical slots settings specified in the problem statement (50, 100, and 3, respectively).

	Max Weight	Max Cost	Critical Slots
	50	100	3
Total Damage	52		
Total Weight		34	
Total Cost			60
Total Critical Slots			2

Following our formatting conventions from Part 2 of this series, the blue cells along the top are our criteria from the problem statement. The grey cells are calculation cells representing the total weight, cost, and critical slot values based on the summations from the quantity table above (i.e. the totals of the "Weight x Quantity," "Cost x Quantity," and "Critical Slots x Quantity" columns. Finally, the orange cell represents the total damage of our SuperTank, based on the total damage from the "Damage x Quantity" column of the table above.

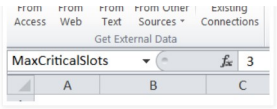
Before we run ahead and solve this, let's take the time to make our spreadsheet a bit more user-friendly. We'll take advantage of Excel's ability to name each cell to give user-friendly names to these 7 cells in the final calculation table. This isn't strictly necessary, but in the long run, it will help make our spreadsheet significantly more understandable if we can give cells readable names such as "MaxCriticalSlots" instead of "F21." To do this, we simply select a cell and go to the name entry box above the spreadsheet and just to the left of the formula bar and type in the new name.

Blog Archive

► 2015 (5)
► 2014 (3)
▼ 2013 (9)
► December (1)
► September (2)
► August (2)
▼ July (4)
Decision Modeling and Optimization in Game Design,...
Decision Modeling and Optimization in Game Design,...
Decision Modeling and Optimization in Game Design,...
Decision Modeling and Optimization in Game Design,...

Contributors

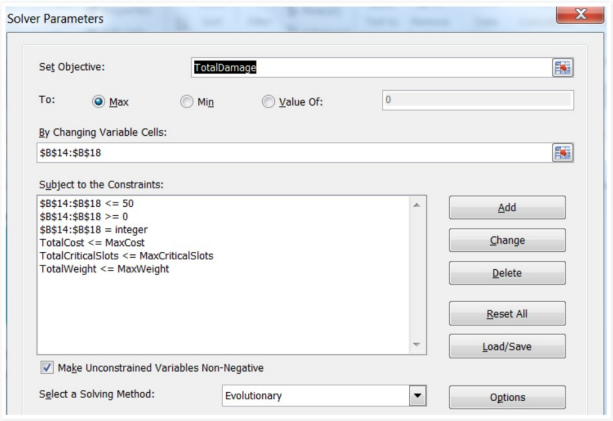
- Paul Tozour
- Sensai Pose
- Unknown



Finally, let's go to Excel Solver and find the solution (go to the right side of the "Data" tab and select "Solver"; if you don't see it, go to Excel Options, select the Add-Ins category, ensure that the "Manage" drop box is set to "Excel Add-Ins," hit "Go..." and ensure that "Solver Add-in" is checked.)

Under "Set Objective," we'll select the orange objective cell and select the "Max" radio button below it. Under "By Changing Variable Cells," we'll select the decision cells (the yellow cells in the "Quantity" column of the second table). Below that, we'll hit the "Add" button to add constraints as follows:

- The decision cells should be between 0 and some reasonable maximum (we pick 50, even though this is probably a much larger upper limit than we need). It's also essential to set the "= integer" constraint on each decision cell, since you cannot have a fractional amount of any given weapon, and since Excel Solver assumes any variable is a real number unless you tell it otherwise, it would undoubtedly try to do this if we let it.
- We also set the total cost, total weight, and total critical slots values to less than the maximums specified in the problem statement. You can see from the dialog image below that these now have the nice user-specified names we added in the table below, making this dialog much more readable.



Now we hit the "Solve" button, and after a brief wait, Solver fills in the "Quantity" values for us, giving us:

- 1 Machine Gun
- 3 Rockets
- 2 MegaRockets
- 1 Laser
- 1 UltraLaser

All of this gives us total damage of 83 and uses exactly 50 tons, 100 credits, and 3 critical slots. You can see that the best solution does not change no matter how much time you give Solver to run. If you reset these values and re-optimize, or if you go to Options and change the random seed, it will still give you these values. We can't be 100% sure this is the optimal solution, but given that Solver seems to be unable to improve on it after repeated optimization passes, it seems quite likely that this is the real optimum and not just a local maximum.

Problem solved!

Additional Uses

What's exciting about this is that we've not only solved the problem much more quickly than we could have done by hand, we've also set it up to allow us to test which weapons are most useful with different (weight, cost, critical slots) settings for a SuperTank. This means that we can relatively easily see and measure the effects of various changes to any of these parameters on the SuperTank, and if we wanted to introduce a new alternate model of SuperTank that was lighter, heavier, or had a different number of critical slots, we could do so very easily.

We can also get a sense of the relative utility of all of our five weapon types as we modify all of these parameters, and quickly identify which weapons are too useful, not useful enough, inappropriately priced for their weight and damage characteristics, and so on.

Again, the point is that this kind of tool allows us to search through the design space much more quickly than we could do by hand. For any incremental design decision we might consider, whether it be changing any of the parameters of the weapons or the SuperTank itself, adding new weapons or SuperTank models, or adding new parameters (say, a Size requirement measured in cubic meters), this gives us a very easy way to get visibility into some of the ramifications of that potential change.

To see what I mean, go to the blue "Max Cost" cell and change it from 100 to 99. Then re-run Solver, and you should now get a very different weapon loadout:

- 0 Machine Guns
- 2 Rockets
- 3 MegaRockets
- 3 Lasers
- 0 UltraLasers

This loadout gets a slightly lower damage score (82 instead of 83), but is radically different from the previous loadout.

If you set Max Cost to 101 or 102 and re-run, chances are that you'll get a configuration similar or identical to the first one; in either case, damage will remain at 83 (actual results may vary since there are several optimal loadouts in these cases). However, if you set Max Cost to 103, you should get:

- 1 Machine Gun
- 4 Rockets
- 2 MegaRockets
- 0 Lasers
- 1 UltraLaser

... which increases our total damage to 84.

This is interesting; this weapon loadout is very different from the first two.

Weapon	MaxCost = 99 TotalDamage = 82	MaxCost = 100 TotalDamage = 83	MaxCost = 103 TotalDamage = 84
Machine Gun	0	1	1
Rockets	2	3	4
MegaRockets	3	2	2
Laser	3	1	0
UltraLaser	0	1	1

As you can see, we get a surprising result: the optimal choice of weapons in our loadout is highly dependent on the SuperTank's parameters and can change dramatically with even a tiny change in those parameters. This also gives us all kinds of other useful information: all five of the weapon types are useful in at least two of the three SuperTank settings, with Rockets and MegaRockets having clear utility in all three. This seems to indicate that all five weapons are well-balanced in the sense that they are all useful relative to one another, while at the same time remaining unique.

Wormhole Teleporters

After looking at the last two examples (the strategy game tax rate example and the SuperTank), you may think these techniques only apply to cases where users have to deal with numbers. Nothing could be further from the truth! As we'll see, there are countless instances where we can gain benefits from optimizing design elements that either do not appear as numbers to the user, or do not seem to involve any numbers at all!

You might also be inclined to conclude that decision modeling only applies when modeling the decisions that players will make in our games. This is also incorrect: in some cases, you can also use it to model an optimize your own decisions as a designer.

Assume that you are working on a massively multiplayer space-based role-playing game. One day your design lead comes to you with a look of worry on his face. "We've just wrapping up a redesign of the Omega Sector," he says, "and we're running into a problem. We're planning to have some wormhole teleporters in that world segment, but we can't agree on where to put them."

"How many teleporters?" you ask.

"We're not sure yet. It will probably be three, but it could be anywhere between 2 and 4. We just don't know right now." Then he lays down a map that looks like this:

	1	2	3	4	5	6	7	8	9	10	11	12
1					*							
2				*					*	*		
3	*					*		*		*		*
4		*					*					
5		*										*
6								*				
7		*								*		
8		*				*						
9												
10								*			*	

"What's that?" you ask.

"It's a map of the Omega Sector. Or at least, the star systems the player can visit within that quadrant. We need you to figure out which cells should contain wormholes."

"OK, what are the rules for wormhole placement? And can I have a wormhole in the same quadrant as a star system?"

"We want you to place the wormholes in a way that minimizes the distance between any star system and the nearest wormhole. And yes, you can put them in the same quadrant as a star system; they're just small teleporters hanging in space, so you can put them anywhere. And don't forget, we haven't agreed on how many wormholes this sector should have yet, so try to give me solutions for 2, 3, and 4 wormholes."

How would you frame this problem, and how would you solve it?

Optimize Me Up, Scotty!

Let's start by setting up the decision cells. We'll call the four wormhole teleporters 'A,' 'B,' 'C,' and 'D.' We know that each teleporter is essentially nothing more than an (x,y) coordinate on the Omega Sector star map above. We also know that we'll need some way to specify how many of these four teleporters are actually active, so we'll add a cell to let us specify the number of teleporters. We will use teleporter 'D' only in the case where we are using 4 wormholes, and we'll use 'C' only when we have 3 or more.

	Teleporter A		Teleporter B		Teleporter C		Teleporter D	
	x	y	x	y	x	y	x	y
Locations	1	1	1	1	1	1	1	1

Number of Teleporters? 3 *Set this to 2, 3 or 4 to change the effective number of teleporters*

Below that, we'll set up a table to figure out the distance from each star system to the nearest teleporter. That table looks like this:

Coord	Px	Py	Dist to A	Dist to B	Dist to C	Dist to D	Dist to Closest
(5,1)	5	1	4.00	4.00	4.00	99.00	4.00
(4,2)	4	2	3.16	3.16	3.16	99.00	3.16
(9,2)	9	2	8.06	8.06	8.06	99.00	8.06
(10,2)	10	2	9.06	9.06	9.06	99.00	9.06
(1,3)	1	3	2.00	2.00	2.00	99.00	2.00
(6,3)	6	3	5.39	5.39	5.39	99.00	5.39

On the left side, in blue, we have the coordinates of each star system on the map. Each star system is represented in one row. We simply typed this in from the Omega Sector map we were handed above.

To the right of that, we calculate the distance to each of the four teleporters A, B, C, and D. This is simply the Pythagorean theorem, calculated as the square root of the horizontal and vertical distance between the star system and the teleporter:

```
=SQRT(($B14-Ax)^2+($C14-Ay)^2)
```

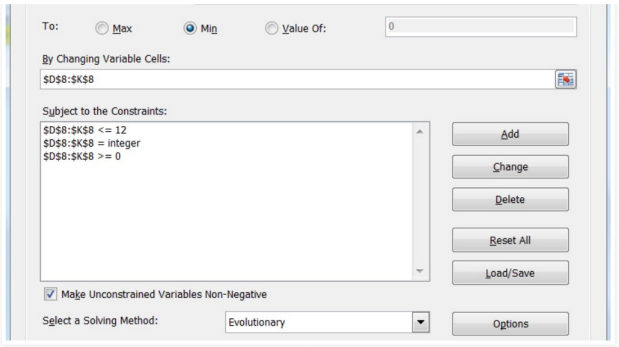
(Don't worry - I promise that this is as complicated as the math will get in this series!)

We get the X and Y coordinates of each star system from the blue cells in the table above, and we get the X and Y coordinates of each teleporter (the cells named "Ax" and "Ay" for teleporter A in the SQRT() function above) from the yellow decision cells at the top.

Finally, we take the minimum of these four values in the "Dist to Closest" column, which simply uses the MIN() function to determine the minimum of the four values immediately to its left. We then sum that entire column at the bottom; this sum is our objective cell.

You may have also noticed that in the screenshot above, the "Dist to D" cells all have the value 99. The reason for this is that we use the "Number of Teleporters?" cell in the section at the top of the decision model to let us tweak the number of teleporters we are considering. If the number of teleporters is 2, we use the value '99' in both the "Dist to C" and "Dist to D" columns, while if it is 3, we use '99' in the "Dist to D" column only. This ensures that each star system will ignore any extraneous teleporters when calculating the distance to the closest teleporter in the case of 2 or 3 teleporters.

Now, we run Solver, as before:



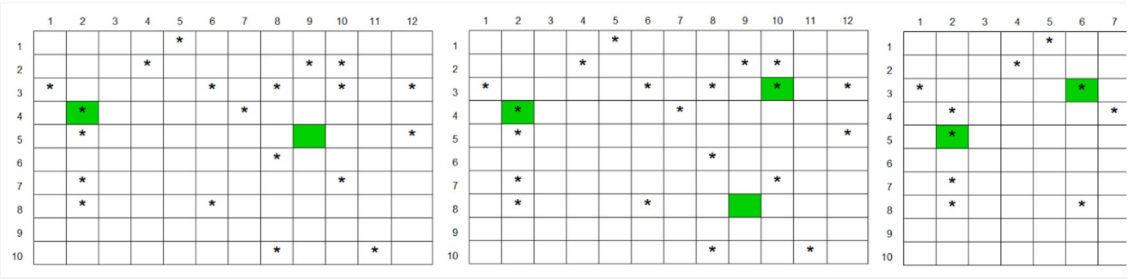
The objective cell is the sum at the bottom of our "Dist to Closest" column. Note that unlike other examples, we want to use "To: Min" in the radio button for this, because we want the minimum distance from all the star systems to our teleporters, not the maximum.

Below that, we specify the decision cells ("By Changing Variable Cells") as the eight yellow decision cells for the X and Y coordinates of wormholes A, B, C, and D. In the constraints section at the bottom, we constrain each of our coordinates to be an integer between 0 and 12. Note that we are using an integer constraint on these decision cells because we are assuming our design lead simply wants to know which cell each teleporter should be in, but we could just as easily skip this constraint if our design lead wanted to know a real-valued location.

If we set the "Number of Teleporters?" cell to 2, 3, and 4, and re-run Solver at each setting, we get the following configurations:

Teleporters	Average Distance	Teleporter Locations
2	55.71	(2,4) and (9,5)
3	42.60	(2,4), (9,8), and (10,3)
4	33.08	(2,5), (6,3), (9,8), and (10,3)

Armed with this information, you can go back to your design lead and show the optimal locations to place any number of teleporters between 2 and 4. Here is what these optimal wormhole locations look like on the map (shown in green) for 2, 3, and 4 wormholes, respectively.



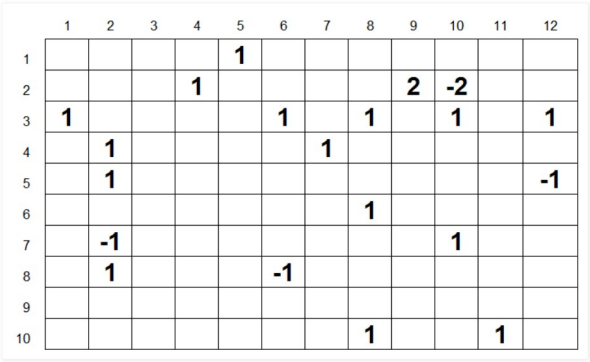
You can download the spreadsheet for this example here.

Did I Mention There Are Ninjas?

"OK, that's terrific," your design lead replies, but you can see a slight look of anguish on his face. "But, uhh ... well, I forgot to tell you some of these star systems are inhabited by Space Ninjas. And we actually want the systems with Space Ninjas to be farther away from the wormholes, because we don't want players to feel too threatened."

"Oh. Well, that kind of throws a monkey wrench into things."

"Yeah. Also, some star systems have 2 colonies in them instead of one, so it makes it twice as important for them to be closer to the wormhole teleporters. Or twice as important for them to be farther, in the case of that one star system that has 2 Space Ninja colonies. Here's what the map looks like now:"



	1	2	3	4	5	6	7	8	9	10	11	12
1					1							
2				1					2	-2		
3	1					1		1		1		1
4		1					1					
5		1										-1
6								1				
7		-1								1		
8		1				-1						
9												
10								1			1	

He continues: "Every negative number is a Space Ninja colony. The system with a '2' has two human colonies, while the '-2' has two Space Ninja Colonies. So, can you tell us where to put the teleporters in this case?"

"Tell me you've at least decided whether there will be 2, 3, or 4 teleporters by now," you reply snarkily.

"No such luck, I'm afraid."

Solving For Ninjas

In order to solve this, we need to add a new column to our table to represent the weightings in the table above. We will call this the "multiplier." We will then multiply this value by the value in the "Dist to Closest" column

When we do this, though, "Dist to Closest" changes its meaning slightly. It's not really the distance to the closest star system, since for Space Ninja star systems, it's -1 times that. It's more of a generalized "score," so let's call it that instead.

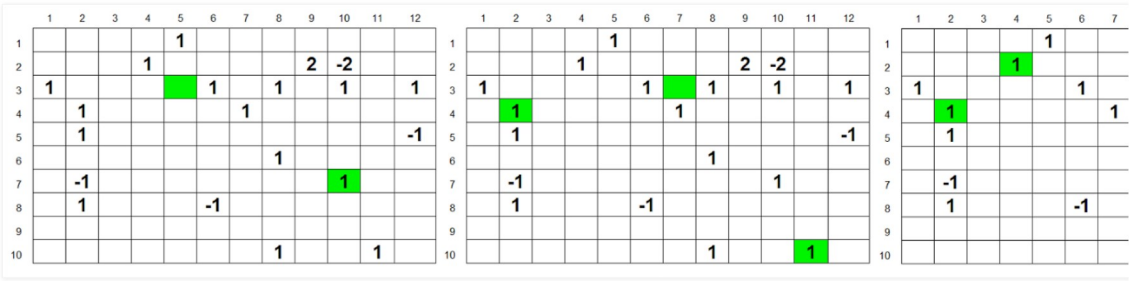
(10,2)	10	2	8.25	6.00	8.06	2.24	2.24	-2	-4.47
(1,3)	1	3	1.41	3.16	12.21	7.00	1.41	1	1.41
(6,3)	6	3	4.12	2.24	8.60	2.00	2.00	1	2.00
(10,3)	10	3	8.06	6.08	7.07	2.00	2.00	1	2.00
(12,3)	12	3	10.05	8.06	7.07	4.00	4.00	1	4.00

In this way, the score now represents an aggregate value. By minimizing it, we ensure that Solver attempts to be as close as possible to human-colonized star systems and as far as possible from Space Ninja-occupied systems simultaneously.

Now we get the following results:

Teleporters	"Score"	Teleporter Locations
2	55.71	(5,3) and (10,7)
3	16.35	(2,4), (7,3), and (11,10)
4	11.90	(2,4), (4,2), (8,3), and (11,10)

As you can see, this gives us a significantly different wormhole configuration in each case from the simpler, pre-Space-Ninja version.



The spreadsheet for this extended version of the teleporter example can be downloaded here.

As you can see, our decision model was able to very quickly solve this non-trivial problem, and we could adapt it to changing requirements remarkably quickly.

This problem is from a class of problems called "facility location problems," which are very well-studied in the operations management field. But as you can see, it has potential applications in game design and level design as well, and is easy (to the point of being trivial) to set up in Excel.

Tune in next time, when we'll apply decision modeling and optimization to a challenging game balancing problem for multiple classes in player-vs-player (PvP) combat for a simplified role-playing game.

-Paul Tozour

Part 4 in this series is now available here.

This article was edited by Professor Christopher Thomas Ryan, Assistant Professor of Operations Management at the University of Chicago Booth School of Business.

Posted by Paul Tozour at 9:30 AM

Labels: Part 3

6 comments:

Fhyl July 22, 2013 at 7:50 AM

Fascinating read.
How are these tools not given more attention? Or are they something all experienced designers know about without ever talking about it?

Reply

Replies

Paul Tozour July 22, 2013 at 7:52 AM

I think very few designers actually know that they exist or understand how to use them in depth. The surveys I've passed around seem to indicate that the subject matter is more or less unknown to designers.

Most likely, the few who do, either haven't had the time to discuss them or haven't been willing to do so.

Reply

Anonymous September 25, 2013 at 4:56 AM

Nice article! I can't believe I didn't know about the solvers in excel before :(. I am going to have some fun with these.

The LP Simplex solver (select from the Solving Method dropdown) is designed to solve linear problems like this (simple combinations of quantities with multipliers). It has less knobs and dials than the others, and if it converges it will be a guaranteed optimal solution. And it should always converge for well behaved problems like the above.

Reply

Replies

Paul Tozour September 25, 2013 at 5:48 AM

Thanks for the compliment, Huw!

Yes, the LP Simplex solver can work for some problems, but it's generally too simple to work on the kinds of complex, nonlinear problems we discuss in this series. They generally require the Evolutionary or GRG solver.

Anonymous September 25, 2013 at 6:08 AM

Oops, should have read the whole article. Only the SuperTank problem is linear.

The Wormhole problem is not linear and LP will complain. I couldn't really get GRG to converge (it does seem to find an optimal solution but struggles to satisfy the integer constraint, I think), so Evolutionary worked best for me here.

Reply

Unknown April 19, 2016 at 9:56 PM

This comment has been removed by a blog administrator.

Reply

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)