Mini Project 5 – Monster Diagnosis

Michael Lukacsko
mlukacsko3@gatech.edu

1 AGENT IMPLEMENTATION

The monster diagnosing agent implemented for this assignment operates by performing a combinatorial search to identify the smallest subset of diseases that can explain a patient's symptoms. Using Python's itertools.combinations, the agent systematically generates all possible combinations of diseases, starting with single diseases and progressing to larger sets until it finds the minimal subset that accounts for the observed symptoms. Once a combination is generated, the agent employs a helper function, matches_patient_symptoms, which uses a constraint satisfaction approach to determine if the selected diseases collectively match the patient's symptoms.

Within the matches_patient_symptoms function, the agent utilizes defaultdict from Python's collections library to track cumulative vitamin effects for each disease combination. For each disease, it iterates through each vitamin effect where + increases the vitamin's count by 1 (indicating an elevated effect) and - decreases the count by 1 (indicating a reduced effect). Vitamins marked as 0 (normal) remain unaffected. After applying the effects from all diseases in the combination, the agent evaluates if the cumulative effects match the patient's symptom profile. A positive net effect for a vitamin must correspond to a + symptom, a negative net effect to a -, and a zero net effect to a 0. If all vitamins match, the function confirms that the disease combination is a valid explanation for the patient's symptoms. By combining systematic search and constraint satisfaction techniques, the agent effectively narrows down, and returns, the smallest set of diseases to explain the symptoms.

2 AGENT PERFORMANCE

After implementing the agent, it successfully passed all local tests, demonstrating its initial accuracy and functionality. Moreover, the autograder results confirm the agent's strong performance across all test cases, achieving both correct answers and optimal solutions for all the 20 diagnosis tasks performed by the autograder. This level of success indicates that the agent is highly effective for the scope of cases tested.

3 AGENT EFFICIENCY

The performance of my agent demonstrates initial efficiency when handling smaller disease sets, such as the provided sets 1 and 2, which consist of fewer diseases and simpler symptom combinations. In these tests, the agent consistently produces results within a very short time frame, often close to 0 seconds or a few milliseconds per disease-patient pair. This consistency indicates that the agent is highly efficient for smaller datasets, managing simpler conditions with minimal computational overhead.

In addition to the 2 test cases provided, 7 additional test cases were created to evaluate the agent's performance with increasingly complex disease sets. The 2 test cases provided included 5 diseases, while the additional tests expanded the scope to include disease sets with 8 to 10 total diseases. This increase in complexity was designed to test the agent's ability to handle larger and more diverse datasets. As the tests progressed to these larger sets, specifically test sets 7, 8, and 9, the performance noticeably degraded. As can be seen in figure 1 below, timing results for these larger sets show execution times increasing to between 0.05 and 0.15 seconds per disease-patient pair, indicating that the agent requires significantly more time to process each case as the number of diseases and symptom combinations grow. This decrease in performance reflects the additional computational burden imposed by the increased size and complexity of the disease sets, requiring the agent to evaluate a greater number of patterns and symptom matches for each case.

The time complexity of the agent can be expressed as O(n×m), where n represents the number of diseases and m the number of symptoms for each disease. As n grows, the agent's runtime scales linearly with it, assuming each symptom comparison operation is constant time. However, the presence of multiple patients and the structure of each test case introduce further complexity, which exacerbates runtime growth as the dataset expands. Figure 1 below highlights this increase, with clear spikes beginning around test cases 60 and beyond, correlating with the larger and more complex disease sets.

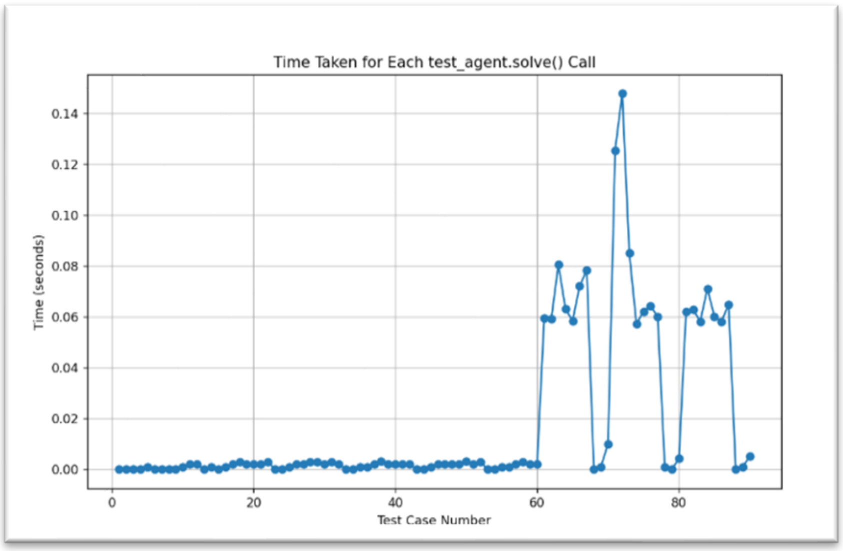

Figure 1 — Time taken for test sets. 10 total diseases and 10 total patients.

4 YOU ARE NOT SO CLEVER, AGENT!

In its current form, my agent is more exhaustive than clever. As such, it employs a straightforward, systematic approach rather than relying on clever or optimized strategies. The agent simply compares the symptoms of each patient with the symptom profiles of each disease in the set. As figure 1 above shows, this approach is effective for smaller datasets, but it lacks any heuristic-based shortcuts or optimized search techniques that could reduce the number of comparisons as the dataset grows.

One potential improvement, which would make this agent a bit cleverer, could be to implement a filtering mechanism to eliminate diseases that clearly do not match early in the comparison process, thereby reducing the number of full comparisons required. For instance, if a disease profile contains a unique symptom signature that doesn't align with the patient's symptoms, the agent could exclude that disease immediately without needing a full comparison.

5 AGENT VS HUMAN

This agent's implementation is more exhaustive and systematic than a human approach would be for the same problem. A human, when faced with diagnosing based on a large set of symptoms, would typically rely on heuristics, intuition, and process of elimination. For example, a human might first look for distinctive or "key" symptoms that point strongly to certain diseases, quickly narrowing down the possibilities without going through every symptom one by one. Additionally, humans often rely on prior experience, which allows them to recognize patterns or make educated guesses based on partial information, something this agent currently doesn't do.

In contrast to a human approach, the agent follows a brute-force approach, comparing every symptom against every possible disease symptom systematically, regardless of how distinctive or common symptoms are. This approach is thorough but inefficient, especially as the number of diseases and symptoms grows.

Lastly, humans also have the ability to adaptively switch strategies if an initial approach seems inefficient. For instance, if a human notices that certain symptom combinations are rare or indicative of specific diseases, they might prioritize checking those symptoms first. This adaptive flexibility is something my agent lacks; it doesn't dynamically adjust its comparison order based on prior comparisons or observed patterns.