Calculating the highest DpR in D&D 5th Edition

Frederic Frenkel

Albert-Ludwigs-Universität Freiburg, DE

Abstract. The abstract should briefly summarize the contents of the paper in 15–250 words.

Keywords: TTRPG · Simulation · Optimization

Table of Contents

1 Introduction

5e (D&D 5th Edition) is sometimes criticized for the martial gap [1], the observation that the classes relying on magic feel much more powerful at higher levels, being able to bend reality with spells like ”wish”, while martial classes may get to do one more attack in a round.

The goal of this project is to calculate the best DpR (Damage per Round) martial character in 5e.
For this, we will generate all[1] possible combinations of characters, meaning that every decision made in character creation and at level up will multiply the number of characters we will need to rank.

[1] We heavily reduce our amount of starting points, and what characters we keep level-to-level, explained here

2 The Rules

The basic combat analysis we will cover relies on the Basic Rules of D&D 5e. [2] The system uses multiple types of dice, denoted as xdy, where x is the number of dice, and y is the highest face of the die.
A character has 6 Ability Scores, ranging from 8 to 20. Each score has a modifier, which is calculated as

$$modifier = floor((Score - 10)/2) \tag{1}$$

We will use Point-buy, where you can choose any number from 8 to 15 as your starting scores.
You have 27 points to spend, where a Score of 8 costs nothing, and every step until 13 costs 1 more point, 14 and 15 both costing 2 points to reach instead.

A character starts at level 1, and can reach Level 20, gaining feats and features at certain levels, depending on their class.
A character can multiclass, if their appropriate Ability Score is high enough.
They start the new class at level 1, and gain those features if they take more levels in that class. The 20 level restriction applies to the character, meaning that the levels of classes combined cannot exceed 20.
Classes have subclasses that grant access different extra features.

In combat, a character can use actions he knows to make Attack Rolls on the enemy. We roll 1d20, add any bonuses (like a relevant modifier), and hit if our result is equal to or higher then the enemies Armor Class.
If we have advantage, we roll 2d20 and take the higher.
If the die lands on a 20, we always hit, and any dice used to calculate damage are doubled.
A one is always a miss.
Certain features can increase the number of crit dice, or expand the range that results in a crit.
Crit dice enhanced by features only count towards **one** weapon die, meaning that a character with a greatsword (2d6) will do 5d6 instead of the usual 4d6, but a character with a greataxe (1d12) will do 3d12 instead of 2d12.
This enhancement does not add to crits for any other dice added, even if they are part of weapons damage. A Barbarians elemental cleaver adds 1d6 to his weapon damage, but a crit with a greataxe is still 3d12 + 2d6, not + 3d6.

In a round, a character gets an action, a bonus action, and a reaction.
The action can be used for making an attack. Bonus actions are usually unused, but feats (gained at some levels) like Polearm Master allow a character to make a second attack as a bonus action, if their weapon is classified as a Polearm.
Some features also allow the character to add damage once per round without costing an action, like the Rogue's sneak attack.
Every class has one Ability Score that is their Primary Score, which they can add to attack and damage rolls.
Another way to use the level-up feats is an ASI (Ability Score Improvement), which allows you to increase a score by 2 points, or two scores by 1 point.

3 The Math

We will go through this analytically, as using random number generation will lead to the same results with much more processing.
The average value of each die is found by

$$dieAverage = (dieFace/2) + 0.5 \tag{2}$$

This is used to assign each set of die associated with damage a static float value.

To calculate the chance to hit, we will find the amount of die sides that would result in a hit, and divide by 20.

Table 1. Breakdown of formula used in program.

Partial Formula	Explanation
AC - ToHit	We use all modifiers, and see what we need to roll (AC of 19 vs +5 to hit, we need to roll a 14)
21 - acDifference	This inverts our result, giving the sides that are hits (21 - 14 is 7, we hit on all 7 sides in [14, 20])
max(hitSides, 21-critRange)	If we have a high difference, we might hit on ”-2 sides of the die”. We always have at least 1 side that hits (20, possibly 19 / 18)
min(max(...), 19)	A one always misses. We cant have more then 19 sides.
min(...) / 20	Now that we know how many sides hit, we get the hit chance by dividning by sides on the die.

$$hitChance = min(max(21 - (AC - toHit), 21 - critRange), 19)/20 \tag{3}$$

For hit chance with advantage, we can just use the inclusion-exclusion principle on our last result:

$$\begin{aligned} \text{chanceAdvantage} &= (P(\text{hit}) \textit{ or } P(\text{hit})) - (P(\text{hit}) \textit{ and } P(\text{hit})) \\ &= 2 * P(\text{hit}) - P(\text{hit})^2 \\ &= P(\text{hit}) * (2 - P(\text{hit})) \end{aligned}$$

Crit chances are a bit easier, we only need

Partial Formula	Explanation
critRange-1	If we crit on 20, there are 19 sides that dont crit.
1-(critRange-1)/20	The inverse of our chance not to crit.

Advantage can use the same formula as hit chances.

The hit chances were validated by comparing them to values from Can I hit This? [3]

4 The Implementation

4.1 Reduction

As we will very quickly reach a high level of complexity, we need to reduce our possible starting points.

Limited Features There are features that can be used only a limited number of times a day, while others increase the damage of every attack a character makes.

We only consider those that are available consistently through combat.
This will be more accurate in some ways, as a character that does 80 damage in the first round with resources but then only 10 damage for every following round may be less usefull in longer combat then a character that does 40 damage every round until the end of combat.
But this also omits some features from different classes, like the Paladins[2] Smite, which uses a limited resource that would do a good amount of extra damage every day. That will not be considered, as it is hard to say how long a combat may last, how many combats may happen until the resources get refilled, etc.

Bad Decisions We will only choose good Ability Scores. If a class has no benefit from a certain Score, we will not create a version of a character in that class that wastes points on it. (Based on advice from RPGBot.net [4])

Pre-calculations All the hit and crit chances are done at the start of the program and stored in a hashmap, as we need to calculate the damage (and therefore hit chance) for every single character we generate at every level.
This replaces multiple equations with a lookup of O(1).

Culling To ensure that the program runs in the 10 minute window, we provide a Settings file. One can choose how many characters of each class will be kept after ranking, meaning that only the top x strongest characters of every class will be used as a bases for the choices at the next level, stopping exponential growth as we raise levels.
Another parameter can be set that makes the culling only occur on every nth level after the first, improving the chances for multiclassing while not completely disabling culling.

Copies Copying is necessary, as we want to branch characters of in all directions they can access. We can't allow two derivatives of one character to be influenced by each others features. Many objects don't need copies, like classes. The dicts and lists that contain them still have to be replicated shallowly.

This is by far the most expensive step. Any future optimization should focus on this aspect.

[2] Paladins are not implemented at the time of writing

4.2 Syntax

To allow anyone to easily add new classes, subclasses, actions and more, we read them from folders in the same directory as the program, meaning none of the attributes are hard coded.

Features Our most basic structure is the feature.
A feature can be gained from a Class, Race, or Feat.
The feature's name is the files name.
Features are applied to characters in 4 ways:

```
1. var[path/variableName value type]
2. method[path/methodName(value=type|value2=type2|...)]
3. feature[featureName holeValue|holeValue2|...]
4. action[actionName]
```

1. A feature can change a variable of the character, like adding 1d6 to their weapon damage. For example, barbarians get extra crit dice, the feature referenced in their class contains:

```
var[battleStats/extraCritDice ? int]
```

This feature can add to a characters extraCritDice counter, which is located in their battleStats variable.
Here we encounter holes. Features can leave holes to be filled by the provider of the feature. If there are holes, they are filled from left to right, top to bottom with values that are given. (So in the order they appear in the features file.)
Only values can be holes, the type should always be clear.

2. A feature can use a method of the character, like adding an extra attack. If a character gets extra attack from two classes, they do not stack, instead using the highest count out of the ones given by the classes.
We use the method to add the Attack to a dict and then re-calculate the actual counter with the highest dict value.

```
method[battleStats/addExtraAttack(?=str)]
```

Here, the hole is for the name of the class that wants to add the extra attack.
Methods can have multiple parameters, which are split by "|".

3. A feature can have subfeatures for convinience, like the savage attacks feature half-orcs get. This feature adds one crit die to their counter - but we have a feature for that - as defined in example 1. We simply call that feature and fill its hole.

```
feature[ExtraCritDice 1]
```

4. A feature can give a character access to a new action, like the rogue's sneak attack.

```
action[SneakAttack]
```

Any number of any of these four can appear in a single feature file.

The type given for each value can be an int, str, bool, attrType, or dice. The values for dice are written in the **xdy notation**, attributes (Ability Scores) as their first 3 letters, so Str, Dex, Con, Int, Wis, or Cha.

Requirements Requirements are implemented similarly to Features.
They are used by Classes, Feats, and Actions.
A requirement consists of 3 parts, the value to check, an operator, and a value / type pair. To add a requirement, we use

```
Req: <characterValue operator compareValue typeOfValue>
```

The character value is a method or variable, as described in the last section.
In requirements, there are no holes, and character variables don't need a value / type pair, as we are asking for them instead of adding to them.
The operators available are =, <, and >.
For equality, the compare value can have multiple items separated by "|", in that case they should all have the same type. The requirement is fulfilled if one of them is the same as the characters value.
One example, a class may require to have a 13 Strength to be used by a character. Our requirement would look like this:

```
Req: method[attr/getStat(Str=attrTpye)] > 12 int
```

The method called gives the Ability Score of a character, which combines the base points from character creation, and any bonuses added through Race, Feats, or ASI's.
Another example, the Polearm Master feat requires the character to use Polearms, resulting in this requirement:

```
Req: var[battleStats/weapon/wType] = Polearm str
```

Requirements don't have their own files, they are conditions only found in others.

Classes can have multiple requirements, like Fighters, who need either 13 Str or Dex:

```
Req: method[attr/getStat(Str=attrTpye)] > 12 int
or Req: method[attr/getStat(Dex=attrTpye)] > 12 int
```

Or Paladins, who require both 13 Str and Cha:

```
Req: method[attr/getStat(Str=attrTpye)] > 12 int
and Req: method[attr/getStat(Dex=attrTpye)] > 12 int
```

This is specific to classes, other users of Requirements can have any number of `Req:` sections. In that case all requirements have to be fulfilled to return true.

Feats have requirements and can give access to actions and features.

Races can increase a characters Ability Scores, and may use any of the 6 first letter shorthands combined with the value added (e.g.: `Cha 2`), or `All x` if all stats increase by the same value (like Standard Human).

They can also grant Features. As those are in a dedicated `Features` section we will omit the `feature[...]` syntax, instead using the shortend

```
featureName>holeValue|holeValue2|...
```

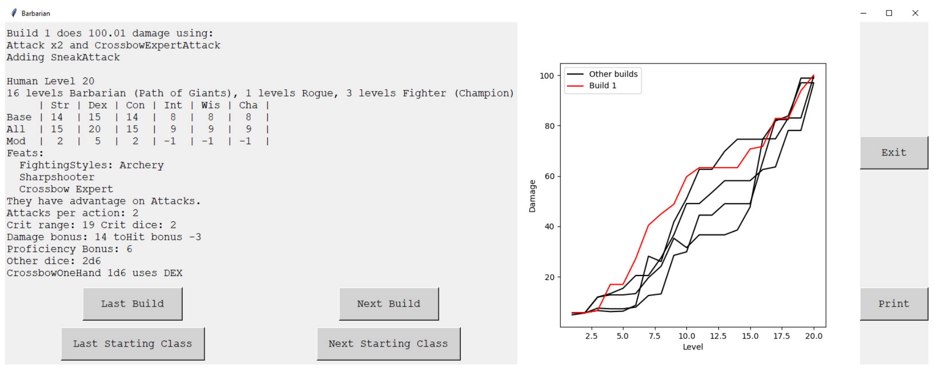
Or just the features name (file name) if the feature has no holes.

4.3 Output

To see our results, we pick the top 5 ”builds” (Combination of all choices that resulted in the character).

To have a more interesting graph, we select our top five builds with the requirement that the next best build always has to do less damage then the previous, as we don’t want to end up with almost the exact same character 5 times.

While characters may have been culled due to having lower damage, the ranked character list still exists, those characters only aren’t used for further offspring.
This makes it possible to revisit the results we reached at any level we have calculated on our way to the chosen level.


```
Build 1 does 100.01 damage using:
Attack x2 and CrossbowExpertAttack
Adding SneakAttack

Human Level 20
16 levels Barbarian (Path of Giants), 1 levels Rogue, 3 levels Fighter (Champion)
     | Str | Dex | Con | Int | Wis | Cha |
Base | 14  | 15  | 14  |  8  |  8  |  8  |
All  | 15  | 20  | 15  |  9  |  9  |  9  |
Mod  |  2  |  5  |  2  | -1  | -1  | -1  |
Feats:
  FightingStyles: Archery
  Sharpshooter
  Crossbow Expert
They have advantage on Attacks.
Attacks per action: 2
Crit range: 19 Crit dice: 2
Damage bonus: 14 toHit bonus -3
Proficiency Bonus: 6
Other dice: 2d6
CrossbowOneHand 1d6 uses DEX
```

Fig. 1. While going through the list of builds, the current build is highlighted in red, while others are black.

The buttons and window are made with TKinter, the graphs using matplotlib. This is the best build the program finds, given the current, limited, options.

5 Conclusion

While the classes / features that are available are heavily limited by our computational bounds and Limited Features rule, we can get some very usable characters that - in cases - can do great and reliable damage.

The rogue class had the highest difficulty multiclassing, as the immediate damage increase from sneak attack at every second level (+1d6 per Round) seemed to outweigh anything gained from taking 3 - 6 levels in another class, leading to the lowest damage output because of our more greedy implementation.

The fighter on the other hand couldn't stop giving most its levels to Barbarians, as the low number of feats made their high number of ASI's relatively useless after they had maxed out their primary and secondary Ability Score.

The winner of this setup seemed to be a ranged Barbarian, taking 1-3 levels of fighter for their Archery Fighting Style, increasing their to hit modifier by 2, which can have noticable effects in 5e.

Reaching 100.1 points of damage every round seems fantastic, especially when comparing this to the normal DpR of characters with "High Damage" - as given by RPGBot.net [5] - which starts at just 66.7 DpR!
Needles to say, we reached our goal in making a character that can be just as impressive as some casters are.

6 Themes

Table 2. Topics coverage

Topics	Usage
Linux	Only used to confirm work, as I didn't want to use VM's during debug for performance.
Text Editor	VSCode was used. Nothing of note.
Git	Used for submission.
Docker	Not used.
Automation	All code is written in python. Automation is the main goal for this project.
Gnuplot	I used pyplot to display the best results.
LaTeX	The report is beeing written on Overleaf.
LLM	Not used.

7 Setup, Recommendations

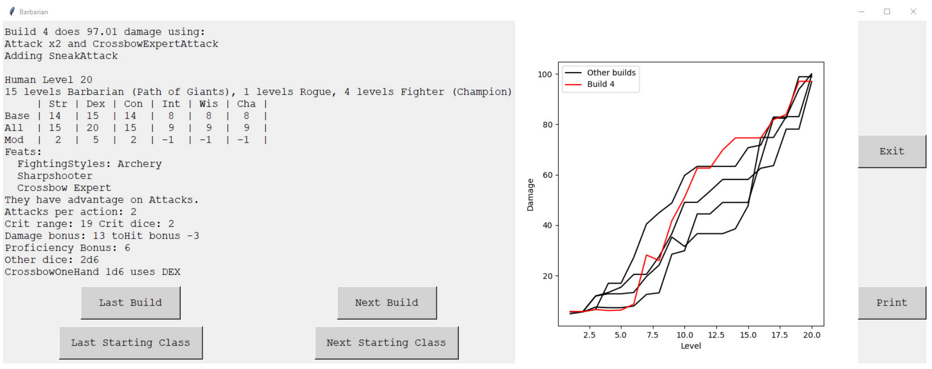
My personal setup used an AMD Ryzen 5600X and 32 GB of RAM.
The best settings I could use under 10 minutes were: 2500 max characters, cleansing every 4 levels, keeping only characters with unique damage values.
This took 7m13s.
These results seemed to have no noticable improvement over using 1500 characters or only 3 level cleanses, so I would recommend sticking to those, as they are much faster without any information loss at high levels.

References

1. https://www.dndbeyond.com/forums/d-d-beyond-general/general-discussion/158756-the-martial-caster-divide
   Discussion on martial gap.
2. https://www.dndbeyond.com/sources/basic-rules
   D&D 5e is owned by Wizards of the Coast, I claim no affiliation or ownership over their content.
3. http://canihitthis.com/
   Used to validate math for hit chances.
4. https://rpgbot.net/dnd5/
   Used for picking good starting points. (This website gives advice on what Ability Scores are most important for different classes, what subclasses are good, etc).
5. https://rpgbot.net/dnd5/characters/fundamental_math/
   This was specifically used as a basis for determining enemy AC.

8 Other examples of builds

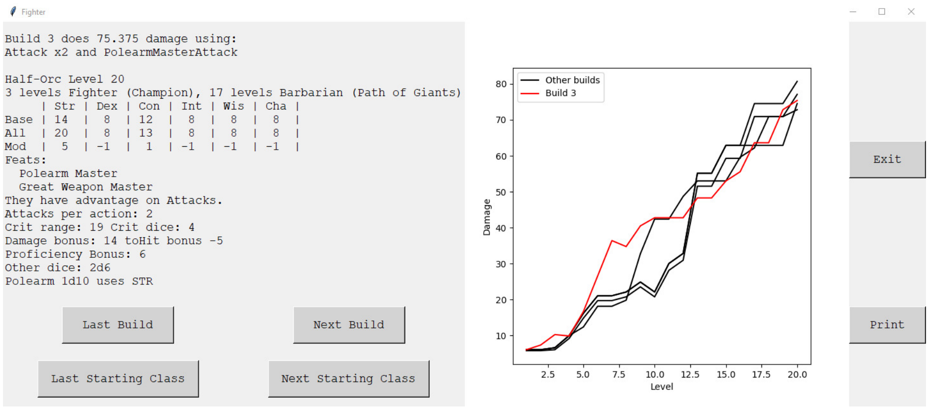
These results were achieved with a character limit of 1500, cleanses every 3 levels, keeping only characters with a unique amount of damage.
It took 2m28s.



```
Build 4 does 97.01 damage using:
Attack x2 and CrossbowExpertAttack
Adding SneakAttack

Human Level 20
15 levels Barbarian (Path of Giants), 1 levels Rogue, 4 levels Fighter (Champion)
     | Str | Dex | Con | Int | Wis | Cha |
Base | 14  | 15  | 14  |  8  |  8  |  8  |
All  | 15  | 20  | 15  |  9  |  9  |  9  |
Mod  |  2  |  5  |  2  | -1  | -1  | -1  |
Feats:
  FightingStyles: Archery
  Sharpshooter
  Crossbow Expert
They have advantage on Attacks.
Attacks per action: 2
Crit range: 19 Crit dice: 2
Damage bonus: 13 toHit bonus -3
Proficiency Bonus: 6
Other dice: 2d6
CrossbowOneHand 1d6 uses DEX
```

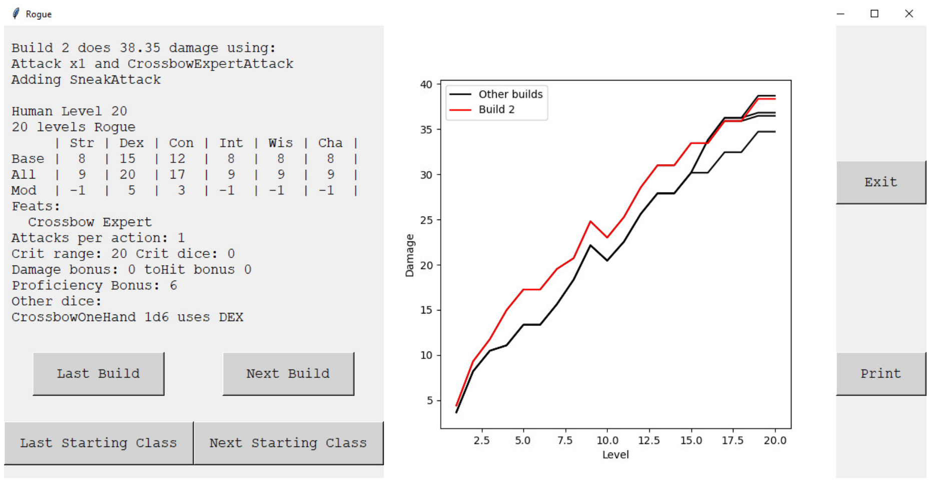
Fig. 2. Another Barbarian example that exhibits a less smoothe progression.



```
Build 3 does 75.375 damage using:
Attack x2 and PolearmMasterAttack

Half-Orc Level 20
3 levels Fighter (Champion), 17 levels Barbarian (Path of Giants)
     | Str | Dex | Con | Int | Wis | Cha |
Base | 14  |  8  | 12  |  8  |  8  |  8  |
All  | 20  |  8  | 13  |  8  |  8  |  8  |
Mod  |  5  | -1  |  1  | -1  | -1  | -1  |
Feats:
  Polearm Master
  Great Weapon Master
They have advantage on Attacks.
Attacks per action: 2
Crit range: 19 Crit dice: 4
Damage bonus: 14 toHit bonus -5
Proficiency Bonus: 6
Other dice: 2d6
Polearm 1d10 uses STR
```

Fig. 3. This Fighter got caught in the trap of becoming a Barbarian, as mentioned in the conclusion.



```
Build 2 does 38.35 damage using:
Attack x1 and CrossbowExpertAttack
Adding SneakAttack

Human Level 20
20 levels Rogue
     | Str | Dex | Con | Int | Wis | Cha |
Base |  8  | 15  | 12  |  8  |  8  |  8  |
All  |  9  | 20  | 17  |  9  |  9  |  9  |
Mod  | -1  |  5  |  3  | -1  | -1  | -1  |
Feats:
  Crossbow Expert
Attacks per action: 1
Crit range: 20 Crit dice: 0
Damage bonus: 0 toHit bonus 0
Proficiency Bonus: 6
Other dice:
CrossbowOneHand 1d6 uses DEX
```

Fig. 4. Here a Rogue build, this will be the only one I include, as they all look like this.