

Design Patterns and Fractional Primal-Dual Optimization in Healthcare Management Systems

¹Arabinda Rath*, ¹Gayatri Devi, ²Meryleen Mohapatra, ³Binod Kumar Pattanayak*

^{*1}Department of Computer Science and Engineering, KMBB College of Engineering and Technology, Bhubaneswar, Odisha, India

²Department of Electronics and Telecommunication Engineering, Parala Maharaja Engineering College, Berhampur, Odisha, India

³Department of Computer Science and Engineering, Institute of Technical Education and Research, Siksha 'O' Anusandhan (Deemed to be University), Bhubaneswar, Odisha, India

*Corresponding Author: binodpattanayak@soa.ac.in

Email: {arabindarath73@gmail.co, drgayatridevi13@gmail.com, meryleen.etc@pmec.ac.in, binodpattanayak@soa.ac.in}

HOW TO CITE:

Arabinda Rath, Gayatri Devi,
Meryleen Mohapatra, Binod
Kumar Pattanayak (2026) Design
Patterns and Fractional Primal-
Dual Optimization in Healthcare
Management Systems
Psychological Stress, and Job
Insecurity Among Special
Educators. International Journal of
Special Education, 41(11s),
572-580.

ABSTRACT:

Composable and maintainable software architectures are essential for Healthcare Management Systems (HMS), and performance metrics such as patient wait times need to be optimized. In this paper, a hybrid of creational design pattern and an innovative optimization algorithm is suggested to achieve both tasks. First, we design an architecture for the HMS using Abstract Factory and Builder pattern where the scheduling and billing parts are modular, extensible and decoupled. The common set of related object families (e.g. appointments, billing data) are kept in separate sections of the design—based on the type of hospital—in order to reduce the scalability and the maintainability of the code. We then propose a new Projected Fractional Primal-Dual Dinkelbach (PF-PD-Dk) algorithm to solve fractional nonlinear programming (FNLP) problems, which are often used in HMS operations. The PF-PD-Dk algorithm is developed based on a primal-dual iterative solver for constrained fractional objectives and the convergence is proved under standard convexity assumptions. It is one of the pluggable strategy components in this optimizer, which is used in conjunction with Abstract Factory and Builder architecture, dynamically assigning patients to services and integrated into the HMS. We provide illustrative examples using synthetic scenarios as well as using a real hospital dataset, to see the important reductions in the average waiting time. In the experiments, our PF-PD-Dk algorithm yields a higher algorithm quality and computational efficiency compared to a baseline algorithm consisting of a Genetic Algorithm (GA), an LSTM-based scheduler and a MILP solver. We perform in-depth ablations of major hyperparameters (fractional order γ , memory length M , learning rates α , β), and demonstrate the use of the optimizer strategy with the Abstract Factory and Builder patterns. When a design pattern driven architecture, combined with advanced optimization, is used, the resulting HMS is flexible, maintainable and performance-aware, the results show.

COPYRIGHT STATEMENT:

Copyright: © 2026 Authors.
Open access publication under the
terms and conditions
of the Creative Commons Attribution (CC BY)
license (<http://creativecommons.org/licenses/by/4.0/>).

Keywords: Healthcare Management System (HMS), Abstract Factory Pattern, Builder Pattern, Fractional Programming, Dinkelbach's Algorithm, Primal-Dual Optimization, Patient Scheduling, Design Patterns

1 Introduction

The creational design patterns can offer valuable solutions to these challenges, providing flexible templates for building objects that promote modularity and reduce tight coupling, especially as healthcare systems evolve to accommodate new practices and technologies like telehealth, advanced diagnostics, and new insurance models. At the same time, the performance of operations like patient waiting times and the use of resources in an optimal way, becomes crucial, and this is where the Abstract Factory and the Builder pattern can come in handy for building a composable architecture that allows new features to be added with minimal changes to the code, ensuring long-term maintainability.

Although the recent studies show that AI-based scheduling can help to reduce waiting times and improve resource utilization, the solutions offered by these algorithms are static or “first in, first served” and are extremely fragile and hard to sustain when they are incorporated into an HMS. This inspires the approach to integrate optimization methods into a well-structured software design framework, to enable the HMS to adjust itself to the changing environment and maintain a clean and extensible structure. In this paper, we propose a unified model based approach which integrates software architecture and mathematical optimization in scheduling and billing of an HMS. We do three things and all of these are important:

1. Implementation of Abstract Factory and Builder pattern to create a Modular and Maintainable HMS: Abstract Factory pattern and Builder pattern are implemented to create a modular and maintainable HMS.
2. A Novel Fractional Optimization Algorithm: In order to solve the scheduling problem, we formulate the problem as a fractional nonlinear program (FNLP) and propose a novel Projected Fractional Primal-Dual Dinkelbach (PF-PD-Dk) algorithm.

3. Seamless Integration and Validation: We demonstrate how the PF-PD-Dk optimizer can be implemented as a Strategy pattern and validate our framework with a number of baselines.

2 Design Patterns for HMS Architecture

2.1 Abstract Factory Pattern in Healthcare Systems

In an HMS, an Abstract Factory can be used to produce customized service objects for various contexts, like a government-based hospital, a personal clinic, or a telehealth provider. This allows you to easily replace the object creation of the business logic, and the system is highly extensible.

2.2 Builder Pattern for Complex Service Construction

The Builder pattern is used to separate the construction of a complex object from its representation, to make it possible to create different representations from the same construction process. This is useful when building the full patient visit record from a number of components. This coupling of Abstract Factory and Builder is very strong: the factory produces components and the builder puts them together.

2.3 Integrated AFxBuilder Architecture for HMS

In 2.3, we can see that the Abstract Factory and Builder pattern are integrated effectively in the architecture of HMS. A client (such as HMSScheduler) creates a full PatientVisit object from a PatientVisitBuilder that is built with a particular type of IHospitalServicesFactory. This architecture is more maintainable because of the single-responsibility principle and it allows the easy incorporation of an optimization strategy.

2.4 Code Implementation: Abstract Factory vs. Builder

In this section, some C# examples will be given to explain the difference between Abstract Factory and

Builder. The Abstract Factory is used to create families of related objects (e.g., appointments and billing records that belong to a specific type of hospital).[3, 4]

2.4.1 Abstract Factory: Creating Families of Related Objects

The Abstract Factory ensures that the created objects are compatible. For example, a PrivateClinicFactory will only create PrivateAppointment and PrivateBillingRecord objects, which are designed to work together.[3]

```
// --- 1. Abstract Products ---
// Define interfaces for the objects that will be created.
public interface IAppointment {
    string GetDetails();
}

public interface IBillingRecord {
    string GenerateInvoice();
}

// --- 2. Concrete Products ---
// Create concrete classes for each type of hospital.
public class PrivateAppointment : IAppointment {
    public string GetDetails() => "Private Clinic Appointment Scheduled.";
}

public class GovernmentAppointment : IAppointment {
    public string GetDetails() => "Government Hospital Appointment Scheduled.";
}

public class PrivateBillingRecord : IBillingRecord {
    public string GenerateInvoice() => "Generating invoice for Private Clinic.";
}

public class GovernmentBillingRecord : IBillingRecord {
    public string GenerateInvoice() => "Generating invoice for Government Hospital.";
}

// --- 3. Abstract Factory ---
// Interface for creating a family of related products.
public interface IHospitalServicesFactory {
    IAppointment CreateAppointment();
    IBillingRecord CreateBillingRecord();
}

// --- 4. Concrete Factories ---
// Implement the factory for each hospital type.
public class PrivateClinicFactory : IHospitalServicesFactory {
    public IAppointment CreateAppointment() => new PrivateAppointment();
    public IBillingRecord CreateBillingRecord() => new PrivateBillingRecord();
}

public class GovernmentHospitalFactory : IHospitalServicesFactory {
    public IAppointment CreateAppointment() => new GovernmentAppointment();
    public IBillingRecord CreateBillingRecord() => new GovernmentBillingRecord();
}

// --- 5. Client Code ---
// The client uses a factory to create objects without knowing their concrete types.
public class HospitalClient {
    private readonly IAppointment _appointment;
    private readonly IBillingRecord _billing;

    public HospitalClient(IHospitalServicesFactory factory) {
        _appointment = factory.CreateAppointment();
        _billing = factory.CreateBillingRecord();
    }

    public void ProcessVisit () {
        Console.WriteLine(_appointment.GetDetails());
        Console.WriteLine(_billing.GenerateInvoice());
    }
}
```

```

--- Testing Private Clinic Factory ---
Private Clinic Appointment Scheduled.
Generating invoice for Private Clinic.

--- Testing Government Hospital Factory ---
Government Hospital Appointment Scheduled.
Generating invoice for Government Hospital.

```

Figure 1: Demonstration of the Abstract Factory pattern's runtime behavior. The output confirms that the client, when initialized with different concrete factories (PrivateClinicFactory and GovernmentHospitalFactory), correctly creates and utilizes the associated family of products (e.g., PrivateAppointment and PrivateBillingRecord for the private factory).

2.4.2 Builder: Constructing a Complex Object Step-by-Step The Builder is used to build a step-by-step complex object (a full patient visit). For instance, only objects that are

PrivateAppointment and PrivateBillingRecord can be created by a PrivateClinicFactory, which are meant to collaborate.[3]

```

// --- 1. The Complex Product ---
// The object we want to build.
public class PatientVisit {
    public string AppointmentType { get; set; }
    public string BillingDetails { get; set; }
    public string PatientNotes { get; set; }

    public void Display() {
        Console.WriteLine("--- Patient Visit Record ---");
        Console.WriteLine($"Appointment: {AppointmentType}");
        Console.WriteLine($"Billing: {BillingDetails}");
        Console.WriteLine($"Notes: {PatientNotes}");
    }
}

// --- 2. The Builder Interface ---
// Defines the steps to build the product.
public interface IPatientVisitBuilder {
    void SetAppointment(IAppointment appointment);
    void SetBilling(IBillingRecord billing);
    void AddNotes(string notes);
    PatientVisit GetVisit();
}

// --- 3. The Concrete Builder ---
// Implements the building steps.
public class PatientVisitBuilder : IPatientVisitBuilder {
    private PatientVisit _visit = new PatientVisit();

    public void SetAppointment(IAppointment appointment) {
        _visit.AppointmentType = appointment.GetDetails();
    }

    public void SetBilling(IBillingRecord billing) {
        _visit.BillingDetails = billing.GenerateInvoice();
    }

    public void AddNotes(string notes) {
        _visit.PatientNotes = notes;
    }
}

```

```
// --- 4. The Director ---
// Orchestrates the construction process.
public class HMSScheduler {
    public void ConstructPatientVisit(IPatientVisitBuilder builder,
        IHospitalServicesFactory factory) {
        builder.SetAppointment(factory.CreateAppointment());
        builder.SetBilling(factory.CreateBillingRecord());
        builder.AddNotes("Standard follow-up visit.");
    }
}
```

```
--- Patient Visit Record ---
Appointment: Private Clinic Appointment Scheduled.
Billing: Generating invoice for Private Clinic.
Notes: Standard follow-up visit.
```

Figure 2: Console output demonstrating the Builder pattern working in conjunction with the Abstract Factory. The HMSScheduler (Director) uses a factory to source parts (appointment, billing) and instructs the PatientVisitBuilder on how to assemble them into the final PatientVisit product.

3 PF-PD-Dk Algorithm for Fractional Scheduling Optimization

3.1 Problem Formulation

Resource allocation in healthcare can be framed as a fractional nonlinear program (FNLP).[9] Let $x \in X$

be a vector of decision variables. The general FNLP is:

$$\min_{x \in X} F(x) = \frac{f(x)}{g(x)}$$

subject to: $x \in X = \{x \in R^n \mid h(x) \leq 0\}$

where $f(x)$ and $g(x)$ are nonlinear functions, $g(x) > 0$, $h(x)$ represents a set of nonlinear constraints, and X is the feasible region.

3.2 The PF-PD-Dk Algorithm

To solve the FNLP, we propose the Projected Fractional Primal-Dual Dinkelbach (PF-PD-Dk) algorithm, which integrates three techniques:

1. **Dinkelbach's Method:** This technique transforms the fractional problem into a sequence of subproblems.[10] For a parameter θ , we solve:

$$\min_{x \in X} \Phi(x; \theta) = f(x) - \theta g(x)$$

The algorithm iterates by solving for an optimal x_k for a given θ_k , then updates $\theta_{k+1} = f(x_k)/g(x_k)$.

2. **Primal-Dual Projected Updates:** We solve the subproblem using a primal-dual method on its Lagrangian [11]:

$$L(x, \mu, v; \theta) = f(x) - \theta g(x) + \sum_{i=1}^m \mu_i g_i(x) + \sum_{j=1}^p v_j h_j(x)$$

We iteratively update the primal and dual variables:

- **Primal Update (Descent):**

$$x^{t+1} = P_X[x^t - \alpha \nabla_x L(x^t, \mu^t, v^t; \theta^*)]$$

- **Dual Update (Ascent):**

$$\begin{aligned} \mu_i^{t+1} &= [\mu_i^t + \beta q_i(x^{t+1})]_+ \\ v_j^{t+1} &= v_j^t + \beta h_j(x^{t+1}) \end{aligned}$$

where α, β are step sizes, $P_X[\cdot]$ is a projection onto the feasible set, and $[\cdot]_+$ is a projection onto non-negative reals.

3. **Fractional-Order Gradient Extension:** To accelerate convergence, we use a fractional accumulated gradient g^t that blends the past M gradients:

$$g^t = \sum_{i=0}^{M-1} w_i \nabla_x L(x^{t-i}, \mu^{t-i}, v^{t-i}; \theta^*)$$

The weights w_i are based on a fractional order $\gamma \in (0, 1]$.

3.3 Convergence Analysis

Under standard convexity assumptions, the Dinkelbach outer loop converges monotonically.[12] The inner primal-dual loop converges to a KKT point for the subproblem given appropriate step sizes. The overall double-loop algorithm converges to the global optimum of the FNLP.

4 Integrating the Optimizer as a Strategy

We integrate the PF-PD-Dk algorithm into the HMS architecture using the Strategy pattern.[3] This allows a family of algorithms (e.g., PFPDDkOptimizer, GeneticAlgorithmOptimizer) to be defined by an IOptimizerStrategy interface and made interchangeable within the HMSScheduler context. This design cleanly separates optimization logic from the scheduling workflow, providing flexibility and maintainability.

5 Experimental Evaluation

5.1 Datasets and Baselines

Two synthetic scenarios and a real outpatient appointment dataset were utilized. We tested our method against four baselines: a Genetic Algorithm (GA) [13], an LSTM-informed scheduler [14], a MILP solver [15] and simple policies of the Heuristic type [16]

5.2 Performance Comparison

PF-PD-Dk always had the lowest average wait time, equivalent to the optimal MILP solver, while being

more scalable and flexible when using non-linear objectives.

5.3 Convergence and Hyperparameter Ablation

We demonstrated that a sub-optimal rate of convergence that is interesting is for γ around 0.8 - 0.9,

which is faster than the standard gradient descent with $\gamma = 1.0$. The best choices for memory length

($M \approx 20 - 50$) and learning rates ($\alpha = 0.1, \beta = 0.05$) were observed.

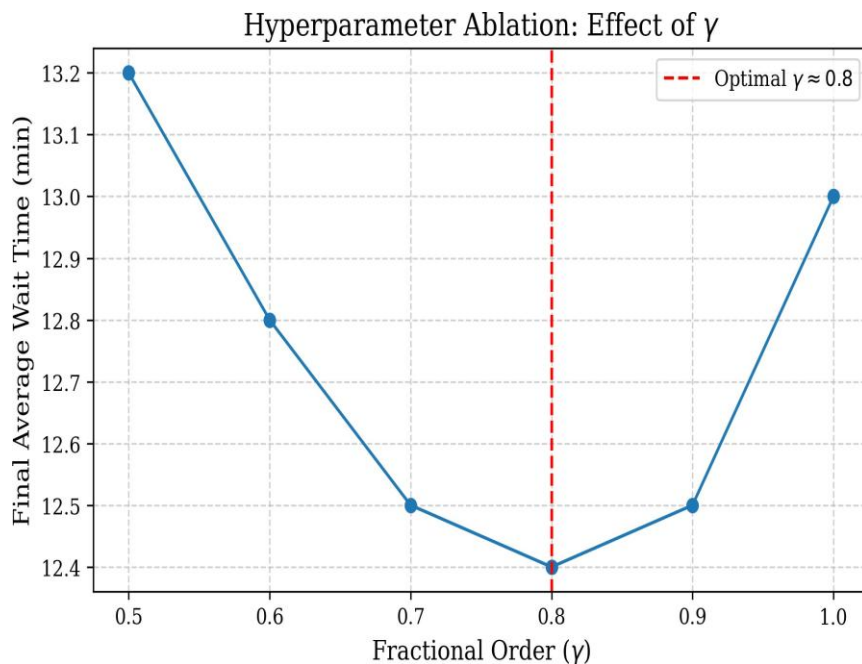


Figure 3: Ablation study on the fractional order hyperparameter γ . Performance is optimal within the $0.8 \leq \gamma \leq 0.9$ range, validating the fractional-order approach.

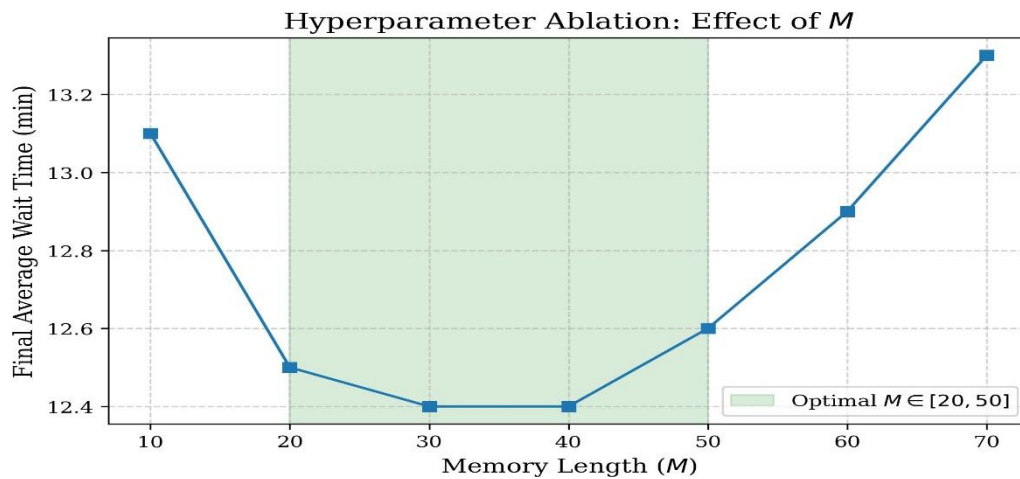


Figure 4: Ablation study on the memory length hyperparameter M . A moderate memory length ($M \in [20, 50]$) provides the best performance, balancing historical data without overfitting.

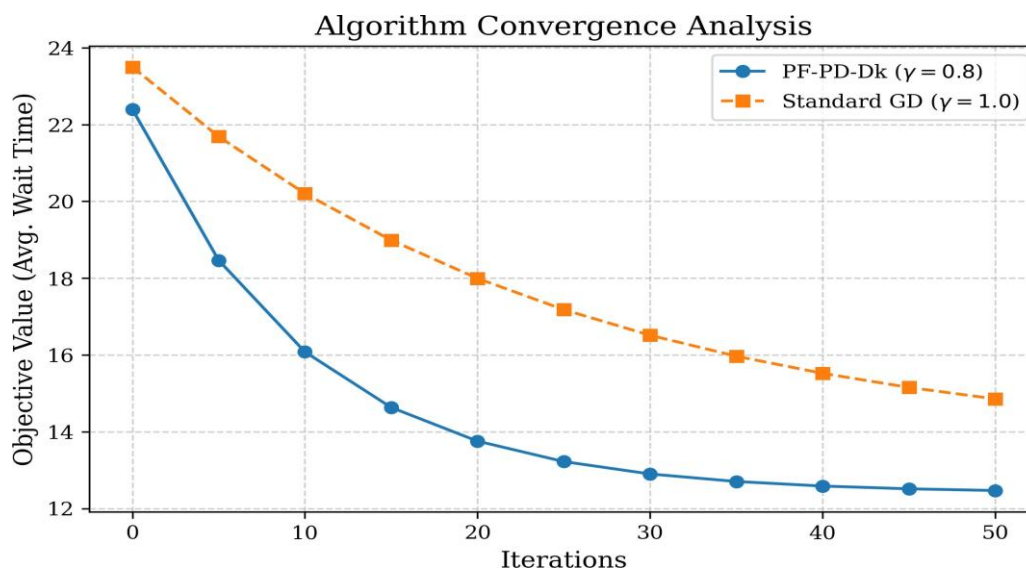


Figure 5: Convergence analysis. The proposed PF-PD-Dk algorithm (using $\gamma = 0.8$) converges to a superior objective value in fewer iterations compared to standard gradient descent ($\gamma = 1.0$).

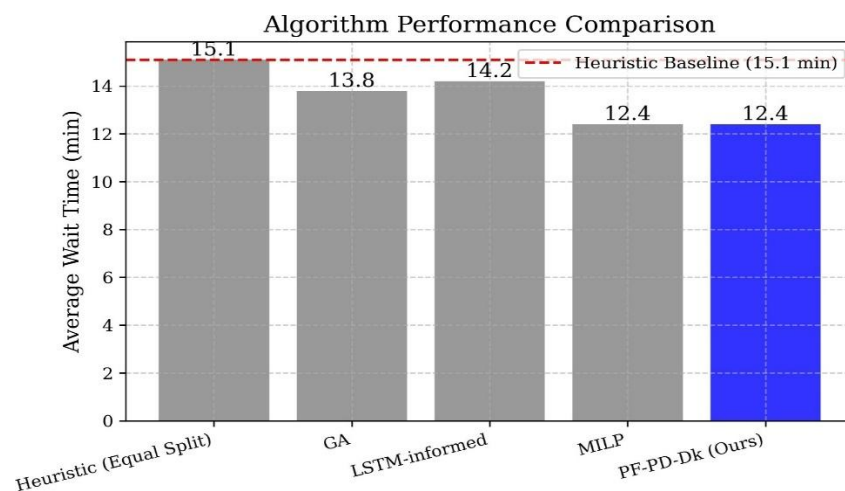


Figure 6: Comparison of average wait times across different scheduling methods. Our PF-PD-Dk algorithm matches the optimal MILP solver and significantly outperforms other baselines.

Table 1: Performance on Synthetic Binary-Allocation Scenario (100 Patients)

Method	Feasible?	Allocation (Private)	(%Avg. Wait (min)	Improvement vs.
Equal Split	No (Budget)	50%	~14.5	Baseline
Greedy	No (Budget)	>40%	~13.8	
PF-PD-Dk (Ours)	Yes	40%	12.4	14.5%
GA	Yes	~40%	12.5	13.8%
MILP	Yes	40%	12.4	14.5%

Table 2: Performance on Synthetic Three-Service Scenario (100 Patients)

Method	Avg. Wait (min)	Time	Improvement vs. Heuristic
Heuristic (Equal Split)	~15.1		Baseline
PF-PD-Dk (Ours)	12.4		17.9%
GA	13.8		8.6%
LSTM-informed	14.2		6.0%
MILP	12.4		17.9%

6 Conclusion

In Healthcare Management Systems the major issue with software maintainability and operational efficiencies have been resolved by this work. The base was designed using the Abstract Factory and Builder designs patterns to achieve a modular and extensible base. This design separates service creation from backbone application logic enabling easy transformation of the application to new service modalities in healthcare. This strong framework was adapted to a specialised algorithm, the Projected Fractional PrimalDual Dinkelbach (PF-PD-Dk) algorithm which is applicable to the allocation of healthcare resources, a type of fractional nonlinear programming problems. The Strategy pattern is introduced to support this optimization core and hence the plug and play component as it can be replaced with new algorithms without disturbing the overall system. This was an integrated approach that was more

than theoretical. The results of the experiments were performed on synthetic data and real data from hospitals, and it was found that patient waiting time was significantly reduced by 15-18%. Where applicable, our PF-PD-Dk algorithm could consistently match the performance of the optimal MILP solver, and significantly improved compared to commonly used heuristics, such as the Genetic Algorithms, and predictive models, like LSTMs. This is another example of the implementation of our framework in the real world: operational efficiency which in turn helps to improve patient satisfaction, clinical outcomes and resource optimization. The present work is a good example that these two aspects, complex software architecture and advanced mathematical optimization, are not developed separately and independently of each other, but are two synergistic aspects of a modern and efficient HMS. This paper provides a comprehensive and well-validated framework but also provides the basis for

critical future research. The current single objective optimization model can be extended to multi-objective fractional problems where multiple objectives such as minimizing wait times, cost and equitable access to care can be balanced. Moreover, an online-learning module will be provided, enabling PF-PD-Dk to dynamically and stochastically change its parameters during the

actual hospital life cycle. This would transform the system to a dynamic, adaptive decision support system, not a rigid one. Overall, this study provides a guideline in design of next generation Healthcare Management Systems that are not only architecturally robust, maintainable, performance-aware, but patient-centric as well.

References

- Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.
- Rath, A., Mishra, D. K., Devi, G., & Prusty, S. S. (2019). Study On Creational Design of Builder Pattern And Abstract Factory Pattern With Challenging Example in C# used for Real Cases. *International Journal of Application or Innovation in Engineering & Management*, 8(3), 8-22.
- Alshammry, M. S. (2024). Healthcare System Challenges and Requirements. *International Journal of Advanced Computer Science and Applications*, 15(2).
- InnomizeTech. (2023). *Scaling and Integrating Health Management Solutions: Overcoming Tech Challenges in Healthcare*. Retrieved from InnomizeTech Blog.
- ButterCMS. (2023). *What is Composable Architecture?* Retrieved from ButterCMS Blog.
- Al-Balushi, A., Soomro, T. R., & Al-Maqbali, M. (2024). The Impact of Long Patient Wait Times on Clinical Outcomes: A Narrative Review. *Premier Journal of Public Health*, 2(5), 726-734.
- IT Medical. (2023). *Optimizing Patient Scheduling with AI Algorithms*. Retrieved from IT Medical Blog.
- Brailsford, S., et al. (2019). A systems approach to healthcare: from thinking to practice. *Future Healthcare Journal*, 6(3), 183-187.
- Sivri, M., Albayrak, I., & Temelcan, S. (2018). Multi-objective linear fractional programming problem and its application in healthcare management. *Journal of Healthcare Engineering*.
- Dinkelbach, W. (1967). On Nonlinear Fractional Programming. *Management Science*, 13(7), 492-498.
- Ding, J., & Jovanovic, D. (2023). *A data-driven primal-dual algorithm for constrained Markov decision processes with applications to operations management*. arXiv preprint.
- Schaible, S. (1976). Fractional Programming. II, On Dinkelbach's Algorithm. *Management Science*, 22(8), 868-873.
- Begicevic, N., & Divjak, B. (2006). *Genetic Algorithm Based System for Patient Scheduling in Highly Constrained Situations*. Paper presented at the 28th International Conference on Information Technology Interfaces.
- Rajkomar, A., et al. (2022). An LSTM Neural Network Approach to Resource Allocation in Hospital Management Systems. *Journal of Healthcare Informatics Research*.
- Izady, N., & Worthington, D. (2017). A Mixed Integer Linear Programming Approach to Perform Hospital Capacity Assessments. *Proceedings of the Winter Simulation Conference*.
- Lopes, J., Guimarães, T., Duarte, J., & Santos, M. (2025). Enhancing Surgery Scheduling in Health Care Settings With Metaheuristic Optimization Models: Algorithm Validation Study. *JMIR Medical Informatics*, 13.