PROJECT PROPOSAL

Artificial Intelligence

BS(CS)-5B

AIR UNIVERSITY ISLAMABAD

Group Members

Name	Registration-ID
Afia Aziz	231561
Zumer Dhillun	231597
Zoya Azad	231579

Submitted to: Ma'am Hareem Kibriya

System Design Document: Medi-Match

AI-Integrated Hospital Resource Optimization & Emergency Triage Platform

1. Introduction

1.1 Purpose

The purpose of this document is to provide a detailed technical blueprint for the Medi-Match system. It outlines the architectural decisions, the algorithmic methodologies (Fuzzy Logic, Rule-Based Expert Systems, and Genetic Algorithms), and the data structures that enable the platform to optimize hospital resources and automate emergency triage.

1.2 System Overview

Medi-Match is a dual-purpose medical management system. It provides an administrative interface for scheduling hospital resources (doctors, patients, and beds) and a clinical interface for the immediate triage and specialist recommendation of emergency arrivals. The system bridges the gap between high-level resource management and real-time clinical decision support.

1.3 Design Goals

- **Decoupled Intelligence:** Separation of the user interface (C#) from the optimization logic (Python).
- **Algorithmic Flexibility:** Support for both rapid heuristic scheduling and deep evolutionary optimization.
- **Clinical Accuracy:** High-fidelity symptom-to-specialist mapping based on medical priority.
- **Visual Evidence:** Real-time feedback for administrators to verify AI performance.

2. System Architecture

The Medi-Match architecture follows a **Layered Hybrid Design**, utilizing the strengths of the .NET ecosystem for the frontend and the Python data science ecosystem for the backend.

2.1 The UI Layer (C# / WPF / XAML)

The presentation layer is built using Windows Presentation Foundation (WPF).

- **Responsibility:** Handling user input, managing application state, rendering data grids, and displaying generated reports.
- **User Controls:** The system utilizes modular UserControls (e.g., EmergencyTriage.xaml) to allow for an extensible navigation system within a single main window.

2.2 The Intelligence Layer (Python 3.x)

The "brain" of the system consists of multiple Python engines.

- **Heuristic Engine (scheduler.py):** Fast, rule-based logic for immediate assignments.
- **Optimization Engine (scheduler_ga.py):** Uses meta-heuristic search to solve complex allocation problems.
- **Triage Engine (triage_calculator.py):** A rule-based expert system for clinical urgency calculation.

2.3 The Communication Layer (JSON/CSV)

Data exchange between the UI and the AI engine is handled through standardized JSON files.

- **input.json:** Sent from C# to Python containing resource parameters.
- **output.json:** Sent from Python back to C# containing the optimized schedule.
- **metrics.csv:** Statistical data used to populate the Graph and Metrics tables.

3. Algorithmic Methodology

Medi-Match does not rely on a single algorithm; instead, it uses an **Ensemble Approach** to handle different types of data complexity.

3.1 Rule-Based Expert System (Mapping & Triage)

The system uses a Knowledge Base to perform "Hard Constraints" matching.

- **Methodology:** The system maintains a dictionary of medical specialties mapped to specific conditions (e.g., Neurology: ["Stroke", "Head Injury"]).
- **Logic:** When a patient presents a disease, the system queries the knowledge base. If an exact match is found, the patient is flagged as a "Perfect Match." If no specialist is present, the system triggers a "Referral Needed" status.
- **Triage Rules:** Symptoms are assigned weights based on clinical severity. These weights are aggregated to form a raw score, which is then mapped to clinical priorities (Critical, Urgent, Routine).

3.2 Fuzzy Logic Inference

Urgency in a hospital is rarely binary (0 or 1). Medi-Match utilizes **Fuzzy Logic** to handle the "shades of grey" in patient severity.

- **Fuzzification:** The calculate_fuzzy_score function takes an integer urgency (1–10) and applies a linear membership function to map it to a value in the range [0.0, 1.0].

- **Inference:** This fuzzy score acts as a multiplier in the doctor assignment logic. A higher fuzzy score increases the "weight" of a patient, ensuring they "pull" a more senior or specialized doctor toward them in the scoring matrix.

3.3 Genetic Algorithm (GA) Methodology

To solve the **NP-Hard** problem of global resource allocation, the system uses an evolutionary strategy.

1. **Representation:** Each individual (chromosome) in the population is a list where the index represents a patient and the value represents a doctor index.
2. **Fitness Function:** Maximizes (Specialty Matches + Seniority Bonuses) - (Load Imbalance + Referral Penalties).
3. **Selection:** Uses **Tournament Selection** to ensure only the strongest schedules "survive" to the next generation.
4. **Crossover:** Uses **Two-Point Crossover** to combine parts of two successful schedules.
5. **Mutation:** A low-probability mutation rate (6%) introduces random changes to prevent the AI from getting stuck in a local optimum.

4. Detailed Component Design

4.1 Module 1: Hospital Resource Scheduler

This module is responsible for administrative data entry.

- **Dynamic UI Generation:** C# logic generates individual input cards for each doctor and patient based on the "Total Count" entered by the user.
- **Validation:** Ensures that the number of urgency values matches the number of patients before invoking the AI.

4.2 Module 2: The Emergency Triage Engine

This is the clinical core of the system.

- **Weighting Matrix:** Clinical symptoms like Unconscious (9.0) carry more weight than Dizziness (1.5).
- **Synergy Multiplier:** If a patient has multiple critical symptoms (e.g., Chest Pain + Difficulty Breathing), the AI applies an exponential multiplier, recognizing that the combination is more dangerous than the sum of its parts.
- **Specialist Recommendation:** Based on the highest-weighted symptom, the engine maps the patient to a specific specialist (e.g., Cardiologist, Trauma Surgeon).

4.3 Module 3: Visualization & Analytics

The Graph module fix ensures data is no longer static.

- **Data Source:** Python calculates the "Cumulative Quality" of the schedule at each patient assignment.
- **Visualization:** Matplotlib generates a PNG image showing the quality curve.
- **Bust-Caching Logic:** To ensure the website/UI displays the new graph every time, a timestamp-based naming or refreshing strategy is used.

5. Justification for AI Techniques

5.1 Why a Genetic Algorithm for Scheduling?

In a hospital with 10 doctors and 100 patients, the number of possible scheduling combinations is astronomical 10^{100} A standard "Brute Force" search would take years to compute. A Genetic Algorithm allows Medi-Match to explore the most promising "evolutionary paths" of a schedule, finding a 95% optimal solution in mere seconds.

5.2 Why Fuzzy Logic for Urgency?

Traditional logic says a "Level 7" urgency is strictly different from a "Level 8." However, in medicine, these levels often overlap. Fuzzy logic allows the system to treat urgency as a continuous spectrum, preventing "cliff-edge" decision-making where a slightly less urgent patient is completely ignored.

5.3 Why Rule-Based Logic for Triage?

Deep Learning (Neural Networks) are "Black Boxes"—it is often impossible to know why they made a certain decision. In medical triage, this is dangerous. A **Rule-Based System** is transparent. We can audit the weights and rules to see exactly why a patient was labeled "Critical," ensuring medical safety and accountability.

6. Data Design & Security

6.1 Data Structures (JSON Schema)

This structure ensures that the system is **Stateless**. The AI engine does not need to remember previous runs; it receives everything it needs in one JSON package, making the system highly reliable and easy to debug.

6.2 Report Generation

The system automates clinical documentation.

- **HTML Reporting:** Provides a professional, print-ready document for the patient's medical record.
- **Plain Text Backup:** Ensures compatibility with legacy hospital systems that cannot read HTML.

7. Performance and Scalability

Medi-Match is designed to scale:

- **Small Clinics:** The Heuristic Scheduler provides instant results for low patient volumes.
- **Major Hospitals:** The Genetic Algorithm can be tuned (by increasing population size and generations) to handle hundreds of patients across multiple departments.
- **Modular Specialists:** New specialties (e.g., Oncology, Radiology) can be added to the Python SPECIALTY_CONDITIONS dictionary without requiring any changes to the C# user interface or the core GA logic.

8. AI Knowledge Representation (Frames & Semantic Networks)

A. Frame-Based Representation

Frames represent the "Slots and Fillers" of our medical entities. These are the data structures our Python engine uses to process patients and doctors.

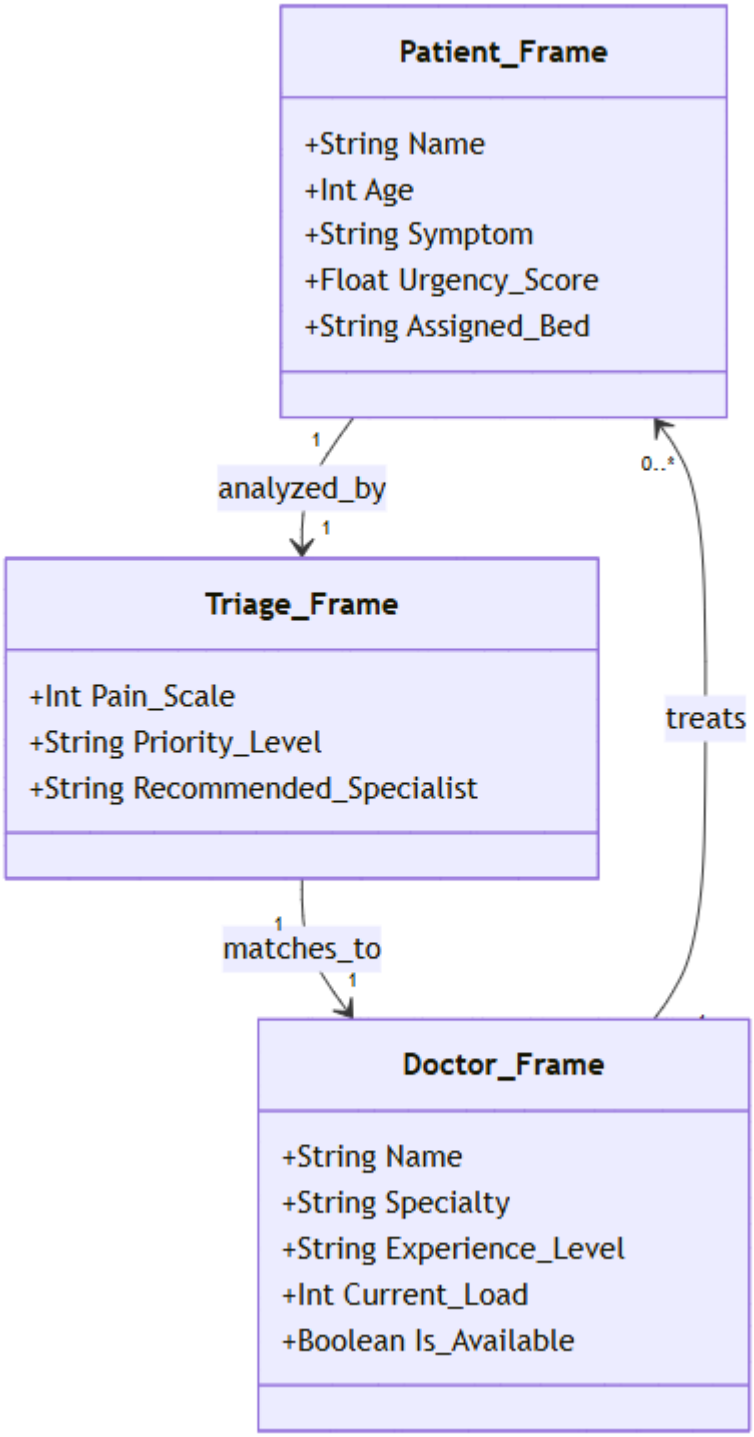

Figure 3.1: Frame-Based Knowledge Representation for Medi-Match

The internal data structure of the Medi-Match AI is organized into a series of interconnected **Frames**. Each frame consists of specific **Slots** (attributes) that the logic engine populates with **Fillers** (data) during execution.

- **Patient Frame:** Captures clinical and administrative data, integrating the fuzzy urgency score used for prioritization.
- **Triage Frame:** Represents the expert system's assessment of symptoms, mapping clinical inputs to priority levels and specialist requirements.
- **Doctor Frame:** Acts as the resource tracking frame, storing specialty constraints and dynamic load metrics used by the Genetic Algorithm to prevent resource exhaustion.

The associations between these frames (analyzed_by, matches_to, and treats) represent the logical flow of the scheduling and triage sub-systems, ensuring a cohesive data-driven approach to hospital management."

B. Semantic Network Diagram

The Semantic Network represents the relationships between medical concepts.

- **Is-A Relationship:** Pneumonia *is-a* General Disease.
- **Treats Relationship:** Cardiologist *treats* Chest Pain.
- **Assigned-To Relationship:** Patient *is assigned to* Bed.

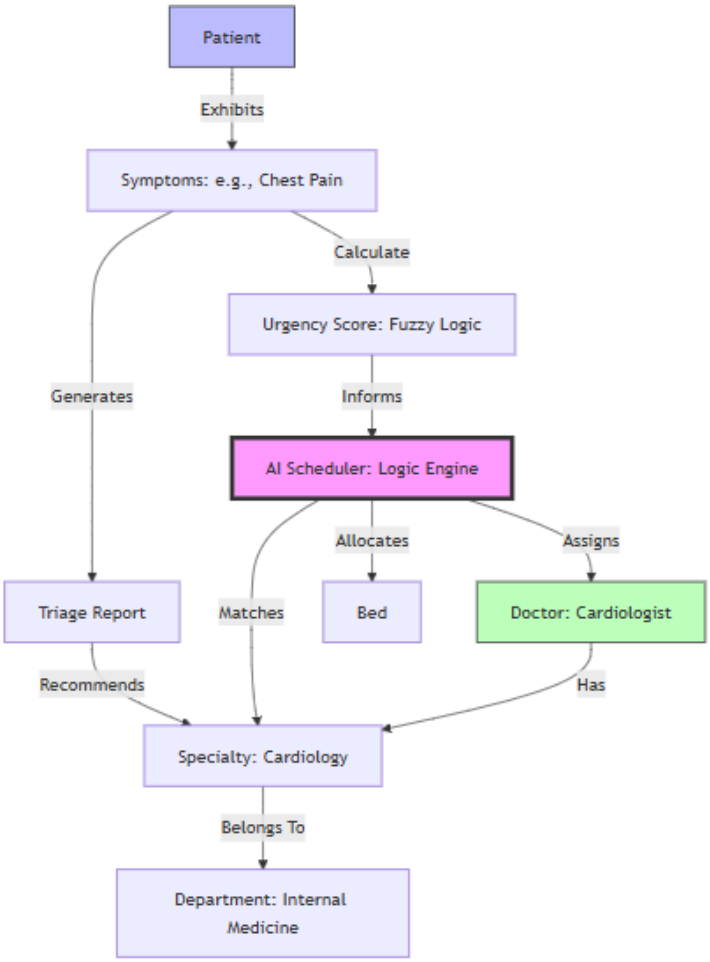

Figure 3.2: Semantic Network of the Medi-Match Knowledge Base

The diagram above illustrates the Semantic Network used by the Medi-Match AI. This network represents the 'Knowledge' the system possesses about the medical domain.

- **Triage Path:** The system captures patient symptoms to generate a Triage Report, which uses a rule-based approach to recommend a specialist.
- **Scheduling Path:** Symptoms are processed via a Fuzzy Logic membership function to determine an Urgency Score. This score informs the AI Scheduler (Logic Engine), which then performs a greedy heuristic or genetic optimization to assign a specific Doctor and allocate a Bed.
- **Hierarchical Mapping:** The network defines the structural relationship where a Cardiologist is a type of doctor that possesses a Specialty (Cardiology), which is categorized under the Department of Internal Medicine. This ontological structure ensures that patients are never assigned to the wrong clinical department."

8. Conclusion

The design of Medi-Match represents a sophisticated intersection of modern software engineering and classical AI. By combining the transparency of **Rule-Based Systems**, the nuance of **Fuzzy Logic**, and the optimization power of **Genetic Algorithms**, the system provides a robust solution to the most pressing problems in hospital resource management. The decoupled architecture ensures that as medical protocols evolve, the system can be updated with minimal disruption to the hospital's operational workflow.