Pokémon, Possibilities and P vs NP

How to Build a Mathematically Optimal Pokémon Team*

Connor Donoghue

April 2026

Contents

1 Introduction **2**

2 The Question **2**

3 The (Very) Short Answer **2**

4 The Long Answer **2**
4.1 Checking Our Answer 2
4.2 Iterating Through Teams 4
4.3 When Can We Solve It 5
4.4 It's Easy to Check—Is It Easy to Solve? 6

5 P vs NP **6**
5.1 The History of P vs NP 6
5.2 Our Problem: NP or not NP 8
5.3 What If I Solve It? 9

*All work produced in this paper is original and my own, including mathematical proofs and calculations.

[0] Image source: https://champions.pokemon.com/en-us/

1 Introduction

Imagine you are a child, curating the perfect army of Pokémon to battle against your friends. Without realising it, you have just stumbled upon one of the most famous unsolved mathematics problems of the present era—and if you can solve it, you may be entitled to $1,000,000.

2 The Question

In the Pokémon VGC (Video Game Championship) format, players engage in one-versus-one doubles matches in which their Pokémon use moves to deal damage to their opponent's Pokémon until either player resigns or either player's full selection of Pokémon has been knocked out. Each player brings a team of six Pokémon to the battle but can only select four members of their team per round to use, usually chosen based on a variety of factors, e.g. preferential type-matchups or different play-styles.

Some Pokémon species are more viable due to, what I'm going to call, their 'species properties': inherent strengths and weaknesses that players generally cannot alter. These include typing, base stats, move pool and abilities. Players can customise each individual Pokémon beyond their fixed species properties by choosing four moves from the species' move pool, an ability (if a species has multiple), a held item, nature, and stat point[1] spread[2]. I will refer to these customisable properties as 'player properties'.

The question we have is simple: how can we arrange a team of six Pokémon species (out of 1025[3][4]), and how can we customise their player properties, such that we have created the mathematically optimal team.

3 The (Very) Short Answer

With our current mathematical system, we might never know the answer to our question. Checking a team's performance is easy, but generating the perfect team from nothing is much harder.

4 The Long Answer

4.1 Checking Our Answer

In order to solve our problem, it is best to first find out how we can check if our solution is correct once we find it. Think of it as creating a machine with many

[1] Formerly EVs (Effort Values).

[2] Prior to the release of Pokémon Champions, April 8th 2026, players could also choose an IV (Individual Value) spread.

[3] As of April 2026, per Pokémon national dex.

[4] Figure does not include alternate forms.

programs, whereby one can input a VGC team, and it will assign it a score. If we can find the team with the highest score and verify such score is correct, we have thus found the optimal team.

Competitive teams need balanced offense and defense, which can be checked in constant time via table lookups.

```
types = ['normal','fire','water','grass','electric','ice','fighting','poison',
         'ground','flying','psychic','bug','rock','ghost','dragon','dark','steel','fairy']
typeindex = {t: i for i, t in enumerate(types)}
covered, coveredall = [], True
coverage = {
    'normal':  [1,1,1,1,1,1,1,1,1,1,1,1,0.5,0,1,1,0.5,1],
    'fire':    [1,0.5,0.5,2,1,2,1,1,1,1,1,2,0.5,1,0.5,1,2,1],
    'water':   [1,2,0.5,0.5,1,1,1,1,2,1,1,1,2,1,0.5,1,1,1],
    'grass':   [1,0.5,2,0.5,1,1,1,0.5,2,0.5,1,0.5,2,1,0.5,1,0.5,1],
    'electric':[1,1,2,0.5,0.5,1,1,1,0,2,1,1,1,1,0.5,1,1,1],
    'ice':     [1,0.5,0.5,2,1,0.5,1,1,2,2,1,1,1,1,2,1,0.5,1],
    'fighting':[2,1,1,1,1,2,1,0.5,1,0.5,0.5,0.5,2,0,1,2,2,0.5],
    'poison':  [1,1,1,2,1,1,1,0.5,0.5,1,1,1,1,0.5,0.5,1,1,0,2],
    'ground':  [1,2,1,0.5,2,1,1,2,1,0,1,0.5,2,1,1,1,2,1],
    'flying':  [1,1,1,2,0.5,1,2,1,1,1,1,2,0.5,1,1,1,0.5,1],
    'psychic': [1,1,1,1,1,1,2,2,1,1,0.5,1,1,1,1,1,0,0.5,1],
    'bug':     [1,0.5,1,2,1,1,0.5,0.5,1,0.5,2,1,1,0.5,1,2,0.5,0.5],
    'rock':    [1,2,1,1,1,2,0.5,1,0.5,2,1,2,1,1,1,1,0.5,1],
    'ghost':   [0,1,1,1,1,1,1,1,1,1,2,1,1,2,1,0.5,1,1],
    'dragon':  [1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,1,0.5,0],
    'dark':    [1,1,1,1,1,1,0.5,1,1,1,2,1,1,2,1,0.5,1,0.5],
    'steel':   [1,0.5,0.5,1,0.5,2,1,1,1,1,1,1,2,1,1,1,0.5,2],
    'fairy':   [1,0.5,1,1,1,1,2,0.5,1,1,1,1,1,1,2,2,0.5,1]
}
for _ in range(0,6):
    type1 = input()
    type2 = input()
    if type2 == '':
        for i, list in coverage.items():
            if list[typeindex[type1]] < 1:
                covered.append(i)
    else:
        for i, list in coverage.items():
            if list[typeindex[type1]]*list[typeindex[type2]] < 1:
                covered.append(i)
    print("----------------")
for type in types:
    if not type in covered:
        coveredall = False
        print(type)
print(coveredall)
```

Figure 1: My Python Program

This program has complexity $O(1)$ and finds unresisted types among a team of six. A slightly altered variant can be used to find a team's offensive coverage given their damage-dealing moves' types, once again, in constant time.

We can implement many kinds of these programs in our machine (e.g. speed priority rankings, weakness overlap scores, team versatility scores, etc) to produce an accurate team score. This machine would always be able to output scores in a finite, short amount of time, due to programs' complexity being polynomial, linear or constant in total species or types. Thus, given a solution's score, we could 'easily' verify if that score is correct.

4.2 Iterating Through Teams

Let's try and simplify our problem.
We need to choose 6 distinct species out of 1025 total:

$$\text{Unique Species Team } = \binom{1025}{6} = \boxed{1.587 \times 10^{15}}$$

Each Pokémon can hold one item out of a total of 146[5] [6]. This means that the total amount of permutations for held items across a team is at least:

$${}^{146}P_6 = \boxed{8.728 \times 10^{12}}$$

Moreover, each Pokémon has 66 stat points to assign to its 6 stats, each between 0 and 32.

$$x_1 + x_2 + x_3 + x_4 + x_5 + x_6 = 66$$

$$\text{Where } 0 \leq x_i \leq 32$$

Some strategies may leave unused points, such as to avoid speed ties[7], so we introduce a slack variable s_1 such that:

$$x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + s_1 = 66$$

$$\text{Where } 0 \leq x_i \leq 32 \text{ and } 0 \leq s_1$$

First, we use the stars and bars formula to calculate the total number of ways to assign points to stats, ignoring the limit of 32 points per stat:

$$\binom{66 + 7 - 1}{7 - 1} = \binom{72}{6}$$

Then, we find the number of combinations in which a particular stat, say x_1, exceeds 32, and multiply it by six to get the number of combinations for which any single stat is greater than 32. Set:

$$y_1 = x_1 - 33$$

$$y_1 + x_2 + x_3 + x_4 + x_5 + x_6 + s_1 = 66 - 33 = 33$$

And find non-negative integer solutions:

$$\binom{33 + 7 - 1}{7 - 1} = \binom{39}{6}$$

[5] Source: https://www.serebii.net/itemdex/list/holditem.shtml

[6] That is, excluding trivial items such as those meant for use in showcases, or items specific to individual Pokémon or a subset of them, whose description begins, "To be held by a Pokémon"

[7] Where 2 Pokémon with the very same speed stat must make a move - this is decided by a coin flip in the game's code, an RNG aspect many players try to avoid.

Multiply by six:

$$6 \cdot \binom{39}{6}$$

Then we consider cases where 2 stats are greater than 32. This can only happen in cases where both stats are assigned 33 points and all others have none:

$$\binom{6}{2} = 15$$

Subtracting cases where a stat is over-assigned from the total non-negative integer combinations, and adding cases where two stats are over-assigned (to avoid double subtraction), we get:

$$\text{Unique SP spreads} = \binom{72}{6} - 6\binom{39}{6} + 15 = 1.367 \times 10^8$$

Finally, to get the number of unique stat point spreads across a team of six Pokémon, we simply find:

$$((1.367 \times 10^8)^6) = \boxed{6.515 \times 10^{48}}$$

Each Pokémon can also have one of 25 natures, giving us:

$$\text{Unique Team Nature Combinations} = 25^6 = \boxed{2.441 \times 10^8}$$

We can multiply all of our boxed results out to find the number of combinations of species, held items, SP spreads and natures across a team of six Pokémon to be at least:

$$\boxed{1.825 \times 10^{85}}$$

Remember, this excludes moveset and ability combinations—which are arguably more important than some of the previous aspects—since the total learnable moves and abilities is different for each individual species. Either way, our result is a hugely large number, which is currently impossible to compute.

4.3 When Can We Solve It

Reducing our problem to lower the number of combinations is certainly possible. For instance, not all SP totals are useful—some strategies want to lower their speed so may use less than around 44 points, whilst typical teams will use at least 62 points per Pokémon[8].

However, this does almost nothing towards helping us brute-force the problem. Reducing candidate teams to one in a million, it would still take El Capitan, the world's most powerful supercomputer[9], over 3.49×10^{79} years to

[8] 62 rather than 64 as some players may leave one or two points out for matchups against very specific Pokémon

[9] As of April 2026. Source: https://top500.org/lists/top500/2025/11/

compute—even when rounding up El Capitan's speed from 1.8 Exaflops/sec to 2 and assuming single-step team effectiveness checks (it would take hundreds). For reference, if you took one step for every lifespan of the universe and only added a millilitre of water to the oceans every time you circumnavigated Earth, then every time you filled the oceans you took a step around the Earth in an alternate universe, moving a millimetre into space every circumnavigation, you would have travelled the distance to the moon over one-hundred million times.

We're going to have to use a different method if we want our answer to predate the heat death of the universe: the problem is... no one has discovered one yet.

4.4 It's Easy to Check—Is It Easy to Solve?

As we saw earlier, verifying a team's score is fast. Despite this, the sheer number of adjustable dimensions when team-building makes an exhaustive search infeasible. Finding the perfect team therefore scales exponentially, and no known shortcut exists besides reducing the search space and brute-forcing—very characteristic of an NP-hard or NP-complete problem.

"Just because a problem's solution can be verified in polynomial time, does that mean it can be solved in polynomial time?"

That's the $1,000,000 question.

5 P vs NP

5.1 The History of P vs NP

The year is 1955, and John Nash, who would go on to be awarded both Nobel and Abel prizes for developments in unrelated areas, is writing a letter to the USA's NSA[10], suspecting that cracking a code of great complexity would take an exponentially longer length of time as the length of the key increases[11]. One year later, Kurt Gödel would write to John von Neumann (founder of modern computer architecture and widely considered an all-time polymath genius), questioning whether proving a conjecture is achievable in quadratic or linear time and suggesting that, if so, proof-generation could be automated[12].

Fast-forward to the late 1960s and early 1970s, and computer scientists are eagerly classifying their problem-solving programs into 'fast' and 'slow'. They have lots of fast algorithms—such as list-sorting and matrix multiplication—but

[10] National Security Agency.

[11] Source: https://bit.ly/NASA-John-Nash

[12] https://bit.ly/ecommons-pvsnp

also a handful of impractically slow algorithms, such as sudoku solvers[13] [14].

It was only 1971 when Stephen Cook published, "The Complexity of Theorem-Proving Procedures", that a formal definition of P vs NP was introduced[15] (Leonid Levin also independently introduced the problem in 1973[16]). Nowadays, algorithms can be sorted into a variety of subclasses, depending on their complexity. You can see some of them in the below figure[17].

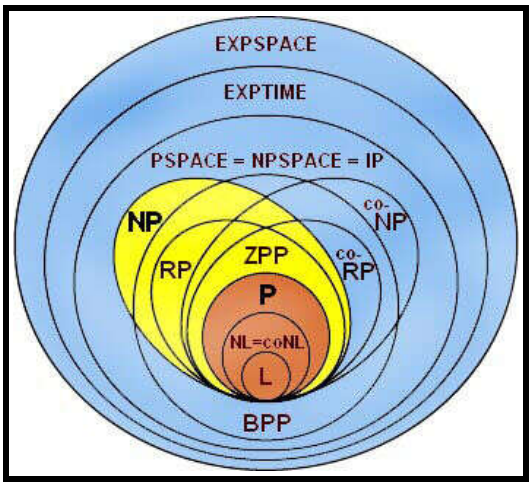

Figure 2: Selection of Major Complexity Classes

P, or polynomial, problems can be entirely solved in polynomial time[18], also including linear and constant time problems. NP problems have solutions whose *correctness* can be verified in polynomial time, for example, given a sudoku puzzle, it is easy to tell if the puzzle has been correctly completed: check if every row, column, diagonal and 3x3 square has integers one to nine once each. P is thus a subclass of NP, because if you can solve a problem quickly, you can certainly verify it quickly. NP-complete problems are the hardest problems within NP, for which we don't currently have an algorithm that can solve their worst cases in polynomial time. They are still within NP, however. NP-hard problems are those that are at least as hard as NP-complete problems, but do not necessarily have easily verifiable solutions[19].

[13] Many of the problems first thought to be slow turned out, with time, to be faster, the most famous example being the discrete Fourier transform (DFT), however just as many seemed unable to be simplified.

[14] For further reading, see Cooley and Tukey's introduction of the fast Fourier transform (FFT): https://bit.ly/fast-fourier-transform

[15] Source: https://bit.ly/4soPyT4

[16] Source: https://bit.ly/3PV07zD

[17] Source: https://www.jeremykun.com/2012/02/29/other-complexity-classes/

[18] A time length proportionate to n^x, denoted $O(n^x)$.

[19] That is, not necessarily NP, so their solution verification may not be in the P class.

The dilemma is: despite seeming so distinct, it has not yet been proved whether or not P = NP. Yet, if only a single NP-complete problem can be solved in polynomial time, then every single NP problem can be reduced to it in polynomial time and in turn solved in polynomial time—resulting in P = NP.

The irony is that, as Gödel hinted in 1956, finding a proof for a conjecture is itself an NP-hard problem.

5.2 Our Problem: NP or not NP

So, we know our Poké-problem is extremely difficult to compute, and conjecture that it is NP-hard, maybe even NP-complete. The question is, can we prove it?

Proof.

Theorem 1. *Building an optimal Pokémon team is an NP problem.*

Let the candidate solution be a team $T = (X_1, X_2, \ldots, X_6)$, where each Pokémon $X_i \in D$.

Each Pokémon in D is fully specified by:

$$D = \{(p, h, s, n, a, m_1, m_2, m_3, m_4)\}$$

where p is the species, $h \in H$ is the held item, $s \in S$ is the stat spread, $n \in N$ is the nature, $a \in A$ is the ability, and $m_1, \ldots, m_4 \in M$ are the moves.

Constraints for a valid team include:

1. No two X_i share the same species p.
2. No two X_i share the same held item h.
3. Each Pokémon X_i can only have moves and abilities allowed by its species' domain D_i.

To verify a candidate team T, we check the above constraints and compute the team's score. Since there are only six variables, checking all constraints and computing the score has polynomial complexity relative to $|D|, |H|, |S|, |N|, |A|, |M|$. Therefore, the problem is in NP.

Theorem 2. *Building an optimal Pokémon team is an NP-hard problem.*

We reduce from the general Constraint Satisfaction Problem (CSP), which is known to be NP-complete. An instance consists of:

1. Variables $Y_1, \ldots, Y_n$, and domains D_1, D_i where $Y_i \in D_i$.
2. Constraints $C_1, \ldots, C_m$ restricting allowed variable assignment combinations.

Reduction Construction: Given a CSP instance, construct a team problem as follows:

1. Let each Pokémon X_i correspond to a CSP variable.
2. Let the Pokémon $X_i \in D$ correspond to the domain values of the variables, extended with valid held items H, stat spreads S, natures N, abilities A, and moves M.
3. Encode CSP constraints as Pokémon constraints:
   (a) No two $X_i.p$ or $X_i.h$ are equal.
   (b) Each species p_i can only learn moves in its move pool D_i.
   (c) More complex CSP constraints can be encoded using additional properties which differ per Pokémon (e.g., abilities, natures, or type coverage).

Correctness of the reduction:

- Any CSP solution corresponds to a valid Pokémon team satisfying all constraints.
- Any valid Pokémon team satisfying all constraints corresponds to a CSP solution (assign each X_i to the CSP variable's domain value it represents).

Since the reduction can be performed in polynomial time, and CSP is NP-complete, the Pokémon team problem is NP-hard.

Conclusion: The Pokémon team-building problem is in NP and NP-hard; hence, it is NP-complete. □

5.3 What If I Solve It?

Solving P vs NP would have a profound impact on the world, changing the way we live.

If you were to find P = NP, all the problems previously considered 'hard' would no longer take supercomputers months or years to solve, now just seconds or hours. This would result in massive advances in drug discovery, protein folding (and thus cancer curing), logistics, AI and more. Revolutionarily if not controversially, algorithms could create 'perfect' literature, music and art instantly. However, this would come at the cost of an immediate collapse in cryptography, with current encryption, which relies on some problems being 'hard', becoming easily decryptable.

If you proved P ≠ NP, you would stabilise cryptography by confirming what many already believe—some problems are inherently 'hard', and would prevent the possibility that AI-powered vulnerability-exploitation and fake news become even easier to generate and less detectable. Unfortunately, we would miss out on all of the benefits of P = NP, meaning some problems remain impossible to solve efficiently.

Either way, you would earn the $1,000,000 Clay prize, worldwide recognition and intellectual 'immortality'.