Dark Souls 'Damage Optimisation' Problem - Solution

prattling-pate

June 2024

Contents

1 Introduction

Dark Souls (2011) is a video-game and video-game franchise developed by From-Software and published by Bandai Namco. It is an RPG (Role Playing Game) based about careful and deliberate gameplay as well as player character statistics which players are free to choose to increase one at a time during gameplay. Players typically are confused as to how exactly this works and may spend hours on player made wiki entries stating how exactly to level up individual stats to maximise the damage dealt by the player. Additionally, non-damage outputting statistics such as "Vitality" and "Endurance" affect critical player features such as total damage the player can take before dying and the amount of stamina the player character has.

This is not to be used as much of a very useful tool as it is not difficult to appoint points in a decent manner, especially at high levels where the points given become arbitrary if a player is knowledgeable. This is made in order to test the suitability of linear programming in video game skill point appointment in general to produce a general solution to the problem later on.

2 Linear Programming

2.1 First Strategy

The setup of this problem is that the player needs to choose to appoint skills one at a time into one of 8 skills per level, with each skill doing something else. Each of the rewards the skills produce are quantifiable, we can therefore use a certain type of problem solving using a linear programming setup to maximise these rewards put into a multi-variable function with certain constraints acting on these.

2.2 Linear Programming

2.2.1 What is Linear Programming?

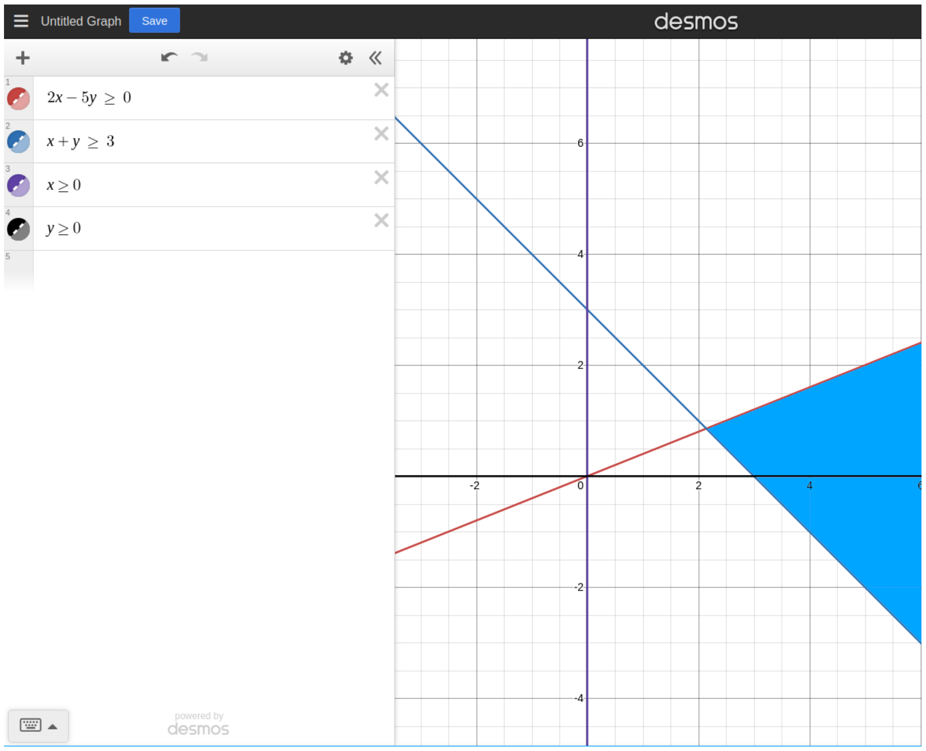
Linear programming is an approach to optimising a linear multi-variable function subject to an array of linear inequalities called constraints. Where a given optimisation problem is called a linear program.

2.2.2 Solving linear programs

To illustrate this section consider the following linear program:

$$\begin{array}{lr} \text{min} & 2x + y \\ \text{subject to} & x + y \geq 3 \\ & 2x - 5y \geq 0 \\ & x, y \geq 0 \end{array}$$

This is asking us to solve the optimisation problem of optimising $f(x, y) = 2x + y$ subject to the inequalities $x + y \geq 3$, $2x - 5y \geq 0$ and $x, y, \geq 0$. This can actually be solved graphically by simply drawing the region of $\mathbb{R}^2$ that the constraints prescribe.



$2x - 5y \geq 0$

$x + y \geq 3$

$x \geq 0$

$y \geq 0$

Figure 1: A Desmos graphic of the lines representing the constraints of the above linear program (blue region).

Then we can draw lines parallel to the direction in which the objective function $f(x, y)$ increases which will be represented by vectors pointing in the direction:

$$\nabla f(x, y) = (\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}) = (2, 1)$$

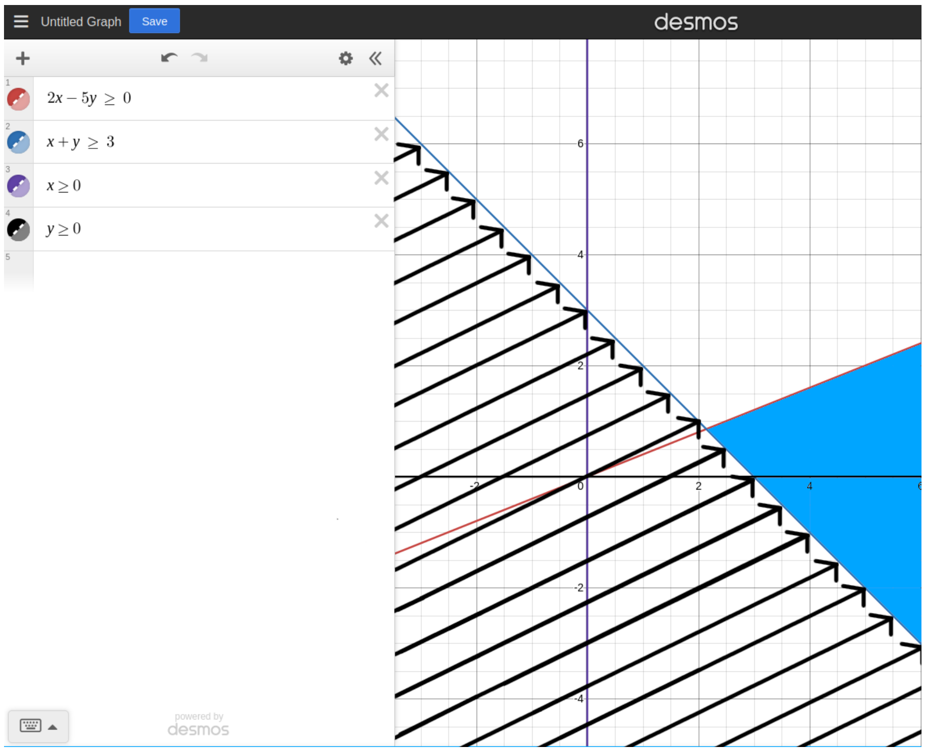


Figure 2: A Desmos graphic of the lines representing the constraints of the above linear program (blue region) along with arrows denoting in which direction the objective function increases most.

Looking at these we want to minimise the total of the objective function given it is some point in the blue region, it is easy to notice that this exists on the lower-left edge of the region. Testing this we can look at the points of this edge which are described by the line:

$$y = -x + 3, \quad \frac{15}{7} \leq x \leq 3$$

Note that if we substitute this expression for the edge into $f(x, y)$ we get

$$f(x, y(x)) = 2x + (-x + 3) = x + 3$$

.

So the objective function is non-constant along this line and is minimised by x being minimal. Hence the minimum value of the objective function along the given constraints is $x = \frac{15}{7}$ so $y = -(\frac{15}{7}) + 3 = \frac{6}{7}$.

So finally we have found the solution to the linear program of $(x, y) = (\frac{15}{7}, \frac{6}{7})$ and this solution is unique.

However, we cannot use the graphical method for problems with more than 3 variables as it is simply visually impossible to represent higher dimensional objects. However there is a method for solving such linear programs with similar intuitions called the simplex method.

The core idea of the simplex method is that simple solutions to these linear programs can be found at the vertices of the shape described by the constraints of the program, moving between these vertices until an optimal solution can be found.

This method will not be fully explained or derived here as it is a lengthy explanation. For more information see Linear Programming in the bibliography [Chv83].

2.2.3 Setting up the damage optimisation program

The following is the standard form of a linear program used for producing solutions:

$$\begin{array}{ll} \text{min} & \boldsymbol{cx} \\ \text{subject to} & A\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{x} \geq 0 \end{array}$$

Where A $\in \mathbb{R}^{m \times n}$, $\boldsymbol{x}$, $\boldsymbol{b}$, $\boldsymbol{c} \in \mathbb{R}^n$, A represents the matrix of constraints while b and c represent the constraint equations and costs of each decision variable respectively. We however will constrain this problem to the integer space rather than the real space as it makes no sense to have "half a point" in a skill in the Dark Souls franchise.

Hence the problem is now:

$$\begin{array}{ll} \text{min} & \boldsymbol{cx} \\ \text{subject to} & A\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{x} \geq 0 \end{array}$$

Where A $\in \mathbb{Z}^{m \times n}, \boldsymbol{x}, \boldsymbol{b}, \boldsymbol{c} \in \mathbb{Z}^n$.

In order to create our linear program we need to identify what function we will be optimising and what constraints it is subject to. Obviously, we want to optimise the total damage output of a given weapon and we want to do this subject to the constraints of the levelling system in Dark Souls.

2.2.4 Setting up the damage optimisation program - constraints

Firstly we will look at creating constraints for our linear program (as discussing the objective function will take much more time):

1. We cannot have a skill be a higher level than 99,
2. We cannot have a skill be below the maximum between what the class of the player starts at or what the minimum weapon requirement is for our given weapon.
3. We must use all skill levels given to us, so the sum of all skill points must be equal to the default skill points in the skills and the number of skill points we will use.

We can easily characterize these constraints using algebraic inequalities. Firstly let's define some notation that will help us (some of these will be reiterated in the next section):

- Let l be the number of skill points devoted to the problem.
- Let c_i be the class minimum skill points for skill i.
- Let r_i be the requirement for the given weapon to be used for skill i.
- Let S be the set of all skills in the problem.
- Let $\boldsymbol{x}$ be the vector of skill points (each entry is related to some $i \in S$).

Now we can clearly write each of the three constraints algebraically as the following:

1. $x_i \leq 99, \forall i \in S$
2. $x_i \geq \max\{c_i, r_i\}, \forall i \in S$
3. $\sum_{i \in S} x_i = l + \sum_{i \in S} c_i$

2.2.5 Setting up the damage optimisation program - objective function

By looking at data mining efforts by the player-base of Dark Souls we have the exact formula used in calculating the damage of a given weapon in Dark Souls [Wik].

Let's define some variables/constants to make this easier:

- Let $P = \{\text{STR}, \text{DEX}\}$ be the set of physical attributes
  - Write STR as 1, DEX as 2 throughout the project.
- Let $M = \{\text{INT}, \text{FTH}\}$ be the set of magic attributes

    – Write INT as 3, FTH as 4 throughout the project.

- Let $S = P \cup M$ be the set of damage attributes
- Let f_P be the damage rating function for physical attributes
- Let f_M be the damage rating function for magic attributes
- Let $k_i \in (0, 2]$ be the damage scaling multiplier of the skill $i \in S$ due to the given weapon[1]
- Let B_P be the base physical damage of the given weapon
- Let B_M be the base magic damage of the given weapon

Now by the efforts of the community [Wik] the formula for the damage is:

$$f(\boldsymbol{x}) = B_P \cdot (1 + \sum_{i \in P} k_i f_P(x_i)) + B_M \cdot (1 + \sum_{i \in M} k_i f_M(x_i) \tag{1}$$

Where $\boldsymbol{x}$ is the vector of skill points of every skill $i \in S$. We want to simplify this down into the form $f(\boldsymbol{x}) = \boldsymbol{c} \cdot \boldsymbol{x}$ where $\boldsymbol{c}$ is a vector of real numbers in order for this to be compatible with the simplex method for solving linear programs.

Firstly we need to prove some properties of the simplex method that will help us prove that a simpler function will provide an equivalent solution to the linear program, firstly let's define a term.

Definition 2.1 (Solution equivalency on a linear program). *Let f, $g : \mathbb{R}^n \to \mathbb{R}$, f and g are called solution equivalent on a linear program P if and only if solving P with either f or g as the objective function provides the exact same solution to problem P (where the constraints of P are kept the same on both functions). Let $\cong$ denote the solution equivalency between two functions on a linear program P.*

Lemma 2.1 (Irrelevancy of linearity). *Given a linear multi-variable function $f : \mathbb{R}^n \to \mathbb{R}$, two distinct functions g_1, $g_2 : \mathbb{R}^n \to \mathbb{R}$ of the form $g_1(\boldsymbol{x}) = a \cdot f(\boldsymbol{x}) + b$ a, $b \in \mathbb{R}$ and $a > 0$ are solution equivalent on any linear program.*

Proof. Let $f : \mathbb{R}^n \to \mathbb{R}$ and g_1, $g_2 : \mathbb{R}^n \to \mathbb{R}$ such that $g_1(\boldsymbol{x}) = a \cdot f(\boldsymbol{x}) + b$ where a, $b \in \mathbb{R}$ and $a > 0$ and $g_2(\boldsymbol{x}) = c \cdot f(\boldsymbol{x}) + d$ where c, $d \in \mathbb{R}$ and $c > 0$.

Recall that solving linear programs involves following the optimising direction of the objective function until we meet a point in the feasible region which minimises the objective function. We find the optimising direction by taking the gradient of the objective function.

[1] Note that each k_i is given by a piece-wise function taking in discrete letter values, outputting the midpoint of the possible scaling of a weapon of that grade. More info available on the wiki [Wik] [Fex]

Observe:

$$\nabla g_1(\boldsymbol{x}) = \nabla(a \cdot f(\boldsymbol{x}) + b) = a\nabla f(\boldsymbol{x}) \text{ (By properties of partial derivatives)}$$

$$\nabla g_2(\boldsymbol{x}) = \nabla(c \cdot f(\boldsymbol{x}) + d) = c\nabla f(\boldsymbol{x}) \text{ (By properties of partial derivatives)}$$

Now we may take the norm's of these vectors to get the unique direction's in which they point:

$$\nabla \hat{g}_1(\boldsymbol{x}) = \frac{1}{|\nabla g_1(\boldsymbol{x})|}\nabla g_1(\boldsymbol{x})$$

$$\nabla \hat{g}_2(\boldsymbol{x}) = \frac{1}{|\nabla g_2(\boldsymbol{x})|}\nabla g_2(\boldsymbol{x})$$

Now substituting each expression for gradient:

$$\nabla \hat{g}_1(\boldsymbol{x}) = \frac{1}{|a\nabla f(\boldsymbol{x})|}a\nabla f(\boldsymbol{x}) = \frac{a}{a|\nabla f(\boldsymbol{x})|}\nabla f(\boldsymbol{x}) = \frac{\nabla f(\boldsymbol{x})}{|\nabla f(\boldsymbol{x})|}$$

$$\nabla \hat{g}_2(\boldsymbol{x}) = \frac{1}{|c\nabla f(\boldsymbol{x})|}c\nabla f(\boldsymbol{x}) = \frac{c}{c|\nabla f(\boldsymbol{x})|}\nabla f(\boldsymbol{x}) = \frac{\nabla f(\boldsymbol{x})}{|\nabla f(\boldsymbol{x})|}$$

Hence we get that each of the function g_1, g_2 point in the same direction, hence resulting in the same optimal solution given the same constraints of a linear program. So $g_1 \cong g_2$ on any linear program. $\square$

Using Lemma 2.1 we can rewrite the original function to be of the form $f(\boldsymbol{x}) = \boldsymbol{c} \cdot \boldsymbol{x}$.

Observe:

$$f(\boldsymbol{x}) = B_P \cdot (1 + \sum_{i \in P} k_i f_P(x_i)) + B_M \cdot (1 + \sum_{i \in M} k_i f_M(x_i)$$

$$\iff f(\boldsymbol{x}) = B_P + B_P \sum_{i \in P} k_i f_P(x_i) + B_M + B_M \sum_{i \in M} k_i f_M(x_i)$$

$$\iff f(\boldsymbol{x}) \cong B_P \sum_{i \in P} k_i f_P(x_i) + B_M \sum_{i \in M} k_i f_M(x_i)$$

$$\iff f(\boldsymbol{x}) \cong (k_1 B_P, k_2 B_P, k_3 B_M, k_4 B_M) \cdot (f_P(x_1), f_P(x_2), f_M(x_3), f_M(x_4))$$

Hence we can instead solve the linear program using:

$$g(\boldsymbol{x}) = (k_1 B_P, k_2 B_P, k_3 B_M, k_4 B_M) \cdot (f_P(x_1), f_P(x_2), f_M(x_3), f_M(x_4)) \quad (2)$$

Notice that we are still not at the correct form. Here we run into some issues trying to further simplify the function. If f_P and f_M were perfectly linear functions then we could do that easily, however the definitions of the two functions are as follows [Wik] [Fex]:

$$f_P(x) = \begin{cases} 0.005x & \text{if } 1 \leq x \leq 10 \\ 0.05 + 0.035(x-10) & \text{if } 10 < x \leq 20 \\ 0.4 + 0.0225(x-20) & \text{if } 20 < x \leq 40 \\ 0.85 + 0.0025(x-40) & \text{if } 40 < x < 99 \\ 1 & \text{if } x = 99 \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

$$f_M(x) = \begin{cases} 0.005x & \text{if } 1 \leq x \leq 10 \\ 0.05 + 0.0225(x-10) & \text{if } 10 < x \leq 30 \\ 0.5 + 0.015(x-30) & \text{if } 30 < x \leq 50 \\ 0.8 + 0.0041(x-50) & \text{if } 50 < x < 99 \\ 1 & \text{if } x = 99 \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

So the functions are actually continuous[2] and linear (assuming x is real and not discrete like it will be), but only over parts of their domain due to their piece-wise definitions. The reason for this is a concept known as "soft caps" in skill leveling in Dark Souls (This allows for very accurate damage optimisation). Hence we need to discuss some alternative ways to approach this linear program.

3 Problems in linear programming

In order to solve our optimisation for damage we must be able to properly handle the fact that the objective function changes as our decision variables change. These are different for each of the two subsets of skills P and M.

One way we may handle this is by approximation of each function to a linear function allowing us to apply the simplex method without any problems, however we lose the previously mentioned encoded "soft caps" in skill leveling with this, although this will result in faster time to solve.

Another way we may handle this is by running the simplex method recursively on the problem every time the objective function changes due to the interval x_i exists in for one of the skills $i \in S$, then by appropriately modifying the simplex method instance and continuing to solve until this happens again

[2] f_M is technically not continuous as it is laid out here as the 0.0041 coefficient on the interval $50 < x < 99$ is an approximation of the true coefficient for nice layout, the function itself is actually continuous.

or the problem is solved. However this would cause an upper bound of 16 simplex method switches occurring, this also may not be guaranteed to provide a solution every-time (as I haven't proven this to be true).

3.1 Solving the problem

Let's try to solve this problem with the outlined solutions above.

3.1.1 Approximation to a linear function

We can simply approximate $f_P(x)$ and $f_M(x)$ both to be the identity function $f(x) = x$ as graphically comparing this to the piece-wise function it is a "close enough" approximation for the purposes of our solution.

From here we can apply the simplex method without problem so this is our first working solution for the problem.

3.1.2 Recursive application of the simplex method on a piece-wise linear function

Firstly it's easy to see that this will take a bit more work, however it will be a more useful algorithm due to the "soft caps"[3] of Dark Souls being directly built into this rather than being ignored in the previous solution (although this may not affect the solution provided by the method).

In order to provide this we must switch the reduced cost vector of the simplex instance (referred to as a tableau), ensuring we have it adjusted accordingly to new objective function and current solution. Then we will simply reapply the simplex method until the function switches or an optimal solution is finally given.

Algorithm 3.1 (Piece-Wise Linear Objective function Simplex method)**.**

1. *Find the current objective function by looking at the piece-wise definition of the damage rating functions.*
2. *Apply the two phase simplex method*
3. *If the objective function changes due to the piece-wise damage rating functions being altered then go back to step 1.*

Eventually this will yield an optimal solution for all feasible problems.

With both of these approaches we may finally formally construct our linear programs to be solved to accomplish this optimisation problem.

[3]A "soft cap" in role playing video games are typically points at which levelling a skill produces diminishing returns, there may be multiple soft caps for each skill as there are in Dark Souls

4 Setting up the linear program formally

So far we have been setting up in very vague terms what we will be doing to solve the problem. However now I will lay out every step of preparation explicitly:

4.1 Setting up the constraints

In linear programming to solve a problem we want to be able to express the constraints as a matrix A paired with a vector $\boldsymbol{b}$ with the relationship $A\boldsymbol{x} = \boldsymbol{b}$ where $\boldsymbol{x}$ is the vector of decision variables (in this case the skill points in every skill). To do this we must convert some of our constraints from inequalities to inequalities by adding additional decision variables:

1. $x_i \leq 99, \forall i \in S \implies x_i + y_i = 99, \forall i \in S$
2. $x_i \geq \max\{c_i, r_i\} \forall i \in S \implies x_i - s_i = \max\{c_i, r_i\}, \forall i \in S$
3. The third constraint is already described by an equality

$$\sum_{i \in S} x_i = l + \sum_{i \in S} c_i$$

We can encode this into a matrix of size 9×12 as each via matrix multiplication we have the following:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \end{pmatrix} = \begin{pmatrix} 99 \\ 99 \\ 99 \\ 99 \\ \max\{c_i, r_i\} \\ \max\{c_i, r_i\} \\ \max\{c_i, r_i\} \\ \max\{c_i, r_i\} \\ l + \sum_{i \in S} c_i \end{pmatrix} \tag{5}$$

Where $x_1, ..., s_4 \in \mathbb{Z}^+$.

This matrix equation expresses all of our constraints exactly.

4.2 Setting up the objective function

When using the simplex method that I will be personally using (which minimises a given linear function of objective functions) we must express the objective function in the following way:

$$\min \boldsymbol{c} \cdot \boldsymbol{x}$$

Where $\boldsymbol{c} \in \mathbb{R}$, $\boldsymbol{x} \in \mathbb{Z}^+$. $\boldsymbol{x}$ is the vector of decision variables and $\boldsymbol{c}$ represents the coefficients in front of each decision variable in the linear objective function.

Our objective function for the optimisation damage problem is

$$g(\boldsymbol{x}) = (k_1 B_P, k_2 B_P, k_3 B_M, k_4 B_M) \cdot (f_P(x_1), f_P(x_2), f_M(x_3), f_M(x_4))$$

We also want to maximise this function (i.e. maximise damage output) so the objective function part of the problem will be formulated the following way:

$$-\max_{\boldsymbol{x}} -g(\boldsymbol{x})$$

Note that we still need to convert our objective function g into the form $\boldsymbol{c} \cdot \boldsymbol{x}$. This is where this section splits into two sections for each approach we have discussed so far.

4.2.1 Approximation method

If we use the linear approximation that $f_P(x) \approx x$ and $f_M(x) \approx x$ then we can easily convert into this form:

$$g(\boldsymbol{x}) \approx (k_1 B_P, k_2 B_P,_3 B_M, k_4 B_M) \cdot (x_1, x_2, x_3, x_4) \tag{6}$$

We can then keep this function constant for the whole duration of the problem and simply apply the two phase simplex method to find a solution to the total problem:

$$\begin{aligned} \text{min} \quad & \boldsymbol{cx} \\ \text{subject to} \quad & A\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{x} \geq 0 \end{aligned}$$

Where

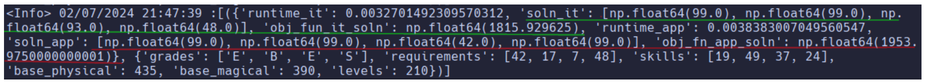
$$A = \begin{pmatrix} 1&0&0&0&1&0&0&0&0&0&0&0 \\ 0&1&0&0&0&1&0&0&0&0&0&0 \\ 0&0&1&0&0&0&1&0&0&0&0&0 \\ 0&0&0&1&0&0&0&1&0&0&0&0 \\ 1&0&0&0&0&0&0&0&-1&0&0&0 \\ 0&1&0&0&0&0&0&0&0&-1&0&0 \\ 0&0&1&0&0&0&0&0&0&0&-1&0 \\ 0&0&0&1&0&0&0&0&0&0&0&-1 \\ 1&1&1&1&0&0&0&0&0&0&0&0 \end{pmatrix} b = \begin{pmatrix} 99 \\ 99 \\ 99 \\ 99 \\ \max\{c_i,r_i\} \\ \max\{c_i,r_i\} \\ \max\{c_i,r_i\} \\ \max\{c_i,r_i\} \\ l+\sum_{i \in S} c_i \end{pmatrix} c = \begin{pmatrix} k_1 B_P \\ k_2 B_P \\ k_3 B_M \\ k_4 B_M \end{pmatrix} x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \end{pmatrix}$$

4.2.2 Iterative method

This is very similar to the approximation method with identical setup except for the fact that we keep our objective function g as the non-linear piecewise function it is and apply algorithm 3.1.2.

5 Testing of algorithms

After testing these two approaches it is noticed that the approximate solution is actually more effective than the iterative solution for some special cases.



```
<Info> 02/07/2024 21:47:39 :[({'runtime_it': 0.0032701492309570312, 'soln_it': [np.float64(99.0), np.float64(99.0), np.
float64(93.0), np.float64(48.0)], 'obj_fun_it_soln': np.float64(1815.929625), 'runtime_app': 0.0038383007049560547,
'soln_app': [np.float64(99.0), np.float64(99.0), np.float64(42.0), np.float64(99.0)], 'obj_fn_app_soln': np.float64(1953.
9750000000001)}, {'grades': ['E', 'B', 'E', 'S'], 'requirements': [42, 17, 7, 48], 'skills': [19, 49, 37, 24],
'base_physical': 435, 'base_magical': 390, 'levels': 210})]
```

Figure 3: A screenshot of raw data results from running iterative algorithm (red) and approximative algorithm (green) in Python 3.12, shows solution for each skill [STR, DEX, INT, FTH] and damage done by the solution.

This is likely because the objective function for the iterative method incentivises ignoring the points with the least skill points in them due to the lower slope of the damage rating functions at the lowest points of the domains. Hence what we should actually do is decrease the slopes as we gain larger skill points (i.e. incentivise increasing lower skills as they will lead to larger reward). This is an approach unusual to a linear programming problem, however since this is technically non-linear programming (but very similar) we must take other approaches. Let's see how this approach may improve or worsen the given results.

5.1 Problems with the iterative method

However, with the iterative method we can actually see that it does not properly optimise the problem no matter how the objective functions are modified (i.e. changing to a strictly decreasing linear piece-wise function from what the functions normally are to better encode skill "soft caps", this will be discussed further in the next approach).

6 Linear Piece-wise Programming

As we saw in the previous sections simplifying the problem down to a linear program and solving using the simplex method does provide solutions, however these are not optimal solutions as it is easy for me to personally change the variables to get a higher damage number by manually changing these solutions a bit. This is because the "soft cap" information is lost when approximating and the iterative changing of the objective function does not work as I thought it would (likely due to some loss in rigour of the method, in which I hadn't proven that this approach would even work). Hence in this section we will step back into the abstraction of the problem as a geometric optimization problem in an n-dimensional space as in section 2.1.

6.1 Approaching the problem without the simplex method

From section 2.1 we can get the function we want to optimise and the constraints on which to optimize it on (which we have used in previous sections for approximate solutions).

6.1.1 Restatement of the problem

The optimisation problem is the following:

$$\begin{array}{lrl} \text{max} & f(\boldsymbol{x}) & \\ \text{subject to} & x_i \leq 99, & \forall i \in S \\ & x_i \geq \max\{c_i, r_i\}, & \forall i \in S \\ & \displaystyle\sum_{i \in S} x_i = l + \sum_{i \in S} c_i & \\ & x_i \geq 1, & \forall i \in S \end{array}$$

Where $f(\boldsymbol{x}) = B_P \cdot (1 + \sum_{i \in P} k_i f_P(x_i)) + B_M \cdot (1 + \sum_{i \in M} k_i f_M(x_i)$.

6.1.2 Geometric intuition for a solution

We can solve the given problem from the previous section by using a similar intuition as we used in section 2.1, where we can view the constraints as describing an n-dimensional space (where n is the number of skills we are optimizing, in this case 4) of all possible vector solutions (where each component is the number of skill points in each skill, so a vector in $\mathbb{R}^n$), and somewhere in this space lies some optimal solution.

We can do this by moving our point from our starting solution (the default skills of the player) l times in the integer lattice of solutions in the prescribed space (i.e. in the space described by the constraints there are discrete points which are consist of integer vector entries in $\mathbb{Z}^n$). The idea will be to move in the unit direction (i.e. increase a vector component of our proposed solution by 1) of greatest change (which will increase f by the greatest amount) until we use up all of our level points l. In the following section we will discuss how we will do this.

6.1.3 Overview of solution

Generally speaking, what we want to do is in l steps increment the skills vector by 1 in a unit direction of $\mathbb{Z}^n$ and ensuring this is the optimal movement to do by verifying that this will increase the damage the most.

To determine at each step what direction to move the solution we can use calculus to determine this by using partial derivatives. This comes from the idea that the partial derivative of any multi variable function tells us the instantaneous rate of change of the function if we move in the direction of any of the unit vectors of the domain of the function (in this case we can think of this as $\mathbb{R}^n$, for the sake of continuity of the function, which we later confine to $\mathbb{Z}^n$).

So what we will do is take the (local[4]) partial derivatives of f with respect to each unit vector and compare which of them is greatest (i.e. moving in that direction, f will increase by the largest amount). Since f is linear piece-wise then the partial derivatives of the function will all be real constants which are easy to compare using the standard relations ($>, <$ etc.). So now we can mathematically describe our algorithm for solving this optimisation problem:

6.1.4 Mathematical prescription of the algorithmic solution

Algorithm 6.1.
Let $\boldsymbol{x} = (x_1, \ldots, x_n)$ be the current skill vector (initially starts at default skills of player character, i.e. $x_i = c_i$ initially for all $i \in S$).
Let l be the number of skill levels to spend.
Repeat the following l times:

1. *Enumerate all local partial derivatives (around $\boldsymbol{x}$) of f, and pick the skill $i \in S$ with the following:*

$$\arg\max_{i \in S} \frac{\partial f}{\partial x_i}(\boldsymbol{x})$$

2. *Set $\boldsymbol{x} = (x_1, \ldots, x_i + 1, \ldots, x_n)$.*

The final state of the vector $\boldsymbol{x}$ will be the optimal solution to the maximisation problem.

[4]The partial derivative of f depends on what domain the skills vector resides in, as f is a piece-wise function. Hence we will only focus on the derivatives around where the skills vector currently is (as we want which direction to move in locally to our current skills vector).

Note: For the above algorithm we will define the enumerated local partial derivatives with the following rather than the standard definition as otherwise the derivatives will be undefined at the boundaries of f's piece-wise domains:

$$\frac{\partial f}{\partial x_i} = \lim_{h \to 0^+} \frac{f(x_1, \ldots, x_i + h, \ldots, x_n) - f(x_1, \ldots, x_i, \ldots, x_n)}{h} \quad \text{for} \quad f : \mathbb{R}^n \to \mathbb{R}$$

Bibliography

[Chv83] V. Chvátal. *Linear Programming*. Series of books in the mathematical sciences. W. H. Freeman, 1983. ISBN: 9780716715870. URL: `https://books.google.co.uk/books?id=DN20_tW_BV0C`.

[Fex] Dark Souls Community Wiki Fextralfe. *Dark Souls: Weapon Scaling wiki*. URL: `https://darksouls.wiki.fextralife.com/Weapon+Scaling`. (accessed: 07/02/2024).

[Wik] Dark Souls Community Wiki. *Dark Souls: Parameter Bonus wiki*. URL: `https://darksouls.fandom.com/wiki/Parameter_Bonus`. (accessed: 06/14/2024).