The National Higher School of Artificial Intelligence

Introduction to AI Course Project

Spring 2025

Introduction to AI Project Paper

Project Title:

Predicting Athlete Performance and Injury Risk Using Search Techniques and CSP

Student list:

Full name	**Group**
Belbakhouche Akram Khaled (Team Leader)	04
Kadouci Abdelhak	08
Zerraf Bdreddine	04
Hamadi Mohammed Abdellah	04
Kerai Yassine	04
Blaha Ibrahim	04

Abstract

This report presents a group project that we named ”PERFOMAX”, developed as part of the Introduction to Artificial Intelligence Mini Project. The system uses physiological measurements and past performance data to forecast athlete outcomes and evaluate injury risks. It creates training plans that are optimally balanced between safety constraints and performance goals by utilizing machine learning models and intelligent search algorithms Constraint Satisfaction Problem solvers

Keywords: Injury Prediction, Athlete Monitoring, Optimization, Artificial Intelligence, Search Algorithms, ...

1 Introduction

PERFOMAX utilizes physiological metrics together with training chronological information to predict athlete performance and evaluate potential injury risks. Through the fusion of machine learning models with intelligent search algorithms, PERFOMAX produces training programs which improve performance levels and decrease both fatigue and injury potential. Through the use of real-world data and constraint-based reasoning, the system helps athletic managers make better decisions about athlete management.

2 State of the Art

Modern athlete monitoring together with performance optimization and injury prevention largely depend on artificial intelligence and data-driven methods. Research and development teams concentrate on converting extensive athlete data streams into practical performance insights because competitive sports actively adopt new technology for minimal improvements. The current state of injury prediction faces various obstacles due to the infrequent nature of injury events that complicate accurate forecasting. Our dataset contains a notable disparity between injury labels and non-injury records. Tim Gabbett presented the Acute:Chronic Workload Ratio (ACWR) as a novel tool for measuring training stress in his foundational work published in 2016. Recent training load and chronic conditioning share a fundamental connection that serves as the basis for this concept. The model suggests that rapid acute load growth increases injury potential whereas gradual acute load growth builds resilience. The model became highly favored across sports teams for its user-friendly design and easy-to-understand nature. However, latest research indicates that ACWR appears limited when utilized independently. The stability of this ratio decreases under low chronic workload conditions and it fails to include personalized recovery data alongside key contextual factors which are sleep patterns and fatigue indicators and past injury cases. ACWR functions as a practical tool yet its ability to forecast outcomes remains inconsistent when evaluated against complex machine learning methods in various sports and test populations. In a nottable contribution, Rossi and colleagues from 2020 conducted an extensive study which evaluated supervised machine learning methods for professional football injury prediction. They used detailed GPS-derived training data with subjective wellness scores and game performance metrics to create decision trees and gradient boosting machines for injury prediction. The researchers demonstrated that machine learning systems produce better results than traditional statistical methods when using feature engineering techniques that incorporate rolling averages together with lag values and contextual flags. However, In their comprehensive database, injury events made up only a minor portion of all recorded data. The prediction of sports injuries faces a significant barrier due to this unequal distribution of class samples.

Machine learning models which process imbalanced datasets demonstrate a tendency to prioritize majority classes (non-injury cases) thus producing misleading high accuracy scores along with low recall rates for injury instances. The application of these models becomes especially challenging during practical use because missed injuries (false negatives) hold greater importance than falsely identified injury cases (false positives). When examining their extensive database the researchers discovered that injury events comprised only a small percentage of the total recorded information. The prediction of sports injuries encounters a significant challenge when class samples are distributed unevenly. Imbalanced datasets cause machine learning models to favor majority classes which leads to high accuracy rates but low recall rates for injury instances. Practical deployment of these models poses a special difficulty because false negatives have greater significance than false positives. When it comes to our system, PERFOMAX brings forward a significant development at this point. The system analyzes performance changes and injury risks through machine learning while incorporating these predictive results into a search-based planning model. The system utilizes various algorithms including A* Search, Greedy Search and CSP solvers to investigate multiple training schedules while considering fatigue and risk thresholds as physiological constraints. PERFOMAX differs from standard prediction systems because it takes an active approach to generate optimally designed training schedules which assist practitioners in transforming insights into concrete action steps. Perfecting AI models through predictive methods together with optimization and simulation tools enables PERFOMAX to deliver a complete solution which predicts results and provides secure improvement recommendations. The combination of analytics with planning capabilities forms the unique aspect of its functionality and establishes a fresh direction for AI support in athletic management.

3 Data Set Building Report

3.1 . Data Collection, Cleaning & Feature Engineering

The link to our data
We downloaded the raw dataset from the ResearchGate repository A large-scale multivariate soccer athlete health, performance, and position monitoring dataset". And we use the following files:

- **Daily Training Load** (`daily_load.csv`), per player.
- **Session Durations** (`session.json`), aggregated to daily `total_duration`.
- **Game Performance** (`game-performance.csv`), with offensive and defensive scores combined into `performance_metric`.
- **Injuries** (`injury.csv`), flagged per date.
- **Wellness** (fatigue, sleep duration, sleep quality, stress), each as wide CSVs.

Using Python's `zipfile` and pandas, we merge them based on player IDs and dates, imputed missing wellness values with column medians, and filled missing injury flags and session durations with zeros.

Feature engineering steps included:

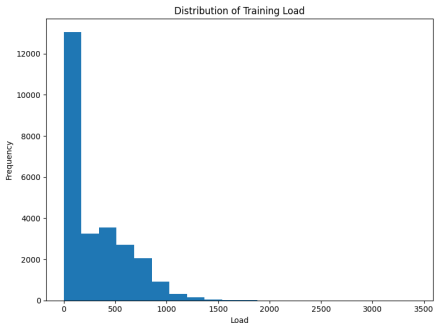
- *Action* labels: `Game` if a performance metric exists; `Rest` if load = 0; otherwise `Train_Low/Med/High` by load terciles.
- Rolling 7-day means for load and wellness; 1-day lags for load, injury flag, and performance.
- Cumulative injury counts, days since last game, and a rest-day indicator.
- Removal of any rest-only blocks longer than 90 days to exclude off-season gaps.

3.2 . Characteristics of the Dataset

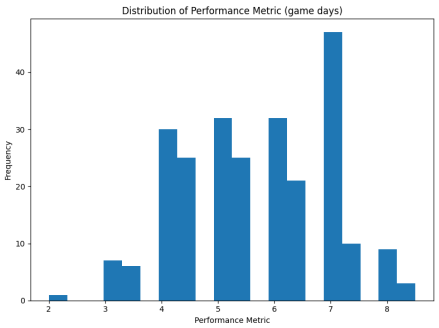
The dataset comprises 50 unique player IDs observed from 2020-01-01 to 2021-12-31, resulting in 26,089 player–day records. Each record includes player ID's, action, training loads, performance (recorded only on game days), injury indicators, session durations, wellness scores, rolling and lag features, and date.

3.3 . Summary Statistics & Visualizations

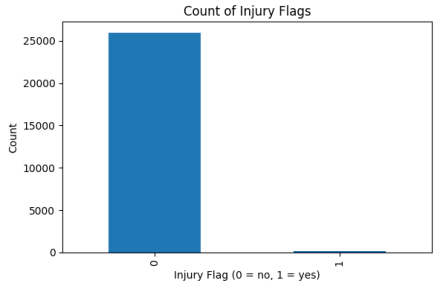
Feature	count	mean	std	min	25%	50%	75%	max
load	26089	272.45	321.28	0.0	150.0	180.0	490.0	3420.0
performance_metric	248	5.66	1.32	2.0	4.5	5.5	7.0	8.5
total_duration	26089	45.89	48.75	0.0	0.0	50.0	90.0	380



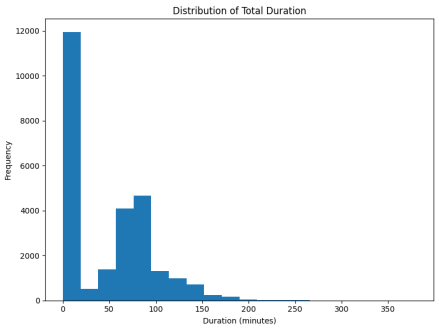
(a) Training Load



(b) Performance Metric (game days)



(c) Injury Flags



(d) Total Duration

3.4 . Use of the data

The data was used to train a RandomForestRegressor model of the performance, fatigue, and injury risk deltas to use them in search algorithms.

4 Problem Solving techniques (2.5 pages max)

4.1 Training Plan Optimization as a Search Problem

4.1.1 Search Problem Definition

- **State:** (day, fatigue, risk, performance, history)
- **Actions:** (intensity, duration) where intensity $\in \{0.0, 0.3, 0.6, 0.9\}$ and duration $\in \{0, 60, 120\}$
- **Transition Model:** ML-based models predict changes in fatigue (ΔF), performance (ΔP), and injury risk based on current state and action
- **Goal Test:** day = target_day **and** performance ≥ target_performance **and** fatigue ≤ max_fatigue **and** risk ≤ max_risk
- **Cost Function:**
  For training days: the cost is based on the gap between the new and old metrics.
  For rest days: the cost is based on recovery efficiency and performance decrease. This was done to ensure that the search does pick rest actions when running the algorithm.

4.2 Informed Search (and UCS) Evaluation functions

4.2.1 Heuristic function

The heuristic function considers the distance to the goal in terms of performance, fatigue and risk. For states that are already at the goal it is null. And for states that cannot possibly reach the goal within the current time frame, the heuristic gets significantly increased to stop the Algorithm from exploring this path.

4.2.2 Admissibility and Consistency

Since our informed algorithms use graph searching, then the consistency of the heuristic will imply its admissibility. We need to show that for any state s and its successor state s' reached by action a, the inequality $h(s) \leq c(s,a) + h(s')$ holds, where h is our heuristic function and c is our cost function.
We consider two cases:
Case 1: Terminal State (remaining_days$(s) = 0$**)** if the search meet a goal state, the heuristic will evaluate to 0, else it would penalize large performance gap. This will ensure the consistency of the heuristic
Case 2: Non-terminal State (remaining_days$(s) > 0$**)**

1. **Minimum Cost to Goal:** The minimum cost per day ($c_{\min} = 1.0$) is less than or equal to any actual action cost

2. **Reachability Penalty:** The reachability penalty is consistent because:

   - If a state cannot possibly provide a solution, all the branches from that state are pruned.
   - For reachable states, the penalty is 0.

3. **Risk Penalty:** The risk penalty in the cost function matches the heuristic penalty.

4. **Fatigue Penalty:** The fatigue penalty in the cost function matches the heuristic penalty.

This means that the heuristic applies the same penalties as the cost function, but takes the minimal cost (1 per day) in the search. This means that for any state s and its successor s' via action a:

$$h(s) \leq c(s,a) + h(s')$$

4.3 Uninformed Search Algorithms

Three uninformed search algorithms were run:

- DFS: Stack for frontier management, explores deepest paths first
- BFS: Queue for frontier management, explores shortest path
- UCS: Priority queue, finds optimal cost solution

Important design decisions:

- Rounding of state to reduce search space
- History tracking to prevent cycles
- Limiting depth to prevent infinite search
- Target parameters can be configured

4.4 Training Plan Optimization as a CSP

The approach uses backtracking search to find an optimal sequence of training activities that maximizes performance while respecting fatigue and injury risk constraints.

4.4.1 CSP Problem Definition

- **Variables:** training day starting from Training_1 through Training_n, where n is our target day.
- **Domains:** All the possible pairs of (intensity, duration)
- **State Representation:** (day, fatigue, risk, performance, history)
- **Constraints:** fatigue $\leq$ max_fatigue **and** risk $\leq$ max_risk
- **Objective:** maximizing the performance.

4.4.2 Heuristic Domain Value Ordering

The `_get_ordered_domain_values()` function orders actions (training options) by a heuristic value, this value prioritizes actions that maximize performance while balancing other constraints. The scoring is a weighted factor of performance gain, training efficiency, fatigue/risk headroom, recovery needs, and long term potential. The weights for each component were carefully chosen through data analysis and experimental validation to achieve an optimal balance. This function has the main role of maximizing the objective function (maximizing the performance).

4.5 Genetic Algorithm for Athlete Training Schedule Optimization

4.5.1 Genetic Algorithm Approach

Genetic algorithm evolves day-by-day training plans to boost performance while managing fatigue and injury through selection, crossover and mutation.

4.5.2 Genetic Algorithm Operators

Each individual is a chromosome—an action list of length `target_day`—evolved via rank-based selection to maintain diversity, random pairing for efficient mate choice, two-point crossover (with $p_c = 0.8$) to recombine schedule segments, and low-rate mutation ($p_m = 0.01$) that randomly flips daily actions to prevent premature convergence and balance exploration with exploitation.

5 Application Design (2.5 pages max)

5.1 Architecture Overview

With a Flask backend and a modern web frontend, the application uses a client-server architecture. With real-time data processing and visualization features, the system is made to give coaches and athletes access to training plan optimization algorithms through an easy-to-use interface.

5.2 Backend Implementation

5.2.1 Server Framework

Flask, a lightweight Python web framework, is used to build the backend. It manages user sessions, serves web pages, and responds to HTTP requests. Because of the framework's simplicity, integrating it with our search algorithms is simple and it still performs well. Flask's integrated session interface with safe secret keys for data encryption is used to implement session management.

5.2.2 Core Components

The application consists of three main components:

Authentication System:

- User registration and login functionality with session-based authentication
- Secure password management
- Currently using local storage with planned database migration

Search Algorithms Integration: - Informed and Uninformed search algorithm integration - CSP integration for training plan optimization - Genetic algorithm integration - Real-time algorithm process tracking

Data Management:

- Local storage for user's search history
- Dynamic performance tracking during search
- Future database integration planned for scalability

5.3 Frontend Implementation

5.3.1 User Interface

The frontend consists of several key pages: - Home page with project overview - Login/Signup interface - Dashboard with multiple views (overview, schedule, predictions, history) - Search interface for algorithm execution - Documentation page

The interface uses modern CSS for responsive design and includes interactive components for better user engagement.

5.3.2 Data Visualization

Chart.js is implemented for real-time data visualization:

- Line charts for performance progression
- Area charts for fatigue and risk levels

- Bar charts for training intensity distribution
- Radar charts for multi-dimensional metrics
- Real-time updates during algorithm execution

5.3.3 Search Interface and Real-time Updates

The search interface features:

- Configurable input parameters (target performance, days, etc.)
- Algorithm selection dropdown
- Real-time progress tracking

WebSocket Implementation:

- Establishes persistent connection between client and server
- Server sends algorithm progress updates (nodes explored, current state)
- Client receives updates and dynamically updates charts
- Progress bars and metrics update in real-time

5.4 Technologies Used

Backend:

- Python 3.x with Flask framework
- JSON for data exchange
- WebSocket for real-time communication
- Python's datetime for time management

Frontend:

- HTML5 and CSS3 for structure and styling
- JavaScript for dynamics
- Chart.js for data visualization

5.5 System Flow

- User authentication - Access to dashboard with multiple sections - Executing Algorithms through dynamic search page - Real-time results visualization - Performance monitor and performance tracker

5.6 Future Improvements

- Database integration for user data storage - Enhanced real-time visualization capabilities - Additional algorithm visualization options - Performance optimization for large datasets - Mobile application development

6 Results and Analysis

6.1 Comparison Metrics

To study the effectiveness of the algorithms, we have chosen the following metrics:

- **Execution time:** the time each algorithm took for a sample problem.
- **Output quality:** the actual schedule generated by the search algorithms.

6.2 Evaluation Criteria

Additionally, we analyze:

- **Rest/Training Balance:** The ratio and scheduling of recovery versus effort days.
- **Workout Intensity Distribution:** Spread of training intensities across the plan.
- **Workload:** The total workload across the training plan.

The following algorithms were tested: A*, Greedy best-first, Genetic, Uninformed Search(BFS) and CSP.
We chose the following parameters to run the searches:
Target day: 7
Target performance: 6.8
Injury and fatigue thresholds: 0.35 and 3 respectively.
CSP backtracking time limit: 120s
Initial state:
(5.5, 0.2, 1.5) for (performance, injury risk, fatigue)

6.3 Comparison and evaluation:

Greedy best-first and CSP provided their solutions in a faster time than other algorithms. A* did not take much longer in this test case, but for other conditions it has taken much longer. BFS and Genetic took the longest.

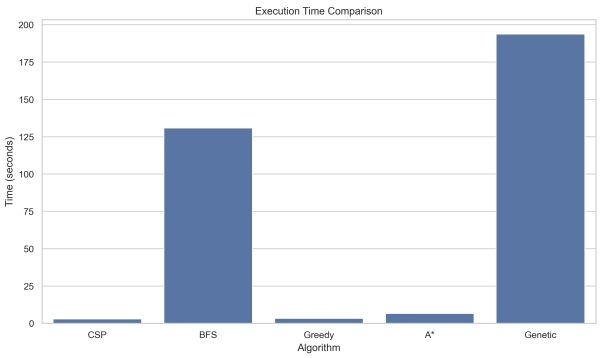


Figure 2: Comparison of execution times across algorithms

Schedule properties comparison summarizes key metrics including rest/training balance and workload distribution. A* and CSP solutions generally provide better-balanced schedules with appropriate rest days interspersed with training days, while maintaining effective workload progression.

As expected, even with the smaller depth the BFS search still explored the most amount of nodes, since it brute forces its way through the search space depth by depth. After that, the A* search explored a lot of nodes, but not nearly close to the uninformed search.

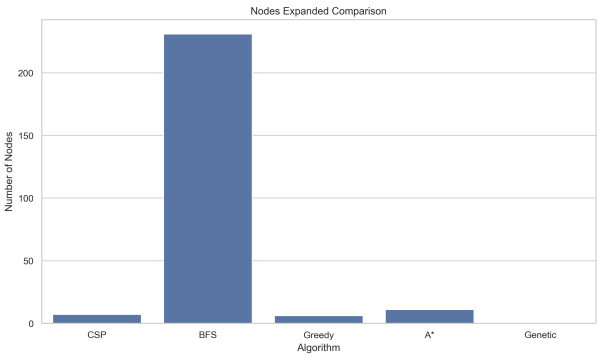


Figure 3: Number of nodes expanded during search

A* and genetic algorithms exploded way above the target, while the greedy algorithm barely reached it. CSP could not reach the target performance as the duration way short. But for other test cases it offered great solutions with fast runtime.

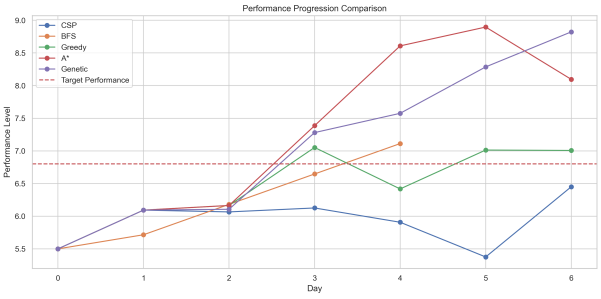


Figure 4: Performance progression across training days

The injury risk progression demonstrates how well each algorithm manages injury risk throughout the schedule, with optimal solutions maintaining risk below the 0.35 threshold.

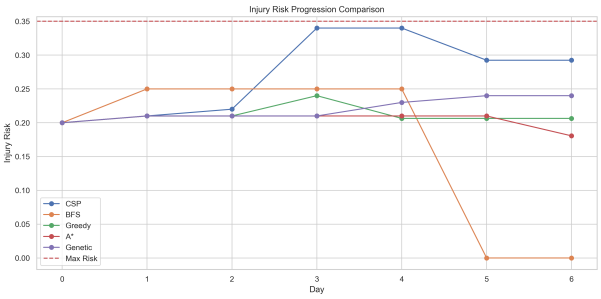


Figure 5: Injury risk progression across training days

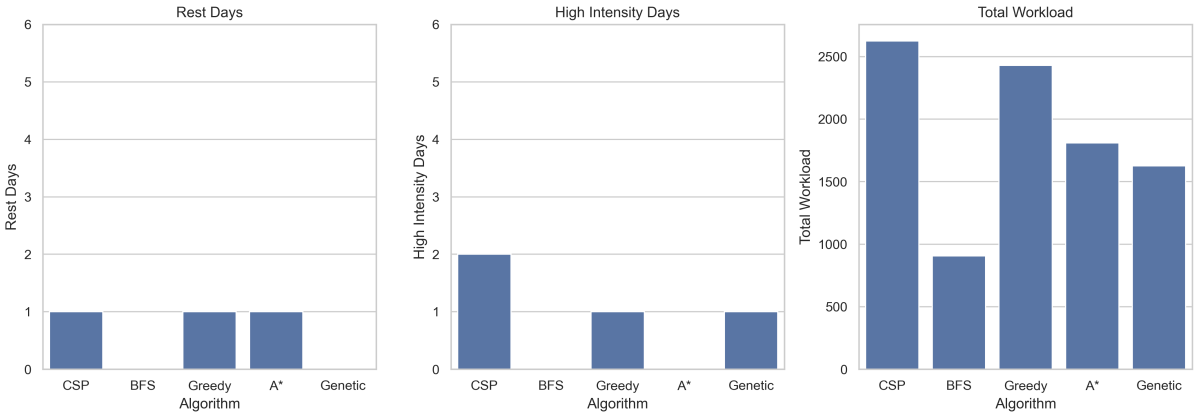


Figure 6: Evaluation of stats of algorithms: Key schedule properties by algorithm

Table 1: Algorithm Comparison Summary

Algorithm	**Quality**	**Performance**	**Usage**	**Execution Time (d = 30)**
A* Search	Optimal, balanced solution	High performance, low risk	Advanced users prioritizing quality	10 min (with caching)
Greedy	Good balanced solution	Fast execution time	Time-sensitive applications	2 min
Genetic	Great despite limitations	Strong results, customisable but requires time	Applications needing runtime/quality tradeoff	30 min (30 generations)
Uninformed	Passable solution	Very slow execution	Small problems requiring complete exploration	more than 2 hours
CSP	Great solutions for longer schedules	Good balance of metrics and speed	General-purpose reliable results	32 sec

7 Discussion

Experiment outcomes show that the algorithm selection greatly impacts performance and quality of training schedule produced by our PERFOMAX system. In terms of execution speed, output quality, and context applicability, each strategy has its own pros and cons.

7.1 Algorithm Performance

Among these tested algorithms, the Greedy best-first search and the constraint satisfaction problem solver algorithm CSP showed the best balance of speed and effectiveness. Greedy search's rapid results with reasonable scheduling, making it ideal for time constrained scenarios where a quick, feasible plan is acceptable. CSP, while slightly more limited in short time windows, generated schedules with excellent structure and constraint adherence, particularly in longer training horizons.

A* Search produced high-quality schedules with well-distributed rest and training days and consistently kept injury risk within safe thresholds. However, its computational cost increased substantially with larger state spaces or tighter constraints, making it less suitable for real-time or resource-limited environments unless caching or pruning techniques are applied.

Genetic algorithms yielded strong and customizable results, often exceeding target performance goals. However, they required significantly more time to converge, especially for higher-quality solutions. This makes them suitable for offline optimization or exploratory planning but less practical for immediate feedback.

Uninformed search (BFS), as expected, was the least efficient. Its exhaustive exploration guarantees completeness but comes at the cost of excessive computation time and memory usage. This method is only viable for small problem instances or for validation purposes where completeness is critical.

In terms of schedule quality, A*, CSP, and Genetic algorithms were generally more successful at managing workload progression and maintaining a healthy rest-to-training balance. These algorithms produced plans that respected physiological limits while gradually building toward the target performance. Notably, CSP was especially effective in managing injury risk, thanks to its constraint-driven design.

7.2 Limitations and Future Work

Despite these promising results, the current implementation has several limitations. The predictive models (ΔP, ΔF, ΔR) are dependent on the quality and balance of the input data. The underrepresentation of injury cases in our dataset, although handled with care, may still affect the generalization of injury risk predictions. Additionally, while the search algorithms simulate training outcomes well, they rely on static thresholds and fixed assumptions about sleep, stress, and recovery. Real-world variability—such as external factors, psychological state, or unmeasured wellness markers—is not yet incorporated.

For future work, we plan to integrate more personalized models using athlete-specific baselines and expand the dataset to better capture rare events such as injuries. Exploring hybrid approaches (e.g., combining CSP with heuristic guidance or integrating reinforcement learning) could further improve both solution quality and efficiency. Another avenue is to develop an interactive user interface for coaches and sports scientists, enabling real-time schedule adjustment and model transparency. Lastly, conducting real-world validation with athlete feedback will be key to refining PERFOMAX into a deployable decision-support system.

8 Conclusion

Sports science makes a substantial leap forward through the implementation of our system which solves the real-world problem of creating athlete-specific training programs. The system uses data analysis with advanced search algorithms A* and CSP and Genetic algorithms to create training schedules which achieve optimal performance results and minimize injury risks. Experimental findings show how selecting different algorithms leads to variations in both schedule quality and efficiency levels; CSP together with Greedy best-first search provide the best compromise for practical applications while A* delivers superior performance in top-quality constraint-based planning scenarios. Our project is currently expanding and can easily adapt from new contributes as well as account for new ideas and improvements

9 References

Gabbett, T. J. (2016). The training–injury prevention paradox: Should athletes be training smarter and harder? *British Journal of Sports Medicine, 50*(5), 273–280. https://doi.org/10.1136/bjsports-2015-095788

Rossi, A., Perri, E., Pappalardo, L., Cintia, P. (2020). Effective injury forecasting in soccer with GPS training data and machine learning. *PLOS ONE, 15*(6), e0233180. https://doi.org/10.1371/journal.pone.0233180

OpenAI. (2023). *ChatGPT* (version) [Large language model]. https://chat.openai.com/chat

ResearchGate. (2024). A large-scale multivariate soccer athlete health, performance, and position monitoring dataset. https://www.researchgate.net/publication/381008141_A_large-scale_multivariate_soccer_athlete_health_performance_and_position_monitoring_dataset

The link to our data: https://drive.google.com/drive/folders/1ZHqc8W9ctrbH6VLQajVcR0kamPIEACXS?usp=drive_link

Appendix A: A few screen shots of your system with one brief sentence explaining each.

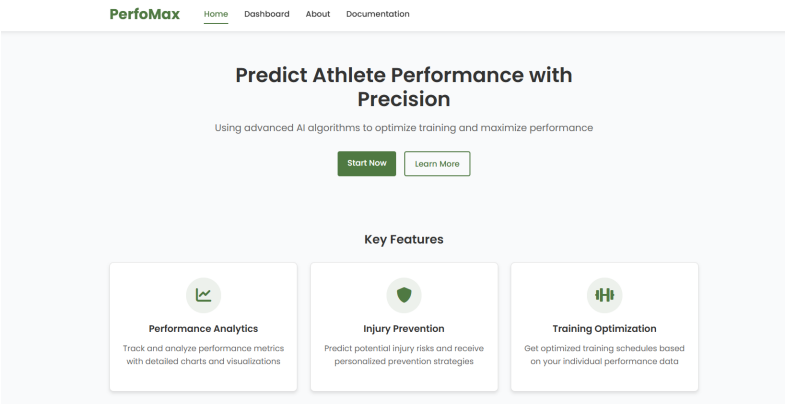


Figure 7: The initial landing page where users can log in to access the athlete monitoring system.

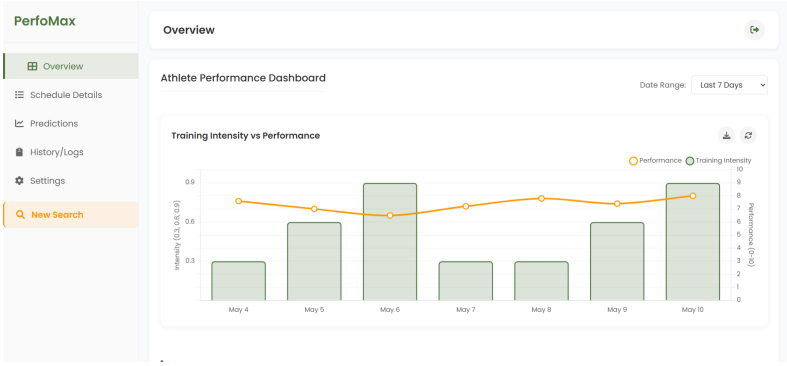


Figure 8: The main dashboard providing an overview of key metrics and athlete status at a glance.

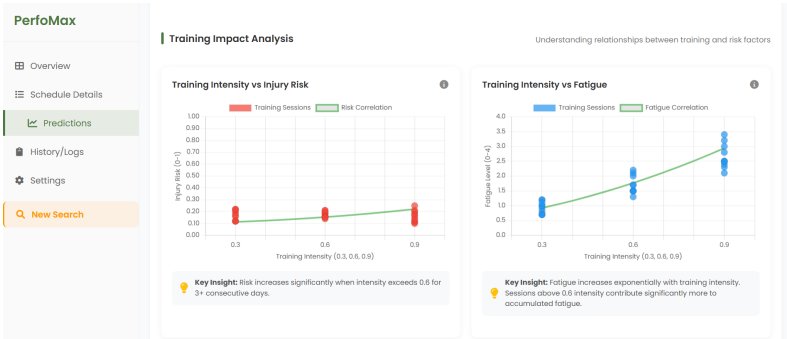


Figure 9: Interactive visualization interface allowing coaches to analyze the relationship between injury risk factors and athletic performance.

Appendix B:

- **Belbakhouche Akram Khaled**
  - A* implementation with caching
  - Genetic helper functions (random individual, fitness evaluation, etc.)
  - Tournament selection crossover
  - Cost design and implementation
  - Front-end implementation
- **Hamadi Mohammed Abdellah**
  - Node.py implementation
  - Problem.py implementation
  - Genetic logic and operators
  - Alternative model to speed up genetic algorithm
  - Front-end implementation
- **Blaha Ibrahim**
  - Heuristic design and implementation
  - Greedy Best-First implementation
  - Assisted in algorithm comparison, evaluation, and testing
  - Website design and documentation page
- **Zerraf Bdreddine**
  - Uninformed search implementation (BFS, DFS, UCS)
  - Assisted in algorithm comparison, evaluation, and testing
  - Data gathering
  - Website design and API integration assistance
- **Kerai Yassine**
  - Data preprocessing, cleaning, and visualization
  - Feature engineering
  - Model training and evaluation (performance, risk, fatigue)
  - Back-end implementation
- **Kadouci Abdelhack**
  - CSP definition and implementation
  - Design of test functions for search algorithms
  - Problem formulation
  - Back-end implementation