Maths behind a MOBA: Using Linear Programming to model and solve a problem

Zachary Knight

December 10, 2015

1 Introduction

MOBA (Multiplayer Online Battle Arena) games have become increasingly popular over the last few years, and the one of my personal preference is called SMITE. In SMITE, players take control of deities from numerous pantheons across the world and work together to destroy objectives at the opposite end of the battlefield, while stopping the enemy team from doing the same to their side. The main way of gaining an advantage is by purchasing items, which give increases to certain attributes that each god has. These attributes include:

- Power - Increases the damage output of an attack
- Attack Speed - How many attacks a god can fire per second
- Lifesteal - The percentage of damage dealt to enemy gods that is returned back to the player as health
- Critical Strike Chance - The chance of an attack dealing double the damage that it would normally do

It should be noted that all these items are bought using Gold, which is mainly gained over time, at a rate of 3 per second. There are five main roles to play as in a game of SMITE, and the main one I have been playing is the role of ADC (All Damage Carry). Arguably the most important role on the team, this means that I will be dealing the most damage for my team and getting the most kills, while preventing the enemy team's ADC from getting anywhere. But what items should I buy to stop them from doing that? In this essay, I will be using Linear Programming to model numerous items to buy and decide which ones will give me the best advantage, while at the same time keeping the price as cheap as possible to ensure a constant increase of stats over the course of the game.

[1]

2 Modelling the problem

I have figured out my solutions with help from Vince Knight's blog post on linear programming [3]. It should be noted that I am modelling this for strictly one-on-one fights against the enemy ADC. This is done not only for practical purposes in terms of the coding involved, but also because as the ADC is one of the most important team roles, he needs to consistently win against his enemy counterpart, meaning that they will fall behind in levels and gold. So I have 16 items, each giving bonuses to some of the four stats I mentioned above, while also having a name and a cost for each item. I am going to store each attribute into a vector and store all 16 items' value of that attribute inside of it. The vectors n, c, p, a, l and d are the vectors for all of the items' name, cost, power, attack speed, lifesteal and critical strike chance respectively. We will also include a vector x, with x_i being the number of item $_i$ bought, this will be the final solution vector, with a 1 in an element of the vector meaning that I should buy the item.

n = ('Asi', 'Deathbringer', 'Devourer's Gauntlet', 'The Executioner', 'Golden Bow', 'Rage', 'Heartseeker', 'Hydra's Lament', 'Bloodforge', 'Soul Eater', 'Malice', 'Brawler's Beat Stick', 'Hastened Fatalis', 'Titan's Bane', 'Odysseus' Bow', 'Transcendence')
c = (1780, 3150, 2050, 2200, 2330, 2755, 1890, 2150, 2265, 2000, 2900, 2400, 2140, 2050, 2450, 2600)
p = (0, 50, 55, 30, 35, 30, 35, 30, 40, 0, 50, 40, 0, 30, 0, 66)
a = (20, 0, 0, 20, 0, 0, 0, 0, 15, 0, 0, 30, 0, 40, 0)

l = (15, 0, 25, 0, 0, 0, 0, 0, 15, 20, 0, 0, 0, 0, 0, 0)
d = (0, 20, 0, 0, 10, 30, 0, 0, 0, 0, 20, 0, 0, 0, 0, 0)
x = (1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1)

As we can only buy whole amounts of an item, this will be solved using Integer Linear Programming, where the problem's solutions are given as integers.

The first few constraints are setting up the fact that you can buy a maximum of five items and you can only buy an item once.

With each $x_i \leq 1$:

$$\sum_{i=1}^{16} x_i \leq 5$$

Next we should get the main objective added, we are trying to minimise the cost of all the items bought:

$$minimise \quad C(x) = \sum_{i=1}^{16} c_i x_i$$

And as we want to get as many stat increases as possible, we should buy exactly 5 items:

$$\sum_{i=1}^{16} x_i = 5$$

To solve this problem I am going to put these vectors and constraints into sagemath, and solve it using the MixedIntegerLinearProgram method, which will a problem for a variable, given certain constraints and objectives.

3 The code & Finding the solution

```
names = ['Asi', 'Deathbringer', 'Devourers Gauntlet', 'The Executioner', 'Golden Bow', 'Rage', 'Heartseeker',
'Hydras Lament', 'Bloodforge', 'Soul Eater', 'Malice', 'Brawlers Beat Stick', 'Hastened Fatalis', 'Titans Bane',
'Odysseus Bow', 'Transcendence']  # The name of each item stored in a vector
cost = [1780, 3150, 2050, 2200, 2330, 2755, 1890, 2150, 2265, 2000, 2900, 2400, 2140, 2050, 2450, 2600]
# The cost vector
temp = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
# Set the constraint for how many item spaces each item fill when it is bought
attackspeed = [20, 0, 0, 20, 0, 0, 0, 0, 15, 0, 0, 30, 0, 40, 0]
# Sets the attack speed increase for each item
power = [0, 50, 55, 30, 35, 30, 35, 30, 40, 0, 50, 40, 0, 30, 0, 66]
# Sets the power increase for each item
lifesteal = [15, 0, 25, 0, 0, 0, 0, 0, 15, 20, 0, 0, 0, 0, 0, 0]
# Sets the lifesteal increase for each item
critchance = [0, 20, 0, 0, 10, 30, 0, 0, 0, 0, 20, 0, 0, 0, 0, 0]
# Sets the critical strike chance increase for each item
items = 5

length = len(cost)  # Get the number of items in the list



Totalcost = lambda x: sum([x[i] * cost[i] for i in range(length)])  # The cost function
Totalitems = lambda x: sum([x[i] * temp[i] for i in range(length)])  # The item quantity function

# Create and solve the optimisation problem
problem = MixedIntegerLinearProgram(maximization=False)  # Create an optimisation problem with minimization as
the goal
x = problem.new_variable(integer=True)  # Create an integer variable
solution
for i in range(length):
    problem.add_constraint(x[i] <= 1)  # Constraint that means there can be at most one of an item
problem.add_constraint(Totalitems(x) == items)  # The total number of items constraint
problem.set_objective(Totalcost(x))  # The objective function
problem.solve()  # Solve the optimisation problem

print 'Items to buy: '
for i, y in problem.get_values(x).iteritems():
    print '%s = %s' % (names[i], int(round(y)))  # Prints the name of the item at that position, and then
prints whether the item is to be bought (1) or not (0)
```

This returns the following:
9770.0
Items to buy: Asi = 1, Deathbringer = 0, Devourer's Gauntlet = 1, The Executioner = 0, Golden Bow = 0, Rage = 0, Heartseeker = 1, Hydra's Lament = 0, Bloodforge = 0, Soul Eater = 1, Malice = 0, Brawler's Beat Stick = 0, Hastened Fatalis = 0, Titan's Bane = 1, Odysseus' Bow = 0, Transcendence = 0,

Meaning that our five items to buy are Asi, Devourer's Gauntlet, Heartseeker, Soul Eater and Titan's Bane, costing a total of 9770 gold. These are the five cheapest items to buy from the store, however this doesn't mean they are necessarily the best items. For example, while these items do give a great amount of lifesteal (an extra 60 percent), it doesn't offer much in terms of power and attack speed, and doesn't give any chance of a critical strike at all. So, to counterbalance this, I am going to include some extra constraints that:

1. Limits the lifesteal to being between 0 and 31 (non-inclusive).
2. Ensures that the chance of a critical strike is greater than 0.
3. Makes the power increase at least 50.
4. Makes the attack speed increase at least 25.

The following code are the four constraints added to the sagemaths program.

```
Totalattackkspeed = lambda x: sum([x[i] * attackspeed[i] for i in range(length)])  # Define the attack speed
function
Totalpower = lambda x: sum([x[i] * power[i] for i in range(length)])  # Define the power
function
Totallifesteal = lambda x: sum([x[i] * lifesteal[i] for i in range(length)])  # Define the lifesteal function
Totalcritchance = lambda x: sum([x[i] * critchance[i] for i in range(length)])  # Define the critical strike
chance function

problem.add_constraint(30 >= Totallifesteal(x) >= 0)  # Sets the total lifesteal constraint
problem.add_constraint(Totalcritchance(x) >= 0)  # Sets the total crit. strike chance constraint
problem.add_constraint(Totalpower(x) >= 50)  # Sets the total power constraint
problem.add_constraint(Totalattackspeed(x) >= 25)  # Sets the total attack speed constraint
```

These new constraints added onto the previous ones return the following:
Items to buy: Asi = 1, Deathbringer = 0, Devourer's Gauntlet = 0, The Executioner = 0, Golden Bow = 1, Rage = 0, Heartseeker = 1, Hydra's Lament = 0, Bloodforge = 0, Soul Eater = 0, Malice = 0, Brawler's Beat Stick = 0, Hastened Fatalis = 1, Titan's Bane = 1, Odysseus' Bow = 0, Transcendence = 0

4 Conclusion

This means that the five items to buy as an ADC to fight against the enemy team's ADC are Asi, Golden Bow, Heartseeker, Hastened Fatalis and Titan's Bane, giving the following stats:
Cost: 10190, Power Increase: 100, Attack Speed Increase: 50, Lifesteal Increase: 15, Critical Strike Chance Increase: 10

[Asi] [Golden Bow] [Heartseeker] [Hastened Fatalis] [Titan's Bane] [2]

I know that this is a pretty accurate solution because I use all of these items in my "build" except for Golden Bow. However, I cannot completely rely on this solution for a couple of reasons. This is firstly because of the fact that I have assumed that each battle I'm in I am only against the enemy ADC, meaning that this may not be guaranteed to be effective in a team dynamic. Also, each item in game comes with a passive ability, granting an additional statistic increase in certain situations, e.g. hitting an enemy removes your movement penalty when firing for a short time. These are near impossible to code in sagemath, because they are interaction dependent, meaning there are too many variables involved that are constantly changing. This is why I believe the solution I have found is the optimal solution, and I have found it using a combination of both computer and linear programming.

References

[1] Image of SMITE's map. http://www.pcgamer.com/smites-new-conquest-map-revealed/. Accessed: 2015-12-06.

[2] Item icons. http://smite.gamepedia.com/Category:Passive_item_icons. Accessed: 2015-12-08.

[3] Un peu de math... wizards, giants, linear programming and sage. http://drvinceknight.blogspot.co.uk/2014/05/wizards-giants-linear-programming-and.html. Accessed: 2015-12-07.