At the time of writing, the next open case for model enumeration in the queen domination problem is for n=19. Our goal is to address this case.

Please note that while model enumeration for n=19 remains open, the value of γ(19) is known. Specifically, γ(19) is reported to be 10 in the literature, based on exhaustive search methods. However, since this value was determined using advanced search techniques and complex code, which might contain errors, there is some uncertainty about whether γ(19) is truly 10. This uncertainty is particularly relevant if we plan to use this information to tackle the next open case.

The literature provides a theorem stating that γ(19) is at least 9. Additionally, since a solution of size 10 has been found for a 19×19 chessboard, and verifying that this solution is a domination set is straightforward, we can conclude that γ(19) is either 9 or 10.

Here, one natural way to go through is as follows:

1- Assume γ(19) is 9 and encode the problem into SAT.
2- Use a SAT solver to determine if the instance is SAT or UNSAT.
3- If the instance is SAT, perform model enumeration.
4- If the instance is UNSAT, then γ(19) must be 10, and model enumeration should be performed under that assumption.

Given the computational intensity of each SAT solver call for a 19×19 chessboard, our goal is to minimize the number of such calls. To optimize the above approach, we propose starting with the assumption that γ(19) is 10. In other words, while we should not blindly accept the results reported in the literature—due to potential bugs in their code and lack of independent verification—it is reasonable to initially assume that γ(19) is 10. This assumption is based on the premise that, until proven otherwise, the reported value might be correct and serves as a plausible starting point for our analysis.

If γ(19)=10 proves to be correct, we can avoid the additional computation required if we were to test γ(19)=9 first. Should our assumption be incorrect and γ(19) turn out to be 9, we can still determine all satisfying solutions of size 9 with relatively minimal additional computation. This strategy helps us manage our resources efficiently while addressing the problem.

The opportunistic approach we propose is as follows:

1- Assume γ(19) is 10 and encode the problem into SAT.
2- Since this instance is known to be satisfiable, perform model enumeration on it.
3- Check each solution for redundancy (i.e., check if removing a queen still maintains a domination set).

4- If no solutions are redundant, then γ(19) is 10. If redundant solutions are found, then γ(19) is 9, and removing redundant queens from all redundant solutions will yield the model enumeration for γ(19)=9.

In the n_19_gamma_10_stat_results.json file, cubes generated from a single call (referred to as round 0 of cube generation) to the cube generator march_cu are stored. For a large instance like model enumeration with n=19, we observed that some cubes consumed more resources—such as time and sapce—than acceptable, while many others performed as expected. For cubes that did not meet our expectations, we recursively split them further in subsequent rounds.

Unfortunately, we did not save the cubes generated in these subsequent rounds but only recorded some statistics on their performance. This was a mistake on my part when designing the experiment, as I did not consider saving all relevant information. As a result, some details are missing, although the remaining information is still sufficient to report the solver's performance when solving the instance.

Additionally, storing and sharing resolution proofs requires substantial storage. Therefore, we followed a common practice in the literature: after verifying the SAT solver's output for the instance, we deleted the proof to conserve memory.

There were a few cubes for which the SAT solver's output could not be verified or was incorrect. Assuming that any bugs were likely in the SAT solver rather than the verification process, we reran the SAT solver multiple times with different configurations and attempted to verify the instances. This approach successfully resolved the issues, leading to correct solutions and verification of the cubes, albeit with increased computational cost. Unfortunately, we did not track or mark these specific cubes, nor did we record the number of retries or the additional computations required. The statistics provided pertain only to the final successful run.

The n_19_gamma_10_stat_results.json file contains several key pieces of information, currently, organized under the following keys: 'models', 'model_enum_formula', and 'cubes'.

- 'models': This section provides access to the models found during model enumeration for n=19 and gamma=10.
- 'model_enum_formula': This key contains the SAT formula for n=19 and gamma=10, concatenated with the negation of the models found. This combined formula is what

we pass to the solver to verify that no other solutions exist beyond those stored in models.

- 'cubes': This section stores the cubes generated from a single call to the cube generator march_cu, referred to as round 0 of cube generation.

Each cube within the 'cubes' section is identified by a unique cube ID. If you load this file into Python and store it in a variable named data, each cube ID in data['cubes'] contains a dictionary with the following information:

- data['cubes'][cube_id]['cube']: A list of unit clauses representing the cube itself.
- data['cubes'][cube_id]['DRAT_Proof_Size_Bytes']: A list of proof sizes. If empty, this may indicate that the instance has not been solved yet or that we used on-the-fly verification instead of writing the proof to a file, so there is no proof for this cube yet. If there is exactly one item in the list, it means the corresponding cube was solved without further splitting in subsequent rounds. Otherwise, it means that the cube was split further (though the exact subcubes were not saved), and each item in the list corresponds to the proof size of each subcube.
- data['cubes'][cube_id]['Total_Process_Time_Seconds']: A list of process times in seconds for the cube itself (if it was resolved), and for the subcubes if it was further split; otherwise, the list is empty if it has not been resolved yet.
- data['cubes'][cube_id]['Total_Real_Time_Seconds']: A list of real time spent resolving the cube in seconds, and for the subcubes if it was further split; otherwise, the list is empty if it has not been resolved yet.
- data['cubes'][cube_id]['Maximum_Resident_Set_Size_MB']: A list of the maximum resident set sizes in megabytes for resolving the cube, and for the subcubes if it was further split; otherwise, the list is empty if it has not been resolved yet.
- data['cubes'][cube_id]['Verification_Time_Seconds']: A list of real time spent verifying the cube in seconds, and for the subcubes if it was further split; otherwise, the list is empty if it has not been verified yet.

TODO:

1. I missed recording the time it took to perform model enumeration for this case using Unidom. There is probably some report available in Compute Canada that I should try to locate and add to this report. However, as a reminder, it took a few hours and, in the worst case, less than a day to solve the instance, which is not significant when considering all the time spent verifying the results.

2. I should also add the time spent on cube generation and subsequent cube generation.
3. There remain some cubes that have not been resolved and verified yet. I need to complete this.